

App Service documentation

Azure App Service enables you to build and host web apps, mobile back ends, and RESTful APIs in the language stack of your choice without managing infrastructure, including .NET, Java (Java SE, Tomcat, and JBoss EAP), Node.js, Python, and PHP. It can also run any language stack with custom containers. It provides auto-scaling and high availability, supports both Windows and Linux, and enables automated deployments from GitHub, Azure DevOps, or any Git repo. Learn how to use Azure App Service with our quickstarts, tutorials, and samples.

About App Service

OVERVIEW

[Introduction to App Service](#)

CONCEPT

[Azure App Service plans](#)

[App Service Environments](#)

[Compare Azure hosting options for web apps](#)

WHAT'S NEW

[Azure updates ↗](#)

Create your first app

GET STARTED

[Getting started with Azure App Service](#)

QUICKSTART

[ASP.NET](#)

[Java](#)

[Node.js](#)

[Python](#)

[PHP](#)

[WordPress](#)

[Custom container](#)

Build a CRUD app



[ASP.NET and Azure SQL Database](#)

[ASP.NET Core and Azure SQL Database](#)

[Java Spring Boot and Azure Cosmos DB](#)

[Node.js and MongoDB](#)

[Python \(Django\) and PostgreSQL](#)

[PHP and MySQL](#)



[REST API](#)

Secure and deploy



[Security in Azure App Service](#)



[Secure with custom domain and certificate](#)

[Continuous deployment](#)

[Upload content with FTP](#)

Getting started with Azure App Service

Article • 02/06/2025

Azure App Service is a fully managed platform as a service (PaaS) for hosting web applications.

ASP.NET or ASP.NET Core

Use the following resources to get started with .NET.

[+] Expand table

Action	Resources
Create your first .NET app	Use one of the following tools: <ul style="list-style-type: none">- Visual Studio- Visual Studio Code- Command line- Azure PowerShell- Azure portal
Deploy your app	<ul style="list-style-type: none">- Configure ASP.NET- Configure ASP.NET core- GitHub actions
Monitor your app	<ul style="list-style-type: none">- Log stream- Diagnose and solve tool
Add domains & certificates	<ul style="list-style-type: none">- Map a custom domain- Add an SSL certificate
Connect to a database	<ul style="list-style-type: none">- .NET with Azure SQL Database- .NET Core with Azure SQL Database
Custom containers	<ul style="list-style-type: none">- Linux - Visual Studio Code- Windows - Visual Studio
Review best practices	<ul style="list-style-type: none">- Scale your app- Deployment- Security- Virtual Network

Next step

[Learn about App Service](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

App Service overview

Article • 01/24/2025

ⓘ Note

Starting June 1, 2024, newly created App Service apps can generate a unique default hostname that uses the naming convention <app-name>-<random-hash>. <region>.azurewebsites.net. Existing app names remain unchanged. For example:

myapp-ds27dh7271aah175.westus-01.azurewebsites.net

For more information, see [Unique Default Hostname for App Service Resource](#).

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, Node.js, Python, or PHP. Applications run and scale with ease on both Windows and [Linux](#)-based environments.

App Service adds the power of Microsoft Azure to your application, including improved security, load balancing, autoscaling, and automated management. Additionally, you can take advantage of its DevOps capabilities, such as continuous deployment from Azure DevOps, GitHub, Docker Hub, and other sources, package management, staging environments, custom domains, 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?

Azure App Service is a fully managed platform as a service (PaaS) offering for developers. 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, Node.js, Python, and PHP. 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 sidecar containers of your choice. 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 the global Microsoft datacenter infrastructure, and the App Service [SLA](#) promises high availability.
- **Connections to SaaS platforms and on-premises data** - Choose from [many hundreds of 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 Network](#).
- **Security and compliance** - App Service is [ISO](#), [SOC](#), and [PCI compliant](#). Create [IP address restrictions](#) and [managed service identities](#). Protect against [subdomain takeovers](#).
- **Authentication** - [Authenticate users](#) using the built-in authentication component. Authenticate users with [Microsoft Entra ID](#), [Google](#), [Facebook](#), [X](#), or [Microsoft accounts](#).
- **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.
- **Java tools integration** - Develop and deploy to Azure without leaving your favorite development tools, such as Maven, Gradle, Visual Studio Code, IntelliJ, and Eclipse.
- **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 a microservice architecture, consider [Azure Kubernetes 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 among these Azure services, see [Azure App Service, Azure Kubernetes Service, Virtual Machines, and other 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: .NET Core, Java (Tomcat, JBoss EAP, or Java SE with an embedded web server), Node.js, Python, and PHP. Run `az webapp list-runtimes --os linux` to view the latest languages and supported versions. If the runtime your application requires isn't 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're deprecated by the maintaining organization or found to have significant vulnerabilities. These options are hidden to guide customers to the latest runtimes, where they'll 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's 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 isn't supported on the [Shared](#) pricing tier.
- The Azure portal shows only features that currently work for Linux apps. As features are enabled, they're activated on the portal.
- When deployed to built-in images, your code and content are allocated a storage volume for web content, backed by Azure Storage. The disk latency of this volume is higher and more variable than the latency of the container filesystem. Apps that require heavy read-only access to content files might benefit from the custom

container option, which places files in the container filesystem instead of on the content volume.

App Service Environment

App Service Environment is an Azure App Service feature that provides a fully isolated and dedicated environment for running App Service apps with improved security at high scale. Unlike the App Service offering, where supporting infrastructure is shared, with App Service Environment, compute is dedicated to a single customer. For more information on the differences between App Service Environment and App Service, see the [comparison](#).

Next step

Create your first web app.

[Getting started](#)

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

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

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

[Node.js \(on Windows or Linux\)](#)

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

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

[HTML](#)

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

Feedback

Was this page helpful?

[!\[\]\(4146d17f71dced09c6ad789cacceaa6d_img.jpg\) Yes](#)

[!\[\]\(9db214d549b9aeebe72aa11d3a5c4b1a_img.jpg\) No](#)

App Service Environment overview

Article • 09/23/2024

An App Service Environment is an Azure App Service feature that provides a fully isolated and dedicated environment for running App Service apps securely at high scale. Unlike the App Service public multitenant offering where supporting infrastructure is shared, with App Service Environment, compute is dedicated to a single customer. For more information on the differences between App Service Environment and App Service public multitenant, see the [comparison of the offerings](#).

ⓘ Note

This article covers the features, benefits, and use cases of App Service Environment v3, which is used with App Service Isolated v2 plans.

An App Service Environment can host your:

- Windows web apps
- Linux web apps
- Docker containers (Windows and Linux)
- Functions
- Logic apps (Standard) - in [supported regions ↗](#)

App Service Environments are appropriate for application workloads that require:

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

An App Service Environment can host applications from only one customer, and they do so on one of their virtual networks. Customers have fine-grained control over inbound and outbound application network traffic. Applications can establish high-speed secure connections over VPNs to on-premises corporate resources.

Usage scenarios

App Service Environments have many use cases, including:

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

There are many networking features that enable apps in a multitenant App Service to reach network-isolated resources or become network-isolated themselves. These features are enabled at the application level. With an App Service Environment, no added configuration is required for the apps to be on a virtual network. The apps are deployed into a network-isolated environment that's already on a virtual network. If you really need a complete isolation story, you can also deploy your App Service Environment onto dedicated hardware.

Dedicated environment

An App Service Environment is a single-tenant deployment of Azure App Service that runs on your virtual network.

Applications are hosted in App Service plans, which are created in an App Service Environment. An App Service plan is essentially a provisioning profile for an application host. As you scale out your App Service plan, you create more application hosts with all the apps in that App Service plan on each host. A single App Service Environment v3 can have up to 200 total App Service plan instances across all the App Service plans combined. A single App Service Isolated v2 (Iv2) plan can have up to 100 instances by itself.

When you're deploying onto dedicated hardware (hosts), you're limited in scaling across all App Service plans to the number of cores in this type of environment. An App Service Environment that's deployed on dedicated hosts has 132 vCores available. I1v2 uses two vCores, I2v2 uses four vCores, and I3v2 uses eight vCores per instance. Only I1v2, I2v2, and I3v2 SKU sizes are available on App Service Environment deployed on dedicated hosts.

Virtual network support

The App Service Environment feature is a deployment of Azure App Service into a single subnet on a virtual network. When you deploy an app into an App Service Environment, the app is exposed on the inbound address that's assigned to the App Service Environment. If your App Service Environment is deployed with an internal virtual IP (VIP) address, the inbound address for all the apps is an address in the App Service

Environment subnet. If your App Service Environment is deployed with an external VIP address, the inbound address is an internet-addressable address, and your apps are in a public Domain Name System.

The number of addresses that are used by an App Service Environment v3 in its subnet varies, depending on the number of instances and the amount of traffic. Some infrastructure roles are automatically scaled, depending on the number of App Service plans and the load. The recommended size for your App Service Environment v3 subnet is a /24 Classless Inter-Domain Routing (CIDR) block with 256 addresses in it, because that size can host an App Service Environment v3 that's scaled out to its limit.

The apps in an App Service Environment don't need any features enabled to access resources on the same virtual network that the App Service Environment is in. If the App Service Environment virtual network is connected to another network, the apps in the App Service Environment can access resources in those extended networks. Traffic can be blocked by user configuration on the network.

The multitenant version of Azure App Service contains numerous features to enable your apps to connect to your various networks. With those networking features, your apps can act as though they're deployed on a virtual network. The apps in an App Service Environment v3 don't need any added configuration to be on the virtual network.

A benefit of using an App Service Environment instead of a multitenant service is that any network access controls for the App Service Environment-hosted apps are external to the application configuration. With the apps in the multitenant service, you must enable the features on an app-by-app basis and use role-based access control or a policy to prevent any configuration changes.

Feature differences

App Service Environment v3 differs from earlier versions in the following ways:

- There are no networking dependencies on the customer's virtual network. You can secure all inbound and outbound traffic and route outbound traffic as you want.
- You can deploy an App Service Environment v3 that's enabled for zone redundancy. You set zone redundancy only during creation and only in regions where all App Service Environment v3 dependencies are zone redundant. Zone redundancy is a deployment time only decision. Changing zone redundancy isn't possible after it has been deployed. With zone redundant App Service Environment, each App Service Plan on the App Service Environment needs to have a minimum of three instances so that they can be spread across zones. For

more information, see [Migrate App Service Environment to availability zone support](#).

- You can deploy an App Service Environment v3 on a dedicated host group. Host group deployments aren't zone redundant.
- Scaling is faster than with an App Service Environment v2. Although scaling still isn't immediate, as in the multitenant service, it's a lot faster.
- Front-end scaling adjustments are no longer required. App Service Environment v3 front ends automatically scale to meet your needs and are deployed on better hosts.
- Scaling no longer blocks other scale operations within the App Service Environment v3. Only one scale operation can be in effect for a combination of OS and size. For example, while your Windows small App Service plan is scaling, you could kick off a scale operation to run at the same time on a Windows medium or anything else other than Windows small.
- You can reach apps in an internal-VIP App Service Environment v3 across global peering. Such access wasn't possible in earlier versions.

A few features that were available in earlier versions of App Service Environment aren't available in App Service Environment v3. For example, you can no longer do these actions:

- Perform a backup and restore operation on a storage account behind a firewall.
- Access the FTPS endpoint using a custom domain suffix.

Pricing

With App Service Environment v3, the pricing model varies depending on the type of App Service Environment deployment you have. The three pricing models are:

- **App Service Environment v3:** If the App Service Environment is empty, there's a charge as though you have one instance of Windows I1v2. The one instance charge isn't an additive charge but is applied only if the App Service Environment is empty.
- **Zone redundant App Service Environment v3:** There's a minimum charge of 18 cores. There's no added charge for availability zone support if you have 18 or more cores across your App Service plan instances. If you have fewer than 18 cores across your App Service plans in the zone redundant App Service Environment, the difference between 18 cores and the sum of the cores from the running instance count is charged as additional Windows I1v2 instances.
- **Dedicated host App Service Environment v3:** With a dedicated host deployment, you're charged for two dedicated hosts per our pricing when you create the App

Service Environment v3 and then, as you scale, you're charged a specialized Isolated v2 rate per vCore. I1v2 uses two vCores, I2v2 uses four vCores, and I3v2 uses eight vCores per instance.

Note

Sample calculations for zone redundant App Service Environment v3 pricing:

1. Your zone redundant App Service Environment v3 has 3 Linux I1v2 instances in a single App Service plan.
 - An I1v2 instance has 2 cores.
 - In total, across your instances, you have 6 cores.
 - $18 \text{ cores} - 6 \text{ cores} = 12 \text{ cores}$
 - 12 cores is equivalent to 6 I1v2 instances.
 - You'll be charged for your 3 Linux I1v2 instances plus 6 additional Windows I1v2 instances.
2. Your zone redundant App Service Environment v3 has 3 Linux I2v2 instances in a single App Service plan.
 - An I2v2 instance has 4 cores.
 - In total, across your instances, you have 12 cores.
 - $18 \text{ cores} - 12 \text{ cores} = 6 \text{ cores}$
 - 6 cores is equivalent to 3 I1v2 instances.
 - You'll be charged for your 3 Linux I2v2 instances plus 3 additional Windows I1v2 instances.
3. Your zone redundant App Service Environment v3 has 4 Linux I3v2 instances in a single App Service plan.
 - An I3v2 instance has 8 cores.
 - In total, across your instances, you have 32 cores.
 - 32 cores is greater than 18 cores
 - You won't be charged for any additional cores.

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

Regions

App Service Environment v3 is available in the following regions:

Azure Public

 Expand table

Region	Single zone support	Availability zone support
Australia Central		
Australia Central 2	 *	
Australia East		
Australia Southeast		
Brazil South		
Brazil Southeast		
Canada Central		
Canada East		
Central India		
Central US		
East Asia		
East US		
East US 2		
France Central		
France South		
Germany North		
Germany West Central		
Israel Central		
Italy North		 **
Japan East		
Japan West		

Region	Single zone support	Availability zone support
Jio India Central	✓ **	
Jio India West	✓ **	
Korea Central	✓	✓
Korea South	✓	
Mexico Central	✓	✓ **
North Central US	✓	
North Europe	✓	✓
Norway East	✓	✓
Norway West	✓	
Poland Central	✓	✓
Qatar Central	✓ **	✓ **
South Africa North	✓	✓
South Africa West	✓	
South Central US	✓	✓
South India	✓	
Southeast Asia	✓	✓
Spain Central	✓	✓ **
Sweden Central	✓	✓
Sweden South	✓	
Switzerland North	✓	✓
Switzerland West	✓	
UAE Central	✓	
UAE North	✓	✓
UK South	✓	✓
UK West	✓	
West Central US	✓	

Region	Single zone support	Availability zone support
West Europe	✓	✓
West India	✓ *	
West US	✓	
West US 2	✓	✓
West US 3	✓	✓

* Limited availability and no support for dedicated host deployments.

** To learn more about availability zones and available services support in these regions, contact your Microsoft sales or customer representative.

Azure Government

[\[+\] Expand table](#)

Region	Single zone support	Availability zone support
US DoD Central	✓	
US DoD East	✓	
US Gov Arizona	✓	
US Gov Texas	✓	
US Gov Virginia	✓	✓

Microsoft Azure operated by 21Vianet

[\[+\] Expand table](#)

Region	Single zone support	Availability zone support
	App Service Environment v3	App Service Environment v3
China East 3	✓	
China North 3	✓	✓

In-region data residency

An App Service Environment only stores customer data including app content, settings, and secrets within the region where it's deployed. All data is guaranteed to remain in the region. For more information, see [Data residency in Azure](#).

Pricing tiers

The following sections list the regional pricing tiers (SKUs) availability for App Service Environment v3.

 Note

Windows Container plans currently do not support memory intensive SKUs.

Azure Public:

 Expand table

Region	Standard	Large	Memory intensive
	I1v2-I3v2	I4v2-I6v2	I1mv2-I5mv2
Australia Central			
Australia Central 2			
Australia East			
Australia Southeast			
Brazil South			
Brazil Southeast			
Canada Central			
Canada East			
Central India			
Central US		*	
East Asia			
East US			
East US 2			

Region	Standard	Large	Memory intensive
France Central	✓	✓	✓
France South	✓	✓	✓
Germany North	✓	✓	✓
Germany West Central	✓	✓	✓
Israel Central	✓	✓	
Italy North	✓	✓	
Japan East	✓	✓	✓
Japan West	✓	✓	✓
Jio India Central	✓	✓	
Jio India West	✓	✓	
Korea Central	✓	✓	
Korea South	✓	✓	✓
Mexico Central	✓	✓	
North Central US	✓	✓	✓
North Europe	✓	✓	✓
Norway East	✓	✓	✓
Norway West	✓	✓	✓
Poland Central	✓	✓	
Qatar Central	✓	✓	
South Africa North	✓	✓	✓
South Africa West	✓	✓	✓
South Central US	✓	✓	✓
South India	✓	✓	
Southeast Asia	✓	✓	✓
Spain Central	✓	✓	
Sweden Central	✓	✓	✓

Region	Standard	Large	Memory intensive
Sweden South	✓	✓	✓
Switzerland North	✓	✓	✓
Switzerland West	✓	✓	✓
UAE Central	✓	✓	
UAE North	✓	✓	✓
UK South	✓	✓	✓
UK West	✓	✓	✓
West Central US	✓	✓ *	
West Europe	✓	✓ *	
West India	✓	✓	
West US	✓	✓	✓
West US 2	✓	✓	✓
West US 3	✓	✓	✓

* Windows Container doesn't support Large skus in this region.

Azure Government:

[\[+\] Expand table](#)

Region	Standard	Large	Memory intensive
	I1v2-I3v2	I4v2-I6v2	I1mv2-I5mv2
US DoD Central	✓	✓ *	
US DoD East	✓	✓ *	
US Gov Arizona	✓	✓ *	
US Gov Texas	✓	✓ *	
US Gov Virginia	✓	✓ *	

Microsoft Azure operated by 21Vianet:

Expand table

Region	Standard	Large	Memory intensive
	I1v2-I3v2	I4v2-I6v2	I1mv2-I5mv2
China East 3		*	
China North 3		*	

Next steps

[Whitepaper on Using App Service Environment v3 in Compliance-Oriented Industries](#)

Feedback

Was this page helpful?

Yes

No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

Choose an Azure compute service

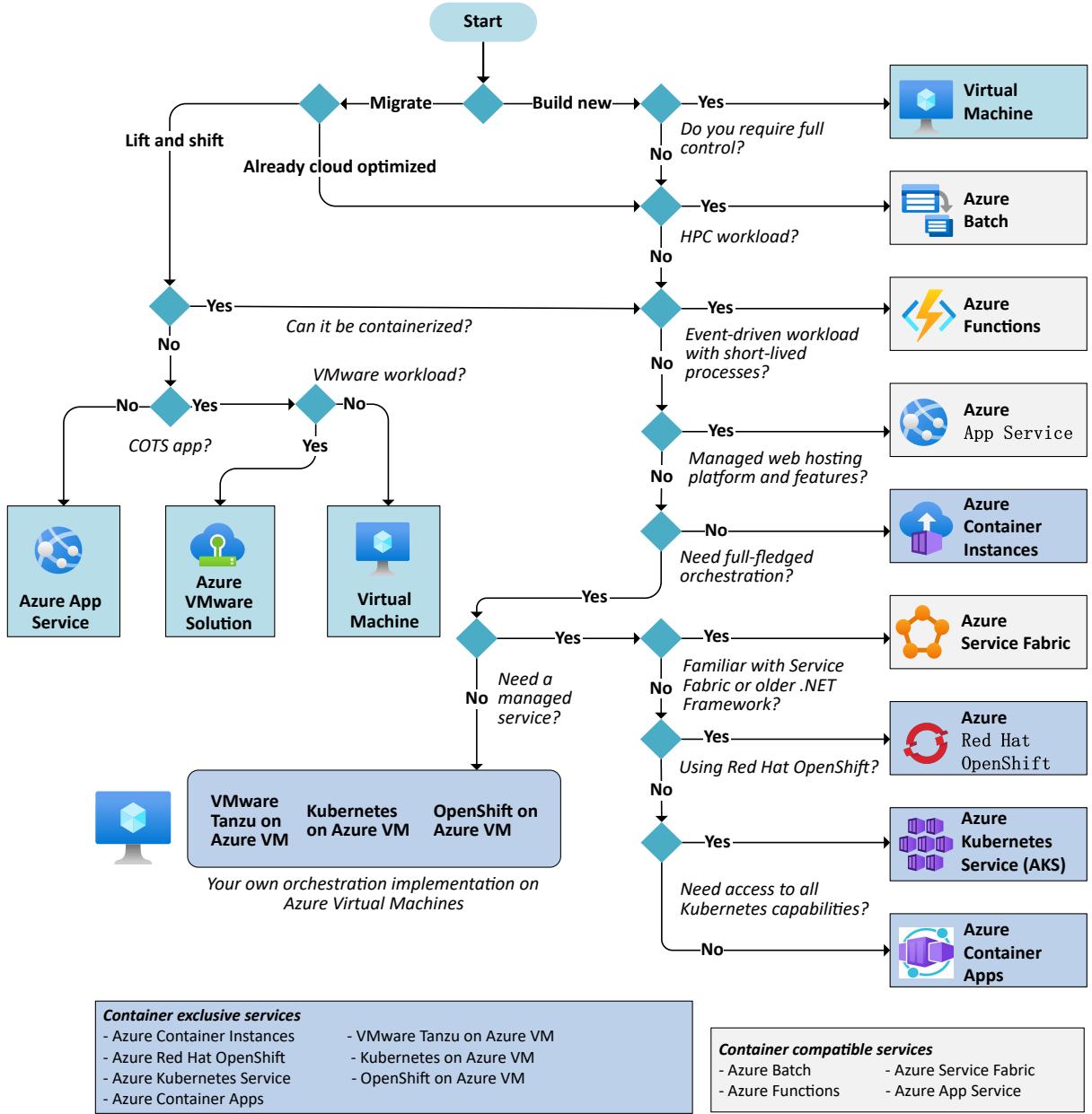
Azure App Service

Azure Kubernetes Service (AKS)

Azure offers many ways to host your application code. The term *compute* refers to the hosting model for the resources that your application runs on. This article helps choose a compute service for your application.

Choose a candidate service

Use the following flowchart to select a candidate compute service.



[Download a Visio file](#) of this decision tree.

This diagram refers to two migration strategies:

- **Lift and shift:** A strategy for migrating a workload to the cloud without redesigning the application or making code changes. It's also called *rehosting*. For more information, see [Azure migration and modernization center](#).
- **Cloud optimized:** A strategy for migrating to the cloud by refactoring an application to take advantage of cloud-native features and capabilities.

The output from this flowchart is your starting point. Next, evaluate the service to see if it meets your needs.

This article includes several tables that can help you choose a service. The initial candidate from the flowchart might be unsuitable for your application or workload. In that case, expand your analysis to include other compute services.

If your application consists of multiple workloads, evaluate each workload separately. A complete solution can incorporate two or more compute services.

Understand the basic features

If you're not familiar with the Azure service selected in the previous section, see this overview documentation:

- [Azure Virtual Machines](#): A service where you deploy and manage virtual machines (VMs) inside an Azure virtual network.
- [Azure App Service](#): A managed service for hosting web apps, mobile app back ends, RESTful APIs, or automated business processes.
- [Azure Functions](#): A service that provides managed functions that run based on a variety of trigger types for event-driven applications.
- [Azure Kubernetes Service \(AKS\)](#): A managed Kubernetes service for running containerized applications.
- [Azure Container Apps](#): A managed service built on Kubernetes, which simplifies the deployment of containerized applications in a serverless environment.
- [Azure Container Instances](#): This service is a fast and simple way to run a single container or group of containers in Azure. Azure Container Instances doesn't provide full container orchestration, but you can implement them without having to provision any VMs or adopt a higher-level service.
- [Azure Red Hat OpenShift](#): A fully managed OpenShift cluster for running containers in production with Kubernetes.
- [Azure Service Fabric](#): A distributed systems platform that can run in many environments, including Azure or on-premises.
- [Azure Batch](#): A managed service for running large-scale parallel and high-performance computing (HPC) applications.
- [Azure VMware Solution](#): A managed service for running VMware workloads natively on Azure.

Understand the hosting models

For hosting models, cloud services fall into three categories:

- **Infrastructure as a service (IaaS)**: Lets you provision VMs along with the associated networking and storage components. Then you can deploy whatever

software and applications you want onto those VMs. This model is the closest to a traditional on-premises environment. Microsoft manages the infrastructure. You still manage the VMs.

- **Platform as a service (PaaS):** Provides a managed hosting environment where you can deploy your application without needing to manage VMs or networking resources. Azure App Service and Azure Container Apps are PaaS services.
- **Functions as a service (FaaS):** Lets you deploy your code to the service, which automatically runs it. Azure Functions is a FaaS service.

① Note

Azure Functions is an [Azure serverless](#) compute offering. To see how this service compares with other Azure serverless offerings, such as Logic Apps, which provides serverless workflows, see [Choose the right integration and automation services in Azure](#).

There's a spectrum from IaaS to pure PaaS. For example, Azure VMs can automatically scale by using virtual machine scale sets. This capability isn't strictly a PaaS, but it's the type of management feature found in PaaS.

There's a tradeoff between control and ease of management. IaaS gives the most control, flexibility, and portability, but you have to provision, configure, and manage the VMs and network components you create. FaaS services automatically manage nearly all aspects of running an application. PaaS falls somewhere in between.

[+] Expand table

Service	Application composition	Density	Minimum number of nodes	State management	Web hosting
Azure Virtual Machines	Agnostic	Agnostic	1 ²	Stateless or stateful	Agnostic
Azure App Service	Applications, containers	Multiple apps per instance by using App	1	Stateless	Built in

Service	Application composition	Density	Minimum number of nodes	State management	Web hosting
Service plan					
Azure Functions	Functions, containers	Serverless 1	Serverless 1	Stateless or stateful ⁶	Not applicable
Azure Kubernetes Service	Containers	Multiple containers per node	3 ³	Stateless or stateful	Agnostic
Azure Container Apps	Containers	Serverless	Serverless	Stateless or stateful	Agnostic
Azure Container Instances	Containers	No dedicated instances	No dedicated nodes	Stateless	Agnostic
Azure Red Hat OpenShift	Containers	Multiple containers per node	6 ⁵	Stateless or stateful	Agnostic
Azure Service Fabric	Services, guest executables, containers	Multiple services per VM	5 ³	Stateless or stateful	Agnostic
Azure Batch	Scheduled jobs	Multiple apps per VM	1 ⁴	Stateless	No
Azure VMware Solution	Agnostic	Agnostic	3 ⁷	Stateless or stateful	Agnostic

Notes

1. If you're using a Consumption plan. For an App Service plan, functions run on the VMs allocated for your App Service plan. See [Choose the correct service plan for](#)

Azure Functions.

2. Higher service-level agreement (SLA) with two or more instances.
3. Recommended for production environments.
4. Can scale down to zero after job completes.
5. Three for primary nodes and three for worker nodes.
6. When using [Durable Functions](#).
7. Require minimum number of [three nodes](#).

Networking

[\[\] Expand table](#)

Service	Virtual network integration	Hybrid connectivity
Azure Virtual Machines	Supported	Supported
Azure App Service	Supported ¹	Supported ²
Azure Functions	Supported ¹	Supported ³
Azure Kubernetes Service	Supported	Supported
Azure Container Apps	Supported	Supported
Azure Container Instances	Supported	Supported
Azure Red Hat OpenShift	Supported	Supported
Azure Service Fabric	Supported	Supported
Azure Batch	Supported	Supported
Azure VMware Solution	Supported	Supported

Notes

1. Requires App Service Environment.
2. Use [Azure App Service Hybrid Connections](#).

3. Requires App Service plan or [Azure Functions Premium plan](#).

DevOps

[+] [Expand table](#)

Service	Local debugging	Programming model	Application update
Azure Virtual Machines	Agnostic	Agnostic	No built-in support
Azure App Service	IIS Express, others ¹	Web and API applications, WebJobs for background tasks	Deployment slots
Azure Functions	Visual Studio or Azure Functions CLI	Serverless, event-driven	Deployment slots
Azure Kubernetes Service	Minikube, Docker, others	Agnostic	Rolling update
Azure Container Apps	Local container runtime	Agnostic	Revision management
Azure Container Instances	Local container runtime	Agnostic	Not applicable
Azure Red Hat OpenShift	Minikube, Docker, others	Agnostic	Rolling update
Azure Service Fabric	Local node cluster	Guest executable, Service model, Actor model, Containers	Rolling upgrade (per service)
Azure Batch	Not supported	Command-line	Not applicable

Service	Local debugging	Programming model	Application update
application			
Azure VMware Solution	Agnostic	Agnostic	No built-in support

Notes

- Options include IIS Express for ASP.NET or node.js (iisnode), PHP web server, Azure Toolkit for IntelliJ, and Azure Toolkit for Eclipse. App Service also supports remote debugging of deployed web app.

Scalability

[\[\] Expand table](#)

Service	Autoscaling	Load balancer	Scale limit ³
Azure Virtual Machines	Virtual machine scale sets	Azure Load Balancer	Platform image: 1,000 nodes per scale set, Custom image: 600 nodes per scale set
Azure App Service	Built-in service	Integrated	30 instances, 100 with App Service Environment
Azure Functions	Built-in service	Integrated	200 instances per function app
Azure Kubernetes Service	Pod autoscaling ¹ , cluster autoscaling ²	Azure Load Balancer or Azure Application Gateway	5,000 nodes when using Uptime SLA
Azure Container Apps	Scaling rules ⁴	Integrated	15 environments per region (default limit), unlimited container apps per environment and replicas per

Service	Autoscaling	Load balancer	Scale limit ³
	container app (depending on available cores)		
Azure Container Instances	Not supported	No built-in support	100 container groups per subscription (default limit)
Azure Red Hat OpenShift	Pod autoscaling, cluster autoscaling	Azure Load Balancer or Azure Application Gateway	250 nodes per cluster (default limit)
Azure Service Fabric	Virtual machine scale sets	Azure Load Balancer	100 nodes per virtual machine scale set
Azure Batch	Not applicable	Azure Load Balancer	Core limit of 900 dedicated and 100 low-priority (default limit)
Azure VMware Solution	Built-in service ⁵	Integrated ⁶	Per VMware vCenter can manage between 3 ~ 16 VMware ESXi hosts

Notes

1. See [Autoscale pods](#).
2. See [Automatically scale a cluster to meet application demands on Azure Kubernetes Service](#).
3. See [Azure subscription and service limits, quotas, and constraints](#).
4. See [Set scaling rules in Azure Container Apps](#).
5. See [Scale a Azure VMware Solution](#).
6. See [VMware NSX](#).

Availability

[] Expand table

Service	Multiregion failover option
Azure Virtual Machines	Azure Traffic Manager, Azure Front Door, and cross-region Azure Load Balancer
Azure App Service	Azure Traffic Manager and Azure Front Door
Azure Functions	Azure Traffic Manager and Azure Front Door
Azure Kubernetes Service (AKS)	Azure Traffic Manager, Azure Front Door, and Multiregion Cluster
Azure Container Apps	Azure Traffic Manager and Azure Front Door
Azure Container Instances	Azure Traffic Manager and Azure Front Door
Azure Red Hat OpenShift	Azure Traffic Manager and Azure Front Door
Azure Service Fabric	Azure Traffic Manager, Azure Front Door, and cross-region Azure Load Balancer
Azure Batch	Not applicable
Azure VMware Solution	Not applicable

For guided learning on service guarantees, see [Core Cloud Services - Azure architecture and service guarantees](#).

Security

Review and understand the available security controls and visibility for each service:

- [Azure Windows virtual machine](#)
- [Azure Linux virtual machine](#)
- [Azure App Service](#)
- [Azure Functions](#)
- [Azure Kubernetes Service](#)
- [Azure Container Instances](#)
- [Azure Service Fabric](#)

- Azure Batch
- Azure VMware Solution

Other criteria

[] Expand table

Service	TLS	Cost	Suitable architecture styles
Azure Virtual Machines	Configured in VM	Windows  , Linux 	N-tier, big compute (HPC)
Azure App Service	Supported	App Service pricing 	Web-queue-worker
Azure Functions	Supported	Functions pricing 	Microservices, event-driven architecture
Azure Kubernetes Service (AKS)	Ingress controller	AKS pricing 	Microservices, event-driven architecture
Azure Container Apps	Ingress controller	Container Apps pricing 	Microservices, event-driven architecture
Azure Container Instances	Use sidecar container	Container Instances pricing 	Microservices, task automation, batch jobs
Azure Red Hat OpenShift	Supported	Azure Red Hat OpenShift pricing 	Microservices, event-driven architecture
Azure Service Fabric	Supported	Service Fabric pricing 	Microservices, event-driven architecture
Azure Batch	Supported	Batch pricing 	Big compute (HPC)
Azure VMware Solution	Configured in VM	Azure VMware Solution pricing 	VM workload based on VMware format

Consider limits and cost

Along with the previous comparison tables, do a more detailed evaluation of the following aspects of the candidate service:

- Service limits
- Cost ↗
- SLA ↗
- Regional availability ↗

Contributors

This article is maintained by Microsoft. It was originally written by the following contributors:

- Ayobami Ayodeji ↗ | Senior Program Manager
- Jelle Druyts ↗ | Principal Service Engineer
- Martin Gjoshevski ↗ | Senior Service Engineer
- Phil Huang ↗ | Senior Cloud Solution Architect
- Julie Ng ↗ | Senior Service Engineer
- Paolo Salvatori ↗ | Principal Service Engineer

To see nonpublic LinkedIn profiles, sign in to LinkedIn.

Next steps

[Core Cloud Services - Azure compute options](#). This Learn module explores how compute services can solve common business needs.

Related resources

- Choose an Azure compute option for microservices
- Lift and shift to containers with Azure App Service
- Technology choices for Azure solutions

Feedback

Was this page helpful?

 Yes

 No

Red Hat JBoss EAP on Azure

Article • 10/30/2024

This article describes the available solutions for hosting JBoss EAP on Azure, including the features and benefits of each option so you can choose the best one for your deployment.

There are three hosting options for JBoss EAP on Azure: App Service, Azure Red Hat OpenShift, and Azure Virtual Machines/VM Scale Sets. All three solutions are jointly developed and supported by Red Hat and Microsoft.

If you're interested in providing feedback or working closely on your migration scenarios with the engineering team developing JBoss EAP on Azure solutions, fill out this short [survey on JBoss EAP migration](#) and include your contact information. Our team of program managers, architects, and engineers will promptly get in touch with you to initiate close collaboration.

JBoss EAP on Azure App Service

Azure App Service is a fully managed platform for web and API applications, with built-in infrastructure maintenance, security patching, and scaling. App Service integrates with networking features such as virtual networks, Private Endpoints, and Hybrid Connections. This integration allows you to secure and isolate your infrastructure as necessary. You can deploy rapidly with GitHub Actions and Azure Pipelines integration, and monitor your applications with Azure Monitor Application Insights. For more information, see [Azure App Service overview](#).

JBoss EAP is available on the Linux variants of Premium v3 and Isolated v2 App Service plans. For more information about these plans, see [Azure App Service Pricing](#). The Isolated plans host your application in a private, dedicated Azure environment. You can purchase Premium v3 and Isolated v2 plans on a Pay-As-You-Go basis, or on one to three-year reservations to reduce costs up to 50%. For more information, see [What are Azure Reservations?](#) and [How reservation discounts apply to Azure App Service](#).

JBoss EAP is offered with versions 7.3 and 7.4 on App Service. As new versions of JBoss EAP are released by Red Hat, they're offered on App Service as part of the regular platform upgrades. For a full list of the minor versions available for JBoss EAP on Azure App Service, go to your JBoss EAP web app in the Azure portal, then select **Settings > Configuration > General Settings > Java Web Server Version**.

JBoss EAP on Azure App Service is jointly supported by Red Hat and Microsoft. When you open a support case on the Azure portal about your JBoss EAP apps, Azure support will automatically contact Red Hat technical support when necessary. This integrated support is provided to all JBoss EAP applications running on App Service, pricing information is available on the [Azure App Service Pricing](#) page. JBoss EAP sites can't opt out of the integrated support, but you can [purchase a reservation](#) for the integrated support to reduce costs.

https://www.youtube-nocookie.com/embed/8b_Wiuw8I-8

JBoss EAP on Azure Red Hat OpenShift

Azure Red Hat OpenShift provides highly available, fully managed OpenShift clusters on demand, monitored and operated jointly by Microsoft and Red Hat. If you're already using or planning to adopt containers/Kubernetes, deploying JBoss EAP on Azure Red Hat OpenShift is a compelling option. Red Hat and Microsoft provide a marketplace solution template that automates common boilerplate provisioning tasks to deploy JBoss EAP on Azure Red Hat OpenShift. The solution can automatically provision an Azure Red Hat OpenShift cluster, the JBoss EAP Operator, a sample application or your own application deployed using Source-to-Image (S2I) technology. You can launch the solution [JBoss EAP on Azure Red Hat OpenShift](#) from the Azure portal.

As an alternative to the solution template, Red Hat and Microsoft also provide a step-by-step guide on how to deploy JBoss EAP on Azure Red Hat OpenShift using Helm Charts instead of the Operator. For more information, see [Deploy a Java application with Red Hat JBoss Enterprise Application Platform \(JBoss EAP\) on an Azure Red Hat OpenShift 4 cluster](#).

JBoss EAP on Azure Virtual Machines

Virtual machines are a mature, proven migration path to the cloud that provides maximum flexibility and control. These factors are especially important for mission-critical workloads most suited to lift-and-shift migration. Microsoft and Red Hat provide robust options for migrating JBoss EAP workloads to Azure Virtual Machines. You can launch the solutions from the Azure portal to deploy the following resources:

- A [single JBoss EAP instance on Azure VM](#).
- A [static JBoss EAP cluster on Azure VMs](#) - that is, a JBoss EAP cluster on a fixed number of VMs, with or without domain mode enabled. This option is very similar to traditional on-premises JBoss EAP clusters.

- A [dynamic JBoss EAP cluster on Azure VM Scale Sets](#). Virtual machine scale sets provide groups of load-balanced virtual machines that can be scaled up or down in response to demand. For more information, see [Azure Virtual Machine Scale Sets](#). The JBoss EAP cluster is formed using Azure Ping and is suitable for stateful applications. This option doesn't support domain mode.

Azure solution templates help accelerate migrating JBoss EAP workloads. The solutions automatically provision several Azure resources to quickly create a JBoss EAP deployment on Azure Virtual Machines or virtual machine scale sets. The automatically provisioned resources include virtual network, storage, network security group, OpenJDK, Red Hat Enterprise Linux (RHEL), JBoss EAP, Azure App Gateway, and database connectivity (Azure SQL, Oracle Database, PostgreSQL, MySQL). The solutions support the latest versions of JBoss EAP 7, OpenJDK 8, and RHEL 8.

The offers require a JBoss EAP subscription and work on a Bring-Your-Own-Subscription (BYOS) basis. For the RHEL part of the offer, you have a choice to use either Pay-As-You-Go (PAYGO) or BYOS. In case of PAYGO, there's an extra hourly RHEL subscription charge for using the offer on top of the normal Azure compute, network and storage costs. To use RHEL BYOS, you must [contact Red Hat](#) to get your subscription enabled on Azure. Once you do so, the RHEL BYOS options will become visible as plans.

Next steps

The following articles provide more information on getting started with these technologies.

- [Quickstart: Create a Java app on Azure App Service](#)
- [Configure a Java app for Azure App Service](#)
- [Quickstart: Deploy a Java application with JBoss EAP on Azure Red Hat OpenShift](#)
- [Quickstart: Deploy a JBoss EAP cluster on Azure Virtual Machines \(VMs\)](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

Quickstart: Deploy an ASP.NET web app

Article • 01/13/2025

ⓘ Note

Starting June 1, 2024, newly created App Service apps can generate a unique default hostname that uses the naming convention `<app-name>-<random-hash>. <region>.azurewebsites.net`. Existing app names remain unchanged. For example:

`myapp-ds27dh7271aah175.westus-01.azurewebsites.net`

For more information, see [Unique Default Hostname for App Service Resource](#).

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

Alternatively, you can deploy an ASP.NET web app as part of a [Windows or Linux container in App Service](#).

💡 Tip

Find GitHub Copilot tips in the Visual Studio, Visual Studio Code, and Azure portal steps.

ⓘ Note

Building .NET 9 (STS) apps with Windows App Service using MSBuild or SCM_DO_BUILD isn't yet supported. Support for these build scenarios is planned after the initial general availability date and by December 4, 2024. Deployments that build outside of App Service through Visual Studio, Visual Studio Code, GitHub Actions, and Azure DevOps are fully supported.

Prerequisites

.NET 8.0

- An Azure account with an active subscription. [Create an account for free ↗](#).
- [Visual Studio 2022 ↗](#) with the **ASP.NET and web development** workload.
- **(Optional)** To try GitHub Copilot, a [GitHub Copilot account ↗](#). A 30-day free trial is available.

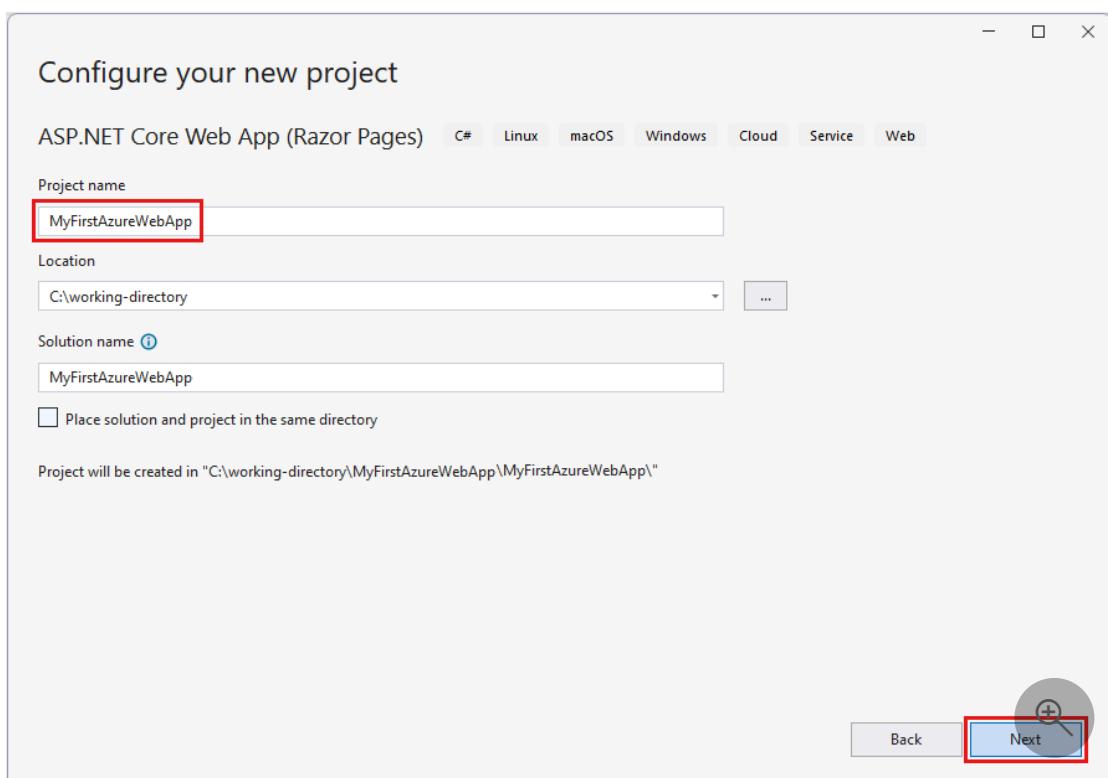
If you already installed Visual Studio 2022:

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

Create an ASP.NET web app

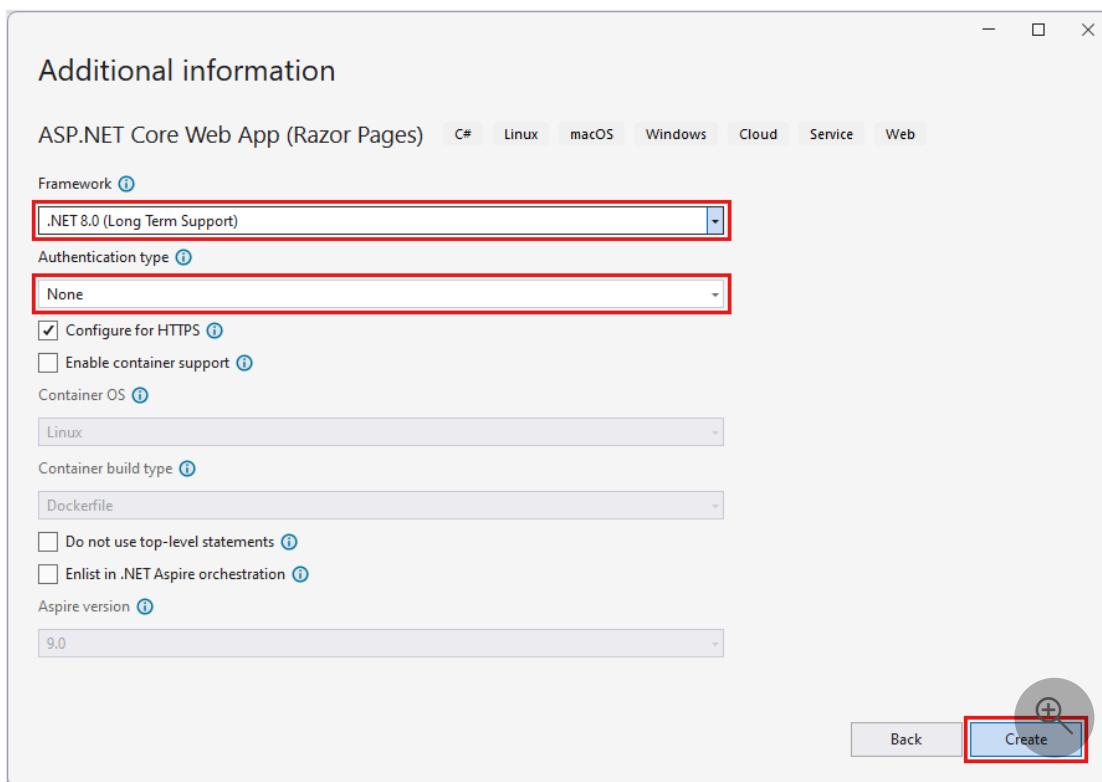
.NET 8.0

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

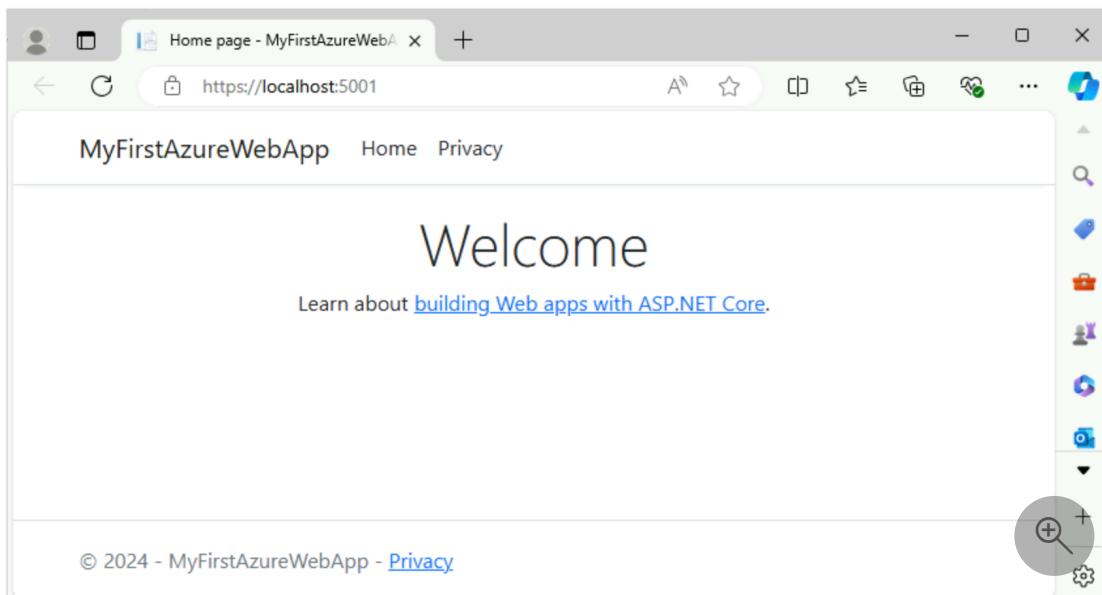


4. Select **.NET 8.0 (Long Term Support)**.

5. Ensure **Authentication type** is set to **None**. Select **Create**.



6. From the Visual Studio menu, select **Debug > Start Without Debugging** to run the web app locally. If you see a message asking you to trust a self-signed certificate, select **Yes**.



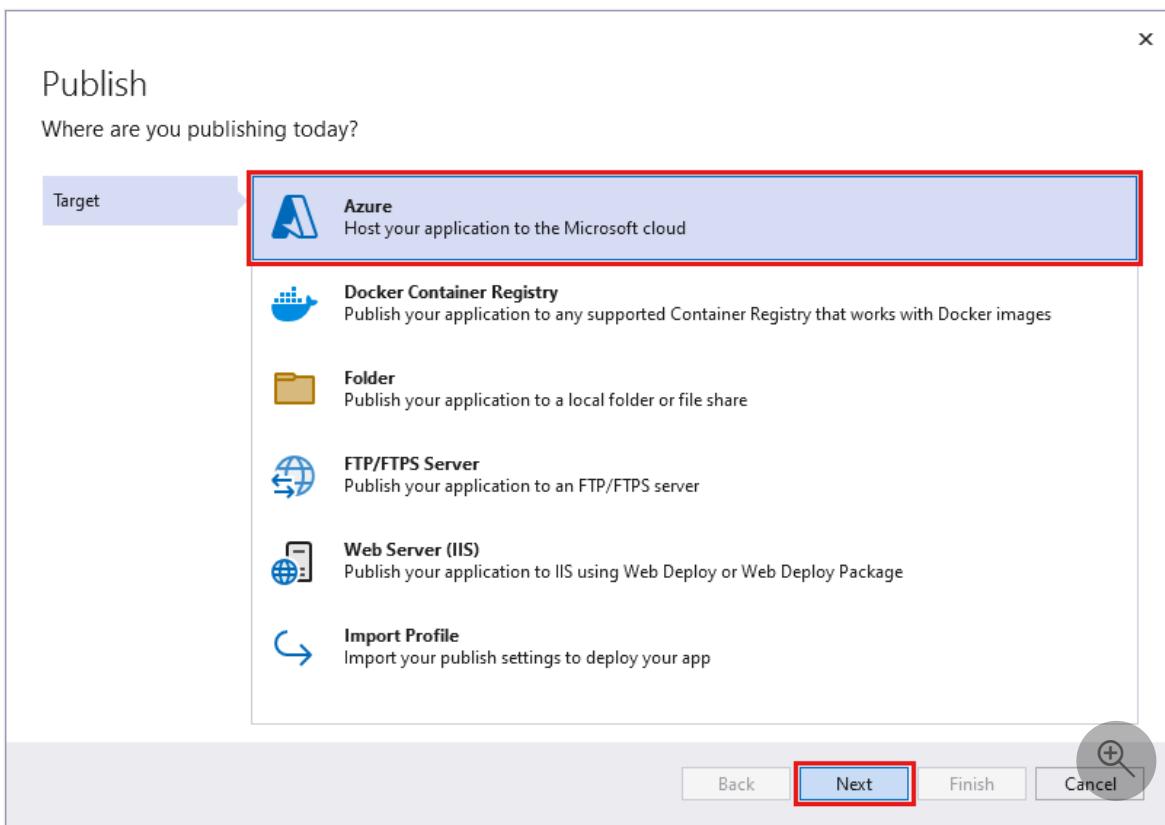
 **Tip**

If you have a GitHub Copilot account, try [getting GitHub Copilot features for Visual Studio](#).

Publish your web app

Follow these steps to create your App Service resources and publish your project:

1. In **Solution Explorer**, right-click the **MyFirstAzureWebApp** project and select **Publish**.
2. In **Publish**, select **Azure** and then **Next**.

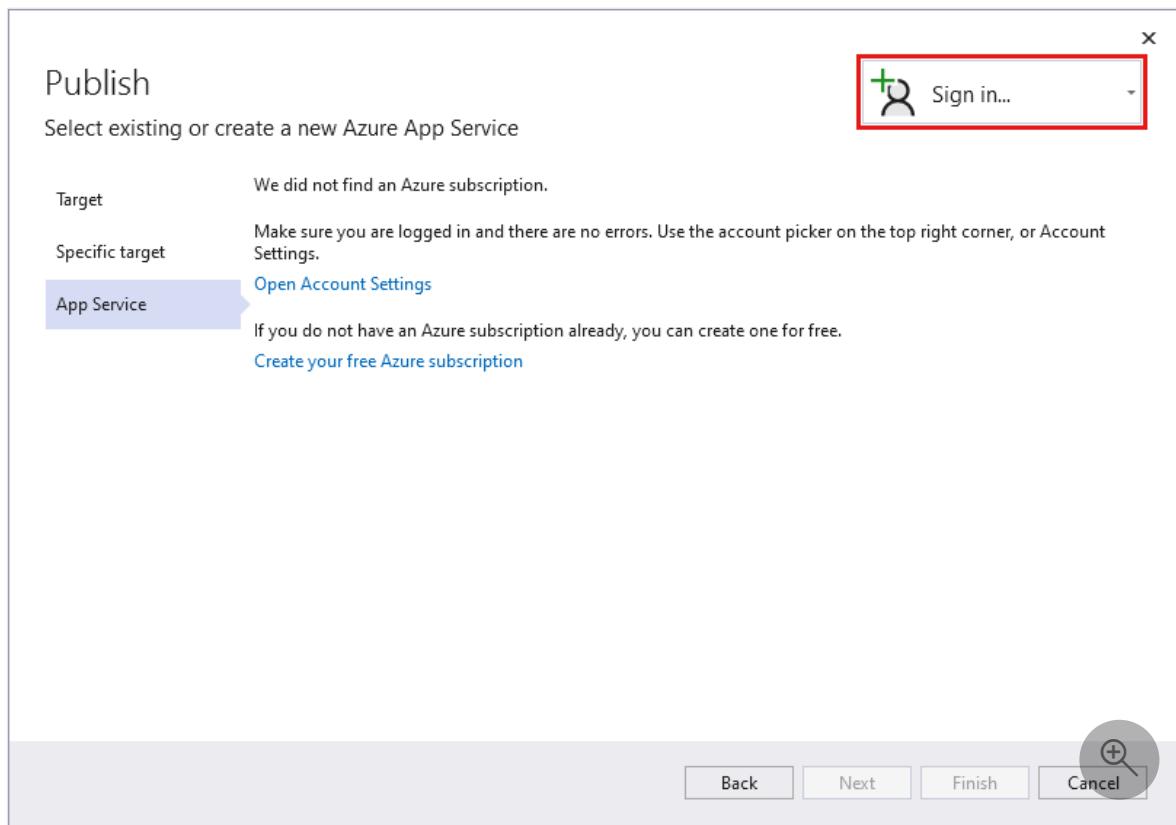


3. Choose the **Specific target**, either **Azure App Service (Linux)** or **Azure App Service (Windows)**. Select **Next**.

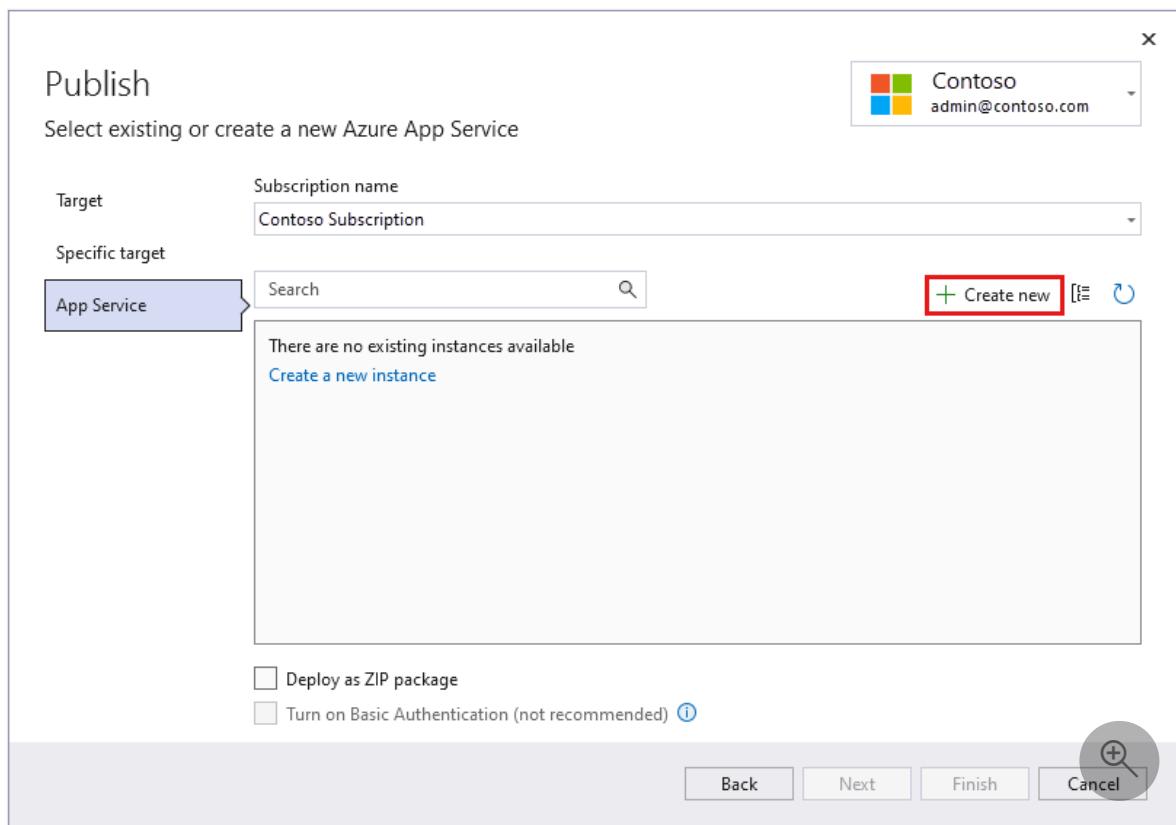
ⓘ Important

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

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



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



6. For **Subscription**, accept the subscription that is listed or select a new one from the drop-down list.
7. For **Resource group**, select **New**. In **New resource group name**, enter *myResourceGroup* and select **OK**.

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

9. In the **Hosting Plan: Create new** dialog, enter the values specified in the following table:

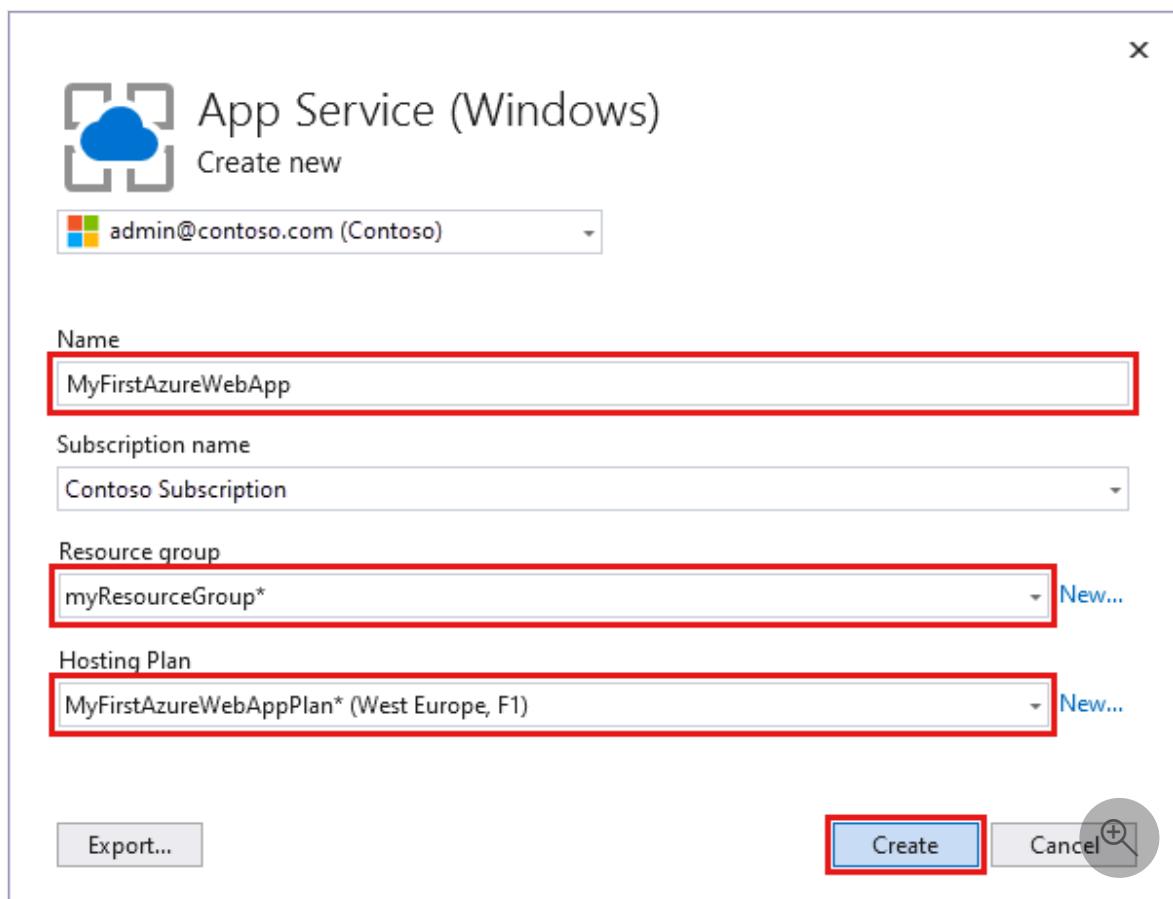
[Expand table](#)

Setting	Suggested value	Description
Hosting Plan	<i>MyFirstAzureWebAppPlan</i>	Name of the App Service plan.
Location	<i>West Europe</i>	The datacenter where the web app is hosted.
Size	Choose the lowest tier.	Pricing tiers define hosting features.

10. In **Name**, enter a unique app name. Include only characters `a-z`, `A-Z`, `0-9`, and `-`.

You can accept the automatically generated unique name. The URL of the web app is `http://<app-name>.azurewebsites.net`, where `<app-name>` is your app name.

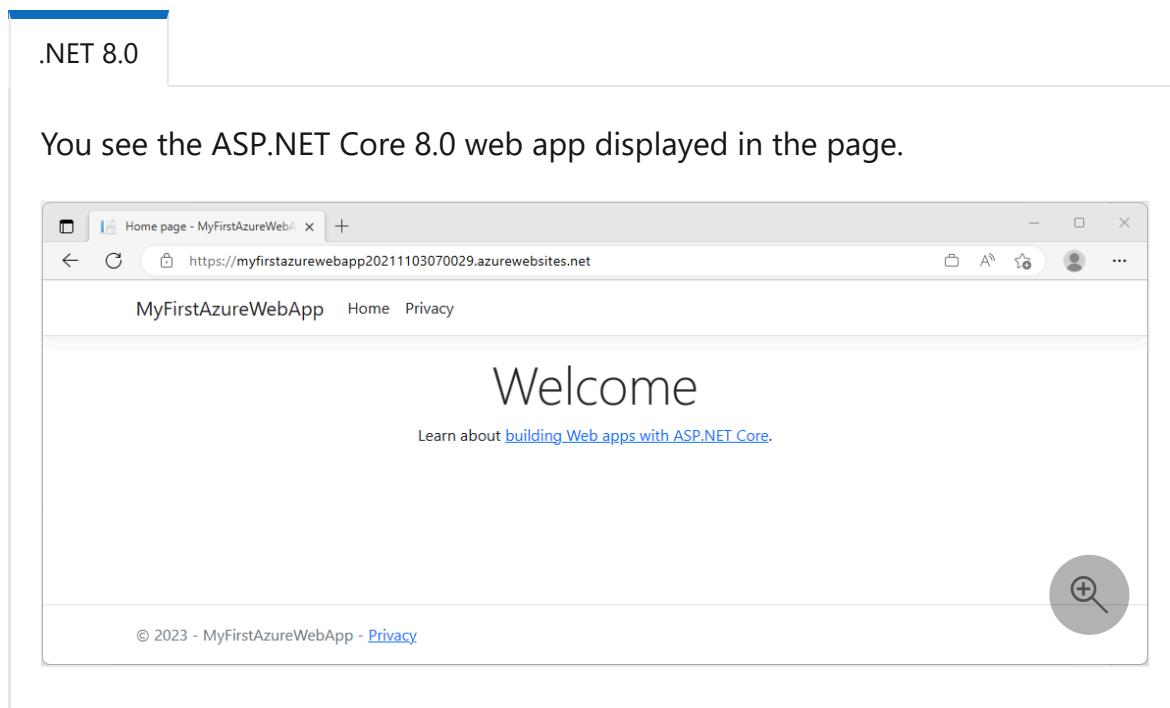
11. Select **Create** to create the Azure resources.



When the process completes, the Azure resources are created for you. You're ready to publish your ASP.NET Core project.

12. In the **Publish** dialog, ensure your new App Service app is selected, then select **Finish**, then select **Close**. Visual Studio creates a publish profile for you for the selected App Service app.
13. In the **Publish** page, select **Publish**. If you see a warning message, select **Continue**.

Visual Studio builds, packages, and publishes the app to Azure, and then launches the app in the default browser.



Update the app and redeploy

Make a change to *Index.cshtml* and redeploy to see the changes. In the .NET 8.0 template, it's in the *Pages* folder. In the .NET Framework 4.8 template, it's in the *Views/Home* folder. Follow these steps to update and redeploy your web app:

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

```
HTML

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

💡 Tip

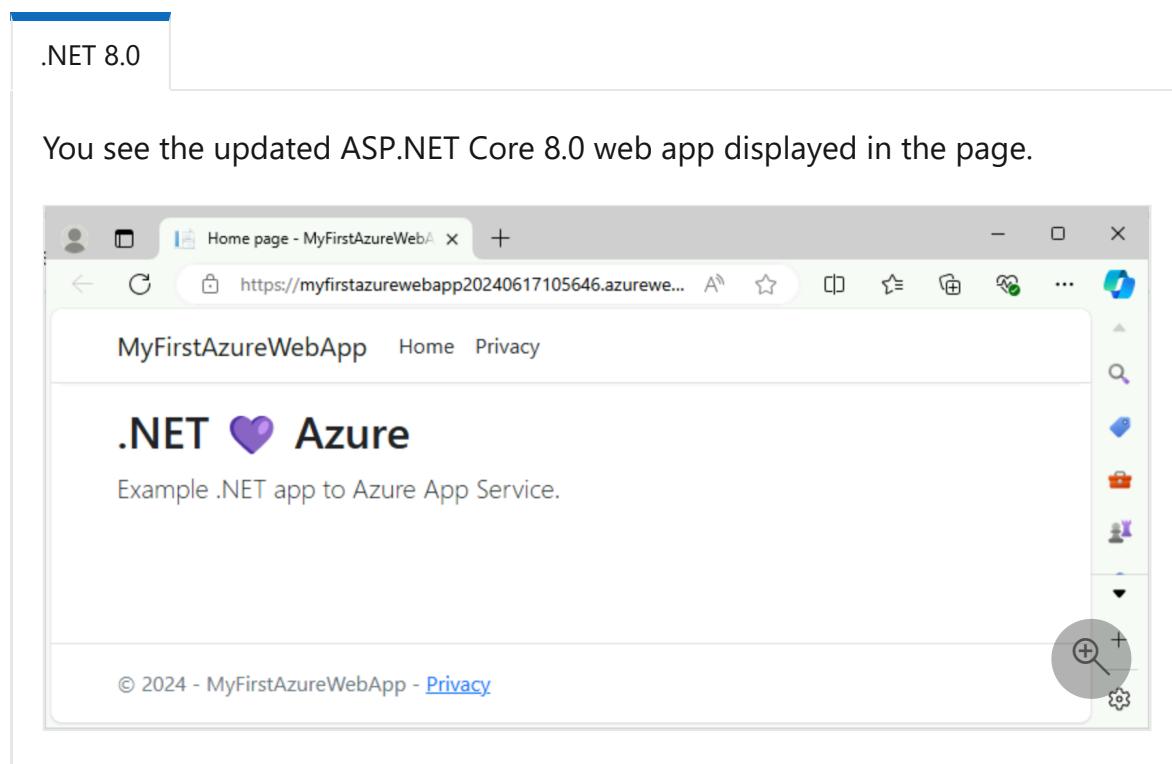
With GitHub Copilot enabled in Visual Studio, try the following steps:

- a. Select the `<div>` element and type `Alt + /`.
- b. Ask Copilot, "Change to a Bootstrap card that says .NET ❤️ Azure."

Save your changes.

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

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



Manage the Azure app

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

The screenshot shows the Microsoft Azure portal interface. At the top, there is a search bar with the text "app services". Below the search bar, the "Azure services" sidebar is visible, featuring a "Create a resource" button and a "More services" button. The main content area shows a list of services under "Services", with "App Services" highlighted and selected. Other listed services include Function App, Reservations, and App Service Certificates. To the right of the service list are links to Azure Data Explorer, Microsoft Entra ID, Enterprise applications, and Resource groups, along with a magnifying glass icon.

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

The screenshot shows the "App Services" list page in the Azure portal. The title bar says "App Services". The main content area displays a table with one record. The columns are "Name", "Status", "Location", "Pricing Tier", and "App Service Plan". The first row shows "MyFirstAzureWebApp" as the name, "Running" as the status, "West Europe" as the location, "Free" as the pricing tier, and "MyFirstAzureWebAppPlan" as the app service plan. The "MyFirstAzureWebApp" entry is highlighted with a red box. The bottom of the page includes navigation buttons for "Page 1 of 1" and a "Give feedback" link.

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

The screenshot shows the "MyFirstAzureWebApp" Overview page in the Azure portal. The title bar says "MyFirstAzureWebApp". The left sidebar has a "Overview" section selected, which includes links for Activity log, Access control (IAM), Tags, Diagnose and solve problems, Microsoft Defender for Cloud, Events (preview), Recommended services (preview), Deployment, Settings, Performance, and App Service plan. The main content area shows the "Essentials" section with details: Resource group (myresourcegroup), Status (Stopped), Location (West Europe), Subscription (Contoso subscription), and Tags (edit). It also includes a note: "Click here to access Application Insights for monitoring and profiling for your app." A "JSON View" link is located in the top right corner of the essentials section. A magnifying glass icon is in the bottom right corner of the page.

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 resource group**, type **myResourceGroup** in the text box, and then select **Delete**.
4. Confirm again by selecting **Delete**.

Next steps

.NET 8.0

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

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

[App Template: ASP.NET Core app with SQL database and App Insights deployed using CI/CD GitHub Actions](#)

[Configure ASP.NET Core app](#)

ⓘ Note: The author created this article with assistance from AI. [Learn more](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

Quickstart: Create a Java app on Azure App Service

Article • 02/08/2024

In this quickstart, you use the [Maven Plugin for Azure App Service Web Apps](#) to deploy a Java web application with an embedded server to [Azure App Service](#). App Service provides a highly scalable, self-patching web app hosting service. Use the tabs to switch between Tomcat, JBoss, or embedded server (Java SE) instructions.

The quickstart deploys either a Spring Boot app, embedded Tomcat, or Quarkus app using the [azure-webapp-maven-plugin](#) plugin.

Note

App Service can host Spring apps. For Spring apps that require all the Spring services, try [Azure Spring Apps](#) instead.



A screenshot of the Azure portal interface. On the left, there's a sidebar with a 'Spring Boot' tab selected. The main area shows a JSON response:

```
1 {  
2   "id": 2,  
3   "content": "Hello, World!"  
4 }
```

There is a gear icon in the top right corner of the main content area.

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

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

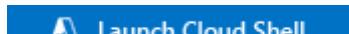
If you don't have an [Azure subscription](#), create an [Azure free account](#) before you begin.

1 - 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:

[+] [Expand table](#)

Option	Example/Link
Select Try It in the upper-right corner of a code or command block. Selecting Try It doesn't automatically copy the code or command 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 use Azure Cloud Shell:

1. Start Cloud Shell.
2. Select the **Copy** button on a code block (or command block) to copy the code or command.
3. Paste the code or command 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 or command.

2 - Get the sample app

Spring Boot

1. Download and extract the [default Spring Boot web application template](#).

This repository is cloned for you when you run the [Spring CLI](#) command

```
spring boot new my-webapp.
```

Bash

```
git clone https://github.com/rd-1-2022/rest-service my-webapp
```

2. Change your working directory to the project folder:

Azure CLI

```
cd my-webapp
```

3 - Configure the Maven plugin

The deployment process to Azure App Service uses your Azure credentials from the Azure CLI automatically. If the Azure CLI isn't installed locally, then the Maven plugin authenticates with OAuth or device sign-in. For more information, see [authentication with Maven plugins](#).

Run the Maven command shown next to configure the deployment. This command helps you to set up the App Service operating system, Java version, and Tomcat version.

Azure CLI

```
mvn com.microsoft.azure:azure-webapp-maven-plugin:2.13.0:config
```

1. For **Create new run configuration**, type Y, then **Enter**.

2. For **Define value for OS**, type 2 for Linux, then **Enter**.

3. For **Define value for javaVersion**, type 1 for Java 17, then **Enter**.

4. For **Define value for pricingTier**, type 9 for P1v2, then **Enter**.

5. For **Confirm**, type Y, then **Enter**.

```
Please confirm webapp properties
AppName : <generated-app-name>
ResourceGroup : <generated-app-name>-rg
Region : centralus
PricingTier : P1v2
OS : Linux
Java Version: Java 17
Web server stack: Java SE
Deploy to slot : false
Confirm (Y/N) [Y]: y
[INFO] Saving configuration to pom.
[INFO] -----
-----
[INFO] BUILD SUCCESS
[INFO] -----
-----
[INFO] Total time: 8.139 s
[INFO] Finished at: 2023-07-26T12:42:48Z
[INFO] -----
-----
```

After you confirm your choices, the plugin adds the above plugin element and requisite settings to your project's `pom.xml` file that configure your web app to run in Azure App Service.

The relevant portion of the `pom.xml` file should look similar to the following example.

XML

```
<build>
  <plugins>
    <plugin>
      <groupId>com.microsoft.azure</groupId>
      <artifactId>azure-webapp-maven-plugin</artifactId>
      <version>x.xx.x</version>
      <configuration>
        <schemaVersion>v2</schemaVersion>
        <resourceGroup>your-resourcegroup-name</resourceGroup>
        <appName>your-app-name</appName>
        ...
      </configuration>
    </plugin>
  </plugins>
</build>
```

You can modify the configurations for App Service directly in your `pom.xml`. Some common configurations are listed in the following table:

Property	Required	Description	Version
<schemaVersion>	false	Specify the version of the configuration schema. Supported values are: v1, v2.	1.5.2
<subscriptionId>	false	Specify the subscription ID.	0.1.0+
<resourceGroup>	true	Azure Resource Group for your Web App.	0.1.0+
<appName>	true	The name of your Web App.	0.1.0+
<region>	false	Specifies the region to host your Web App; the default value is centralus. All valid regions at Supported Regions section.	0.1.0+
<pricingTier>	false	The pricing tier for your Web App. The default value is P1v2 for production workload, while B2 is the recommended minimum for Java dev/test. For more information, see App Service Pricing .	0.1.0+
<runtime>	false	The runtime environment configuration. For more information, see Configuration Details .	0.1.0+
<deployment>	false	The deployment configuration. For more information, see Configuration Details .	0.1.0+

For the complete list of configurations, see the plugin reference documentation. All the Azure Maven Plugins share a common set of configurations. For these configurations see [Common Configurations](#). For configurations specific to App Service, see [Azure Web App: Configuration Details](#).

Be careful about the values of <appName> and <resourceGroup>. They're used later.

4 - Deploy the app

With all the configuration ready in your [pom.xml](#), you can deploy your Java app to Azure with one single command.

1. Build the JAR file using the following command:



```
Spring Boot
```

```
Bash
```

```
mvn clean package
```

2. Deploy to Azure by using the following command:

Bash

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

If the deployment succeeds, you see the following output:

Output

```
[INFO] Successfully deployed the artifact to https://<app-name>.azurewebsites.net
[INFO] -----
-----
[INFO] BUILD SUCCESS
[INFO] -----
-----
[INFO] Total time:  02:20  min
[INFO] Finished at: 2023-07-26T12:47:50Z
[INFO] -----
```

Spring Boot

Once deployment is completed, your application is ready at

<http://<appName>.azurewebsites.net/>. Open the URL

<http://<appName>.azurewebsites.net/greeting> with your local web browser (note the `/greeting` path), and you should see:

```
1 {  
2   "id": 2,  
3   "content": "Hello, World!"  
4 }
```



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

5 - Clean up resources

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

Azure CLI

```
az group delete --name <your resource group name; for example: quarkus-hello-azure-1690375364238-rg> --yes
```

This command might take a minute to run.

Next steps

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

[Set up CI/CD](#)

[Pricing Information](#)

[Aggregate Logs and Metrics](#)

[Scale up](#)

Azure for Java Developers Resources

Configure your Java app

Secure with custom domain and certificate

Deploy a Node.js web app in Azure

Article • 12/02/2024

ⓘ Note

Starting June 1, 2024, all newly created App Service apps will have the option to generate a unique default hostname using the naming convention `<app-name>-<random-hash>. <region>.azurewebsites.net`. Existing app names will remain unchanged.

Example: `myapp-ds27dh7271aah175.westus-01.azurewebsites.net`

For further details, refer to [Unique Default Hostname for App Service Resource ↗](#).

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

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

This video shows you how to deploy a Node.js web app in Azure.

<https://learn-video.azurefd.net/vod/player?id=c66346dd-9fde-4cef-b135-47d3051d5db5&locale=en-us&embedUrl=%2Fazure%2Fapp-service%2Fquickstart-nodejs> ↗

The steps in the video are also described in the following sections.

Set up your initial environment

- Have an Azure account with an active subscription. [Create an account for free ↗](#).
- Install [Node.js LTS](#) ↗ . Run the command `node --version` to verify that Node.js is installed.
- Install [Visual Studio Code](#) ↗ .
- Install the [Azure App Service extension](#) ↗ for Visual Studio Code.

Create your Node.js application

In this step, you create a basic Node.js application and ensure it runs on your computer.

💡 Tip

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

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

Bash

```
npx express-generator myExpressApp --view ejs
```

If this is the first time you've installed the generator, npx will ask you to agree to the installation.

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

Bash

```
cd myExpressApp && npm install
```

3. Update dependencies to the most secure version.

Bash

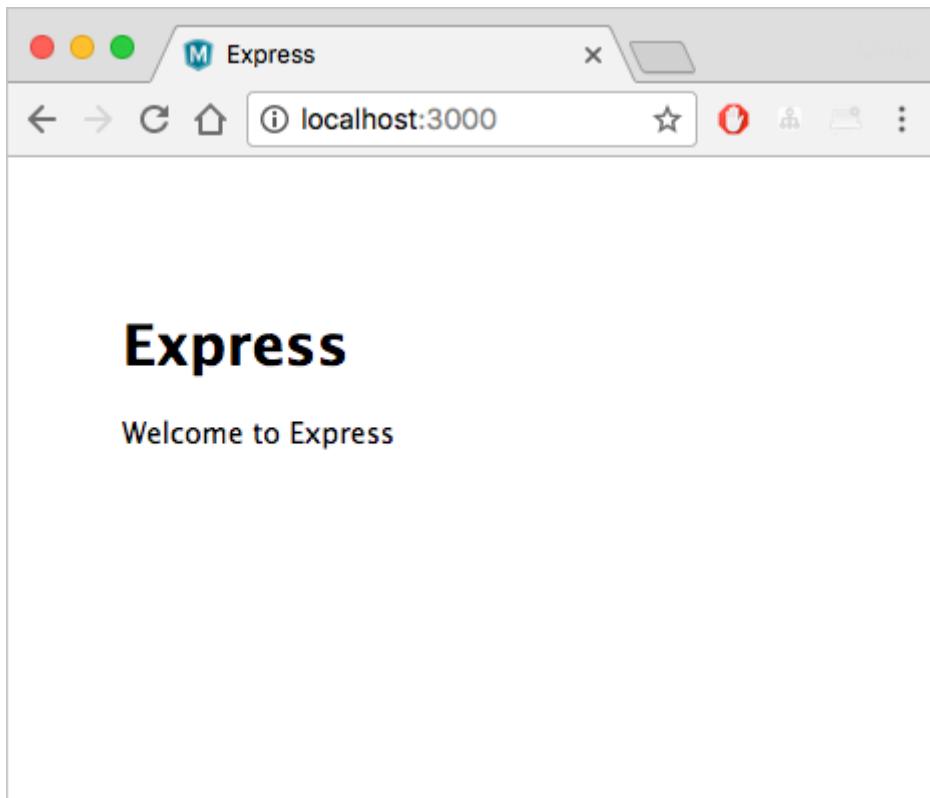
```
npm audit fix --force
```

4. Start the development server with debug information.

Bash

```
DEBUG=myexpressapp:* npm start
```

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



I ran into an issue

Deploy to Azure

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

ⓘ Note

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

Sign in to Azure

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

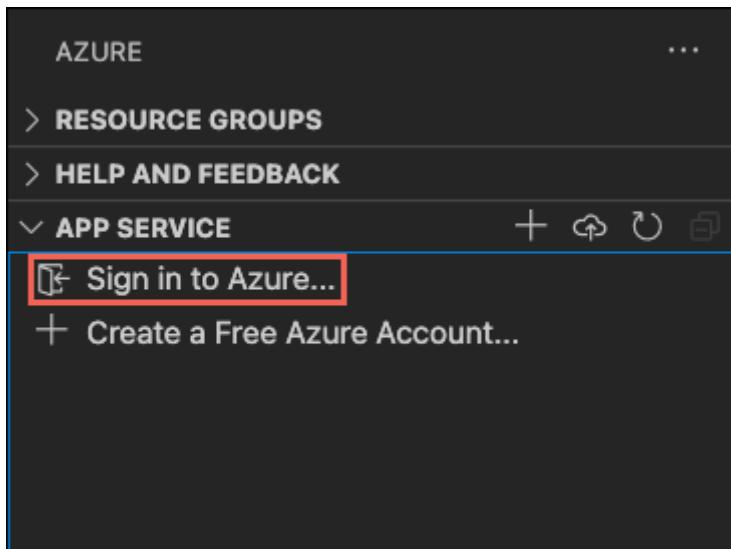
```
Bash
```

```
code .
```

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

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

In Visual Studio Code, you should see your Azure email address in the Status Bar and your subscription in the App Service explorer.



I ran into an issue

Configure the App Service app and deploy code

1. Select the *myExpressApp* folder.

A screenshot of the Azure portal showing the 'Deploy to Windows' blade for creating a new web app. The blade has a title bar with 'Deploy to Windows' and a 'Create New' button. Below the title bar, there are several configuration sections: 'Web App Name' (with placeholder 'myExpressApp'), 'Resource Group' (with placeholder 'myResourceGroup'), 'Subscription' (with placeholder 'mySubscription'), 'Location' (with placeholder 'West Europe'), 'App Service Plan' (with placeholder 'myAppServicePlan'), 'Platform' (set to 'Node.js'), and 'Operating System' (set to 'Windows').

2. Right-click App Services and select **Create new Web App... Advanced**.

3. Type a globally unique name for your web app and select **Enter**. The name must be unique across all of Azure and use only alphanumeric characters ('A-Z', 'a-z', and '0-9') and hyphens ('-'). See [the note at start of this article](#).

4. Select **Create a new resource group**, and then enter a name for the resource group, such as *AppServiceQS-rg*.

5. Select the Node.js version you want. An LTS version is recommended.

6. Select **Windows** for the operating system.

7. Select the location you want to serve your app from. For example, **West Europe**.

8. Select **Create new App Service plan**, enter a name for the plan (such as *AppServiceQS-plan*), and then select **F1 Free** for the pricing tier.

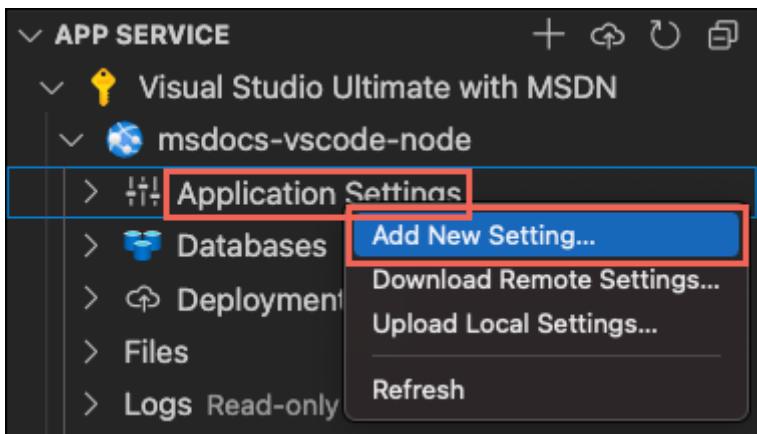
9. For **Select an Application Insights resource for your app**, select **Skip** for now and wait for the resources to be created in Azure.
10. In the popup **Always deploy the workspace "myExpressApp" to <app-name>**, select **Yes**. Doing so ensures that, as long as you're in the same workspace, Visual Studio Code deploys to the same App Service app each time.

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

 **Note**

When deployment completes, your Azure app doesn't run yet because your project root doesn't have a *web.config*. Follow the remaining steps to generate it automatically. For more information, see [You do not have permission to view this directory or page](#) in [Configure a Node.js](#).

11. In the **App Service** explorer in Visual Studio Code, expand the node for the new app, right-click **Application Settings**, and select **Add New Setting**:



12. Enter `SCM_DO_BUILD_DURING_DEPLOYMENT` for the setting key.
13. Enter `true` for the setting value.

This app setting enables build automation at deploy time, which automatically detects the start script and generates the *web.config* with it.
14. In the **App Service** explorer, select the **Deploy to Web App** icon again, and confirm by selecting **Deploy** again.
15. Wait for deployment to complete, and then select **Browse Website** in the notification popup. The browser should display the Express default page.

I ran into an issue

Redeploy updates

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

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

HTML

```
<p>Welcome to <%= title %></p>
```

to

HTML

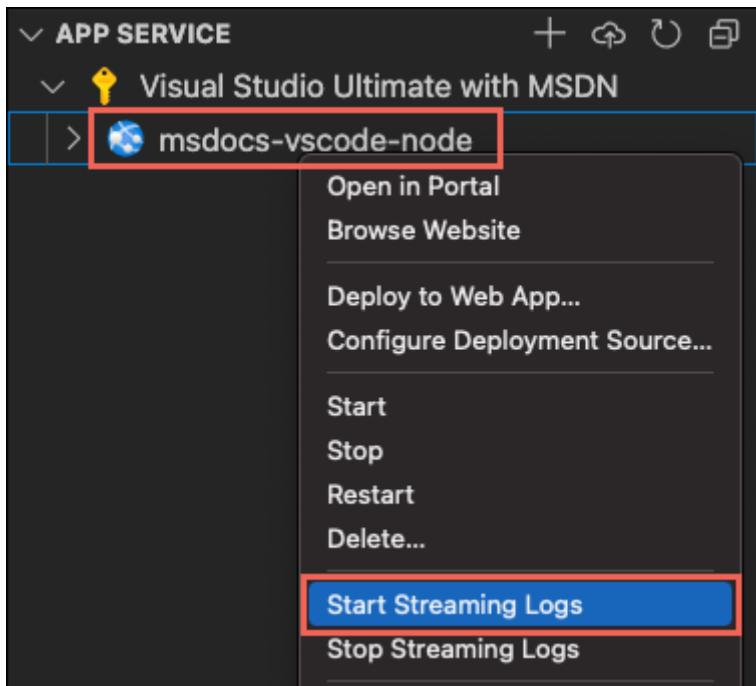
```
<p>Welcome to Azure</p>
```

2. In the **App Service** explorer, select the **Deploy to Web App** icon again, and confirm by selecting **Deploy** again.
3. Wait for deployment to complete, then select **Browse Website** in the notification popup. You should see that the `Welcome to Express` message has been changed to `Welcome to Azure`.

Stream logs

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

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



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

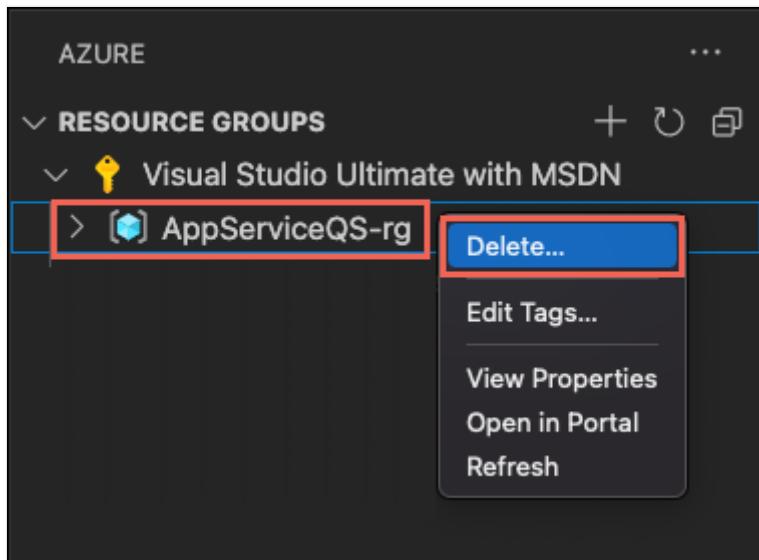
```
Connecting to log stream...
2020-03-04T19:29:44 Welcome, you are now connected to log-streaming
service. The default timeout is 2 hours.
Change the timeout with the App Setting SCM_LOGSTREAM_TIMEOUT (in
seconds).
```

I ran into an issue

Clean up resources

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

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



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

I ran into an issue

Next steps

Congratulations, you've successfully completed this quickstart!

[Deploy a Node.js + MongoDB web app to Azure](#)

[Configure a Node.js app](#)

[Secure your Azure App Service app with a custom domain and a managed certificate](#)

Check out the other Azure extensions.

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

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

ⓘ Note: The author created this article with assistance from AI. [Learn more](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

Quickstart: Deploy a Python (Django, Flask, or FastAPI) web app to Azure App Service

Article • 12/23/2024

ⓘ Note

Starting June 1, 2024, all newly created App Service apps will have the option to generate a unique default hostname using the naming convention <app-name>-<random-hash>. <region>.azurewebsites.net. Existing app names will remain unchanged.

Example: myapp-ds27dh7271aah175.westus-01.azurewebsites.net

For further details, refer to [Unique Default Hostname for App Service Resource ↗](#).

In this quickstart, you deploy a Python web app (Django, Flask, or FastAPI) to [Azure App Service](#). Azure App Service is a fully managed web hosting service that supports Python apps hosted in a Linux server environment.

To complete this quickstart, you need:

- An Azure account with an active subscription. [Create an account for free ↗](#).
- [Python 3.9 or higher ↗](#) installed locally.

ⓘ Note

This article contains current instructions on deploying a Python web app using Azure App Service. Python on Windows is no longer supported.

Sample application

This quickstart can be completed using either Flask, Django, or FastAPI. A sample application in each framework is provided to help you follow along with this quickstart. Download or clone the sample application to your local workstation.

Flask

Console

```
git clone https://github.com/Azure-Samples/msdocs-python-flask-webapp-quickstart
```

To run the application locally:

Flask

1. Go to the application folder:

Console

```
cd msdocs-python-flask-webapp-quickstart
```

2. Create a virtual environment for the app:

Windows

Console

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

3. Install the dependencies:

Console

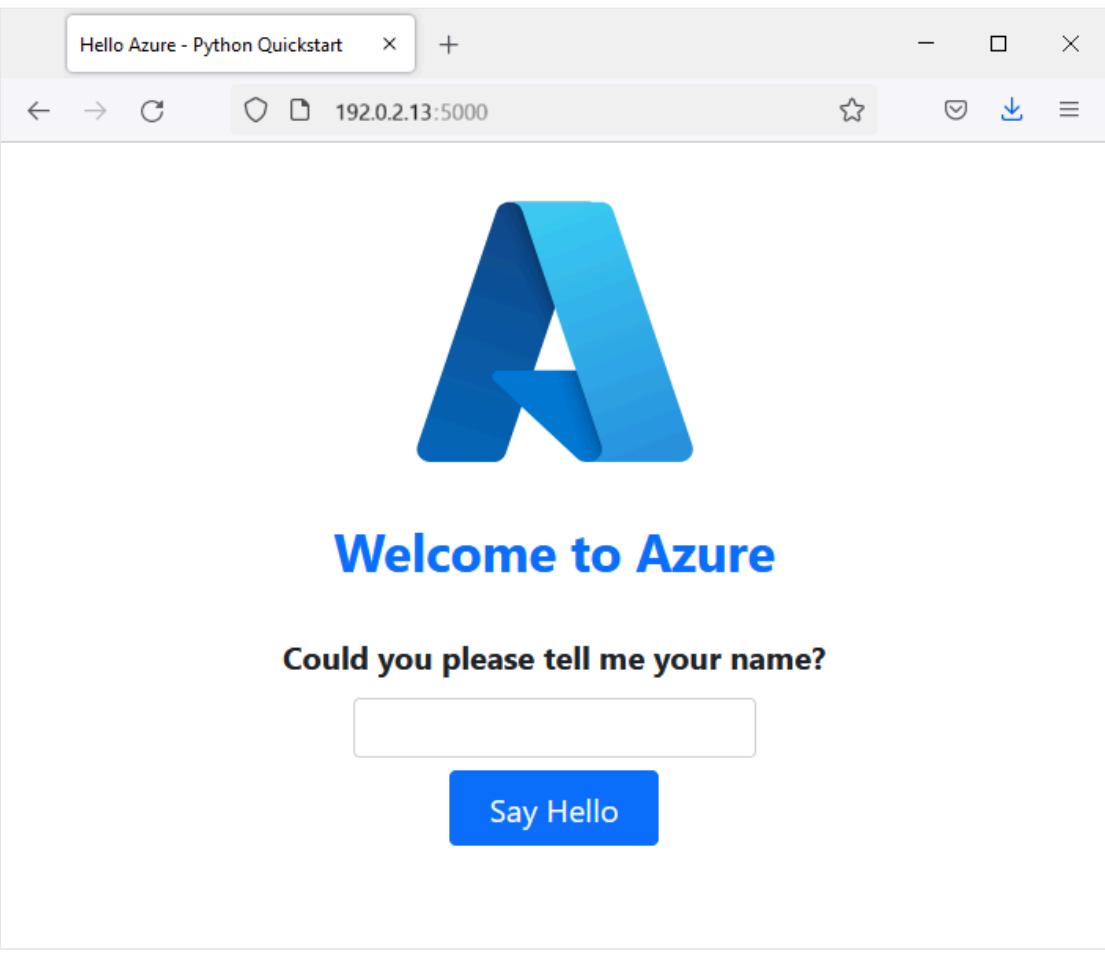
```
pip install -r requirements.txt
```

4. Run the app:

Console

```
flask run
```

5. Browse to the sample application at <http://localhost:5000> in a web browser.



Create a web app in Azure

To host your application in Azure, you need to create an Azure App Service web app in Azure. You can create a web app using the Azure CLI, [VS Code](#), [Azure Tools extension pack](#), or the [Azure portal](#).

Azure CLI

Azure CLI commands can be run on a computer with the [Azure CLI installed](#).

Azure CLI has a command `az webapp up` that will create the necessary resources and deploy your application in a single step.

If necessary, log in to Azure using [az login](#).

Azure CLI

```
az login
```

Create the webapp and other resources, then deploy your code to Azure using [az webapp up](#).

Azure CLI

```
az webapp up --runtime PYTHON:3.9 --sku B1 --logs
```

- The `--runtime` parameter specifies what version of Python your app is running. This example uses Python 3.9. To list all available runtimes, use the command `az webapp list-runtimes --os linux --output table`.
- The `--sku` parameter defines the size (CPU, memory) and cost of the app service plan. This example uses the B1 (Basic) service plan, which will incur a small cost in your Azure subscription. For a full list of App Service plans, view the [App Service pricing](#) page.
- The `--logs` flag configures default logging required to enable viewing the log stream immediately after launching the webapp.
- You can optionally specify a name with the argument `--name <app-name>`. If you don't provide one, then a name will be automatically generated.
- You can optionally include the argument `--location <location-name>` where `<location_name>` is an available Azure region. You can retrieve a list of allowable regions for your Azure account by running the [az appservice list-locations](#) command.

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

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

}
```

```
"os": "<os-type>",
"resourcegroup": "<group-name>",
"runtime_version": "python|3.9",
"runtime_version_detected": "0.0",
"sku": "FREE",
"src_path": "<your-folder-location>"
}
```

ⓘ Note

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

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

Having issues? [Let us know ↗](#).

Deploy your application code to Azure

Azure App Service supports multiple methods to deploy your application code to Azure, including GitHub Actions and all major CI/CD tools. This article focuses on how to deploy your code from your local workstation to Azure.

Deploy using Azure CLI

Since the `az webapp up` command created the necessary resources and deployed your application in a single step, you can move on to the next step.

Having issues? Refer first to the [Troubleshooting guide](#). If that doesn't help, let us know ↗.

Configure startup script

Based on the presence of certain files in a deployment, App Service automatically detects whether an app is a Django or Flask app and performs default steps to run your app. For apps based on other web frameworks like FastAPI, you need to configure a startup script for App Service to run your app; otherwise, App Service runs a default read-only app located in the `opt/defaultsite` folder.

To learn more about how App Service runs Python apps and how you can configure and customize its behavior with your app, see [Configure a Linux Python app for Azure App Service](#).

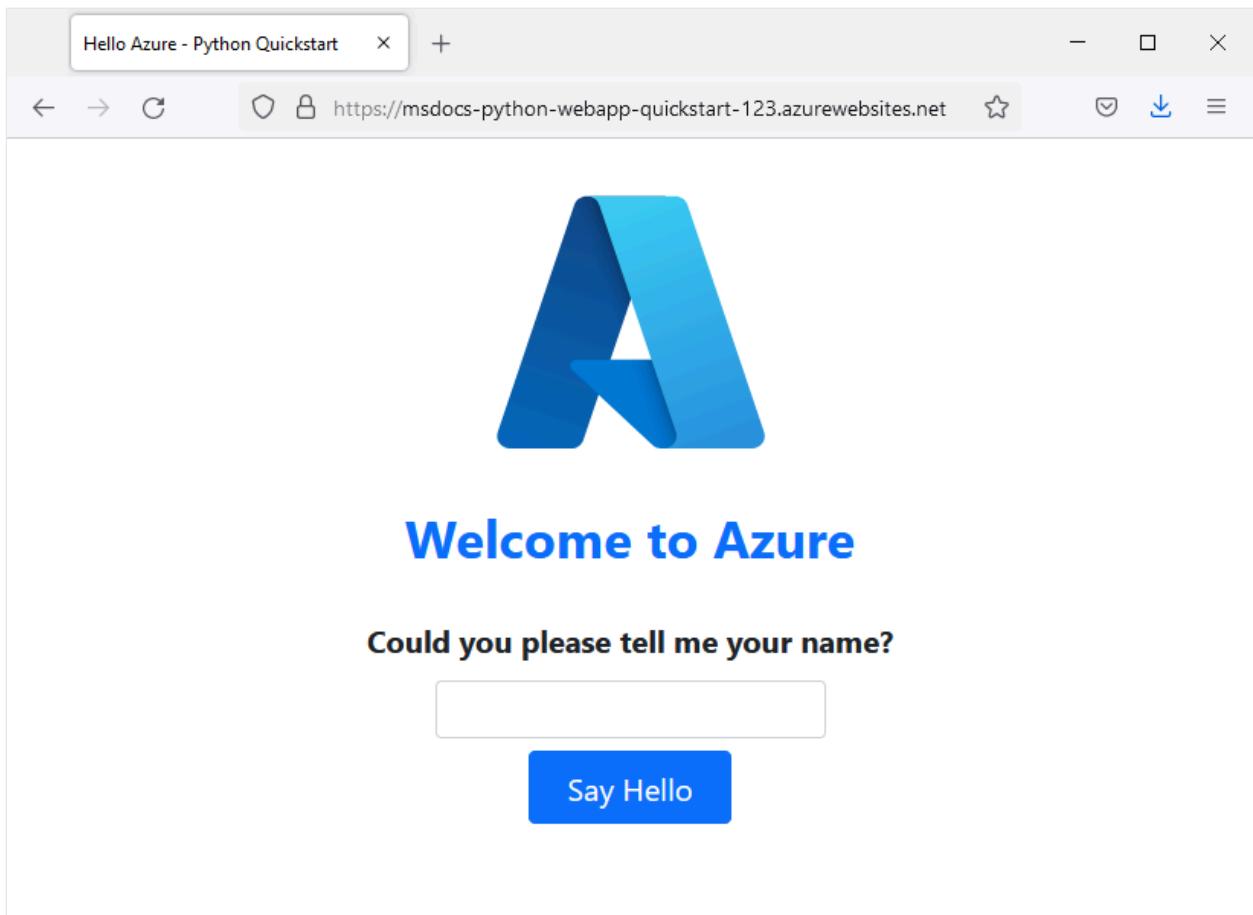
Azure CLI

App Service automatically detects the presence of a Flask app. No additional configuration is needed for this quickstart.

Browse to the app

Browse to the deployed application in your web browser by using the URL `http://<app-name>.azurewebsites.net`. If you see a default app page, wait a minute and refresh the browser.

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



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

Having issues? Refer first to the [Troubleshooting guide](#). If that doesn't help, [let us know ↗](#).

Stream logs

Azure App Service captures all message output to the console to assist you in diagnosing issues with your application. The sample apps include `print()` statements to demonstrate this capability.

```
@app.route('/hello', methods=['POST'])
def hello():
    name = request.form.get('name')

    if name:
        print('Request for hello page received with name=%s' % name)
        return render_template('hello.html', name = name)
    else:
        print('Request for hello page received with no name or blank name
-- redirecting')
        return redirect(url_for('index'))
```

You can review the contents of the App Service diagnostic logs by using the Azure CLI, VS Code, or the Azure portal.

Azure CLI

First, you need to configure Azure App Service to output logs to the App Service filesystem by using the [az webapp log config](#) command.

bash

Azure CLI

```
az webapp log config \
--web-server-logging filesystem \
--name $APP_SERVICE_NAME \
--resource-group $RESOURCE_GROUP_NAME
```

To stream logs, use the [az webapp log tail](#) command.

bash

Azure CLI

```
az webapp log tail \
--name $APP_SERVICE_NAME \
--resource-group $RESOURCE_GROUP_NAME
```

Refresh the home page in the app or attempt other requests to generate some log messages. The output should look similar to the following.

Output

Starting Live Log Stream ---

```
2021-12-23T02:15:52.740703322Z Request for index page received
2021-12-23T02:15:52.740740222Z 169.254.130.1 - - [23/Dec/2021:02:15:52
+0000] "GET / HTTP/1.1" 200 1360 "https://msdocs-python-webapp-
quickstart-123.azurewebsites.net/hello" "Mozilla/5.0 (Windows NT 10.0;
Win64; x64; rv:95.0) Gecko/20100101 Firefox/95.0"
2021-12-23T02:15:52.841043070Z 169.254.130.1 - - [23/Dec/2021:02:15:52
+0000] "GET /static/bootstrap/css/bootstrap.min.css HTTP/1.1" 200 0
"https://msdocs-python-webapp-quickstart-123.azurewebsites.net/"
"Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:95.0) Gecko/20100101
Firefox/95.0"
2021-12-23T02:15:52.884541951Z 169.254.130.1 - - [23/Dec/2021:02:15:52
+0000] "GET /static/images/azure-icon.svg HTTP/1.1" 200 0
"https://msdocs-python-webapp-quickstart-123.azurewebsites.net/"
"Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:95.0) Gecko/20100101
Firefox/95.0"
2021-12-23T02:15:53.043211176Z 169.254.130.1 - - [23/Dec/2021:02:15:53
+0000] "GET /favicon.ico HTTP/1.1" 404 232 "https://msdocs-python-
webapp-quickstart-123.azurewebsites.net/" "Mozilla/5.0 (Windows NT 10.0;
Win64; x64; rv:95.0) Gecko/20100101 Firefox/95.0"

2021-12-23T02:16:01.304306845Z Request for hello page received with
name=David
2021-12-23T02:16:01.304335945Z 169.254.130.1 - - [23/Dec/2021:02:16:01
+0000] "POST /hello HTTP/1.1" 200 695 "https://msdocs-python-webapp-
quickstart-123.azurewebsites.net/" "Mozilla/5.0 (Windows NT 10.0; Win64;
x64; rv:95.0) Gecko/20100101 Firefox/95.0"
2021-12-23T02:16:01.398399251Z 169.254.130.1 - - [23/Dec/2021:02:16:01
+0000] "GET /static/bootstrap/css/bootstrap.min.css HTTP/1.1" 304 0
"https://msdocs-python-webapp-quickstart-123.azurewebsites.net/hello"
"Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:95.0) Gecko/20100101
Firefox/95.0"
2021-12-23T02:16:01.430740060Z 169.254.130.1 - - [23/Dec/2021:02:16:01
+0000] "GET /static/images/azure-icon.svg HTTP/1.1" 304 0
"https://msdocs-python-webapp-quickstart-123.azurewebsites.net/hello"
"Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:95.0) Gecko/20100101
Firefox/95.0"
```

Having issues? Refer first to the [Troubleshooting guide](#). If that doesn't help, let us know ↗.

Clean up resources

When you're finished with the sample app, you can remove all of the resources for the app from Azure. Removing the resource group ensures that you don't incur extra charges and helps keep your Azure subscription uncluttered. Removing the resource

group also removes all resources in the resource group and is the fastest way to remove all Azure resources for your app.

Azure CLI

Delete the resource group by using the [az group delete](#) command.

Azure CLI

```
az group delete \
--name msdocs-python-webapp-quickstart \
--no-wait
```

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

Having issues? [Let us know ↗](#).

Next steps

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

[Configure a Python app](#)

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

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

[Secure an app with a custom domain and certificate](#)

Feedback

Was this page helpful?

 Yes

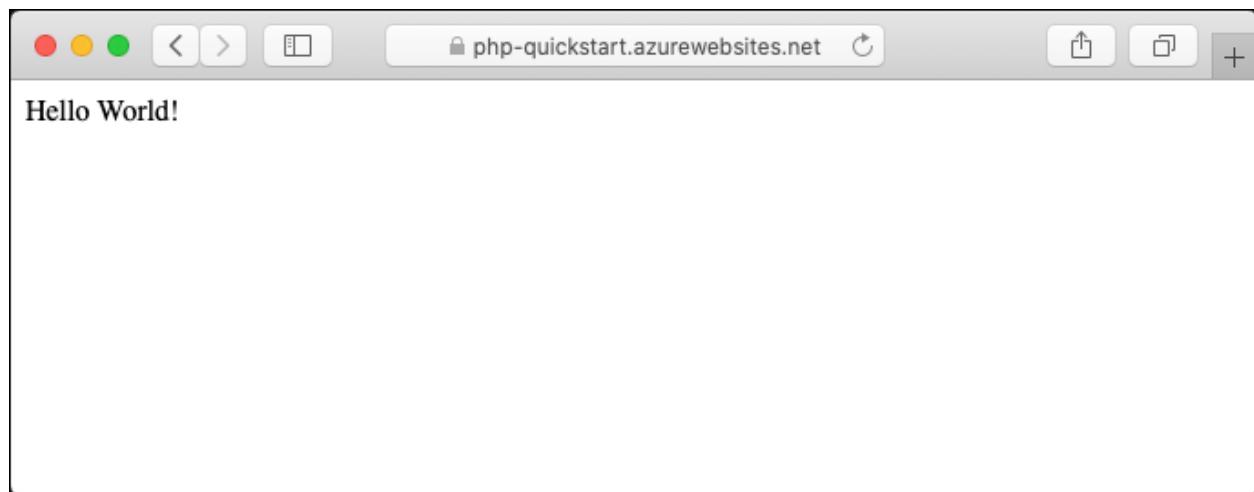
 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

Create a PHP web app in Azure App Service

Article • 01/30/2024

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



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

To complete this quickstart, you need:

- An Azure account with an active subscription. [Create an account for free](#).
- [Git](#)
- [PHP](#)
- [Azure CLI](#) to run commands in any shell to create and configure Azure resources.

1 - Get the sample repository

Azure CLI

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

1. In a terminal window, run the following commands to clone the sample application to your local machine and navigate to the project root.

Bash

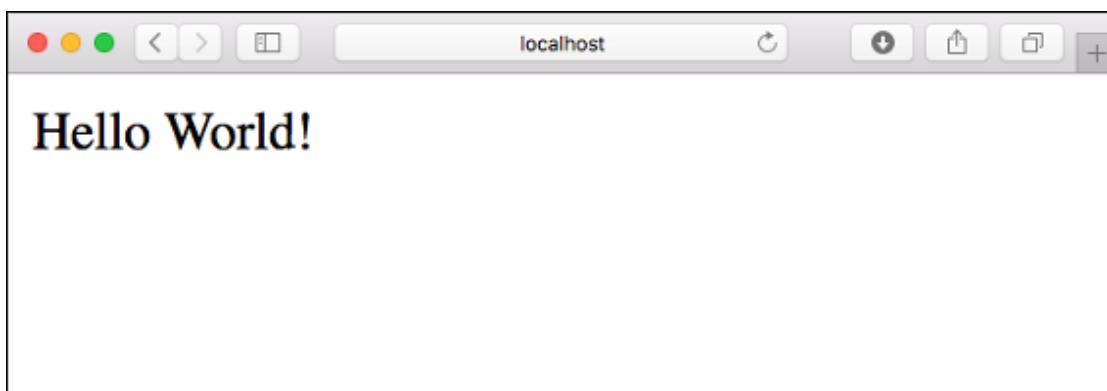
```
git clone https://github.com/Azure-Samples/php-docs-hello-world  
cd php-docs-hello-world
```

2. To run the application locally, use the `php` command to launch the built-in PHP web server.

Bash

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

3. Browse to the sample application at `http://localhost:8080` in a web browser.



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

2 - Deploy your application code to Azure

Azure CLI

Azure CLI has a command `az webapp up` that creates the necessary resources and deploys your application in a single step.

In the terminal, deploy the code in your local folder using the `az webapp up` command:

Azure CLI

```
az webapp up --runtime "PHP:8.2" --os-type=linux
```

- If the `az` command isn't recognized, be sure you have [Azure CLI](#) installed.
- The `--runtime "PHP:8.2"` argument creates the web app with PHP version 8.2.
- The `--os-type=linux` argument creates the web app on App Service on Linux.

- You can optionally specify a name with the argument `--name <app-name>`. If you don't provide one, then a name is automatically generated.
- You can optionally include the argument `--location <location-name>` where `<location_name>` is an available Azure region. You can retrieve a list of allowable regions for your Azure account by running the [az account list-locations](#) command.
- If you see the error, "Could not auto-detect the runtime stack of your app," make sure you're running the command in the code directory (See [Troubleshooting auto-detect issues with az webapp up ↗](#)).

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

```
The webapp '<app-name>' doesn't exist
Creating Resource group '<group-name>' ...
Resource group creation complete
Creating AppServicePlan '<app-service-plan-name>' ...
Creating webapp '<app-name>' ...
Configuring default logging for the app, if not already enabled
Creating zip with contents of dir /home/msangapu/myPhpApp ...
Getting scm site credentials for zip deployment
Starting zip deployment. This operation can take a while to complete ...
Deployment endpoint responded with status code 202
You can launch the app at http://<app-name>.azurewebsites.net
{
  "URL": "http://<app-name>.azurewebsites.net",
  "appserviceplan": "<app-service-plan-name>",
  "location": "centralus",
  "name": "<app-name>",
  "os": "linux",
  "resourcegroup": "<group-name>",
  "runtime_version": "php|8.2",
  "runtime_version_detected": "0.0",
  "sku": "FREE",
  "src_path": "//home//msangapu//myPhpApp"
}
```

ⓘ Note

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

- Create a default resource group.

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

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

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



Congratulations! You deployed your first PHP app to App Service using the Azure portal.

3 - Update and redeploy the app

Azure CLI

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

PHP

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

2. Save your changes, then redeploy the app using the `az webapp up` command again with these arguments:

Azure CLI

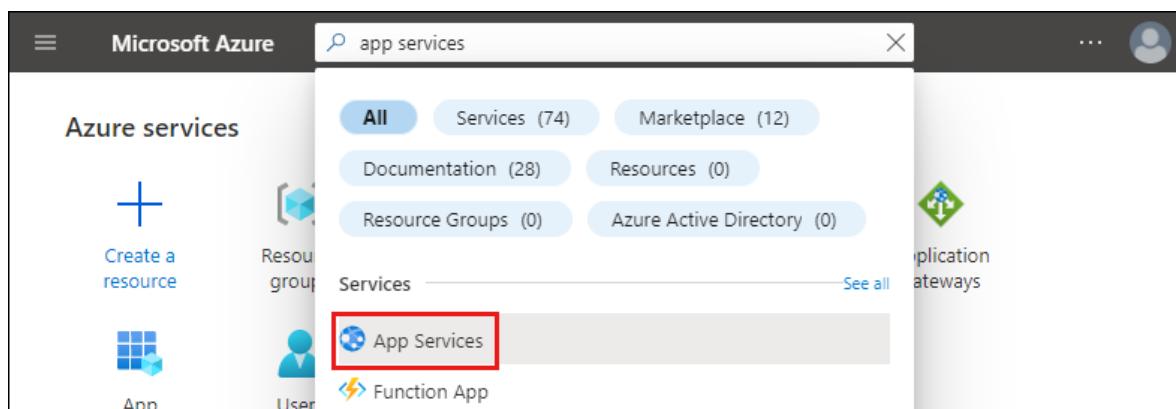
```
az webapp up --runtime "PHP:8.2" --os-type=linux
```

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



4 - Manage your new Azure app

- Go to the Azure portal to manage the web app you created. Search for and select **App Services**.



- Select the name of your Azure app.

The screenshot shows the Microsoft Azure App Services dashboard. At the top, there's a search bar and a user profile icon. Below the header, the title 'App Services' is displayed with a subtitle 'Contoso (contoso.com)'. A navigation bar includes 'Create', 'Manage view', 'Refresh', 'Export to CSV', 'Open query', and a 'More' dropdown. Filter options like 'Subscription == all', 'Add filter', and 'More (2)' are available. The main area lists a single item: 'Name' (php-app-service-demo), 'Status' (Running), 'Location' (West Europe), and 'Pricing Tier' (Free). The row for 'php-app-service-demo' is highlighted with a red box. At the bottom, there are pagination controls ('Page 1 of 1') and a 'List view' button.

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

The screenshot shows the Microsoft Azure App Service overview for 'php-app-service-demo'. The top navigation bar includes 'Search resources, services, and docs' and a user profile icon. Below the title, there are management buttons: 'Browse' (highlighted with a red box), 'Stop', 'Swap', 'Restart', 'Delete', 'Refresh', and a 'More' dropdown. The 'Essentials' section displays resource group information ('Resource group (move) : myResourceGroup') and the current status ('Status : Running'). A 'JSON View' link is also present.

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

5 - Clean up resources

When you're finished with the sample app, you can remove all of the resources for the app from Azure. It helps you avoid extra charges and keeps your Azure subscription uncluttered. Removing the resource group also removes all resources in the resource group and is the fastest way to remove all Azure resources for your app.

Delete the resource group by using the [az group delete](#) command.

Azure CLI

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

This command takes a minute to run.

Next steps

[PHP with MySQL](#)

[Configure PHP app](#)

[Secure with custom domain and certificate](#)

Create a WordPress site

Article • 03/28/2024

[WordPress](#) is an open source Content Management System (CMS) used by over 40% of the web to create websites, blogs, and other applications. WordPress can be run on a few different Azure services: [AKS](#), [Virtual Machines](#), [Azure Container Apps](#) and Azure App Service. For a full list of WordPress options on Azure, see [WordPress on Azure Marketplace](#).

In this quickstart, you'll learn how to create and deploy your first [WordPress](#) site to [Azure App Service on Linux](#) with [Azure Database for MySQL - Flexible Server](#) using the [WordPress Azure Marketplace item by App Service](#). This quickstart uses the **Standard** tier for your app and a **Burstable, B2s** tier for your database, and incurs a cost for your Azure Subscription. For pricing, visit [App Service pricing](#), [Azure Database for MySQL pricing](#), [Content Delivery Network pricing](#), and [Azure Blob Storage pricing](#).

To complete this quickstart, you need an Azure account with an active subscription. [Create an account for free](#).

Create WordPress site using Azure portal

1. To start creating the WordPress site, browse to <https://portal.azure.com/#create/WordPress.WordPress>.

Microsoft Azure Search resources, services, and docs (G+)

Home > Marketplace > WordPress on App Service >

Create WordPress on App Service

Basics Advanced Tags Review + create

WordPress optimized for App Service with best practices for security and performance. [Learn More](#)

Project details

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

Subscription * ⓘ

Resource group * ⓘ (New) Resource group

Hosting details

Select the region for your server and provide a name for Web App. Custom domains can be added later.

Region * South Central US

Name * Web App name. .azurewebsites.net

Hosting plans

This hosting plan dictates what resources are available, what features are enabled and how it is priced.

Hosting plan	Standard PremiumV2 App Service, Burstable MySQL database Change plan
--------------	---

WordPress setup

Select the language you want your website to be in. Provide admin email, username and password that you can use to access WordPress admin dashboard.

Site language	<input type="text"/> English (United States)
Admin email *	<input type="text"/>
Admin username * ⓘ	<input type="text"/>
Admin password * ⓘ	<input type="text"/> Password
Confirm password *	<input type="text"/> Confirm password
Enable multisite ⓘ	<input type="checkbox"/>

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

- In the Basics tab, under Project details, make sure the correct subscription is selected. Select Create new resource group and type `myResourceGroup` for the name.

Basics Advanced Tags Review + create

WordPress optimized for App Service with best practices for security and performance. [Learn More ↗](#)

Project details

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

Subscription * ⓘ

Resource group * ⓘ

[Create new](#)

3. Under **Hosting details**, select a **Region** you want to serve your app from, then type a globally unique name for your web app. Under **Hosting plans**, select **Standard**. Select **Change plan** to view features and price comparisons.

Hosting details

Select the region for your server and provide a name for Web App. Custom domains can be added later.

Region *

Name *

.azurewebsites.net

Hosting plans

This hosting plan dictates what resources are available, what features are enabled and how it is priced.

Hosting plan

Standard

PremiumV2 App Service, Burstable MySQL database

[Change plan](#)

4. Under **WordPress setup**, choose your preferred **Site Language**, then type an **Admin Email**, **Admin Username**, and **Admin Password**. The **Admin Email** is used for WordPress administrative sign-in only. Clear the **Enable multisite** checkbox.

WordPress setup

Select the language you want your website to be in. Provide admin email, username and password that you can use to access WordPress admin dashboard.

Site language

Admin email *

Admin username * ⓘ

Admin password * ⓘ

Confirm password *

Enable multisite ⓘ

5. Select the **Advanced** tab. If you're unfamiliar with an [Azure CDN](#), [Azure Front Door](#), or [Blob Storage](#), then clear the checkboxes. For more details on the Content Distribution options, see [WordPress on App Service ↗](#).

Basics Advanced Tags Review + create

This section allows you to opt for advanced features. The recommended features are selected by default. Please go through the descriptions before making changes.

Azure CDN (Recommended)

Azure Content Delivery Network helps in improving performance, availability, and security by using a distributed network of servers that can store cached content in point-of-presence locations, close to end users. [Learn More](#)

Enable Azure CDN [\(i\)](#)

Azure Front Door (Preview)

Azure Front Door (AFD) provides dynamic site acceleration that reduces response times while also allowing content delivery by caching at nearest edge servers for faster media downloads. [Learn More](#)

Enable AFD [\(i\)](#)

AFD Profiles

Select AFD Profile



You can choose to select either Azure CDN or Azure Front Door.

Azure Blob Storage (Recommended)

Azure Blob Storage allows you to store and access images, videos and other files. This effectively reduces the load on your web server thereby improving performance and user experience. [Learn More](#)

Enable Blob Storage [\(i\)](#)

Storage Account [\(i\)](#)

(New) mydemowpsi23bb6e5df3



Virtual Network

Virtual networks are logically isolated from each other in Azure. You can configure their IP address ranges, subnets, route tables, gateways, and security settings, much like a traditional network in your data center. Virtual machines in the same virtual network can access each other by default. [Learn More](#)

Virtual Network [\(i\)](#)

(New) mydemowpsi-21f6b45092-vnet



Note

The WordPress app requires a virtual network with an address space of /23 at minimum.

6. Select the **Review + create** tab. After validation runs, select the **Create** button at the bottom of the page to create the WordPress site.

Create

< Previous

Next >

7. Browse to your site URL and verify the app is running properly. The site may take a few minutes to load. If you receive an error, allow a few more minutes then refresh the browser.

Hello world!

Welcome to WordPress. This is your first post. Edit or delete it, then start writing!

Published January 24, 2022
Categorized as [Uncategorized](#)

Search

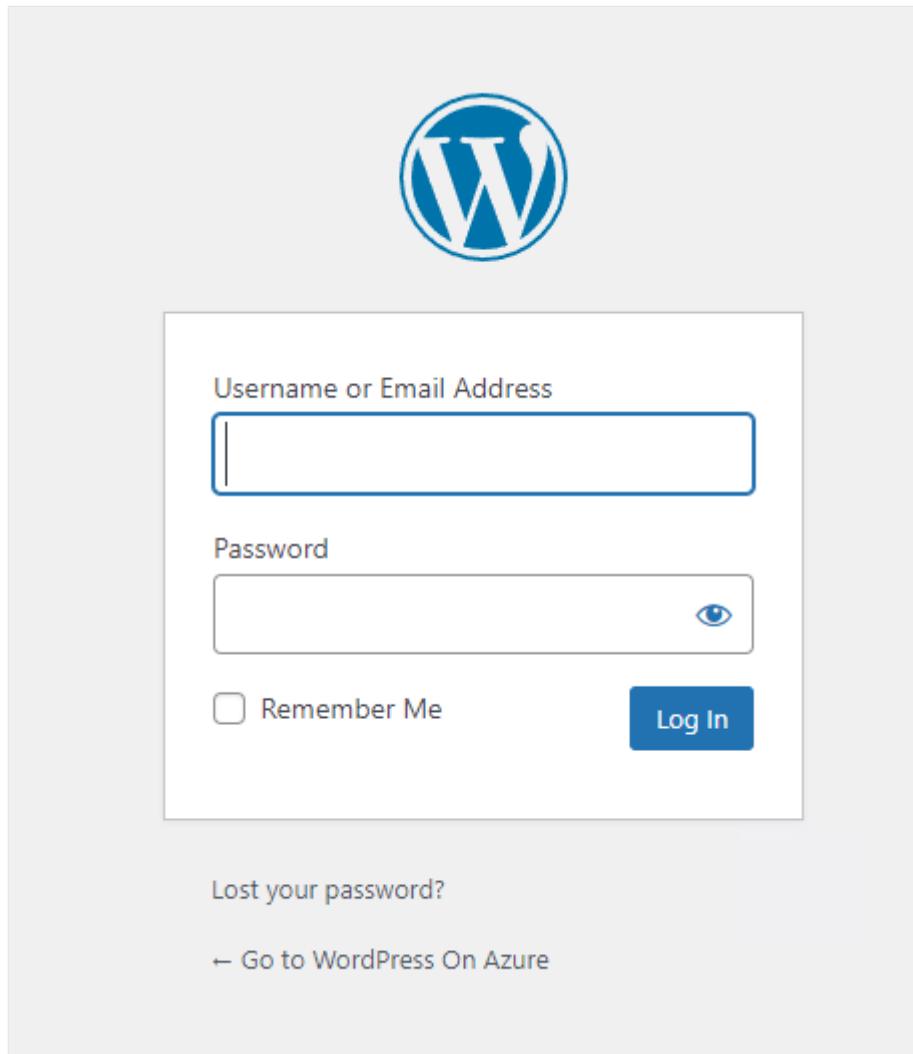
Recent Posts

[Hello world!](#)

Recent Comments

[A WordPress Commenter](#) on [Hello world!](#)

8. To access the WordPress Admin page, browse to `/wp-admin` and use the credentials you created in the [WordPress setup](#) step.



Clean up resources

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

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

The screenshot shows the Azure portal interface for an App Service named 'myWPApp'. On the left, there's a sidebar with links like Overview, Activity log, Access control (IAM), Tags, Diagnose and solve problems, Security, and Events (preview). The main content area has a header with 'Browse', 'Stop', 'Swap', 'Restart', and 'Delete' buttons. Below this, under the 'Essentials' section, it shows the Resource group as 'myResourceGroup' (with a red box around it), Status as 'Running', Location as 'Central US', Subscription as '{your subscription name}', and Subscription ID as '{your subscription id}'. There's also a 'Tags (Edit)' link and a note to 'Click here to add tags'.

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

The screenshot shows the Azure portal interface for a Resource group named 'myResourceGroup'. The sidebar includes links for Overview, Activity log, Access control (IAM), Tags, Resource visualizer, and Events. The main content area has a header with 'Create', 'Edit columns', and a 'Delete resource group' button (highlighted with a red box). Under the 'Essentials' section, it lists the Subscription as '{your subscription name}', Subscription ID as '{your subscription id}', and Tags as 'Click here to add tags'. At the bottom, there are tabs for 'Resources' (which is selected) and 'Recommendations (4)'.

Manage the MySQL flexible server, username, or password

- The MySQL Flexible Server is created behind a private [Virtual Network](#) and can't be accessed directly. To access or manage the database, use [phpMyAdmin](#) that's deployed with the WordPress site. You can access [phpMyAdmin](#) by following these steps:
 - Navigate to the URL: <https://<sitename>.azurewebsites.net/phpmyadmin>
 - Login with the flexible server's username and password

- Database username and password of the MySQL Flexible Server are generated automatically. To retrieve these values after the deployment go to Application Settings section of the Configuration page in Azure App Service. The WordPress configuration is modified to use these [Application Settings](#) to connect to the MySQL database.
- To change the MySQL database password, see [Reset admin password](#). Whenever the MySQL database credentials are changed, the [Application Settings](#) need to be updated. The [Application Settings for MySQL database](#) begin with the `DATABASE_` prefix. For more information on updating MySQL passwords, see [WordPress on App Service ↗](#).

Change WordPress admin password

The [Application Settings](#) for WordPress admin credentials are only for deployment purposes. Modifying these values has no effect on the WordPress installation. To change the WordPress admin password, see [resetting your password ↗](#). The [Application Settings for WordPress admin credentials](#) begin with the `WORDPRESS_ADMIN_` prefix. For more information on updating the WordPress admin password, see [Changing WordPress Admin Credentials ↗](#).

Migrate to App Service on Linux

There's a couple approaches when migrating your WordPress app to App Service on Linux. You could use a WP plugin or migrate manually using FTP and a MySQL client. Additional documentation, including [Migrating to App Service ↗](#), can be found at [WordPress - App Service on Linux ↗](#).

Next steps

Congratulations, you've successfully completed this quickstart!

[Secure with custom domain and certificate](#)

[Tutorial: PHP app with MySQL](#)

[Configure PHP app](#)

Run a custom container in Azure

Article • 06/29/2023

Azure 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 isn't already defined in Azure. This quickstart shows you how to deploy an image from an [Azure Container Registry](#) (ACR) to App Service.

ⓘ Note

For information regarding running containerized applications in a serverless environment, please see [Container Apps](#).

To complete this quickstart, you need:

- 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.

1 - Create a container registry

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

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

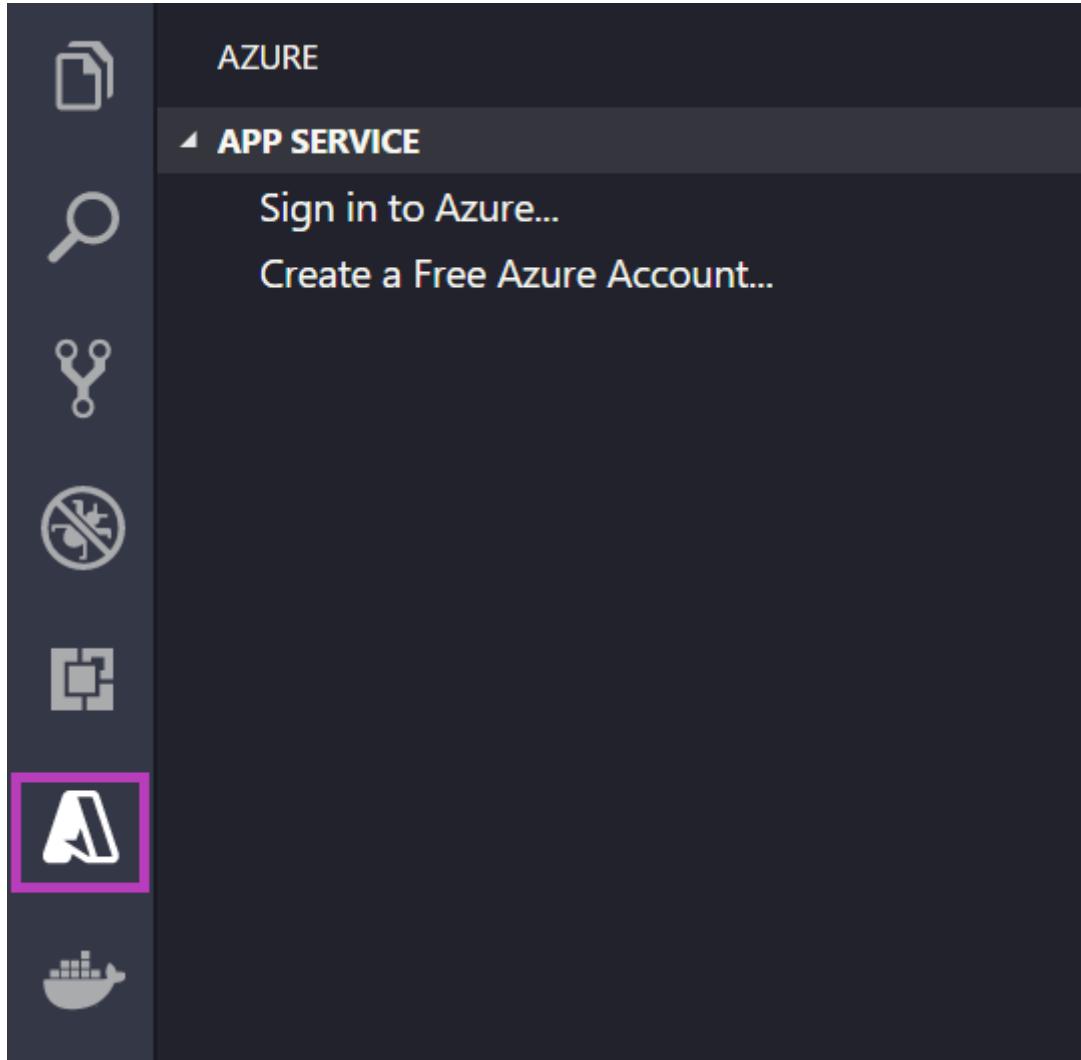
ⓘ Important

Be sure to set the **Admin User** option to **Enable** when you create the Azure container registry. You can also set it from the **Access keys** section of your registry

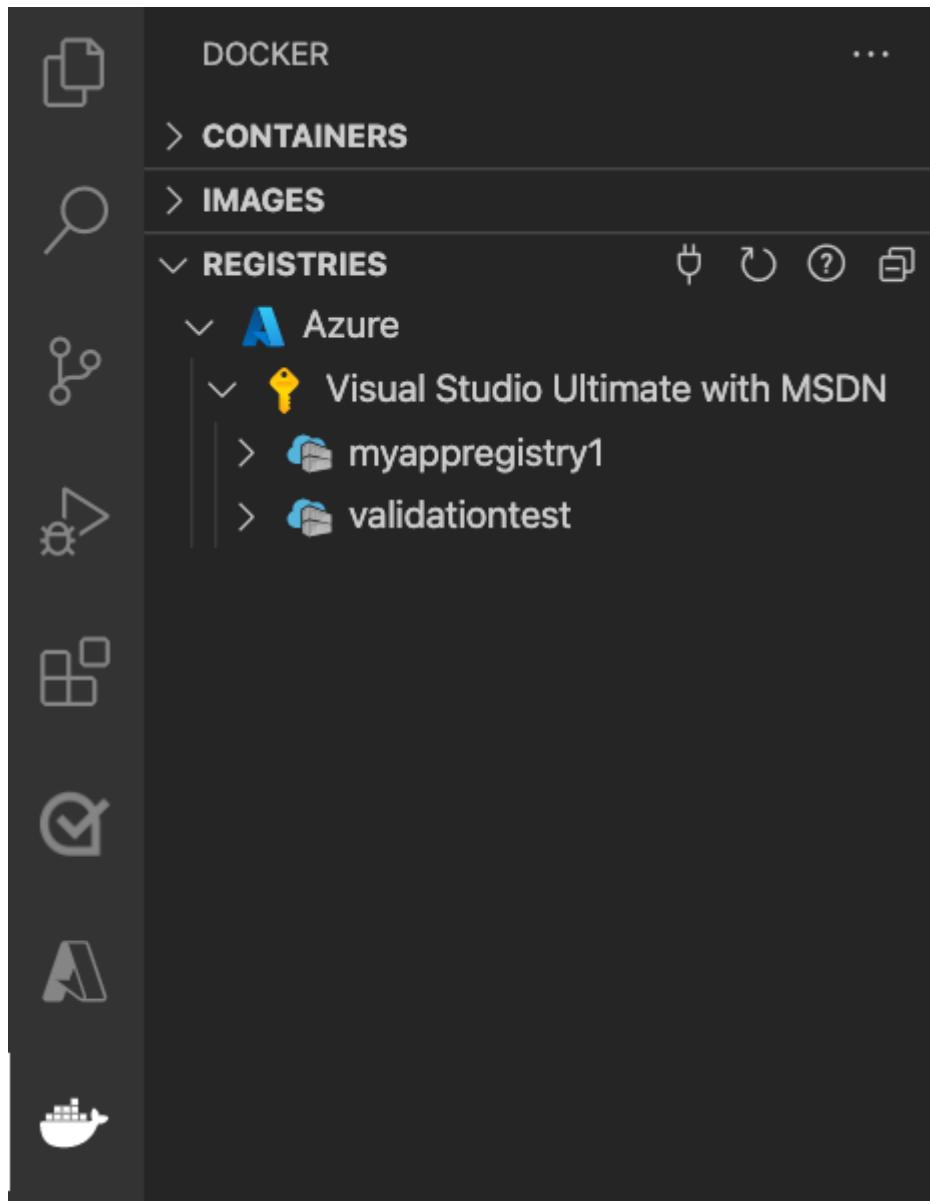
page in the Azure portal. This setting is required for App Service access. For managed identity, see [Deploy from ACR tutorial](#).

2 - Sign in

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



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



3 - Check prerequisites

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

```
Bash
docker --version
```

4 - Create and build image

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

`.NET`

Dockerfile

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

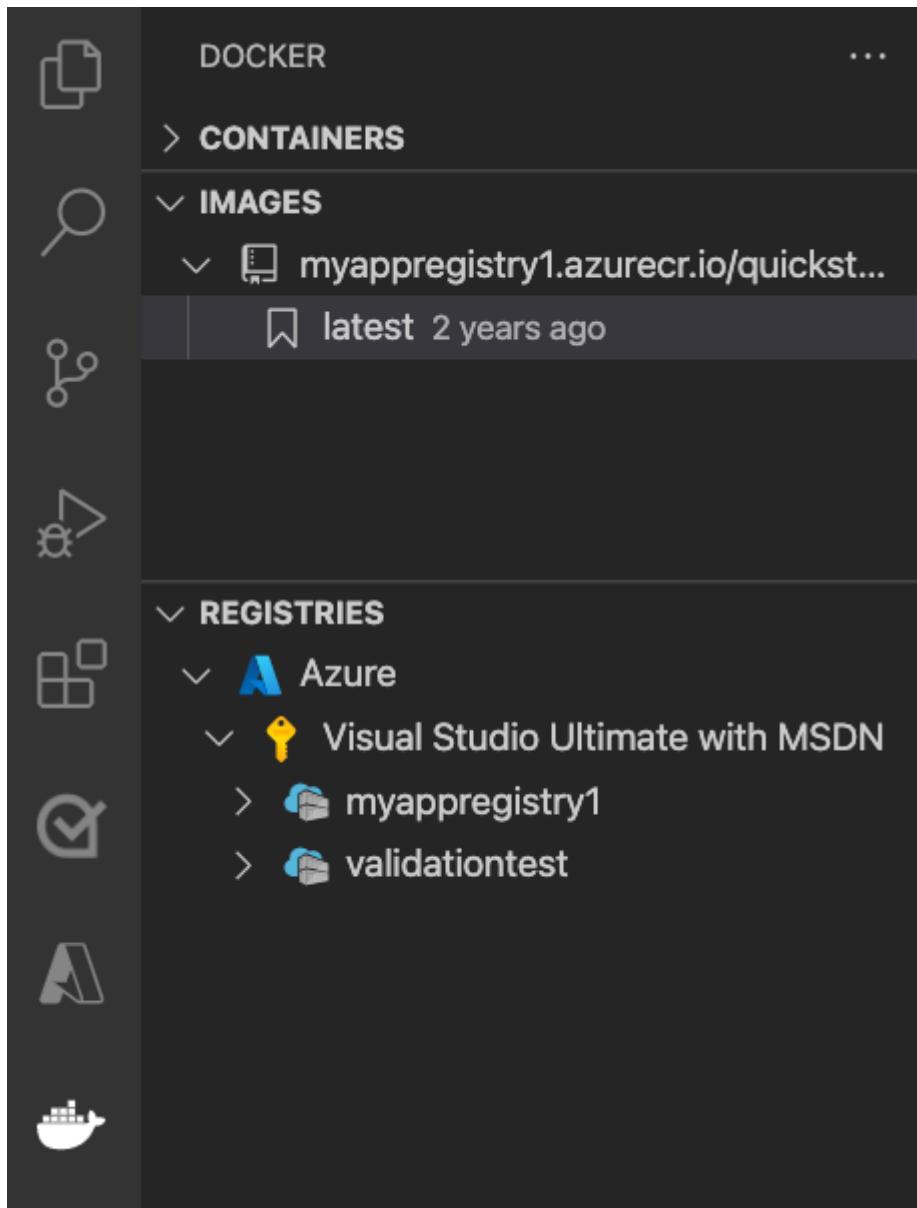
ENV PORT 8080
EXPOSE 8080

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

ENTRYPOINT ["dotnet", "/default/home/hostingstart/hostingstart.dll"]
```

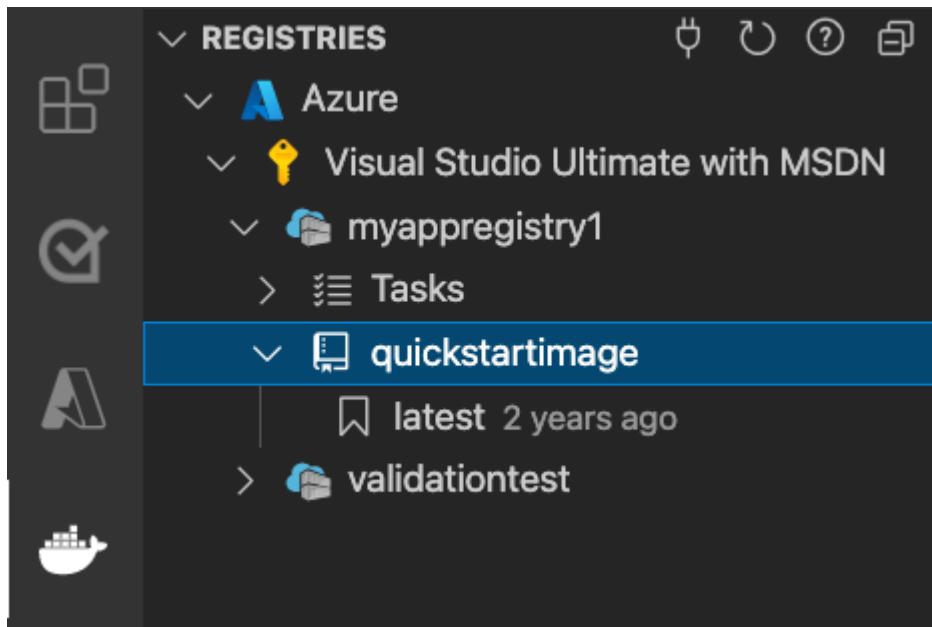
In this Dockerfile, the parent image is one of the built-in .NET containers of App Service. You can find the source files for it [in the Azure-App-Service/ImageBuilder GitHub repository, under GenerateDockerFiles/dotnetcore](#). Its [Dockerfile](#) copies a simple .NET app into `/default/home/hostingstart`. Your Dockerfile simply starts that app.

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



5 - Deploy to container registry

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



6 - Deploy to App Service

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

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

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

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

7 - Browse the website

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

I ran into an issue

8 - Clean up resources

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

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

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

Select **Delete resource group**, type **myResourceGroup** in the text box to confirm, and then select **Delete**.

Next steps

Congratulations, you've successfully completed this quickstart.

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

[Secure with custom domain and certificate](#)

[Migrate to Windows container in Azure](#)

[Integrate your app with an Azure virtual network](#)

[Use Private Endpoints for App Service apps](#)

[Azure Monitor overview](#)

[Application monitoring for Azure App Service overview](#)

[How to use managed identities for App Service and Azure Functions](#)

[Configure custom container](#)

[Multi-container app tutorial](#)

Other Azure extensions:

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

Quickstart: Create App Service app using an ARM template

Article • 02/13/2024

Get started with [Azure App Service](#) by deploying an app to the cloud using an Azure Resource Manager template (ARM template) and [Azure CLI](#) in Cloud Shell. A Resource Manager template is a JavaScript Object Notation (JSON) file that defines the infrastructure and configuration for your project. You incur no costs to complete this quickstart because you use a free App Service tier.

To complete this quickstart, you'll need an Azure account with an active subscription. If you don't have an Azure account, you can [create one for free](#).

Skip to the end

If you're familiar with using ARM templates, you can skip to the end by selecting this

 [Deploy to Azure](#)  button. This button opens the ARM template in the Azure portal.

Basics Review + create

Template

 app-service-docs-linux ↗
3 resources

[Edit template](#) [Edit parameters](#) [Visualize](#)

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

Region * ⓘ ▼

Web App Name ⓘ

Location ⓘ

Sku ⓘ

Linux Fx Version ⓘ

Repo Url ⓘ

[Previous](#) [Next](#) [Review + create](#)

In the Azure portal, select **Create new** to create a new Resource Group and then select the **Review + create** button to deploy the app.

Review the template

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

```
JSON

{
    "$schema": "https://schema.management.azure.com/schemas/2019-04-01/deploymentTemplate.json#",
    "contentVersion": "1.0.0.0",
    "metadata": {
        "_generator": {
            "name": "bicep",
            "version": "0.5.6.12127",
        }
    }
}
```

```
        "templateHash": "10602523904429381366"
    }
},
"parameters": {
    "webAppName": {
        "type": "string",
        "defaultValue": "[format('webApp-{0}', uniqueString(resourceGroup().id))]",
        "minLength": 2,
        "metadata": {
            "description": "Web app name."
        }
    },
    "location": {
        "type": "string",
        "defaultValue": "[resourceGroup().location]",
        "metadata": {
            "description": "Location for all resources."
        }
    },
    "sku": {
        "type": "string",
        "defaultValue": "F1",
        "metadata": {
            "description": "The SKU of App Service Plan."
        }
    },
    "linuxFxVersion": {
        "type": "string",
        "defaultValue": "DOTNETCORE|3.0",
        "metadata": {
            "description": "The Runtime stack of current web app"
        }
    },
    "repoUrl": {
        "type": "string",
        "defaultValue": " ",
        "metadata": {
            "description": "Optional Git Repo URL"
        }
    }
},
"variables": {
    "appServicePlanPortalName": "[format('AppServicePlan-{0}', parameters('webAppName'))]"
},
"resources": [
{
    "type": "Microsoft.Web/serverfarms",
    "apiVersion": "2021-02-01",
    "name": "[variables('appServicePlanPortalName')]",
    "location": "[parameters('location')]",
    "sku": {
        "name": "[parameters('sku')]"
    },
    "identity": {
        "type": "SystemAssigned"
    }
}
]
```

```

    "kind": "linux",
    "properties": {
        "reserved": true
    }
},
{
    "type": "Microsoft.Web/sites",
    "apiVersion": "2021-02-01",
    "name": "[parameters('webAppName')]",
    "location": "[parameters('location')]",
    "properties": {
        "httpsOnly": true,
        "serverFarmId": "[resourceId('Microsoft.Web/serverfarms',
variables('appServicePlanPortalName'))]",
        "siteConfig": {
            "linuxFxVersion": "[parameters('linuxFxVersion')]",
            "minTlsVersion": "1.2",
            "ftpsState": "FtpsOnly"
        }
    },
    "identity": {
        "type": "SystemAssigned"
    },
    "dependsOn": [
        "[resourceId('Microsoft.Web/serverfarms',
variables('appServicePlanPortalName'))]"
    ]
},
{
    "condition": "[contains(parameters('repoUrl'), 'http')]",
    "type": "Microsoft.Web/sites/sourcecontrols",
    "apiVersion": "2021-02-01",
    "name": "[format('{0}/{1}', parameters('webAppName'), 'web')]",
    "properties": {
        "repoUrl": "[parameters('repoUrl')]",
        "branch": "master",
        "isManualIntegration": true
    },
    "dependsOn": [
        "[resourceId('Microsoft.Web/sites', parameters('webAppName'))]"
    ]
}
]
}

```

Two Azure resources are defined in the template:

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

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

Parameters	Type	Default value	Description
webAppName	string	webApp-<uniqueString>	App name based on a unique string value
appServicePlanName	string	webAppPlan-<uniqueString>	App Service Plan name based on a unique string value
location	string	[resourceGroup().location]	App region
sku	string	F1	Instance size (F1 = Free Tier)
linuxFxVersion	string	DOTNETCORE 3.0	"Programming language stack Version"
repoUrl	string		External Git repo (optional)

Deploy the template

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

The following code creates a resource group, an App Service plan, and a web app. A default resource group, App Service plan, and location have been set for you. Replace `<app-name>` with a globally unique app name (valid characters are `a-z`, `0-9`, and `-`).

Run the following commands to create a Python app on Linux:

Azure CLI

```
az group create --name myResourceGroup --location "southcentralus"

az deployment group create --resource-group myResourceGroup --parameters
webAppName=<app-name> linuxFxVersion="PYTHON|3.9" \
--template-uri "https://raw.githubusercontent.com/Azure/azure-quickstart-
templates/master/quickstarts/microsoft.web/app-service-docs-
linux/azuredeploy.json"
```

To deploy a different language stack, update `linuxFxVersion` with appropriate values. Samples are shown in the table. To show current versions, run the following command in the Cloud Shell: `az webapp config show --resource-group myResourceGroup --name <app-name> --query linuxFxVersion`

Language	Example
.NET	linuxFxVersion="DOTNETCORE 3.0"
PHP	linuxFxVersion="PHP 7.4"
Node.js	linuxFxVersion="NODE 10.15"
Java	linuxFxVersion="JAVA 1.8 TOMCAT 9.0"
Python	linuxFxVersion="PYTHON 3.7"

① Note

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

Validate the deployment

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



Microsoft Azure

Your web app is running and waiting for your content

Your web app is live, but we don't have your content yet. If you've already deployed, it could take up to 5 minutes for your content to show up, so come back soon.



.NET Built with .Net Core

Haven't deployed yet?
Use the deployment center to publish code or set up continuous deployment.

Starting a new web site?
Follow our Quickstart guide to get a web app ready quickly.

[Deployment center](#) [Quickstart](#)

Clean up resources

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

Next steps

[Deploy from local Git](#)

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

[Python with Postgres](#)

[PHP with MySQL](#)

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

[Secure with custom domain and certificate](#)

Deployment best practices

Article • 01/24/2025

ⓘ Note

Starting June 1, 2024, newly created App Service apps can generate a unique default hostname that uses the naming convention `<app-name>-<random-hash>. <region>.azurewebsites.net`. Existing app names remain unchanged. For example:

`myapp-ds27dh7271aah175.westus-01.azurewebsites.net`

For more information, see [Unique Default Hostname for App Service Resource](#).

Every development team has unique requirements that can make implementing an efficient deployment pipeline difficult on any cloud service. This article introduces the three main components of deploying to Azure App Service: *deployment sources*, *build pipelines*, and *deployment mechanisms*. This article also covers some best practices and tips for specific language stacks.

Deployment components

This section describes the three main components for deploying to App Service.

Deployment source

A *deployment source* is the location of your application code. For production apps, the deployment source is usually a repository hosted by version control software such as [GitHub](#), [BitBucket](#), or [Azure Repos](#). For development and test scenarios, the deployment source might be [a project on your local machine](#).

Build pipeline

After you decide on a deployment source, your next step is to choose a *build pipeline*. A build pipeline reads your source code from the deployment source and runs a series of steps to get the application in a runnable state.

Steps can include compiling code, minifying HTML and JavaScript, running tests, and packaging components. The specific commands run by the build pipeline depend on

your language stack. You can run these operations on a build server, such as Azure Pipelines, or locally.

Deployment mechanism

The *deployment mechanism* is the action used to put your built application into the `/home/site/wwwroot` directory of your web app. The `/wwwroot` directory is a mounted storage location shared by all instances of your web app. When the deployment mechanism puts your application in this directory, your instances receive a notification to sync the new files.

App Service supports the following deployment mechanisms:

- Kudu endpoints: [Kudu](#) is the open-source developer productivity tool that runs as a separate process in Windows App Service. It runs as a second container in Linux App Service. Kudu handles continuous deployments and provides HTTP endpoints for deployment, such as [zipdeploy/](#).
- FTP and WebDeploy: Using your [site or user credentials](#), you can upload files via [FTP](#) or WebDeploy. These mechanisms don't go through Kudu.

Deployment tools such as Azure Pipelines, Jenkins, and editor plugins use one of these deployment mechanisms.

Use deployment slots

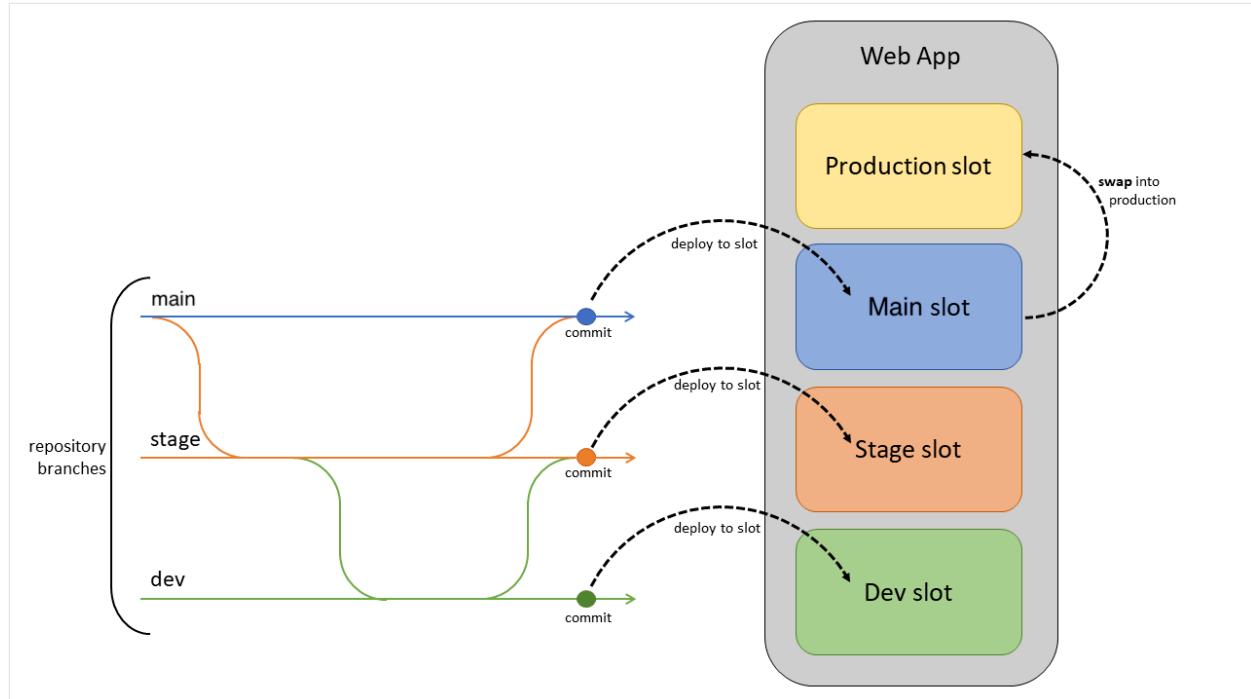
Whenever possible, when you deploy a new production build, use [deployment slots](#). With a Standard App Service Plan tier or better, you can deploy your app to a staging environment, validate your changes, and do smoke tests. When you're ready, swap your staging and production slots. The swap operation warms up the necessary worker instances to match your production scale, which eliminates downtime.

Continuously deploy code

If your project has branches designated for testing, QA, and staging, each of those branches should be continuously deployed to a staging slot. This approach is known as the [Gitflow design](#). This design allows your stakeholders to easily assess and test the deployed branch.

Continuous deployment should never be enabled for your production slot. Instead, your production branch (often main) should be deployed onto a nonproduction slot. When you're ready to release the base branch, swap it into the production slot. Swapping into

production, instead of deploying to production, prevents downtime and allows you to roll back the changes by swapping again.

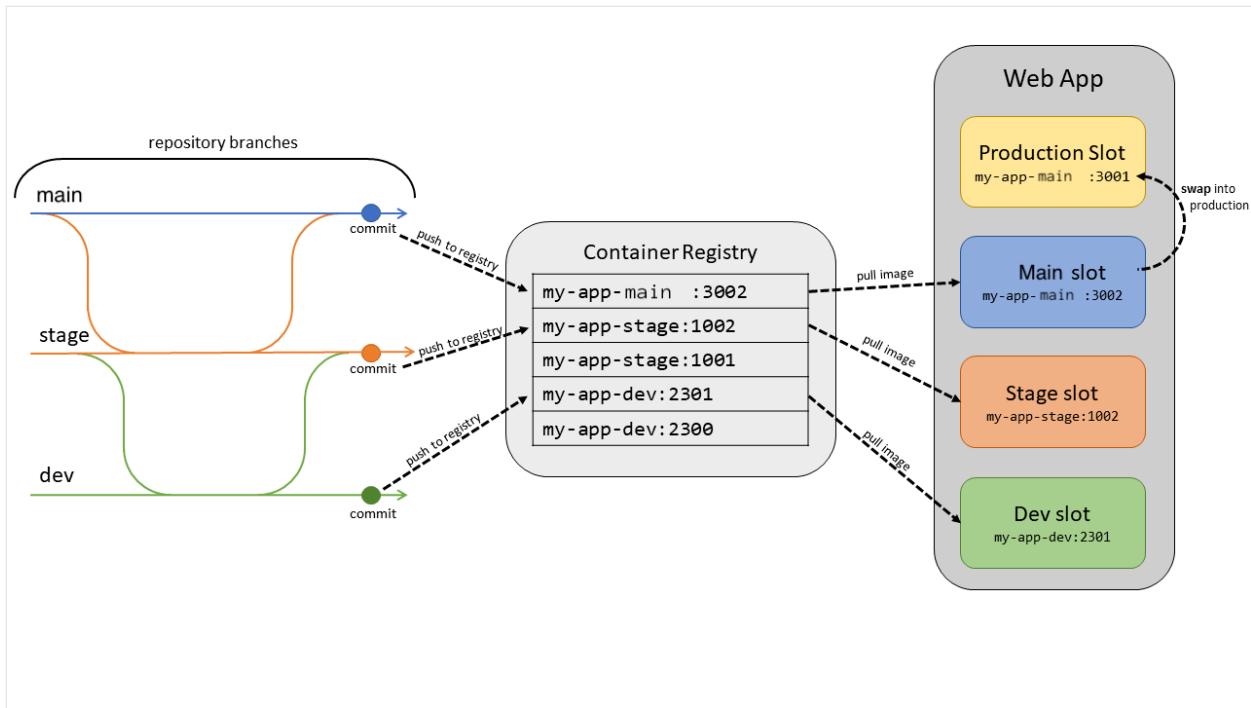


Continuously deploy containers

For custom containers from Docker or other container registries, deploy the image into a staging slot and swap into production to prevent downtime. The automation is more complex than code deployment. You must push the image to a container registry and update the image tag on the webapp.

For each branch you want to deploy to a slot, set up automation to do these tasks on each commit to the branch.

1. **Build and tag the image.** As part of the build pipeline, tag the image with the git commit ID, timestamp, or other identifiable information. It's best not to use the default *latest* tag. Otherwise, it's difficult to trace back what code is currently deployed, which makes debugging more difficult.
2. **Push the tagged image.** After the image is built and tagged, the pipeline pushes the image to the container registry. In the next step, the deployment slot pulls the tagged image from the container registry.
3. **Update the deployment slot with the new image tag.** When this property is updated, the site automatically restarts and pulls the new container image.



This article contains examples for common automation frameworks.

Use Azure DevOps

App Service has [built-in continuous delivery](#) for containers through the Deployment Center. Navigate to your app in the [Azure portal](#). Under **Deployment**, select **Deployment Center**. Follow the instructions to select your repository and branch. This approach configures a DevOps build-and-release pipeline to automatically build, tag, and deploy your container when new commits are pushed to your selected branch.

Use GitHub Actions

You can also automate your container deployment with [GitHub Actions](#). The workflow file builds and tags the container with the commit ID, pushes it to a container registry, and updates the specified web app with the new image tag.

```

YAML

on:
  push:
    branches:
      - <your-branch-name>

  name: Linux Container Node Workflow

jobs:
  build-and-deploy:
    runs-on: ubuntu-latest
    steps:

```

```

# checkout the repo
- name: 'Checkout GitHub Action'
  uses: actions/checkout@main

- uses: azure/docker-login@v1
  with:
    login-server: contoso.azurecr.io
    username: ${{ secrets.REGISTRY_USERNAME }}
    password: ${{ secrets.REGISTRY_PASSWORD }}

- run: |
  docker build . -t contoso.azurecr.io/nodejssampleapp:${{ github.sha }}
}

docker push contoso.azurecr.io/nodejssampleapp:${{ github.sha }}

- uses: azure/webapps-deploy@v2
  with:
    app-name: 'node-rnc'
    publish-profile: ${{ secrets.azureWebAppPublishProfile }}
    images: 'contoso.azurecr.io/nodejssampleapp:${{ github.sha }}'
```

Use other automation providers

The steps listed earlier apply to other automation utilities such as CircleCI or Travis CI. However, you need to use the Azure CLI to update the deployment slots with new image tags in the final step. To use the Azure CLI in your automation script, generate a Service Principal using the following command.

Azure CLI

```
az ad sp create-for-rbac --name "myServicePrincipal" --role contributor \
--scopes /subscriptions/{subscription}/resourceGroups/{resource-group} \
--sdk-auth
```

In your script, sign in using `az login --service-principal`, providing the principal information. You can then use `az webapp config container set` to set the container name, tag, registry URL, and registry password. For more information, see [How to sign in to the Azure CLI on Circle CI ↗](#).

Language-specific considerations

Keep in mind the following considerations for Java, Node, and .NET implementations.

Java

Use the Kudu [zipdeploy](#) API for deploying JAR applications. Use [wardeploy](#) for WAR apps. If you're using Jenkins, you can use those APIs directly in your deployment phase. For more information, see [Deploy to Azure App Service with Jenkins](#).

Node

By default, Kudu runs the build steps for your Node application (`npm install`). If you're using a build service such as Azure DevOps, the Kudu build is unnecessary. To disable the Kudu build, create an app setting, `SCM_DO_BUILD_DURING_DEPLOYMENT`, with a value of `false`.

.NET

By default, Kudu runs the build steps for your .NET application (`dotnet build`). If you're using a build service such as Azure DevOps, the Kudu build is unnecessary. To disable the Kudu build, create an app setting, `SCM_DO_BUILD_DURING_DEPLOYMENT`, with a value of `false`.

Other deployment considerations

Other considerations include local cache and high CPU or memory.

Local cache

Azure App Service content is stored on Azure Storage and is surfaced up in a durable manner as a content share. However, some apps just need a high-performance, read-only content store that they can run with high availability. These apps can benefit from using *local cache*. For more information, see [Azure App Service Local Cache overview](#).

Note

Local cache isn't recommended for content management sites such as WordPress.

To prevent downtime, always use local cache with [deployment slots](#). For information on using these features together, see [Best practices](#).

High CPU or memory

If your App Service Plan is using over 90% of available CPU or memory, the underlying virtual machine might have trouble processing your deployment. When this situation happens, temporarily scale up your instance count to perform the deployment. After the deployment finishes, you can return the instance count to its previous value.

For more information, visit [App Service Diagnostics](#) to find out actionable best practices specific to your resource.

1. Navigate to your Web App in the [Azure portal](#).
2. Select **Diagnose and solve problems** in the left navigation, which opens App Service Diagnostics.
3. Choose **Availability and Performance** or explore other options, such as **High CPU Analysis**.

View the current state of your app in regards to these best practices.

You can also use this link to directly open App Service Diagnostics for your resource:

`https://portal.azure.com/?`

`websitesextension_ext=asd.featurePath%3Ddetectors%2FParentAvailabilityAndPerformance#@microsoft.onmicrosoft.com/resource/subscriptions/{subscriptionId}/resourceGroups/{resourceGroupName}/providers/Microsoft.Web/sites/{siteName}/troubleshoot .`

Related content

- [Environment variables and app settings reference](#)
-

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

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

Article • 09/15/2024

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

In this tutorial, you learn how to:

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

You can complete this tutorial on macOS, Linux, or Windows.

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

Prerequisites

- [Install Git](#).
- [Install the latest .NET Core 3.1 SDK](#).

Create a local ASP.NET Core app

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

Clone the sample application

1. In the terminal window, use `cd` to go to a working directory.
2. Clone the sample repository, and then go to the repository root.

Bash

```
git clone https://github.com/Azure-Samples/dotnet-core-api
```

```
cd dotnet-core-api
```

This repository contains an app that's created based on the tutorial [ASP.NET Core web API documentation with Swagger / OpenAPI](#). It uses a Swagger generator to serve the [Swagger UI](#) and the Swagger JSON endpoint.

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

```
Bash
```

```
git branch -m main
```

 **Tip**

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

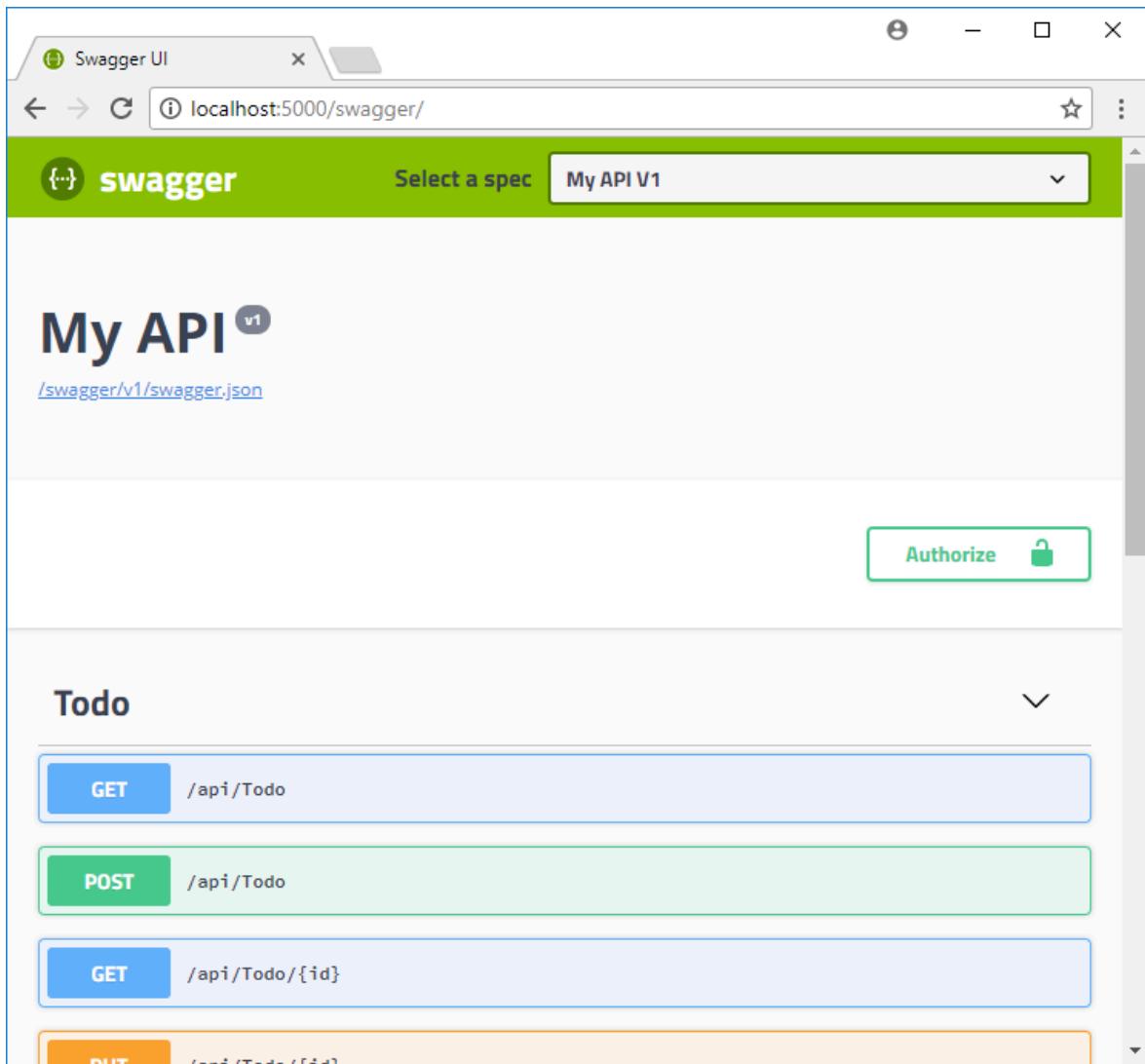
Run the application

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

```
Bash
```

```
dotnet restore  
dotnet run
```

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



3. Navigate to `http://localhost:5000/api/todo` to see a list of ToDo JSON items.
4. Navigate to `http://localhost:5000` and experiment with the browser app. Later, you'll point the browser app to a remote API in App Service to test CORS functionality. Code for the browser app is found in the repository's `wwwroot` directory.
5. To stop ASP.NET Core at any time, select **Ctrl+C** in the terminal.

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:

[] Expand table

Option	Example/Link
Select Try It in the upper-right corner of a code or command block. Selecting Try It doesn't automatically copy the code or command 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 use Azure Cloud Shell:

1. Start Cloud Shell.
2. Select the **Copy** button on a code block (or command block) to copy the code or command.
3. Paste the code or command 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 or command.

Deploy the app to Azure

In this step, you deploy your .NET Core application to App Service.

Configure local Git deployment

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

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

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

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

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

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

Create a resource group

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

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

Azure CLI

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

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

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

Create an App Service plan

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

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

Azure CLI

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

Create a web app

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

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

Azure CLI

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

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

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

```
< JSON data removed for brevity. >  
}
```

ⓘ Note

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

Push to Azure from Git

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

Azure CLI

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

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

Bash

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

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

Bash

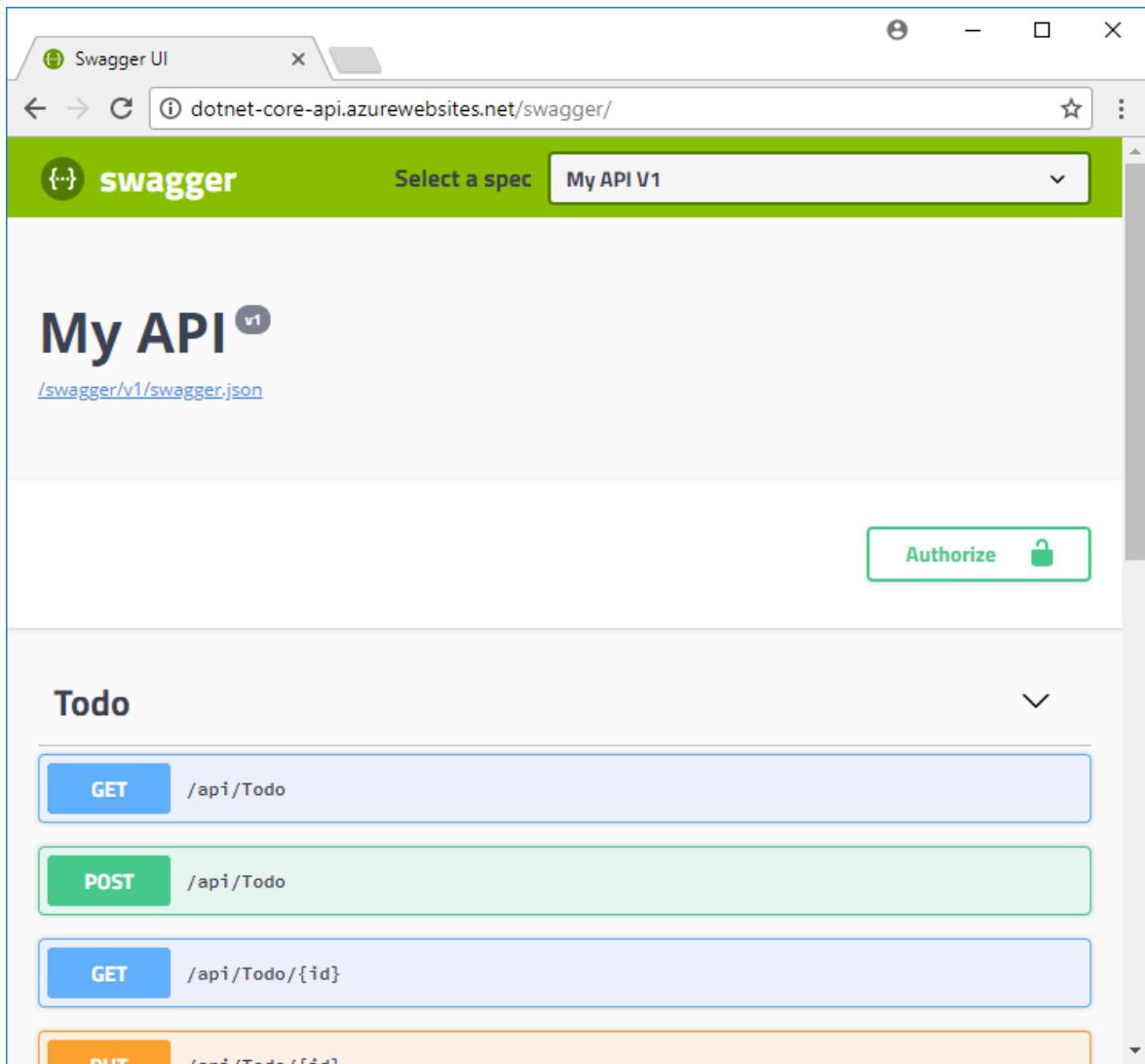
```
git push azure main
```

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

```
Enumerating objects: 83, done.
Counting objects: 100% (83/83), done.
Delta compression using up to 8 threads
Compressing objects: 100% (78/78), done.
Writing objects: 100% (83/83), 22.15 KiB | 3.69 MiB/s, done.
Total 83 (delta 26), reused 0 (delta 0)
remote: Updating branch 'master'.
remote: Updating submodules.
remote: Preparing deployment for commit id '509236e13d'.
remote: Generating deployment script.
remote: Project file path: .\TodoApi.csproj
remote: Generating deployment script for ASP.NET MSBuild16 App
remote: Generated deployment script files
remote: Running deployment command...
remote: Handling ASP.NET Core Web Application deployment with MSBuild16.
remote: .
remote: .
remote: .
remote: Finished successfully.
remote: Running post deployment command(s)...
remote: Triggering recycle (preview mode disabled).
remote: Deployment successful.
To https://<app_name>.scm.azurewebsites.net/<app_name>.git
 * [new branch]      master -> master
```

Browse to the Azure app

1. Navigate to `http://<app_name>.azurewebsites.net/swagger` in a browser and view the Swagger UI.



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

Add CORS functionality

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

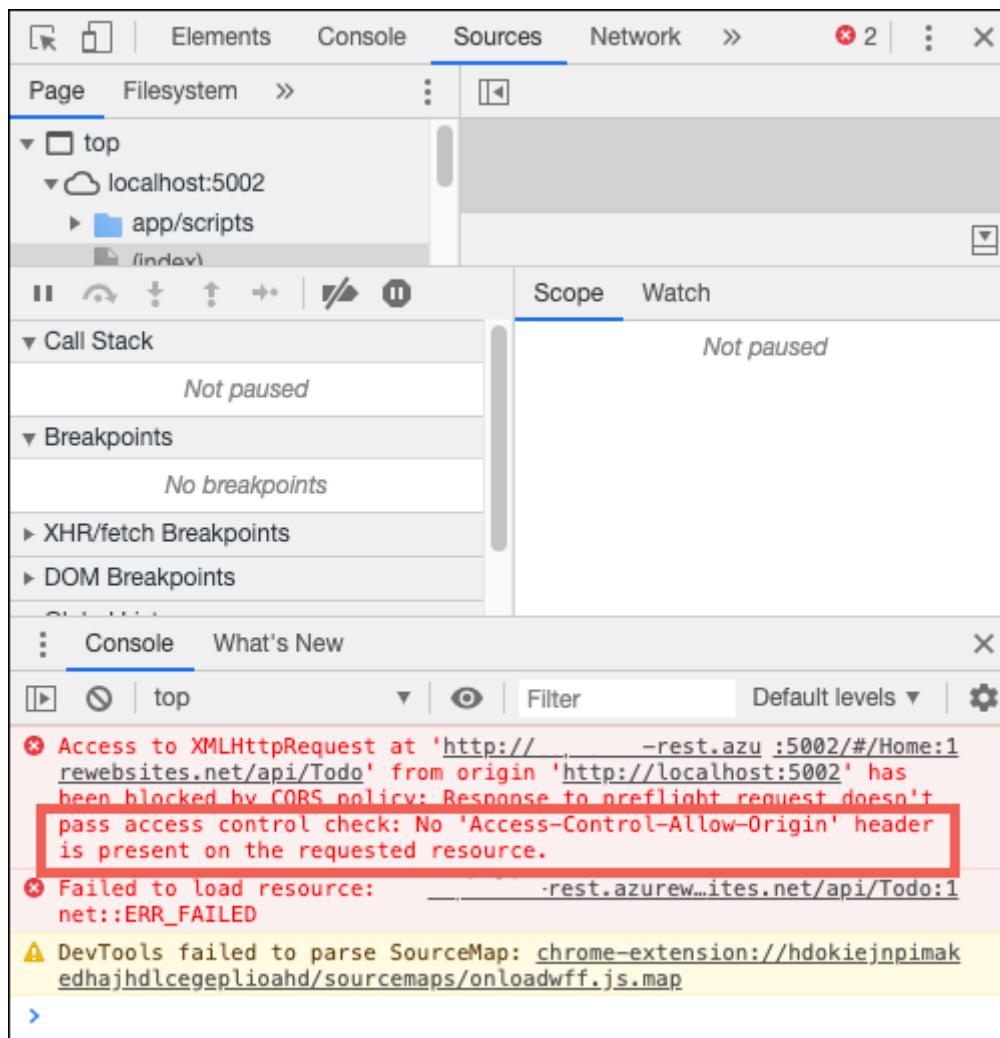
Test CORS in the sample app

1. In your local repository, open `wwwroot/index.html`.
2. On line 51, set the `apiEndpoint` variable to the URL of your deployed API (`http://<app_name>.azurewebsites.net`). Replace `<appname>` with your app name in App Service.
3. In your local terminal window, run the sample app again.

```
Bash
```

```
dotnet run
```

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



The domain mismatch between the browser app (`http://localhost:5000`) and remote resource (`http://<app_name>.azurewebsites.net`) is recognized by your browser as a cross-origin resource request. Also, because the App Service app isn't sending the `Access-Control-Allow-Origin` header, the browser has prevented cross-domain content from loading.

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

Enable CORS

In Cloud Shell, enable CORS to your client's URL by using the [az webapp cors add](#) command. Replace the <app-name> placeholder.

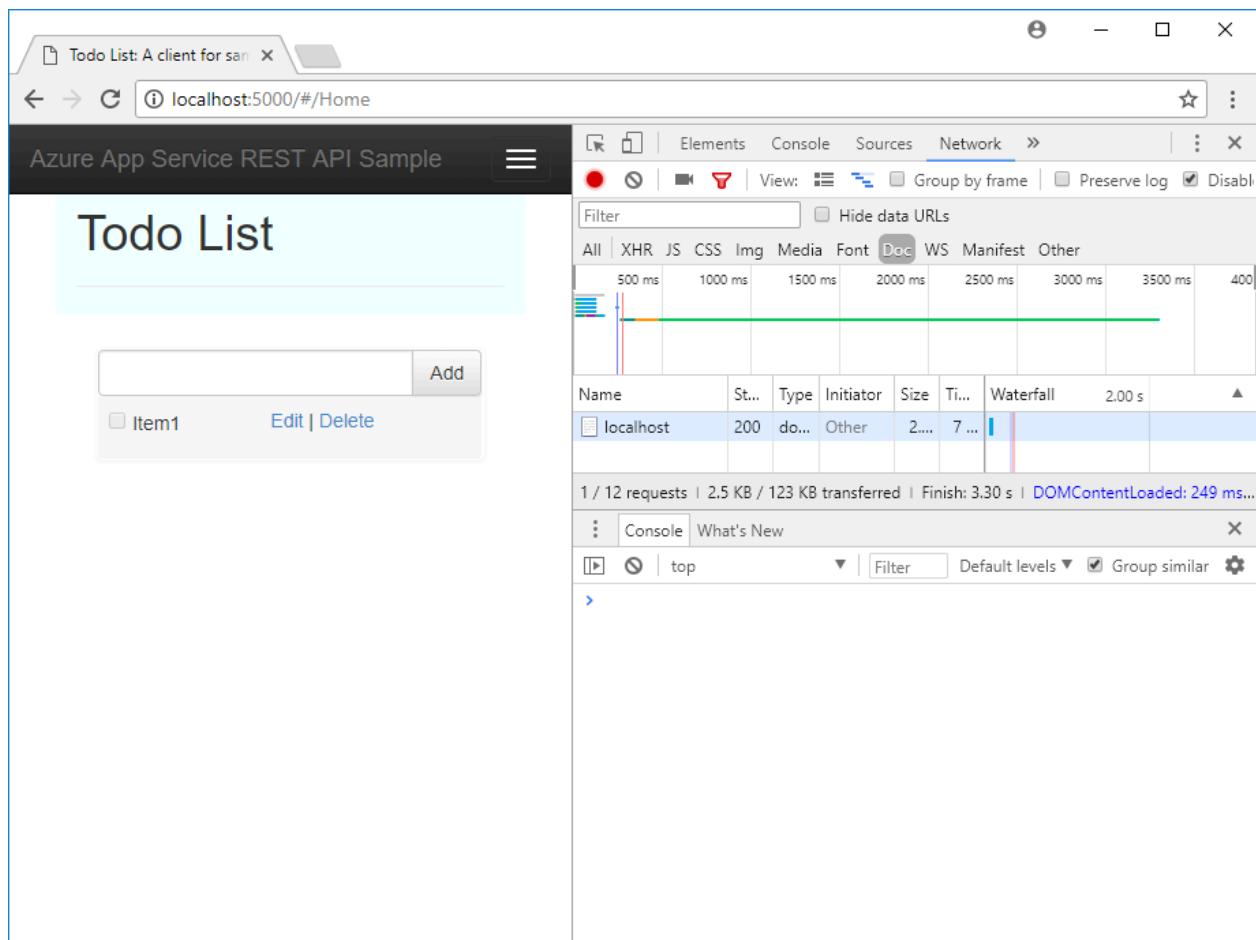
```
Azure CLI

az webapp cors add --resource-group myResourceGroup --name <app-name> --
--allowed-origins 'http://localhost:5000'
```

You can add multiple allowed origins by running the command multiple times or by adding a comma-separated list in `--allowed-origins`. To allow all origins, use `--allowed-origins '*'`.

Test CORS again

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



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

Frequently asked questions

- [App Service CORS vs. your CORS](#)
- [How do I set allowed origins to a wildcard subdomain?](#)
- [How do I enable the ACCESS-CONTROL-ALLOW-CREDENTIALS header on the response?](#)

App Service CORS vs. your CORS

You can use your own CORS utilities instead of App Service CORS for more flexibility. For example, you might want to specify different allowed origins for different routes or methods. Since App Service CORS lets you specify only one set of accepted origins for all API routes and methods, you would want to use your own CORS code. See how CORS is enabled in ASP.NET Core at [Enable CORS](#).

The built-in App Service CORS feature doesn't have options to allow only specific HTTP methods or verbs for each origin that you specify. It will automatically allow all methods and headers for each origin defined. This behavior is similar to [ASP.NET Core CORS](#) policies when you use the options `.AllowAnyHeader()` and `.AllowAnyMethod()` in the code.

ⓘ Note

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

How do I set allowed origins to a wildcard subdomain?

A wildcard subdomain like `*.contoso.com` is more restrictive than the wildcard origin `*`. The app's CORS management page in the Azure portal doesn't let you set a wildcard subdomain as an allowed origin. However, you can do that by using Azure CLI, like so:

Azure CLI

```
az webapp cors add --resource-group <group-name> --name <app-name> --  
allowed-origins 'https://*.contoso.com'
```

How do I enable the ACCESS-CONTROL-ALLOW-CREDENTIALS header on the response?

If your app requires credentials such as cookies or authentication tokens to be sent, the browser might require the `ACCESS-CONTROL-ALLOW-CREDENTIALS` header on the response. To enable this in App Service, set `properties.cors.supportCredentials` to `true`:

Azure CLI

```
az resource update --name web --resource-group <group-name> \
--namespace Microsoft.Web --resource-type config \
--parent sites/<app-name> --set properties.cors.supportCredentials=true
```

This operation isn't allowed when allowed origins include the wildcard origin `'*'`. Specifying `AllowAnyOrigin` and `AllowCredentials` isn't secure. Doing so can result in cross-site request forgery. To allow credentials, try replacing the wildcard origin with [wildcard subdomains](#).

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:

Azure CLI

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

This command may take a minute to run.

Next steps

What you learned:

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

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

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

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

Configure an App Service app

Article • 10/03/2024

ⓘ Note

Starting June 1, 2024, all newly created App Service apps will have the option to generate a unique default hostname using the naming convention `<app-name>-<random-hash>. <region>.azurewebsites.net`. Existing app names will remain unchanged.

Example: `myapp-ds27dh7271aah175.westus-01.azurewebsites.net`

For further details, refer to [Unique Default Hostname for App Service Resource ↗](#).

This article explains how to configure common settings for web apps, mobile back end, or API app. For Azure Functions, see [App settings reference for Azure Functions](#).

Configure app settings

ⓘ Note

- App settings names can only contain letters, numbers (0-9), periods ("."), and underscores ("_")
- Special characters in the value of an App Setting must be escaped as needed by the target OS

For example to set an environment variable in App Service Linux with the value `"pa$$w0rd\"` the string for the app setting should be: `"pa\$\$w0rd\\\"`

In App Service, app settings are variables passed as environment variables to the application code. For Linux apps and custom containers, App Service passes app settings to the container using the `--env` flag to set the environment variable in the container. In either case, they're injected into your app environment at app startup. When you add, remove, or edit app settings, App Service triggers an app restart.

For ASP.NET and ASP.NET Core developers, setting app settings in App Service are like setting them in `<appSettings>` in `Web.config` or `appsettings.json`, but the values in App Service override the ones in `Web.config` or `appsettings.json`. You can keep development settings (for example, local MySQL password) in `Web.config` or `appsettings.json` and

production secrets (for example, Azure MySQL database password) safely in App Service. The same code uses your development settings when you debug locally, and it uses your production secrets when deployed to Azure.

Other language stacks, likewise, get the app settings as environment variables at runtime. For language-stack specific steps, see:

- [ASP.NET Core](#)
- [Node.js](#)
- [PHP](#)
- [Python](#)
- [Java](#)
- [Custom containers](#)

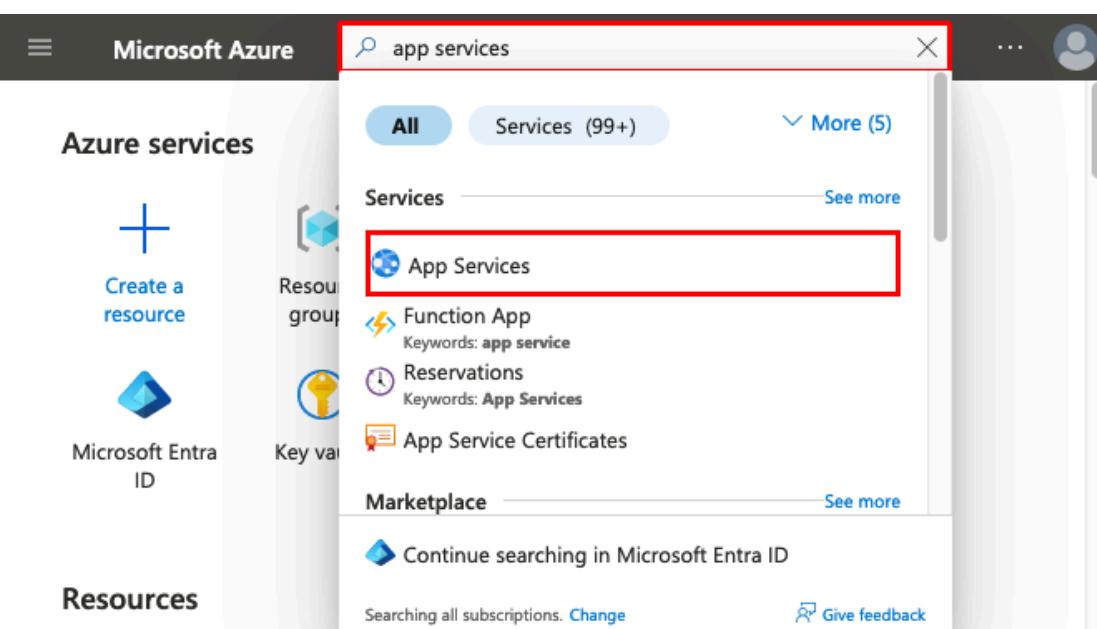
App settings are always encrypted when stored (encrypted-at-rest).

ⓘ Note

If you store secrets in app settings, consider using [Key Vault references](#). If your secrets are for connectivity to back-end resources, consider more secure connectivity options that don't require secrets at all. For more information, see [Secure connectivity to Azure services and databases from Azure App Service](#).

Azure portal

1. In the [Azure portal](#), search for and select **App Services**, and then select your app.



The screenshot shows the Microsoft Azure portal interface. At the top, there's a search bar with the text "app services". Below the search bar, there are three tabs: "All", "Services (99+)", and "More (5)". Under the "Services" tab, there's a list of items. The "App Services" item is highlighted with a red rectangle. Other items listed include "Function App", "Reservations", "App Service Certificates", and "Marketplace". On the left side of the portal, there's a sidebar with sections for "Azure services" (including "Create a resource", "Resource groups", "Microsoft Entra ID", and "Key vault"), "Resources", and "Marketplace".

2. In the app's left menu, select **Environment variables** > **App settings**.

The screenshot shows the Azure portal interface for managing app settings. The left sidebar has a tree view with items like Deployment Center, Performance, Load Testing, Settings, Configuration, Authentication, Application Insights, Identity, and Backups. The 'Settings' node is expanded, and the 'Environment variables' item under it is selected and highlighted with a red box. The main content area is titled 'App settings' and contains a table for managing environment variables. At the top of the table area are buttons for 'Search', 'Add', 'Refresh', 'Show values', and 'Advanced edit'. Below the table are 'Apply' and 'Discard' buttons. At the bottom right of the main area is a link 'Send us your feedback'.

By default, values for app settings are hidden in the portal for security. To see a hidden value of an app setting, select its **Value** field. To see the hidden values of all app settings, select the **Show values** button.

3. To add a new app setting, select **Add**. To edit a setting, click the setting.
4. In the dialog, you can [stick the setting to the current slot](#).

(!) Note

In a default Linux app service or a custom Linux container, any nested JSON key structure in the app setting name like `ApplicationInsights:InstrumentationKey` needs to be configured in App Service as `ApplicationInsights__InstrumentationKey` for the key name. In other words, any `:` should be replaced by `_` (double underscore). Any periods in the app setting name will be replaced with a `_` (single underscore).

5. When finished, select **Apply**. Don't forget to select **Apply** back in the **Environment variables** page.

Edit app settings in bulk

Azure portal

Select the **Advanced edit** button. Edit the settings in the text area. When finished, select **OK**. Don't forget to select **Apply** back in the **Environment variables** page.

App settings have the following JSON formatting:

JSON

```
[  
  {  
    "name": "<key-1>",  
    "value": "<value-1>",  
    "slotSetting": false  
  },  
  {  
    "name": "<key-2>",  
    "value": "<value-2>",  
    "slotSetting": false  
  },  
  ...  
]
```

Configure connection strings

ⓘ Note

Consider more secure connectivity options that don't require connection secrets at all. For more information, see [Secure connectivity to Azure services and databases from Azure App Service](#).

For ASP.NET and ASP.NET Core developers, setting connection strings in App Service are like setting them in `<connectionStrings>` in *Web.config*, but the values you set in App Service override the ones in *Web.config*. You can keep development settings (for example, a database file) in *Web.config* and production secrets (for example, SQL Database credentials) safely in App Service. The same code uses your development settings when you debug locally, and it uses your production secrets when deployed to Azure.

For other language stacks, it's better to use [app settings](#) instead, because connection strings require special formatting in the variable keys in order to access the values.

ⓘ Note

There is one case where you may want to use connection strings instead of app settings for non-.NET languages: certain Azure database types are backed up along with the app *only* if you configure a connection string for the database in your App

Service app. For more information, see [Create a custom backup](#). If you don't need this automated backup, then use app settings.

At runtime, connection strings are available as environment variables, prefixed with the following connection types:

- SQLServer: `SQLCONNSTR_`
- MySQL: `MYSQLCONNSTR_`
- SQLAzure: `SQLAZURECONNSTR_`
- Custom: `CUSTOMCONNSTR_`
- PostgreSQL: `POSTGRESQLCONNSTR_`
- Notification Hub: `NOTIFICATIONHUBCONNSTR_`
- Service Bus: `SERVICEBUSCONNSTR_`
- Event Hub: `EVENTHUBCONNSTR_`
- Document DB: `DOCDBCONNSTR_`
- Redis Cache: `REDISCACHECONNSTR_`

Note

.NET apps targeting PostgreSQL, Notification Hub, Service Bus, Event Hub, Document Db and Redis Cache should set the connection string to **Custom** as workaround for a [known issue in .NET EnvironmentVariablesConfigurationProvider](#)

For example, a MySQL connection string named *connectionstring1* can be accessed as the environment variable `MYSQLCONNSTR_connectionString1`. For language-stack specific steps, see:

- [ASP.NET Core](#)
- [Node.js](#)
- [PHP](#)
- [Python](#)
- [Java](#)
- [Custom containers](#)

Connection strings are always encrypted when stored (encrypted-at-rest).

Note

Connection strings can also be resolved from [Key Vault](#) using [Key Vault references](#).

1. In the [Azure portal](#), search for and select App Services, and then select your app.

The screenshot shows the Microsoft Azure portal's search interface. At the top, there is a search bar with the text "app services". Below the search bar, there are two tabs: "All" (selected) and "Services (99+)". To the right of these tabs is a "More (5)" link. The main area is titled "Services" and contains several items: "App Services" (which is highlighted with a red box), "Function App", "Reservations", "App Service Certificates", and a "Marketplace" section. On the left side of the screen, there is a sidebar with sections for "Azure services" (including "Create a resource", "Resource groups", "Microsoft Entra ID", and "Key vault"), "Resources" (with a "Load Testing" icon), and "Settings" (with "Environment variables" highlighted with a red box). At the bottom of the search results, there is a note "Searching all subscriptions. Change" and a "Give feedback" link.

2. In the app's left menu, select Environment variables > Connection strings.

The screenshot shows the "App settings" page for an Azure app. The left sidebar has sections for "Deployment Center", "Performance" (with "Load Testing" listed), and "Settings" (with "Environment variables" highlighted with a red box). The main content area has tabs for "App settings" and "Connection strings" (which is highlighted with a red box). Below the tabs is a search bar and a row of buttons: "Add", "Refresh", "Show values", and "...". A table follows, with columns for "Name", "Value", and "Deployment slot settir". At the bottom of the table are "Apply" and "Discard" buttons, and a "Send us your feedback" link.

By default, values for connection strings are hidden in the portal for security. To see a hidden value of a connection string, select its **Value** field. To see the hidden values of all connection strings, select the **Show value** button.

3. To add a new connection string, select **Add**. To edit a connection string, select the connection string.
4. In the dialog, you can [stick the connection string to the current slot](#).

- When finished, select **Apply**. Don't forget to select **Apply** back in the **Environment variables** page.

Edit connection strings in bulk

Azure portal

Select the **Advanced edit** button. Edit the connection strings in the text area. When finished, select **Apply**. Don't forget to select **Apply** back in the **Environment variables** page.

Connection strings have the following JSON formatting:

JSON

```
[  
  {  
    "name": "name-1",  
    "value": "conn-string-1",  
    "type": "SQLServer",  
    "slotSetting": false  
  },  
  {  
    "name": "name-2",  
    "value": "conn-string-2",  
    "type": "PostgreSQL",  
    "slotSetting": false  
  },  
  ...  
]
```

Configure language stack settings

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

Configure general settings

In the [Azure portal](#), search for and select **App Services**, and then select your app.

In the app's left menu, select **Configuration > General settings**.

The screenshot shows the Azure App Service configuration interface. On the left, there is a sidebar with various options: Deployment Center, Performance, Load Testing, Settings, Environment variables, Configuration (which is highlighted with a red box), Authentication, Application Insights, Identity, and Backups. The main area has a header with a search bar, refresh, save, discard, and leave feedback buttons. A message at the top states: "Custom Error pages requires a premium App Service Plan." Below this, a blue bar says: "View and edit your application settings and connection strings from Environment variables. Click here to go to Environment Variables menu". The "General settings" tab is selected and highlighted with a blue border. Other tabs include "Default documents", "Path mappings", and "...". Under "Stack settings", there is a dropdown for "Stack" set to "Java" and another dropdown for "Java version" set to "Java 17".

Here, you can configure some common settings for the app. Some settings require you to [scale up to higher pricing tiers](#).

- **Stack settings:** The software stack to run the app, including the language and SDK versions.

For Linux apps, you can select the language runtime version and set an optional **Startup command** or a startup command file.

The screenshot shows the "Stack settings" configuration page for a Python app. It includes fields for "Stack" (set to "Python"), "Major version" (set to "Python 3"), "Minor version" (set to "Python 3.8"), and a "Startup Command" input field which is currently empty. A note below the command field says: "Provide an optional startup command that will be run as part of container startup. [Learn more](#)".

- **Platform settings:** Lets you configure settings for the hosting platform, including:
 - **Platform bitness:** 32-bit or 64-bit. For Windows apps only.
 - **FTP state:** Allow only FTPS or disable FTP altogether.
 - **HTTP version:** Set to 2.0 to enable support for [HTTPS/2](#) protocol.

Note

Most modern browsers support HTTP/2 protocol over TLS only, while non-encrypted traffic continues to use HTTP/1.1. To ensure that client browsers connect to your app with HTTP/2, secure your custom DNS name. For more information, see [Secure a custom DNS name with a TLS/SSL binding in Azure App Service](#).

- **Web sockets:** For [ASP.NET SignalR](#) or [socket.io](#), for example.
- **Always On:** Keeps the app loaded even when there's no traffic. When **Always On** isn't turned on (default), the app is unloaded after 20 minutes without any incoming requests. The unloaded app can cause high latency for new requests because of its warm-up time. When **Always On** is turned on, the front-end load balancer sends a GET request to the application root every five minutes. The continuous ping prevents the app from being unloaded.

Always On is required for continuous WebJobs or for WebJobs that are triggered using a CRON expression.
- **Session affinity:** In a multi-instance deployment, ensure that the client is routed to the same instance for the life of the session. You can set this option to **Off** for stateless applications.
- **Session affinity proxy:** Session affinity proxy can be turned on if your app is behind a reverse proxy (like Azure Application Gateway or Azure Front Door) and you are using the default host name. The domain for the session affinity cookie will align with the forwarded host name from the reverse proxy.
- **HTTPS Only:** When enabled, all HTTP traffic is redirected to HTTPS.
- **Minimum TLS version:** Select the minimum TLS encryption version required by your app.
- **Debugging:** Enable remote debugging for [ASP.NET](#), [ASP.NET Core](#), or [Node.js](#) apps. This option turns off automatically after 48 hours.
- **Incoming client certificates:** require client certificates in [mutual authentication](#).

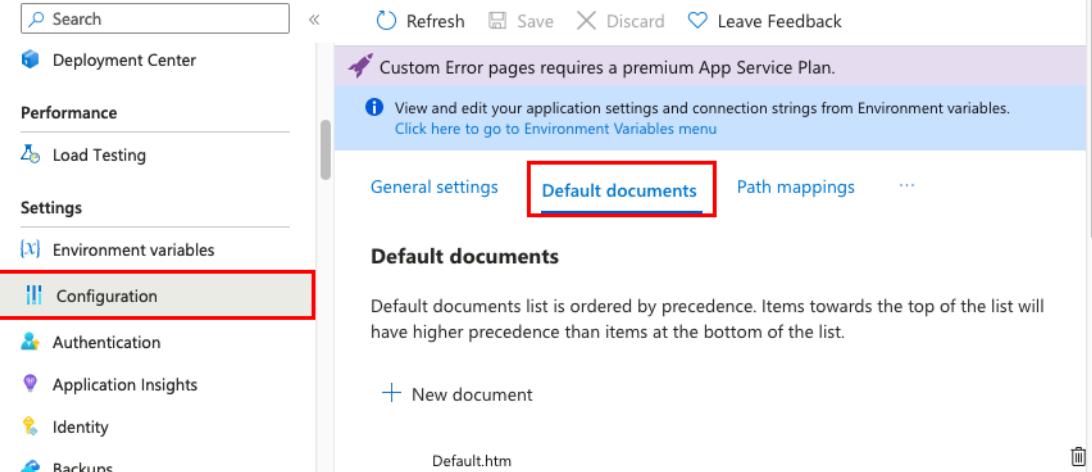
Configure default documents

This setting is only for Windows apps.

The default document is the web page that's displayed at the root URL of an App Service app. The first matching file in the list is used. If the app uses modules that route based on URL instead of serving static content, there's no need for default documents.

Azure portal

1. In the [Azure portal](#), search for and select **App Services**, and then select your app.
2. In the app's left menu, select **Configuration > Default documents**.



3. To add a default document, select **New document**. To remove a default document, select **Delete** to its right.

Map a URL path to a directory

By default, App Service starts your app from the root directory of your app code. But certain web frameworks don't start in the root directory. For example, [Laravel](#) starts in the `public` subdirectory. Such an app would be accessible at

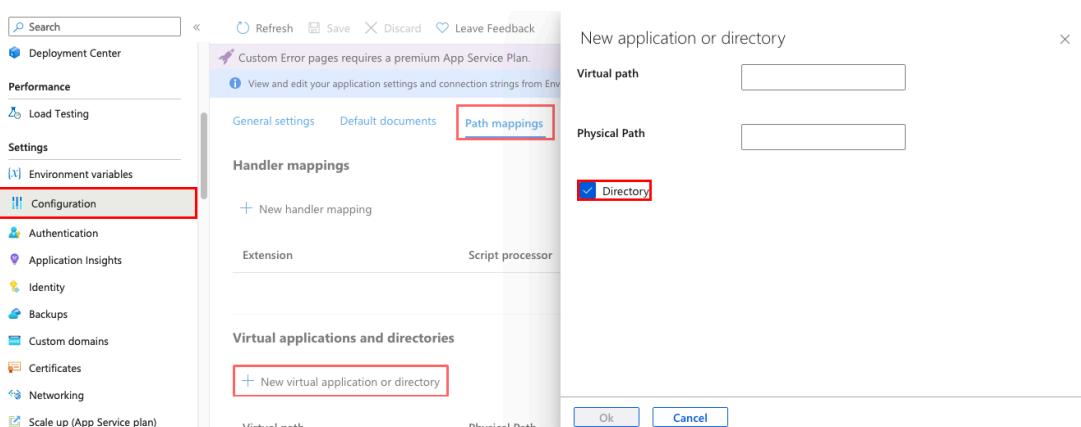
`http://contoso.com/public`, for example, but you typically want to direct `http://contoso.com` to the `public` directory instead. If your app's startup file is in a different folder, or if your repository has more than one application, you can edit or add virtual applications and directories.

Important

Virtual directory to a physical path feature is only available on Windows apps.

Azure portal

1. In the [Azure portal](#), search for and select **App Services**, and then select your app.
2. In the app's left menu, select **Configuration > Path mappings**
3. Select **New virtual application or directory**.
 - To map a virtual directory to a physical path, leave the **Directory** check box selected. Specify the virtual directory and the corresponding relative (physical) path to the website root (`D:\home`).
 - To mark a virtual directory as a web application, clear the **Directory** check box.



4. Select **OK**. Don't forget to select **Save** in the Configuration page.

Configure handler mappings

For Windows apps, you can customize the IIS handler mappings and virtual applications and directories. Handler mappings let you add custom script processors to handle requests for specific file extensions.

To add a custom handler:

1. In the [Azure portal](#), search for and select **App Services**, and then select your app.
2. In the app's left menu, select **Configuration > Path mappings**.

The screenshot shows the 'Configuration' page in the Azure App Service portal. The left sidebar lists various configuration sections: Deployment Center, Performance, Load Testing, Settings, Environment variables, Configuration (which is highlighted with a red box), Authentication, Application Insights, Identity, Backups, Custom domains, and General settings. The main content area displays a message about custom error pages requiring a premium plan, followed by tabs for General settings, Default documents, Path mappings (which is also highlighted with a red box), and Error pages (preview). Below these tabs is a section titled 'Handler mappings' with a 'New handler mapping' button. A table header for 'Handler mappings' includes columns for Extension, Script processor, and Arguments. A note at the bottom states '(no handler mappings to display)'. At the bottom of the page is a section titled 'Virtual applications and directories'.

3. Select **New handler mapping**. Configure the handler as follows:

- **Extension.** The file extension you want to handle, such as `*.php` or `handler.cgi`.
- **Script processor.** The absolute path of the script processor to you. Requests to files that match the file extension are processed by the script processor. Use the path `D:\home\site\wwwroot` to refer to your app's root directory.
- **Arguments.** Optional command-line arguments for the script processor.

4. Select **OK**. Don't forget to select **Save** in the **Configuration** page.

Configure custom containers

- Configure a custom container for Azure App Service
- Add custom storage for your containerized app

Next steps

- Environment variables and app settings reference
- Configure a custom domain name in Azure App Service
- Set up staging environments in Azure App Service
- Secure a custom DNS name with a TLS/SSL binding in Azure App Service
- Enable diagnostic logs
- Scale an app in Azure App Service
- Monitoring basics in Azure App Service
- Change applicationHost.config settings with applicationHost.xdt ↗

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

Use App Configuration references for App Service and Azure Functions (preview)

Article • 03/29/2023

This topic shows you how to work with configuration data in your App Service or Azure Functions application without requiring any code changes. [Azure App Configuration](#) is a service to centrally manage application configuration. Additionally, it's an effective audit tool for your configuration values over time or releases.

Granting your app access to App Configuration

To get started with using App Configuration references in App Service, you'll first need an App Configuration store, and provide your app permission to access the configuration key-values in the store.

1. Create an App Configuration store by following the [App Configuration quickstart](#).

 Note

App Configuration references do not yet support network-restricted configuration stores.

2. Create a [managed identity](#) for your application.

App Configuration references will use the app's system assigned identity by default, but you can [specify a user-assigned identity](#).

3. Enable the newly created identity to have the right set of access permissions on the App Configuration store. Update the [role assignments for your store](#). You'll be assigning `App Configuration Data Reader` role to this identity, scoped over the resource.

Access App Configuration Store with a user-assigned identity

Some apps might need to reference configuration at creation time, when a system-assigned identity wouldn't yet be available. In these cases, a user-assigned identity can

be created and given access to the App Configuration store, in advance. Follow these steps to [create user-assigned identity for App Configuration store](#).

Once you have granted permissions to the user-assigned identity, follow these steps:

1. [Assign the identity](#) to your application if you haven't already.
2. Configure the app to use this identity for App Configuration reference operations by setting the `keyVaultReferenceIdentity` property to the resource ID of the user-assigned identity. Though the property has `keyVault` in the name, the identity will apply to App Configuration references as well.

Azure CLI

```
userAssignedIdentityResourceId=$(az identity show -g MyResourceGroupName -n MyUserAssignedIdentityName --query id -o tsv)
appResourceId=$(az webapp show -g MyResourceGroupName -n MyAppName --query id -o tsv)
az rest --method PATCH --uri "${appResourceId}?api-version=2021-01-01"
--body "{'properties':
{'keyVaultReferenceIdentity':'${userAssignedIdentityResourceId}'}}"
```

This configuration will apply to all references from this App.

Granting your app access to referenced key vaults

In addition to storing raw configuration values, Azure App Configuration has its own format for storing [Key Vault references](#). If the value of an App Configuration reference is a Key Vault reference in App Configuration store, your app will also need to have permission to access the key vault being specified.

Note

The Azure App Configuration Key Vault references concept should not be confused with [the App Service and Azure Functions Key Vault references concept](#). Your app may use any combination of these, but there are some important differences to note. If your vault needs to be network restricted or you need the app to periodically update to latest versions, consider using the App Service and Azure Functions direct approach instead of using an App Configuration reference.

1. Identify the identity that you used for the App Configuration reference. Access to the vault must be granted to that same identity.
2. Create an [access policy in Key Vault](#) for that identity. Enable the "Get" secret permission on this policy. Do not configure the "authorized application" or `applicationId` settings, as this is not compatible with a managed identity.

Reference syntax

An App Configuration reference is of the form

`@Microsoft.AppConfiguration({referenceString})`, where `{referenceString}` is replaced by below:

Reference	Description
string parts	
<code>Endpoint=endpoint;</code>	Endpoint is the required part of the reference string. The value for Endpoint should have the url of your App Configuration resource.
<code>Key=keyName;</code>	Key forms the required part of the reference string. Value for Key should be the name of the Key that you want to assign to the App setting.
<code>Label=label</code>	The Label part is optional in reference string. Label should be the value of Label for the Key specified in Key

For example, a complete reference with `Label` would look like the following,

```
@Microsoft.AppConfiguration(Endpoint=https://myAppConfigStore.azureconfig.io;
Key=myAppConfigKey; Label=myKeysLabel)
```

Alternatively without any `Label`:

```
@Microsoft.AppConfiguration(Endpoint=https://myAppConfigStore.azureconfig.io;
Key=myAppConfigKey)
```

Any configuration change to the app that results in a site restart causes an immediate refetch of all referenced key-values from the App Configuration store.

Source Application Settings from App Config

App Configuration references can be used as values for [Application Settings](#), allowing you to keep configuration data in App Configuration instead of the site config. Application Settings and App Configuration key-values both are securely encrypted at rest. If you need centralized configuration management capabilities, then configuration data should go into App Config.

To use an App Configuration reference for an [app setting](#), set the reference as the value of the setting. Your app can reference the Configuration value through its key as usual. No code changes are required.

💡 Tip

Most application settings using App Configuration references should be marked as slot settings, as you should have separate stores or labels for each environment.

Considerations for Azure Files mounting

Apps can use the `WEBSITE_CONTENTAZUREFILECONNECTIONSTRING` application setting to mount Azure Files as the file system. This setting has additional validation checks to ensure that the app can be properly started. The platform relies on having a content share within Azure Files, and it assumes a default name unless one is specified via the `WEBSITE_CONTENTSHARE` setting. For any requests that modify these settings, the platform will attempt to validate if this content share exists, and it will attempt to create it if not. If it can't locate or create the content share, the request is blocked.

If you use App Configuration references for this setting, this validation check will fail by default, as the connection itself can't be resolved while processing the incoming request. To avoid this issue, you can skip the validation by setting

`WEBSITE_SKIP_CONTENTSHARE_VALIDATION` to "1". This setting will bypass all checks, and the content share won't be created for you. You should ensure it's created in advance.

✖️ Caution

If you skip validation and either the connection string or content share are invalid, the app will be unable to start properly and will only serve HTTP 500 errors.

As part of creating the site, it's also possible that attempted mounting of the content share could fail due to managed identity permissions not being propagated or the virtual network integration not being set up. You can defer setting up Azure Files until later in the deployment template to accommodate for the required setup. See [Azure](#)

[Resource Manager deployment](#) to learn more. App Service will use a default file system until Azure Files is set up, and files aren't copied over so make sure that no deployment attempts occur during the interim period before Azure Files is mounted.

Azure Resource Manager deployment

When automating resource deployments through Azure Resource Manager templates, you may need to sequence your dependencies in a particular order to make this feature work. Of note, you'll need to define your application settings as their own resource, rather than using a `siteConfig` property in the site definition. This is because the site needs to be defined first so that the system-assigned identity is created with it and can be used in the access policy.

Below is an example pseudo-template for a function app with App Configuration references:

JSON

```
{
    "$schema": "https://schema.management.azure.com/schemas/2019-04-01/deploymentTemplate.json#",
    "contentVersion": "1.0.0.0",
    "parameters": {
        "roleNameGuid": {
            "type": "string",
            "defaultValue": "[newGuid()]",
            "metadata": {
                "description": "A new GUID used to identify the role assignment"
            }
        }
    },
    "variables": {
        "functionAppName": "DemoMBFunc",
        "appConfigStoreName": "DemoMBAppConfig",
        "resourcesRegion": "West US2",
        "appConfigSku": "standard",
        "FontNameKey": "FontName",
        "FontColorKey": "FontColor",
        "myLabel": "Test",
        "App Configuration Data Reader": "[concat('/subscriptions/',
subscription().subscriptionId,
'/providers/Microsoft.Authorization/roleDefinitions/', '516239f1-63e1-4d78-a4de-a74fb236a071')]"
    },
    "resources": [
        {
            "type": "Microsoft.Web/sites",
            "name": "[variables('functionAppName')]"
        }
    ]
}
```

```

    "apiVersion": "2021-03-01",
    "location": "[variables('resourcesRegion')]",
    "identity": {
        "type": "SystemAssigned"
    },
    //...
    "resources": [
        {
            "type": "config",
            "name": "appsettings",
            "apiVersion": "2021-03-01",
            //...
            "dependsOn": [
                "[resourceId('Microsoft.Web/sites',
variables('functionAppName'))]",
                "[resourceId('Microsoft.AppConfiguration/configurationStores',
variables('appConfigStoreName'))]"
            ],
            "properties": {
                "WEBSITE_FONTNAME": "[concat('@Microsoft.AppConfiguration(Endpoint=',
reference(resourceId('Microsoft.AppConfiguration/configurationStores',
variables('appConfigStoreName'))).endpoint,';
Key=',variables('FontNameKey'),'; Label=',variables('myLabel'), ')')]",
                "WEBSITE_FONTCOLOR": "[concat('@Microsoft.AppConfiguration(Endpoint=',
reference(resourceId('Microsoft.AppConfiguration/configurationStores',
variables('appConfigStoreName'))).endpoint,';
Key=',variables('FontColorKey'),'; Label=',variables('myLabel'), ')')]",
                "WEBSITE_ENABLE_SYNC_UPDATE_SITE": "true"
            }
        },
        {
            "type": "sourcecontrols",
            "name": "web",
            "apiVersion": "2021-03-01",
            //...
            "dependsOn": [
                "[resourceId('Microsoft.Web/sites',
variables('functionAppName'))]",
                "[resourceId('Microsoft.Web/sites/config',
variables('functionAppName'), 'appsettings')]"
            ]
        }
    ],
    {
        "type": "Microsoft.AppConfiguration/configurationStores",
        "name": "[variables('appConfigStoreName')]",
        "apiVersion": "2019-10-01",
        "location": "[variables('resourcesRegion')]",
        "sku": {
            "name": "[variables('appConfigSku')]"
        }
    }
]
}

```

```

},
//...
"dependsOn": [
    "[resourceId('Microsoft.Web/sites',
variables('functionAppName'))]"
],
"properties": {
},
"resources": [
{
    "type": "keyValues",
    "name": "[variables('FontNameKey')]",
    "apiVersion": "2021-10-01-preview",
    //...
    "dependsOn": [
        ""
    ],
    "properties": {
        "value": "Calibri",
        "contentType": "application/json"
    }
},
{
    "type": "keyValues",
    "name": "[variables('FontColorKey')]",
    "apiVersion": "2021-10-01-preview",
    //...
    "dependsOn": [
        ""
    ],
    "properties": {
        "value": "Blue",
        "contentType": "application/json"
    }
}
]
},
{
    "scope": "
[resourceId('Microsoft.AppConfiguration/configurationStores',
variables('appConfigStoreName'))]"
},
{
    "type": "Microsoft.Authorization/roleAssignments",
    "apiVersion": "2020-04-01-preview",
    "name": "[parameters('roleNameGuid')]",
    "properties": {
        "roleDefinitionId": "[variables('App Configuration Data
Reader')]",
        "principalId": "
[reference(resourceId('Microsoft.Web/sites/',
variables('functionAppName')),"

```

```
'2020-12-01', 'Full').identity.principalId],
    "principalType": "ServicePrincipal"
}
]
}
```

Note

In this example, the source control deployment depends on the application settings. This is normally unsafe behavior, as the app setting update behaves asynchronously. However, because we have included the `WEBSITE_ENABLE_SYNC_UPDATE_SITE` application setting, the update is synchronous. This means that the source control deployment will only begin once the application settings have been fully updated. For more app settings, see [Environment variables and app settings in Azure App Service](#).

Troubleshooting App Configuration References

If a reference isn't resolved properly, the reference value will be used instead. For the application settings, an environment variable would be created whose value has the `@Microsoft.AppConfiguration(...)` syntax. It may cause an error, as the application was expecting a configuration value instead.

Most commonly, this error could be due to a misconfiguration of the [App Configuration access policy](#). However, it could also be due to a syntax error in the reference or the Configuration key-value not existing in the store.

Next steps

[Reference Key vault secrets from App Service](#)

Environment variables and app settings in Azure App Service

Article • 10/16/2024

ⓘ Note

Starting June 1, 2024, all newly created App Service apps will have the option to generate a unique default hostname using the naming convention `<app-name>-<random-hash>. <region>.azurewebsites.net`. Existing app names will remain unchanged.

Example: `myapp-ds27dh7271aah175.westus-01.azurewebsites.net`

For further details, refer to [Unique Default Hostname for App Service Resource](#).

In [Azure App Service](#), certain settings are available to the deployment or runtime environment as environment variables. Some of these settings can be customized when you set them manually as [app settings](#). This reference shows the variables you can use or customize.

App environment

The following environment variables are related to the app environment in general.

[Expand table](#)

Setting name	Description	Example
<code>WEBSITE_SITE_NAME</code>	Read-only. App name.	
<code>WEBSITE_RESOURCE_GROUP</code>	Read-only. Azure resource group name that contains the app resource.	
<code>WEBSITE_OWNER_NAME</code>	Read-only. Contains the Azure subscription ID that owns the app, the resource group, and the webspace.	
<code>REGION_NAME</code>	Read-only. Region name of the app.	
<code>WEBSITE_PLATFORM_VERSION</code>	Read-only. App Service platform version.	
<code>HOME</code>	Read-only. Path to the home directory (for example, <code>D:\home</code> for Windows).	
<code>SERVER_PORT</code>	Read-only. The port the app should listen to.	
<code>WEBSITE_WARMUP_PATH</code>	A relative path to ping to warm up the app, beginning with a slash. The default is <code>/</code> , which pings the root path. The specific path can be pinged by an unauthenticated client, such as Azure Traffic Manager, even if App Service authentication is set to reject unauthenticated clients. (NOTE: This app setting doesn't change the path used by AlwaysOn.)	
<code>WEBSITE_COMPUTE_MODE</code>	Read-only. Specifies whether app runs on dedicated (<code>Dedicated</code>) or shared (<code>Shared</code>) VM/s.	

Setting name	Description	Example
WEBSITE_SKU	Read-only. SKU of the app. Possible values are <code>Free</code> , <code>Shared</code> , <code>Basic</code> , and <code>Standard</code> .	
SITE_BITNESS	Read-only. Shows whether the app is 32-bit (<code>x86</code>) or 64-bit (<code>AMD64</code>).	
WEBSITE_HOSTNAME	Read-only. Primary hostname for the app. Custom hostnames aren't accounted for here.	
WEBSITE_VOLUME_TYPE	Read-only. Shows the storage volume type currently in use.	
WEBSITE_NPM_DEFAULT_VERSION	Default npm version the app is using.	
WEBSOCKET_CONCURRENT_REQUEST_LIMIT	Read-only. Limit for websocket's concurrent requests. For <code>Standard</code> tier and above, the value is <code>-1</code> , but there's still a per VM limit based on your VM size (see Cross VM Numerical Limits).	
WEBSITE_PRIVATE_EXTENSIONS	Set to <code>0</code> to disable the use of private site extensions.	
WEBSITE_TIME_ZONE	By default, the time zone for the app is always UTC. You can change it to any of the valid values that are listed in Default Time Zones . If the specified value isn't recognized, UTC is used.	Atlantic Standard Time
WEBSITE_ADD_SITENAME_BINDINGS_IN_APPHOST_CONFIG	After slot swaps, the app may experience unexpected restarts. This is because after a swap, the hostname binding configuration goes out of sync, which by itself doesn't cause restarts. However, certain underlying storage events (such as storage volume failovers) may detect these discrepancies and force all worker processes to restart. To minimize these types of restarts, set the app setting value to <code>1</code> on all slots (default is <code>0</code>). However, don't set this value if you're running a Windows Communication Foundation (WCF) application. For more information, see Troubleshoot swaps	
WEBSITE_PROACTIVE_AUTOHEAL_ENABLED	By default, a VM instance is proactively "autohealed" when it's using more than 90% of allocated memory for more than 30 seconds, or when 80% of the total requests in the last two minutes take longer than 200 seconds. If a VM instance has triggered one of these rules, the recovery process is an overlapping restart of the instance. Set to <code>false</code> to disable this recovery behavior. The default is <code>true</code> . For more information, see Proactive Auto Heal .	
WEBSITE_PROACTIVE_CRASHMONITORING_ENABLED	Whenever the w3wp.exe process on a VM instance of your app crashes due to an unhandled exception for more than three times in 24 hours, a debugger process is attached to the main worker process on that instance, and collects a memory dump when the worker process crashes again. This memory dump is then analyzed and the call stack of the thread that caused the	

Setting name	Description	Example
	crash is logged in your App Service's logs. Set to <code>false</code> to disable this automatic monitoring behavior. The default is <code>true</code> . For more information, see Proactive Crash Monitoring .	
<code>WEBSITE_DAAS_STORAGE_SASURI</code>	During crash monitoring (proactive or manual), the memory dumps are deleted by default. To save the memory dumps to a storage blob container, specify the SAS URI.	
<code>WEBSITE_CRASHMONITORING_ENABLED</code>	Set to <code>true</code> to enable crash monitoring manually. You must also set <code>WEBSITE_DAAS_STORAGE_SASURI</code> and <code>WEBSITE_CRASHMONITORING_SETTINGS</code> . The default is <code>false</code> . This setting has no effect if remote debugging is enabled. Also, if this setting is set to <code>true</code> , proactive crash monitoring is disabled.	
<code>WEBSITE_CRASHMONITORING_SETTINGS</code>	A JSON with the following format: <pre>{"StartTimeUtc": "2020-02-10T08:21", "MaxHours": "<elapsed-hours-from-StartTimeUtc>", "MaxDumpCount": "<max-number-of-crash-dumps>"}.</pre> Required to configure crash monitoring if <code>WEBSITE_CRASHMONITORING_ENABLED</code> is specified. To only log the call stack without saving the crash dump in the storage account, add <code>,"UseStorageAccount": "false"</code> in the JSON.	
<code>REMOTEDEBUGGINGVERSION</code>	Remote debugging version.	
<code>WEBSITE_CONTENTAZUREFILECONNECTIONSTRING</code>	By default, App Service creates a shared storage for you at app creation. To use a custom storage account instead, set to the connection string of your storage account. For functions, see App settings reference for Functions .	<code>DefaultEndpointsProtocol=https;AccountName=<name>;AccountKey=<key></code>
<code>WEBSITE_CONTENTSHARE</code>	When you use specify a custom storage account with <code>WEBSITE_CONTENTAZUREFILECONNECTIONSTRING</code> , App Service creates a file share in that storage account for your app. To use a custom name, set this variable to the name you want. If a file share with the specified name doesn't exist, App Service creates it for you.	<code>myapp123</code>
<code>WEBSITE_SCM_ALWAYS_ON_ENABLED</code>	Read-only. Shows whether Always On is enabled (1) or not (0).	
<code>WEBSITE_SCM_SEPARATE_STATUS</code>	Read-only. Shows whether the Kudu app is running in a separate process (1) or not (0).	
<code>WEBSITE_DNS_ATTEMPTS</code>	Number of times to try name resolve.	
<code>WEBSITE_DNS_TIMEOUT</code>	Number of seconds to wait for name resolve	

Variable prefixes

The following table shows environment variable prefixes that App Service uses for various purposes.

[Expand table](#)

Setting name	Description
APPSETTING_	Signifies that a variable is set by the customer as an app setting in the app configuration. It's injected into a .NET app as an app setting.
MAINSITE_	Signifies a variable is specific to the app itself.
SCMSITE_	Signifies a variable is specific to the Kudu app.
SQLCONNSTR_	Signifies a SQL Server connection string in the app configuration. It's injected into a .NET app as a connection string.
SQLAZURECONNSTR_	Signifies an Azure SQL Database connection string in the app configuration. It's injected into a .NET app as a connection string.
POSTGRESQLCONNSTR_	Signifies a PostgreSQL connection string in the app configuration. It's injected into a .NET app as a connection string.
CUSTOMCONNSTR_	Signifies a custom connection string in the app configuration. It's injected into a .NET app as a connection string.
MYSQLCONNSTR_	Signifies a MySQL Database connection string in the app configuration. It's injected into a .NET app as a connection string.
AZUREFILESTORAGE_	A connection string to a custom share for a custom container in Azure Files.
AZUREBLOBSTORAGE_	A connection string to a custom storage account for a custom container in Azure Blob Storage.
NOTIFICATIONHUBCONNSTR_	Signifies a connection string to a notification hub in Azure Notification Hubs.
SERVICEBUSCONNSTR_	Signifies a connection string to an instance of Azure Service Bus.
EVENTHUBCONNSTR_	Signifies a connection string to an event hub in Azure Event Hubs.
DOCDBCONNSTR_	Signifies a connection string to a database in Azure Cosmos DB.
REDISCACHECONNSTR_	Signifies a connection string to a cache in Azure Cache for Redis.
FILESHARESTORAGE_	Signifies a connection string to a custom file share.

Deployment

The following environment variables are related to app deployment. For variables related to App Service build automation, see [Build automation](#).

[Expand table](#)

Setting name	Description
DEPLOYMENT_BRANCH	For local Git or cloud Git deployment (such as GitHub), set to the branch in Azure you want to deploy to. By default, it's <code>master</code> .
WEBSITE_RUN_FROM_PACKAGE	Set to <code>1</code> to run the app from a local ZIP package, or set to the URL of an external URL to run the app from a remote ZIP package. For more information, see Run your app in Azure App Service directly from a ZIP package .
WEBSITE_USE_ZIP	Deprecated. Use <code>WEBSITE_RUN_FROM_PACKAGE</code> .
WEBSITE_RUN_FROM_ZIP	Deprecated. Use <code>WEBSITE_RUN_FROM_PACKAGE</code> .

Setting name	Description
SCM_MAX_ZIP_PACKAGE_COUNT	Your app keeps 5 of the most recent zip files deployed using zip deploy . You can keep more or less by setting the app setting to a different number.
WEBSITE_WEBDEPLOY_USE_SCM	Set to <code>false</code> for WebDeploy to stop using the Kudu deployment engine. The default is <code>true</code> . To deploy to Linux apps using Visual Studio (WebDeploy/MSDeploy), set it to <code>false</code> .
MSDEPLOY_RENAME_LOCKED_FILES	Set to <code>1</code> to attempt to rename DLLs if they can't be copied during a WebDeploy deployment. This setting isn't applicable if <code>WEBSITE_WEBDEPLOY_USE_SCM</code> is set to <code>false</code> .
WEBSITE_DISABLE_SCM_SEPARATION	By default, the main app and the Kudu app run in different sandboxes. When you stop the app, the Kudu app is still running, and you can continue to use Git deploy and MSDeploy. Each app has its own local files. Turning off this separation (setting to <code>true</code>) is a legacy mode that's no longer fully supported.
WEBSITE_ENABLE_SYNC_UPDATE_SITE	Set to <code>1</code> ensure that REST API calls to update <code>site</code> and <code>siteconfig</code> are completely applied to all instances before returning. The default is <code>1</code> if deploying with an ARM template, to avoid race conditions with subsequent ARM calls.
WEBSITE_START_SCM_ON_SITE_CREATION	In an ARM template deployment, set to <code>1</code> in the ARM template to pre-start the Kudu app as part of app creation.
WEBSITE_START_SCM_WITH_PRELOAD	For Linux apps, set to <code>true</code> to force preloading the Kudu app when Always On is enabled by pinging its URL. The default is <code>false</code> . For Windows apps, the Kudu app is always preloaded.

Build automation

Kudu (Windows)

Kudu build configuration applies to native Windows apps and is used to control the behavior of Git-based (or ZIP-based) deployments.

[Expand table](#)

Setting name	Description	Example
SCM_BUILD_ARGS	Add things at the end of the msbuild command line, such that it overrides any previous parts of the default command line.	To do a clean build: <code>-t:Clean;Compile</code>
SCM_SCRIPT_GENERATOR_ARGS	Kudu uses the <code>azure site deploymentscript</code> command described here to generate a deployment script. It automatically detects the language framework type and determines the parameters to pass to the command. This setting overrides the automatically generated parameters.	To treat your repository as plain content files: <code>--basic -p <folder-to-deploy></code>
SCM_TRACE_LEVEL	Build trace level. The default is <code>1</code> . Set to higher values, up to <code>4</code> , for more tracing.	<code>4</code>
SCM_COMMAND_IDLE_TIMEOUT	Time out in seconds for each command that the build process launches to wait before without producing any output. After that, the command is considered idle and killed. The default is <code>60</code> (one minute). In Azure, there's also a general idle request timeout that disconnects clients after 230 seconds. However, the command will still continue running server-side after that.	
SCM_LOGSTREAM_TIMEOUT	Time-out of inactivity in seconds before stopping log streaming. The default is <code>1800</code> (30 minutes).	
SCM_SITEEXTENSIONS_FEED_URL	URL of the site extensions gallery. The default is https://www.nuget.org/api/v2/ . The URL of the old feed is http://www.siteextensions.net/api/v2/ .	
SCM_USE_LIBGIT2SHARP_REPOSITORY	Set to <code>0</code> to use git.exe instead of libgit2sharp for git operations.	

Setting name	Description	Example
WEBSITE_LOAD_USER_PROFILE	In case of the error <code>The specified user does not have a valid profile.</code> during ASP.NET build automation (such as during Git deployment), set this variable to <code>1</code> to load a full user profile in the build environment. This setting is only applicable when <code>WEBSITE_COMPUTE_MODE</code> is <code>Dedicated</code> .	
WEBSITE_SCM_IDLE_TIMEOUT_IN_MINUTES	Time out in minutes for the SCM (Kudu) site. The default is <code>20</code> .	
SCM_DO_BUILD_DURING_DEPLOYMENT	With ZIP deploy , the deployment engine assumes that a ZIP file is ready to run as-is and doesn't run any build automation. To enable the same build automation as in Git deploy , set to <code>true</code> .	

Language-specific settings

This section shows the configurable runtime settings for each supported language framework. Additional settings are available during [build automation](#) at deployment time.

.NET

[\[+\] Expand table](#)

Setting name	Description
PORT	Read-only. For Linux apps, port that the .NET runtime listens to in the container.
WEBSITE_ROLE_INSTANCE_ID	Read-only. ID of the current instance.
HOME	Read-only. Directory that points to shared storage (<code>/home</code>).
DUMP_DIR	Read-only. Directory for the crash dumps (<code>/home/logs/dumps</code>).
APP_SVC_RUN_FROM_COPY	Linux apps only. By default, the app is run from <code>/home/site/wwwroot</code> , a shared directory for all scaled-out instances. Set this variable to <code>true</code> to copy the app to a local directory in your container and run it from there. When using this option, be sure not to hard-code any reference to <code>/home/site/wwwroot</code> . Instead, use a path relative to <code>/home/site/wwwroot</code> .
MACHINEKEY_Decryption	For Windows native apps or Windows containerized apps, this variable is injected into app environment or container to enable ASP.NET cryptographic routines (see machineKey Element). To override the default <code>decryption</code> value, configure it as an App Service app setting, or set it directly in the <code>machineKey</code> element of the <code>Web.config</code> file.
MACHINEKEY_DecryptionKey	For Windows native apps or Windows containerized apps, this variable is injected into the app environment or container to enable ASP.NET cryptographic routines (see machineKey Element). To override the automatically generated <code>decryptionKey</code> value, configure it as an App Service app setting, or set it directly in the <code>machineKey</code> element of the <code>Web.config</code> file.
MACHINEKEY_Validation	For Windows native apps or Windows containerized apps, this variable is injected into the app environment or container to enable ASP.NET cryptographic routines (see machineKey Element). To override the default <code>validation</code> value, configure it as an App Service app setting, or set it directly in the <code>machineKey</code> element of the <code>Web.config</code> file.
MACHINEKEY_ValidationKey	For Windows native apps or Windows containerized apps, this variable is injected into the app environment or container to enable ASP.NET cryptographic routines (see machineKey Element). To override the automatically generated <code>validationKey</code> value, configure it as an App Service app setting, or set it directly in the <code>machineKey</code> element of the <code>Web.config</code> file.

WordPress

[Expand table](#)

Application Setting	Scope	Value	Max	Description
<code>WEBSITES_ENABLE_APP_SERVICE_STORAGE</code>	Web App	true	-	When set to TRUE, file contents are preserved during restarts.
<code>WP_MEMORY_LIMIT</code>	WordPress	128M	512M	Frontend or general wordpress PHP memory limit (per script). Can't be more than <code>PHP_MEMORY_LIMIT</code>
<code>WP_MAX_MEMORY_LIMIT</code>	WordPress	256M	512M	Admin dashboard PHP memory limit (per script). Generally Admin dashboard/ backend scripts takes lot of memory compared to frontend scripts. Can't be more than <code>PHP_MEMORY_LIMIT</code> .
<code>PHP_MEMORY_LIMIT</code>	PHP	512M	512M	Memory limits for general PHP script. It can only be decreased.
<code>FILE_UPLOADS</code>	PHP	On	-	Can be either On or Off. Note that values are case sensitive. Enables or disables file uploads.
<code>UPLOAD_MAX_FILESIZE</code>	PHP	50M	256M	Max file upload size limit. Can be increased up to 256M.
<code>POST_MAX_SIZE</code>	PHP	128M	256M	Can be increased up to 256M. Generally should be more than <code>UPLOAD_MAX_FILESIZE</code> .
<code>MAX_EXECUTION_TIME</code>	PHP	120	120	Can only be decreased. Please break down the scripts if it is taking more than 120 seconds. Added to avoid bad scripts from slowing the system.
<code>MAX_INPUT_TIME</code>	PHP	120	120	Max time limit for parsing the input requests. Can only be decreased.
<code>MAX_INPUT_VARS</code>	PHP	10000	10000	-
<code>DATABASE_HOST</code>	Database	-	-	Database host used to connect to WordPress.
<code>DATABASE_NAME</code>	Database	-	-	Database name used to connect to WordPress.
<code>DATABASE_USERNAME</code>	Database	-	-	Database username used to connect to WordPress.
<code>DATABASE_PASSWORD</code>	Database	-	-	Database password used to connect to the MySQL database. To change the MySQL database password, see update admin password . Whenever the MySQL database password is changed, the Application Settings also need to be updated.
<code>WORDPRESS_ADMIN_EMAIL</code>	Deployment only	-	-	WordPress admin email.
<code>WORDPRESS_ADMIN_PASSWORD</code>	Deployment only	-	-	WordPress admin password. This is only for deployment purposes. Modifying this value has no effect on the WordPress installation. To change the WordPress admin password, see resetting your password .
<code>WORDPRESS_ADMIN_USER</code>	Deployment only	-	-	WordPress admin username
<code>WORDPRESS_ADMIN_LOCALE_CODE</code>	Deployment only	-	-	Database username used to connect to WordPress.

Domain and DNS

[Expand table](#)

Setting name	Description	Example
<code>WEBSITE_DNS_SERVER</code>	IP address of primary DNS server for outgoing connections (such as to a back-end service). The default DNS server for App Service is Azure DNS, whose IP address is <code>168.63.129.16</code> . If your app uses VNet integration or is in an App Service environment , it inherits the DNS server configuration from the VNet by default.	<code>10.0.0.1</code>
<code>WEBSITE_DNS_ALT_SERVER</code>	IP address of fallback DNS server for outgoing connections. See <code>WEBSITE_DNS_SERVER</code> .	
<code>WEBSITE_ENABLE_DNS_CACHE</code>	Allows successful DNS resolutions to be cached. By Default expired DNS cache entries will be flushed & in addition to the existing cache to be flushed every 4.5 mins.	

TLS/SSL

For more information, see [Use a TLS/SSL certificate in your code in Azure App Service](#).

[Expand table](#)

Setting name	Description
<code>WEBSITE_LOAD_CERTIFICATES</code>	Comma-separate thumbprint values to the certificate you want to load in your code, or <code>*</code> to allow all certificates to be loaded in code. Only certificates added to your app can be loaded.
<code>WEBSITE_PRIVATE_CERTS_PATH</code>	Read-only. Path in a Windows container to the loaded private certificates.
<code>WEBSITE_PUBLIC_CERTS_PATH</code>	Read-only. Path in a Windows container to the loaded public certificates.
<code>WEBSITE_INTERMEDIATE_CERTS_PATH</code>	Read-only. Path in a Windows container to the loaded intermediate certificates.
<code>WEBSITE_ROOT_CERTS_PATH</code>	Read-only. Path in a Windows container to the loaded root certificates.

Deployment slots

For more information on deployment slots, see [Set up staging environments in Azure App Service](#).

[Expand table](#)

Setting name	Description	Example
<code>WEBSITE_OVERRIDE_STICKY_EXTENSION_VERSIONS</code>	By default, the versions for site extensions are specific to each slot. This prevents unanticipated application behavior due to changing extension versions after a swap. If you want the extension versions to swap as well, set to <code>0</code> on <i>all slots</i> .	
<code>WEBSITE_OVERRIDE_PRESERVE_DEFAULT_STICKY_SLOT_SETTINGS</code>	Designates certain settings as sticky or not swappable by default . Default is <code>true</code> . Set this setting to <code>false</code> or <code>0</code> for <i>all deployment slots</i> to make them swappable instead. There's no fine-grain control for specific setting types.	
<code>WEBSITE_SWAP_WARMUP_PING_PATH</code>	Path to ping to warm up the target slot in a swap, beginning with a slash. The default is <code>/</code> , which pings the root path over HTTP.	<code>/statuscheck</code>
<code>WEBSITE_SWAP_WARMUP_PING_STATUSES</code>	Valid HTTP response codes for the warm-up operation during a swap. If the returned status code isn't in the list, the warmup and swap operations are stopped. By default, all response codes are valid.	<code>200,202</code>
<code>WEBSITE_SLOT_NUMBER_OF_TIMEOUTS_BEFORE_RESTART</code>	During a slot swap, maximum number of timeouts after which we force restart the site on a specific VM instance. The default is <code>3</code> .	

Setting name	Description	Example
<code>WEBSITE_SLOT_MAX_NUMBER_OF_TIMEOUTS</code>	During a slot swap, maximum number of timeout requests for a single URL to make before giving up. The default is <code>5</code> .	
<code>WEBSITE_SKIP_ALL_BINDINGS_IN_APPHOST_CONFIG</code>	Set to <code>true</code> or <code>1</code> to skip all bindings in <code>applicationHost.config</code> . The default is <code>false</code> . If your app triggers a restart because <code>applicationHost.config</code> is updated with the swapped hostnames of the slots, set this variable to <code>true</code> to avoid a restart of this kind. If you're running a Windows Communication Foundation (WCF) app, don't set this variable.	

Custom containers

For more information on custom containers, see [Run a custom container in Azure](#).

[Expand table](#)

Setting name	Description	Example
<code>WEBSITES_ENABLE_APP_SERVICE_STORAGE</code>	For Linux custom containers: set to <code>true</code> to enable the <code>/home</code> directory to be shared across scaled instances. The default is <code>false</code> for Linux custom containers. For Windows containers: set to <code>true</code> to enable the <code>c:\home</code> directory to be shared across scaled instances. The default is <code>true</code> for Windows containers.	
<code>WEBSITES_CONTAINER_START_TIME_LIMIT</code>	Amount of time in seconds to wait for the container to complete start-up before restarting the container. Default is <code>230</code> . You can increase it up to the maximum of <code>1800</code> .	
<code>WEBSITES_CONTAINER_STOP_TIME_LIMIT</code>	Amount of time in seconds to wait for the container to terminate gracefully. Default is <code>5</code> . You can increase to a maximum of <code>120</code> .	
<code>DOCKER_REGISTRY_SERVER_URL</code>	URL of the registry server, when running a custom container in App Service. For security, this variable isn't passed on to the container.	<code>https://<server-name>.azuredcr.io</code>
<code>DOCKER_REGISTRY_SERVER_USERNAME</code>	Username to authenticate with the registry server at <code>DOCKER_REGISTRY_SERVER_URL</code> . For security, this variable isn't passed on to the container.	
<code>DOCKER_REGISTRY_SERVER_PASSWORD</code>	Password to authenticate with the registry server at <code>DOCKER_REGISTRY_SERVER_URL</code> . For security, this variable isn't passed on to the container.	
<code>DOCKER_ENABLE_CI</code>	Set to <code>true</code> to enable the continuous deployment for custom containers. The default is <code>false</code> for custom containers.	
<code>WEBSITE_PULL_IMAGE_OVER_VNET</code>	Connect and pull from a registry inside a Virtual Network or on-premises. Your app will need to be connected to a Virtual Network using VNet integration feature. This setting is also needed for Azure Container Registry with Private Endpoint.	
<code>WEBSITES_WEB_CONTAINER_NAME</code>	In a Docker Compose app, only one of the containers can be internet accessible. Set to the name of the container defined in the configuration file to override the default container selection. By default, the internet accessible container is the first container to define port 80 or 8080, or, when no such container is found, the first container defined in the configuration file.	
<code>WEBSITES_PORT</code>	For a custom container, the custom port number on the container for App Service to route requests to. By default, App Service attempts automatic	

Setting name	Description	Example
	port detection of ports 80 and 8080. This setting isn't injected into the container as an environment variable.	
WEBSITE_CPU_CORES_LIMIT	By default, a Windows container runs with all available cores for your chosen pricing tier. To reduce the number of cores, set to the number of desired cores limit. For more information, see Customize the number of compute cores .	
WEBSITE_MEMORY_LIMIT_MB	By default all Windows Containers deployed in Azure App Service have a memory limit configured depending on the App Service Plan SKU. Set to the desired memory limit in MB. The cumulative total of this setting across apps in the same plan must not exceed the amount allowed by the chosen pricing tier. For more information, see Customize container memory .	

Scaling

[Expand table](#)

Setting name	Description
WEBSITE_INSTANCE_ID	Read-only. Unique ID of the current VM instance, when the app is scaled out to multiple instances.
WEBSITE_IIS_SITE_NAME	Deprecated. Use <code>WEBSITE_INSTANCE_ID</code> .
WEBSITE_DISABLE_OVERLAPPED_RECYCLING	Overlapped recycling makes it so that before the current VM instance of an app is shut down, a new VM instance starts. In some cases, it can cause file locking issues. You can try turning it off by setting to <code>1</code> .
WEBSITE_DISABLE_CROSS_STAMP_SCALE	By default, apps are allowed to scale across stamps if they use Azure Files or a Docker container. Set to <code>1</code> or <code>true</code> to disable cross-stamp scaling within the app's region. The default is <code>0</code> . Custom Docker containers that set <code>WEBSITES_ENABLE_APP_SERVICE_STORAGE</code> to <code>true</code> or <code>1</code> can't scale cross-stamps because their content isn't completely encapsulated in the Docker container.

Logging

[Expand table](#)

Setting name	Description	Example
WEBSITE_HTTPLOGGING_ENABLED	Read-only. Shows whether the web server logging for Windows native apps is enabled (<code>1</code>) or not (<code>0</code>).	
WEBSITE_HTTPLOGGING_RETENTION_DAYS	Retention period in days of web server logs, if web server logs are enabled for a Windows native or Linux app.	10
WEBSITE_HTTPLOGGING_CONTAINER_URL	SAS URL of the blob storage container to store web server logs for Windows native apps, if web server logs are enabled. If not set, web server logs are stored in the app's file system (default shared storage).	
DIAGNOSTICS_AZUREBLOBRETENTIONINDAYS	Retention period in days of application logs for Windows native apps, if application logs are enabled.	10
DIAGNOSTICS_AZUREBLOBCONTAINERSASURL	SAS URL of the blob storage container to store application logs for Windows native apps, if application logs are enabled.	
APPSERVICEAPPLOGS_TRACE_LEVEL	Minimum log level to ship to Log Analytics for the AppServiceAppLogs log type.	

Setting name	Description	Example
DIAGNOSTICS_LASTRESORTFILE	The filename to create, or a relative path to the log directory, for logging internal errors for troubleshooting the listener. The default is <code>logging-errors.txt</code> .	
DIAGNOSTICS_LOGGINGSETTINGSFILE	Path to the log settings file, relative to <code>D:\home</code> or <code>/home</code> . The default is <code>site\diagnostics\settings.json</code> .	
DIAGNOSTICS_TEXTTRACELOGDIRECTORY	The log folder, relative to the app root (<code>D:\home\site\wwwroot</code> or <code>/home/site/wwwroot</code>).	<code>..\..\LogFiles\Application</code>
DIAGNOSTICS_TEXTTRACEMAXLOGFILESIZEBYTES	Maximum size of the log file in bytes. The default is <code>131072</code> (128 KB).	
DIAGNOSTICS_TEXTTRACEMAXLOGFOLDERSIZEBYTES	Maximum size of the log folder in bytes. The default is <code>1048576</code> (1 MB).	
DIAGNOSTICS_TEXTTRACEMAXNUMLOGFILES	Maximum number of log files to keep. The default is <code>20</code> .	
DIAGNOSTICS_TEXTTRACETURNOFFPERIOD	Time out in milliseconds to keep application logging enabled. The default is <code>43200000</code> (12 hours).	
WEBSITE_LOG_BUFFERING	By default, log buffering is enabled. Set to <code>0</code> to disable it.	
WEBSITE_ENABLE_PERF_MODE	For native Windows apps, set to <code>TRUE</code> to turn off IIS log entries for successful requests returned within 10 seconds. This is a quick way to do performance benchmarking by removing extended logging.	

Performance counters

The following are 'fake' environment variables that don't exist if you enumerate them, but return their value if you look them up individually. The value is dynamic and can change on every lookup.

[Expand table](#)

Setting name	Description
WEBSITE_COUNTERS_ASPNET	A JSON object containing the ASP.NET perf counters.
WEBSITE_COUNTERS_APP	A JSON object containing sandbox counters.
WEBSITE_COUNTERS_CLR	A JSON object containing CLR counters.
WEBSITE_COUNTERS_ALL	A JSON object containing the combination of the other three variables.

Caching

[Expand table](#)

Setting name	Description
WEBSITE_LOCAL_CACHE_OPTION	Whether local cache is enabled. Available options are: <ul style="list-style-type: none"> - <code>Default</code>: Inherit the stamp-level global setting. - <code>Always</code>: Enable for the app. - <code>OnStorageUnavailability</code> - <code>Disabled</code>: Disabled for the app.
WEBSITE_LOCAL_CACHE_READWRITE_OPTION	Read-write options of the local cache. Available options are: <ul style="list-style-type: none"> - <code>ReadOnly</code>: Cache is read-only. - <code>WriteButDiscardChanges</code>: Allow writes to local cache but discard changes made locally.

Setting name	Description
WEBSITE_LOCAL_CACHE_SIZEINMB	Size of the local cache in MB. Default is 1000 (1 GB).
WEBSITE_LOCALCACHE_READY	Read-only flag indicating if the app using local cache.
WEBSITE_DYNAMIC_CACHE	Due to network file shared nature to allow access for multiple instances, the dynamic cache improves performance by caching the recently accessed files locally on an instance. Cache is invalidated when file is modified. The cache location is %SYSTEMDRIVE%\local\DynamicCache (same %SYSTEMDRIVE%\local quota is applied). To enable full content caching, set to 1, which includes both file content and directory/file metadata (timestamps, size, directory content). To conserve local disk use, set to 2 to cache only directory/file metadata (timestamps, size, directory content). To turn off caching, set to 0. For Windows apps and for Linux apps created with the WordPress template , the default is 1. For all other Linux apps, the default is 0.
WEBSITE_READONLY_APP	When using dynamic cache, you can disable write access to the app root (D:\home\site\wwwroot or /home/site/wwwroot) by setting this variable to 1. Except for the App_Data directory, no exclusive locks are allowed, so that deployments don't get blocked by locked files.

Networking

The following environment variables are related to [hybrid connections](#) and [VNET integration](#).

[Expand table](#)

Setting name	Description
WEBSITE_RELAYS	Read-only. Data needed to configure the Hybrid Connection, including endpoints and service bus data.
WEBSITE_REWRITE_TABLE	Read-only. Used at runtime to do the lookups and rewrite connections appropriately.
WEBSITE_VNET_ROUTE_ALL	By default, if you use regional VNet Integration , your app only routes RFC1918 traffic into your VNet. Set to 1 to route all outbound traffic into your VNet and be subject to the same NSGs and UDRs. The setting lets you access non-RFC1918 endpoints through your VNet, secure all outbound traffic leaving your app, and force tunnel all outbound traffic to a network appliance of your own choosing.
WEBSITE_PRIVATE_IP	Read-only. IP address associated with the app when integrated with a VNet . For Regional VNet Integration, the value is an IP from the address range of the delegated subnet, and for Gateway-required VNet Integration, the value is an IP from the address range of the point-to-site address pool configured on the Virtual Network Gateway. This IP is used by the app to connect to the resources through the VNet. Also, it can change within the described address range.
WEBSITE_PRIVATE_PORTS	Read-only. In VNet integration, shows which ports are useable by the app to communicate with other nodes.
WEBSITE_CONTENTOVERVNET	If you are mounting an Azure File Share on the App Service and the Storage account is restricted to a VNET, ensure to enable this setting with a value of 1.

Key vault references

The following environment variables are related to [key vault references](#).

[Expand table](#)

Setting name	Description
WEBSITE_KEYVAULT_REFERENCES	Read-only. Contains information (including statuses) for all Key Vault references that are currently configured in the app.
WEBSITE_SKIP_CONTENTSHARE_VALIDATION	If you set the shared storage connection of your app (using WEBSITE_CONTENTAZUREFILECONNECTIONSTRING) to a Key Vault reference, the app can't resolve the key vault reference at app creation or update if one of the following conditions is true: - The app accesses the key vault with a system-assigned identity. - The app accesses the key vault with a user-assigned identity, and the key vault is locked with a

Setting name	Description
<code>VNet</code>	To avoid errors at create or update time, set this variable to <code>1</code> .
<code>WEBSITE_DELAY_CERT_DELETION</code>	This env var can be set to <code>1</code> by users in order to ensure that a certificate that a worker process is dependent upon isn't deleted until it exits.

CORS

The following environment variables are related to Cross-Origin Resource Sharing (CORS) configuration.

[Expand table](#)

Setting name	Description
<code>WEBSITE_CORS_ALLOWED_ORIGINS</code>	Read-only. Shows the allowed origins for CORS.
<code>WEBSITE_CORS_SUPPORT_CREDENTIALS</code>	Read-only. Shows whether setting the <code>Access-Control-Allow-Credentials</code> header to <code>true</code> is enabled (<code>True</code>) or not (<code>False</code>).

Authentication & Authorization

The following environment variables are related to [App Service authentication](#).

[Expand table](#)

Setting name	Description
<code>WEBSITE_AUTH_DISABLE_IDENTITY_FLOW</code>	When set to <code>true</code> , disables assigning the thread principal identity in ASP.NET-based web applications (including v1 Function Apps). This is designed to allow developers to protect access to their site with auth, but still have it use a separate sign-in mechanism within their app logic. The default is <code>false</code> .
<code>WEBSITE_AUTH_HIDE_DEPRECATED_SID</code>	<code>true</code> or <code>false</code> . The default value is <code>false</code> . This is a setting for the legacy Azure Mobile Apps integration for Azure App Service. Setting this to <code>true</code> resolves an issue where the SID (security ID) generated for authenticated users might change if the user changes their profile information. Changing this value may result in existing Azure Mobile Apps user IDs changing. Most apps don't need to use this setting.
<code>WEBSITE_AUTH_NONCE_DURATION</code>	A <code>timespan</code> value in the form <code>_hours_:_minutes_:_seconds_</code> . The default value is <code>00:05:00</code> , or 5 minutes. This setting controls the lifetime of the cryptographic nonce generated for all browser-driven logins. If a sign-in fails to complete in the specified time, the sign-in flow will be retried automatically. This application setting is intended for use with the V1 (classic) configuration experience. If using the V2 authentication configuration schema, you should instead use the <code>login.nonce.nonceExpirationInterval</code> configuration value.
<code>WEBSITE_AUTH_PRESERVE_URL_FRAGMENT</code>	When set to <code>true</code> and users select on app links that contain URL fragments, the sign-in process will ensure that the URL fragment part of your URL doesn't get lost in the sign-in redirect process. For more information, see Customize sign-in and sign-out in Azure App Service authentication .
<code>WEBSITE_AUTH_USE_LEGACY CLAIMS</code>	To maintain backward compatibility across upgrades, the authentication module uses the legacy claims mapping of short to long names in the <code>/auth/me</code> API, so certain mappings are excluded (e.g. "roles"). To get the more modern version of the claims mappings, set this variable to <code>False</code> . In the "roles" example, it would be mapped to the long claim name " <code>http://schemas.microsoft.com/ws/2008/06/identity/claims/role</code> ".
<code>WEBSITE_AUTH_DISABLE_WWWAUTHENTICATE</code>	<code>true</code> or <code>false</code> . The default value is <code>false</code> . When set to <code>true</code> , removes the WWW-Authenticate HTTP response header from module-generated HTTP 401 responses. This application setting is intended for use with the V1 (classic) configuration experience. If using the V2 authentication configuration schema, you should instead use the <code>identityProviders.azureActiveDirectory.login.disableWwwAuthenticate</code> configuration value.

Setting name	Description
<code>WEBSITE_AUTH_STATE_DIRECTORY</code>	A local file system directory path where tokens are stored when the file-based token store is enabled. The default value is <code>%HOME%\Data\auth</code> . This application setting is intended for use with the V1 (classic) configuration experience. If using the V2 authentication configuration schema, you should instead use the <code>login.tokenStore.FileSystem.directory</code> configuration value.
<code>WEBSITE_AUTH_TOKEN_CONTAINER_SASURL</code>	A fully qualified blob container URL. Instructs the auth module to store and load all encrypted tokens to the specified blob storage container instead of using the default local file system.
<code>WEBSITE_AUTH_TOKEN_REFRESH_HOURS</code>	Any positive decimal number. The default value is <code>72</code> (hours). This setting controls the amount of time after a session token expires that the <code>/auth/refresh</code> API can be used to refresh it. Refresh attempts after this period will fail and end users will be required to sign-in again. This application setting is intended for use with the V1 (classic) configuration experience. If using the V2 authentication configuration schema, you should instead use the <code>login.tokenStore.tokenRefreshExtensionHours</code> configuration value.
<code>WEBSITE_AUTH_TRACE_LEVEL</code>	Controls the verbosity of authentication traces written to Application Logging . Valid values are <code>Off</code> , <code>Error</code> , <code>Warning</code> , <code>Information</code> , and <code>Verbose</code> . The default value is <code>Verbose</code> .
<code>WEBSITE_AUTH_VALIDATE_NONCE</code>	<code>true</code> or <code>false</code> . The default value is <code>true</code> . This value should never be set to <code>false</code> except when temporarily debugging cryptographic nonce validation failures that occur during interactive logins. This application setting is intended for use with the V1 (classic) configuration experience. If using the V2 authentication configuration schema, you should instead use the <code>login.nonce.validateNonce</code> configuration value.
<code>WEBSITE_AUTH_V2_CONFIG_JSON</code>	This environment variable is populated automatically by the Azure App Service platform and is used to configure the integrated authentication module. The value of this environment variable corresponds to the V2 (non-classic) authentication configuration for the current app in Azure Resource Manager. It's not intended to be configured explicitly.
<code>WEBSITE_AUTH_ENABLED</code>	Read-only. Injected into a Windows or Linux app to indicate whether App Service authentication is enabled.
<code>WEBSITE_AUTH_ENCRYPTION_KEY</code>	By default, the automatically generated key is used as the encryption key. To override, set to a desired key. This is recommended if you want to share tokens or sessions across multiple apps. If specified, it supersedes the <code>MACHINEKEY_DecryptionKey</code> setting.
<code>WEBSITE_AUTH_SIGNING_KEY</code>	By default, the automatically generated key is used as the signing key. To override, set to a desired key. This is recommended if you want to share tokens or sessions across multiple apps. If specified, it supersedes the <code>MACHINEKEY_ValidationKey</code> setting.

Managed identity

The following environment variables are related to [managed identities](#).

[Expand table](#)

Setting name	Description
<code>IDENTITY_ENDPOINT</code>	Read-only. The URL to retrieve the token for the app's managed identity .
<code>MSI_ENDPOINT</code>	Deprecated. Use <code>IDENTITY_ENDPOINT</code> .
<code>IDENTITY_HEADER</code>	Read-only. Value that must be added to the <code>x-IDENTITY-HEADER</code> header when making an HTTP GET request to <code>IDENTITY_ENDPOINT</code> . The value is rotated by the platform.
<code>MSI_SECRET</code>	Deprecated. Use <code>IDENTITY_HEADER</code> .

Health check

The following environment variables are related to [health checks](#).

[Expand table](#)

Setting name	Description
<code>WEBSITE_HEALTHCHECK_MAXPINGFAILURES</code>	The maximum number of failed pings before removing the instance. Set to a value between <code>2</code> and <code>100</code> . When you're scaling up or out, App Service pings the Health check path to ensure new instances are ready. For more information, see Health check .
<code>WEBSITE_HEALTHCHECK_MAXUNHEALTHYWORKERPERCENT</code>	To avoid overwhelming healthy instances, no more than half of the instances will be excluded. For example, if an App Service Plan is scaled to four instances and three are unhealthy, at most two will be excluded. The other two instances (one healthy and one unhealthy) will continue to receive requests. In the worst-case scenario where all instances are unhealthy, none will be excluded. To override this behavior, set to a value between <code>1</code> and <code>100</code> . A higher value means more unhealthy instances will be removed. The default is <code>50</code> (50%).

Push notifications

The following environment variables are related to the [push notifications](#) feature.

[Expand table](#)

Setting name	Description
<code>WEBSITE_PUSH_ENABLED</code>	Read-only. Added when push notifications are enabled.
<code>WEBSITE_PUSH_TAG_WHITELIST</code>	Read-only. Contains the tags in the notification registration.
<code>WEBSITE_PUSH_TAGS_REQUIRING_AUTH</code>	Read-only. Contains a list of tags in the notification registration that requires user authentication.
<code>WEBSITE_PUSH_TAGS_DYNAMIC</code>	Read-only. Contains a list of tags in the notification registration that were added automatically.

 **Note**

This article contains references to a term that Microsoft no longer uses. When the term is removed from the software, we'll remove it from this article.

Webjobs

The following environment variables are related to [WebJobs](#).

[Expand table](#)

Setting name	Description
<code>WEBJOBS_RESTART_TIME</code>	For continuous jobs, delay in seconds when a job's process goes down for any reason before relaunching it.
<code>WEBJOBS_IDLE_TIMEOUT</code>	For triggered jobs, timeout in seconds, after which the job is aborted if it's in idle, has no CPU time or output.
<code>WEBJOBS_HISTORY_SIZE</code>	For triggered jobs, maximum number of runs kept in the history directory per job. The default is <code>50</code> .
<code>WEBJOBS_STOPPED</code>	Set to <code>1</code> to disable running any job, and stop all currently running jobs.
<code>WEBJOBS_DISABLE_SCHEDULE</code>	Set to <code>1</code> to turn off all scheduled triggering. Jobs can still be manually invoked.
<code>WEBJOBS_ROOT_PATH</code>	Absolute or relative path of webjob files. For a relative path, the value is combined with the default root path (<code>D:/home/site/wwwroot/</code> or <code>/home/site/wwwroot/</code>).

Setting name	Description
<code>WEBJOBS_LOG_TRIGGERED_JOBS_TO_APP_LOGS</code>	Set to true to send output from triggered WebJobs to the application logs pipeline (which supports file system, blobs, and tables).
<code>WEBJOBS_SHUTDOWN_FILE</code>	File that App Service creates when a shutdown request is detected. It's the web job process's responsibility to detect the presence of this file and initiate shutdown. When using the WebJobs SDK, this part is handled automatically.
<code>WEBJOBS_PATH</code>	Read-only. Root path of currently running job (will be under some temporary directory).
<code>WEBJOBS_NAME</code>	Read-only. Current job name.
<code>WEBJOBS_TYPE</code>	Read-only. Current job type (<code>triggered</code> or <code>continuous</code>).
<code>WEBJOBS_DATA_PATH</code>	Read-only. Current job metadata path to contain the job's logs, history, and any artifact of the job.
<code>WEBJOBS_RUN_ID</code>	Read-only. For triggered jobs, current run ID of the job.

Functions

 [Expand table](#)

Setting name	Description
<code>WEBSITE_FUNCTIONS_ARMCACHE_ENABLED</code>	Set to <code>0</code> to disable the functions cache.
<code>WEBSITE_MAX_DYNAMIC_APPLICATION_SCALE_OUT</code>	App settings reference for Azure Functions
<code>AzureWebJobsSecretStorageType</code>	App settings reference for Azure Functions
<code>FUNCTIONS_EXTENSION_VERSION</code>	App settings reference for Azure Functions
<code>FUNCTIONS_WORKER_RUNTIME</code>	App settings reference for Azure Functions
<code>AzureWebJobsStorage</code>	App settings reference for Azure Functions
<code>WEBSITE_CONTENTAZUREFILECONNECTIONSTRING</code>	App settings reference for Azure Functions
<code>WEBSITE_CONTENTSHARE</code>	App settings reference for Azure Functions
<code>WEBSITE_CONTENTOVERVNET</code>	App settings reference for Azure Functions
<code>WEBSITE_ENABLE_BROTLI_ENCODING</code>	App settings reference for Azure Functions
<code>WEBSITE_USE_PLACEHOLDER</code>	App settings reference for Azure Functions
<code>WEBSITE_PLACEHOLDER_MODE</code>	Read-only. Shows whether the function app is running on a placeholder host (<code>generalized</code>) or its own host (<code>specialized</code>).
<code>WEBSITE_DISABLE_ZIP_CACHE</code>	When your app runs from a ZIP package (<code>WEBSITE_RUN_FROM_PACKAGE=1</code>), the five most recently deployed ZIP packages are cached in the app's file system (D:\home\data\SitePackages). Set this variable to <code>1</code> to disable this cache. For Linux consumption apps, the ZIP package cache is disabled by default.

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

Configure error pages on App Service (preview)

Article • 10/25/2024

This article explains how to configure custom error pages on your web app. With App Service you can configure an error page for specific errors that will be presented to users instead of the default error page.

Prerequisite

In this tutorial, we're adding a custom 403 error page to our web app hosted on App Service and test it with an IP restriction. To do so, you need the following:

- a web app hosted on App Service w/ a Premium SKU
- an html file under 10 kb in size

Upload an error page

For this example, we're uploading and testing a 403 error page to present to the user. Name your html file to match the error code (for example, `403.html`). Once you have your html file prepared, you can upload it to your web app. In the configuration blade, you should see an **Error pages (preview)** tab. Click on this tab to view the error page options. If the options are greyed out, you need to upgrade to at least a Premium SKU to use this feature.

Select the error code that you'd like to upload an error page for and click **Edit**. On the next screen, click the folder icon to select your html file. The file must be in html format and within the 10 kb size limit. Find your .html file and click on the **Upload** button at the bottom of the screen. Notice the Status in the table updates from Not Configured to Configured. Then click **Save** to complete the upload.

Confirm error page

Once the custom error page is uploaded and saved, we can trigger and view the page. In this example, we can trigger the 403 error by using an IP restriction.

To set an IP restriction, go to the **Networking** blade and click the **Enabled with access restrictions** link under **Inbound traffic configuration**.

Under the **Site access and rules** section, select the **+Add** button to create an IP restriction.

In the form that follows, you need to change the Action to **Deny** and fill out the **Priority** and **IP Address Block**. In this example, we use the **Inbound address** found on the Networking blade and we're setting it to **/0** (for example, **12.123.12.123/0**). This disables all public access when visiting the site.

Once the Add rule form is filled out, select the **Add rule** button. Then click **Save**.

Once saved, you need to restart the site for the changes to take effect. Go to your overview page and select **browse**. You should now see your custom error page load.

Error codes

App Service currently supports three types of error codes that are available to customize:

[+] Expand table

Error code	description
403	Access restrictions
502	Gateway errors
503	Service unavailable

FAQ

1. I've uploaded my error page, why doesn't it show when the error is triggered?

Currently, error pages are only triggered when the error is coming from the front end. Errors that get triggered at the app level should still be handled through the app.

2. Why is the error page feature greyed out?

Error pages are currently a Premium feature. You need to use at least a Premium SKU to enable the feature.

Feedback

Was this page helpful?

 Yes

 No

Provide product feedback  | Get help at Microsoft Q&A

Configure an ASP.NET app for Azure App Service

Article • 08/08/2024

ⓘ Note

For ASP.NET Core, see [Configure an ASP.NET Core app for Azure App Service](#). If your ASP.NET app runs in a custom Windows or Linux container, see [Configure a custom container for Azure App Service](#).

ASP.NET apps must be deployed to Azure App Service as compiled binaries. The Visual Studio publishing tool builds the solution and then deploys the compiled binaries directly, whereas the App Service deployment engine deploys the code repository first and then compiles the binaries.

This guide provides key concepts and instructions for ASP.NET developers. If you've never used Azure App Service, follow the [ASP.NET quickstart](#) and [ASP.NET with SQL Database tutorial](#) first.

Show supported .NET Framework runtime versions

In App Service, the Windows instances already have all the supported .NET Framework versions installed. To show the .NET Framework runtime and SDK versions available to you, navigate to `https://<app-name>.scm.azurewebsites.net/DebugConsole` and run the appropriate command in the browser-based console:

For CLR 4 runtime versions (.NET Framework 4 and above):

CMD

```
ls "D:\Program Files (x86)\Reference Assemblies\Microsoft\Framework\.NETFramework"
```

Latest .NET Framework version may not be immediately available.

For CLR 2 runtime versions (.NET Framework 3.5 and below):

CMD

```
ls "D:\Program Files (x86)\Reference Assemblies\Microsoft\Framework"
```

Show current .NET Framework runtime version

Run the following command in the [Cloud Shell](#):

Azure CLI

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

A value of `v4.0` means the latest CLR 4 version (.NET Framework 4.x) is used. A value of `v2.0` means a CLR 2 version (.NET Framework 3.5) is used.

Set .NET Framework runtime version

By default, App Service uses the latest supported .NET Framework version to run your ASP.NET app. To run your app using .NET Framework 3.5 instead, run the following command in the [Cloud Shell](#) (`v2.0` signifies CLR 2):

Azure CLI

```
az webapp config set --resource-group <resource-group-name> --name <app-name> --net-framework-version v2.0
```

Access environment variables

In App Service, you can [set app settings](#) and connection strings outside of your app code. Then you can access them in any class using the standard ASP.NET pattern:

C#

```
using System.Configuration;  
...  
// Get an app setting  
ConfigurationManager.AppSettings["MySetting"];  
// Get a connection string  
ConfigurationManager.ConnectionStrings["MyConnection"];  
}
```

If you configure an app setting with the same name in App Service and in `web.config`, the App Service value takes precedence over the `web.config` value. The local `web.config` value lets you debug the app locally, but the App Service value lets your run the app in product with production settings. Connection strings work in the same way. This way, you can keep your application secrets outside of your code repository and access the appropriate values without changing your code.

ⓘ Note

Consider more secure connectivity options that don't require connection secrets at all. For more information, see [Secure connectivity to Azure services and databases from Azure App Service](#).

Deploy multi-project solutions

When a Visual Studio solution includes multiple projects, the Visual Studio publish process already includes selecting the project to deploy. When you deploy to the App Service deployment engine, such as with Git, or with ZIP deploy [with build automation enabled](#), the App Service deployment engine picks the first Web Site or Web Application Project it finds as the App Service app. You can specify which project App Service should use by specifying the `PROJECT` app setting. For example, run the following in the [Cloud Shell](#):

Azure CLI

```
az webapp config appsettings set --resource-group <resource-group-name> --  
name <app-name> --settings PROJECT=<project-name>/<project-name>.csproj
```

Get detailed exceptions page

When your ASP.NET app generates an exception in the Visual Studio debugger, the browser displays a detailed exception page, but in App Service that page is replaced by a generic error message. To display the detailed exception page in App Service, open the `Web.config` file and add the `<customErrors mode="Off"/>` element under the `<system.web>` element. For example:

XML

```
<system.web>  
  <customErrors mode="Off"/>
```

```
</system.web>
```

Redeploy your app with the updated *Web.config*. You should now see the same detailed exception page.

Access diagnostic logs

You can add diagnostic messages in your application code using [System.Diagnostics.Trace](#). For example:

C#

```
Trace.TraceError("Record not found!"); // Error trace  
Trace.TraceWarning("Possible data loss"); // Warning trace  
Trace.TraceInformation("GET /Home/Index"); // Information trace
```

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

Azure CLI

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

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

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

Azure CLI

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

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

ⓘ Note

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

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

More resources

- [Tutorial: Build an ASP.NET app in Azure with SQL Database](#)
 - [Environment variables and app settings reference](#)
-

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

Configure an ASP.NET Core app for Azure App Service

Article • 10/31/2024

ⓘ Note

For ASP.NET in .NET Framework, see [Configure an ASP.NET app for Azure App Service](#). If your ASP.NET Core app runs in a custom Windows or Linux container, see [Configure a custom container for Azure App Service](#).

ASP.NET Core apps must be deployed to Azure App Service as compiled binaries. The Visual Studio publishing tool builds the solution and then deploys the compiled binaries directly, whereas the App Service deployment engine deploys the code repository first and then compiles the binaries.

This guide provides key concepts and instructions for ASP.NET Core developers. If you've never used Azure App Service, follow the [ASP.NET Core quickstart](#) and [ASP.NET Core with SQL Database tutorial](#) first.

Show .NET Core version

To show the current .NET Core version, run the following command in the [Cloud Shell](#):

Azure CLI

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

To show all supported .NET Core versions, run the following command in the [Cloud Shell](#):

Azure CLI

```
az webapp list-runtimes --os linux | grep DOTNET
```

Set .NET Core version

Run the following command in the [Cloud Shell](#) to set the .NET Core version to 8.0:

Azure CLI

```
az webapp config set --name <app-name> --resource-group <resource-group-name> --linux-fx-version "DOTNETCORE|8.0"
```

Customize build automation

ⓘ Note

Building .NET 9 (STS) apps with Windows App Service using MSBuild or SCM_DO_BUILD is not yet supported. Support for these build scenarios will come after the initial GA date and by December 4th, 2024. Deployments that build outside of App Service through Visual Studio, Visual Studio Code, GitHub Actions and Azure DevOps are fully supported.

If you deploy your app using Git, or zip packages [with build automation enabled](#), the App Service build automation steps through the following sequence:

1. Run custom script if specified by `PRE_BUILD_SCRIPT_PATH`.
2. Run `dotnet restore` to restore NuGet dependencies.
3. Run `dotnet publish` to build a binary for production.
4. Run custom script if specified by `POST_BUILD_SCRIPT_PATH`.

`PRE_BUILD_COMMAND` and `POST_BUILD_COMMAND` are environment variables that are empty by default. To run prebuild commands, define `PRE_BUILD_COMMAND`. To run post-build commands, define `POST_BUILD_COMMAND`.

The following example specifies the two variables to a series of commands, separated by commas.

Azure CLI

```
az webapp config appsettings set --name <app-name> --resource-group <resource-group-name> --settings PRE_BUILD_COMMAND="echo foo, scripts/prebuild.sh"
az webapp config appsettings set --name <app-name> --resource-group <resource-group-name> --settings POST_BUILD_COMMAND="echo foo, scripts/postbuild.sh"
```

For other environment variables to customize build automation, see [Oryx configuration](#).

For more information on how App Service runs and builds ASP.NET Core apps in Linux, see [Oryx documentation: How .NET Core apps are detected and built](#).

Access environment variables

In App Service, you can [set app settings](#) outside of your app code. Then you can access them in any class using the standard ASP.NET Core dependency injection pattern:

```
C#  
  
using Microsoft.Extensions.Configuration;  
  
namespace SomeNamespace  
{  
    public class SomeClass  
    {  
        private IConfiguration _configuration;  
  
        public SomeClass(IConfiguration configuration)  
        {  
            _configuration = configuration;  
        }  
  
        public SomeMethod()  
        {  
            // retrieve nested App Service app setting  
            var myHierarchicalConfig =  
_configuration["My:Hierarchical:Config>Data"];  
            // retrieve App Service connection string  
            var myConnString =  
_configuration.GetConnectionString("MyDbConnection");  
        }  
    }  
}
```

If you configure an app setting with the same name in App Service and in *appsettings.json*, for example, the App Service value takes precedence over the *appsettings.json* value. The local *appsettings.json* value lets you debug the app locally, but the App Service value lets you run the app in production with production settings. Connection strings work in the same way. This way, you can keep your application secrets outside of your code repository and access the appropriate values without changing your code.

Note

Consider more secure connectivity options that don't require connection secrets at all. For more information, see [Secure connectivity to Azure services and databases](#)

[from Azure App Service.](#)

ⓘ Note

Note the [hierarchical configuration data](#) in `appsettings.json` is accessed using the `_` (double underscore) delimiter that's standard on Linux to .NET Core. To override a specific hierarchical configuration setting in App Service, set the app setting name with the same delimited format in the key. you can run the following example in the [Cloud Shell](#):

Azure CLI

```
az webapp config appsettings set --name <app-name> --resource-group <resource-group-name> --settings My_Hierarchical_Config_Data="some value"
```

Deploy multi-project solutions

When a Visual Studio solution includes multiple projects, the Visual Studio publish process already includes selecting the project to deploy. When you deploy to the App Service deployment engine, such as with Git, or with ZIP deploy [with build automation enabled](#), the App Service deployment engine picks the first Web Site or Web Application Project it finds as the App Service app. You can specify which project App Service should use by specifying the `PROJECT` app setting. For example, run the following command in the [Cloud Shell](#):

Azure CLI

```
az webapp config appsettings set --resource-group <resource-group-name> --name <app-name> --settings PROJECT="<><project-name>/<project-name>.csproj"
```

Access diagnostic logs

ASP.NET Core provides a [built-in logging provider for App Service](#). In `Program.cs` of your project, add the provider to your application through the `ConfigureLogging` extension method, as shown in the following example:

C#

```
public static IHostBuilder CreateHostBuilder(string[] args) =>
    Host.CreateDefaultBuilder(args)
```

```
.ConfigureLogging(logging =>
{
    logging.AddAzureWebAppDiagnostics();
})
.ConfigureWebHostDefaults(webBuilder =>
{
    webBuilder.UseStartup<Startup>();
});
```

You can then configure and generate logs with the [standard .NET Core pattern](#).

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

Azure CLI

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

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

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

Azure CLI

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

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

ⓘ Note

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

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

For more information on troubleshooting ASP.NET Core apps in App Service, see [Troubleshoot ASP.NET Core on Azure App Service and IIS](#)

Get detailed exceptions page

When your ASP.NET Core app generates an exception in the Visual Studio debugger, the browser displays a detailed exception page, but in App Service that page is replaced by a generic **HTTP 500 or An error occurred while processing your request**. To display the detailed exception page in App Service, Add the `ASPNETCORE_ENVIRONMENT` app setting to your app by running the following command in the [Cloud Shell](#).

Azure CLI

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

Detect HTTPS session

In App Service, [TLS/SSL termination](#) happens at the network load balancers, so all HTTPS requests reach your app as unencrypted HTTP requests. If your app logic needs to know if the user requests are encrypted or not, configure the Forwarded Headers Middleware in *Startup.cs*:

- Configure the middleware with `ForwardedHeadersOptions` to forward the `X-Forwarded-For` and `X-Forwarded-Proto` headers in `Startup.ConfigureServices`.
- Add private IP address ranges to the known networks, so that the middleware can trust the App Service load balancer.
- Invoke the `UseForwardedHeaders` method in `Startup.Configure` before calling other middleware.

Putting all three elements together, your code looks like the following example:

C#

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddMvc();

    services.Configure<ForwardedHeadersOptions>(options =>
    {
        options.ForwardedHeaders =
            ForwardedHeaders.XForwardedFor |
        ForwardedHeaders.XForwardedProto;
        // These three subnets encapsulate the applicable Azure subnets. At
        // the moment, it's not possible to narrow it down further.
        options.KnownNetworks.Add(new
        IPNetwork(IPAddress.Parse("::ffff:10.0.0.0"), 104));
        options.KnownNetworks.Add(new
        IPNetwork(IPAddress.Parse("::ffff:192.168.0.0"), 112));
        options.KnownNetworks.Add(new
        IPNetwork(IPAddress.Parse("::ffff:172.16.0.0"), 108));
    });
}
```

```
    });

}

public void Configure(IApplicationBuilder app, IHostingEnvironment env)
{
    app.UseForwardedHeaders();

    ...

    app.UseMvc();
}
```

For more information, see [Configure ASP.NET Core to work with proxy servers and load balancers](#).

Rewrite or redirect URL

To rewrite or redirect URL, use the [URL rewriting middleware in ASP.NET Core](#).

Open SSH session in browser

To make open a direct SSH session with your container, your app should be running.

Paste the following URL into your browser and replace `<app-name>` with your app name:

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

If you're not yet authenticated, you're required to authenticate with your Azure subscription to connect. Once authenticated, you see an in-browser shell, where you can run commands inside your container.

```
root@9e933156516f:~# service --status-all
[ + ]  apache2
[ - ]  bootlogs
[ - ]  bootmisc.sh
[ - ]  checkfs.sh
[ - ]  checkroot-bootclean.sh
[ - ]  checkroot.sh
[ - ]  hostname.sh
[ ? ]  hwclock.sh
[ - ]  killprocs
[ - ]  motd
[ - ]  mountall-bootclean.sh
[ - ]  mountall.sh
[ - ]  mountdevsubfs.sh
[ - ]  mountkernfs.sh
[ - ]  mountnfs-bootclean.sh
[ - ]  mountnfs.sh
[ - ]  mysql
[ - ]  procps
[ - ]  rc.local
[ - ]  rmmnlogin
[ - ]  sendsigs
[ + ]  ssh
[ + ]  udev
[ ? ]  udev-finish
[ - ]  umountfs
[ - ]  umountnfs.sh
[ - ]  umountroot
[ - ]  urandom
root@9e933156516f:~#
```

ssh://root@ 2222 | SSH CONNECTION ESTABLISHED |

ⓘ Note

Any changes you make outside the `/home` directory are stored in the container itself and don't persist beyond an app restart.

To open a remote SSH session from your local machine, see [Open SSH session from remote shell](#).

robots933456 in logs

You may see the following message in the container logs:

```
2019-04-08T14:07:56.641002476Z "-" - - [08/Apr/2019:14:07:56 +0000] "GET /robots933456.txt HTTP/1.1" 404 415 "-" "-"
```

You can safely ignore this message. `/robots933456.txt` is a dummy URL path that App Service uses to check if the container is capable of serving requests. A 404 response simply indicates that the path doesn't exist, but it lets App Service know that the container is healthy and ready to respond to requests.

Next steps

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

[App Service Linux FAQ](#)

Or, see more resources:

[Environment variables and app settings reference](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

Configure a Node.js app for Azure App Service

Article • 09/13/2024

Node.js apps must be deployed with all the required npm dependencies. The App Service deployment engine automatically runs `npm install --production` for you when you deploy a [Git repository](#), or when you deploy a [Zip package with build automation enabled](#). If you deploy your files using [FTP/S](#), however, you need to upload the required packages manually.

This guide provides key concepts and instructions for Node.js developers who deploy to App Service. If you've never used Azure App Service, follow the [Node.js quickstart](#) and [Node.js with MongoDB tutorial](#) first.

Show Node.js version

To show the current Node.js version, run the following command in the [Cloud Shell](#):

Azure CLI

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

To show all supported Node.js versions, run the following command in the [Cloud Shell](#):

Azure CLI

```
az webapp list-runtimes --os linux | grep NODE
```

Set Node.js version

To set your app to a [supported Node.js version](#), run the following command in the [Cloud Shell](#):

Azure CLI

```
az webapp config set --resource-group <resource-group-name> --name <app-name> --linux-fx-version "NODE|14-lts"
```

This setting specifies the Node.js version to use, both at runtime and during automated package restore in Kudu.

ⓘ Note

You should set the Node.js version in your project's `package.json`. The deployment engine runs in a separate container that contains all the supported Node.js versions.

Get port number

Your Node.js app needs to listen to the right port to receive incoming requests.

App Service sets the environment variable `PORT` in the Node.js container, and forwards the incoming requests to your container at that port number. To receive the requests, your app should listen to that port using `process.env.PORT`. The following example shows how to do that in a simple Express app:

JavaScript

```
const express = require('express')
const app = express()
const port = process.env.PORT || 3000

app.get('/', (req, res) => {
  res.send('Hello World!')
})

app.listen(port, () => {
  console.log(`Example app listening at http://localhost:${port}`)
})
```

Customize build automation

If you deploy your app by using Git, or by using zip packages [with build automation enabled](#), the App Service build automation steps through the following sequence:

1. Run custom script, if one is specified by `PRE_BUILD_SCRIPT_PATH`.
2. Run `npm install` without any flags, which includes npm `preinstall` and `postinstall` scripts and also installs `devDependencies`.
3. Run `npm run build` if a build script is specified in your `package.json`.
4. Run `npm run build:azure` if a build:azure script is specified in your `package.json`.

5. Run custom script if specified by `POST_BUILD_SCRIPT_PATH`.

ⓘ Note

As is noted in the [npm docs](#), scripts named `prebuild` and `postbuild` run before and after `build`, respectively, if specified. `preinstall` and `postinstall` run before and after `install`, respectively.

`PRE_BUILD_COMMAND` and `POST_BUILD_COMMAND` are environment variables that are empty by default. To run pre-build commands, define `PRE_BUILD_COMMAND`. To run post-build commands, define `POST_BUILD_COMMAND`.

The following example uses the two variables to specify a series of commands, separated by commas.

Azure CLI

```
az webapp config appsettings set --name <app-name> --resource-group <resource-group-name> --settings PRE_BUILD_COMMAND="echo foo, scripts/prebuild.sh"
az webapp config appsettings set --name <app-name> --resource-group <resource-group-name> --settings POST_BUILD_COMMAND="echo foo, scripts/postbuild.sh"
```

For information about additional environment variables to customize build automation, see [Oryx configuration](#).

For more information on how App Service runs and builds Node.js apps in Linux, see [Oryx documentation: How Node.js apps are detected and built](#).

Configure Node.js server

The Node.js containers come with [PM2](#), a production process manager. You can configure your app to start with PM2, with npm start, or with a custom command.

[+] Expand table

Tool	Purpose
Run with PM2	Recommended - Production or staging use. PM2 provides a full-service app management platform.
Run with npm start	Development use only.

Tool	Purpose
Run with a custom command	Either development or staging.

Run with PM2

The container automatically starts your app with PM2 when one of the common Node.js files is found in your project:

- *bin/www*
- *server.js*
- *app.js*
- *index.js*
- *hostingstart.js*
- One of the following [PM2 files](#): *process.json* or *ecosystem.config.js*

You can also configure a custom start file with the following extensions:

- A *.js* file
- A [PM2 file](#) with the extension *.json*, *.config.js*, *.yaml*, or *.yml*

ⓘ Note

Starting from **Node 14 LTS**, the container doesn't automatically start your app with PM2. To start your app with PM2, set the startup command to `pm2 start <.js-file-or-PM2-file> --no-daemon`. Be sure to use the `--no-daemon` argument because PM2 needs to run in the foreground for the container to work properly.

To add a custom start file, run the following command in the [Cloud Shell](#):

Azure CLI

```
az webapp config set --resource-group <resource-group-name> --name <app-name> --startup-file "<filename-with-extension>"
```

Run with a custom command

App Service can start your app using a custom command, such as an executable like *run.sh*. For example, to run `npm run start:prod`, run the following command in the [Cloud Shell](#):

Azure CLI

```
az webapp config set --resource-group <resource-group-name> --name <app-name> --startup-file "npm run start:prod"
```

Run with npm start

To start your app using `npm start`, just make sure a `start` script is in the `package.json` file. For example:

JSON

```
{  
  ...  
  "scripts": {  
    "start": "gulp",  
    ...  
  },  
  ...  
}
```

To use a custom `package.json` in your project, run the following command in the [Cloud Shell](#):

Azure CLI

```
az webapp config set --resource-group <resource-group-name> --name <app-name> --startup-file "<filename>.json"
```

Debug remotely

You can debug your Node.js app remotely in [Visual Studio Code](#) if you configure it to run with [PM2](#), except when you run it using a `.config.js`, `.yml`, or `.yaml`.

In most cases, no extra configuration is required for your app. If your app is run with a `process.json` file (default or custom), it must have a `script` property in the JSON root. For example:

JSON

```
{  
  "name": "worker",  
  "script": "./index.js",  
  ...  
}
```

```
...  
}
```

To set up Visual Studio Code for remote debugging, install the [App Service extension](#). Follow the instructions on the extension page and sign in to Azure in Visual Studio Code.

In the Azure explorer, find the app you want to debug, right-click it and select **Start Remote Debugging**. Select **Yes** to enable remote debugging for your app. App Service starts a tunnel proxy for you and attaches the debugger. You can then make requests to the app and see the debugger pausing at break points.

Once finished with debugging, stop the debugger by selecting **Disconnect**. When prompted, you should select **Yes** to disable remote debugging. To disable it later, right-click your app again in the Azure explorer and select **Disable Remote Debugging**.

Access environment variables

In App Service, you can [set app settings](#) outside of your app code. Then you can access them using the standard Node.js pattern. For example, to access an app setting called `NODE_ENV`, use the following code:

```
JavaScript
```

```
process.env.NODE_ENV
```

Run Grunt/Bower/Gulp

By default, App Service build automation runs `npm install --production` when it recognizes that a Node.js app is deployed through Git, or through Zip deployment [with build automation enabled](#). If your app requires any of the popular automation tools, such as Grunt, Bower, or Gulp, you need to supply a [custom deployment script](#) to run it.

To enable your repository to run these tools, you need to add them to the dependencies in `package.json`. For example:

```
JSON
```

```
"dependencies": {  
  "bower": "^1.7.9",  
  "grunt": "^1.0.1",  
  "gulp": "^3.9.1",
```

```
...  
}
```

From a local terminal window, change the directory to your repository root and run the following commands:

Bash

```
npm install kuduscript -g  
kuduscript --node --scriptType bash --suppressPrompt
```

Your repository root now has two additional files: `.deployment` and `deploy.sh`.

Open `deploy.sh` and find the `Deployment` section, which looks like this:

Bash

```
#####
#####  
# Deployment  
# -----
```

This section ends with running `npm install --production`. Add the code section you need to run the required tool *at the end* of the `Deployment` section:

- [Bower](#)
- [Gulp](#)
- [Grunt](#)

See an [example in the MEAN.js sample ↗](#), where the deployment script also runs a custom `npm install` command.

Bower

This snippet runs `bower install`.

Bash

```
if [ -e "$DEPLOYMENT_TARGET/bower.json" ]; then  
  cd "$DEPLOYMENT_TARGET"  
  eval ./node_modules/.bin/bower install  
  exitWithErrorOnAny "bower failed"  
  cd - > /dev/null  
fi
```

Gulp

This snippet runs `gulp imagemin`.

Bash

```
if [ -e "$DEPLOYMENT_TARGET/gulpfile.js" ]; then
  cd "$DEPLOYMENT_TARGET"
  eval ./node_modules/.bin/gulp imagemin
  exitWithErrorOnAnyError "gulp failed"
  cd - > /dev/null
fi
```

Grunt

This snippet runs `grunt`.

Bash

```
if [ -e "$DEPLOYMENT_TARGET/Gruntfile.js" ]; then
  cd "$DEPLOYMENT_TARGET"
  eval ./node_modules/.bin/grunt
  exitWithErrorOnAnyError "Grunt failed"
  cd - > /dev/null
fi
```

Detect HTTPS session

In App Service, [TLS/SSL termination](#) happens at the network load balancers, so all HTTPS requests reach your app as unencrypted HTTP requests. If your app logic needs to check if the user requests are encrypted, inspect the `X-Forwarded-Proto` header.

Popular web frameworks let you access the `X-Forwarded-*` information in your standard app pattern. In [Express](#), you can use [trust proxies](#). For example:

JavaScript

```
app.set('trust proxy', 1)
...
if (req.secure) {
  // Do something when HTTPS is used
}
```

Access diagnostic logs

You can access the console logs generated from inside the container.

First, turn on container logging by running the following command:

Azure CLI

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

Replace `<app-name>` and `<resource-group-name>` with the names appropriate for your web app.

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

Azure CLI

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

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

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

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

URL rewrites

When deploying Node.js apps on Azure App Service for Linux, you might need to handle URL rewrites directly within your application. This is particularly useful for ensuring specific URL patterns are redirected to the correct endpoints without relying on web server configurations. There are several ways to accomplish URL rewrites in Node.js. One example is through the [express-ur rewrite](#) package.

Monitor with Application Insights

Application Insights allows you to monitor your application's performance, exceptions, and usage without making any code changes. To attach the Application Insights agent, go to your web app in the portal, select **Application Insights** under **Settings**, and then select **Turn on Application Insights**. Next, select an existing Application Insights

resource or create a new one. Finally, select **Apply** at the bottom. To instrument your web app using PowerShell, see [these instructions](#)

This agent will monitor your server-side Node.js application. To monitor your client-side JavaScript, [add the JavaScript SDK to your project](#).

For more information, see the [Application Insights extension release notes](#).

Troubleshooting

When a working Node.js app behaves differently in App Service or has errors, try the following:

- [Access the log stream](#).
- Test the app locally in production mode. App Service runs your Node.js apps in production mode, so you need to make sure that your project works as expected in production mode locally. For example:
 - Depending on your `package.json`, different packages might be installed for production mode (`dependencies` vs. `devDependencies`).
 - Certain web frameworks might deploy static files differently in production mode.
 - Certain web frameworks might use custom startup scripts when running in production mode.
- Run your app in App Service in development mode. For example, in [MEAN.js ↗](#), you can set your app to development mode at runtime by [setting the NODE_ENV app setting](#).

robots933456 in logs

You may see the following message in the container logs:

```
2019-04-08T14:07:56.641002476Z "-" - - [08/Apr/2019:14:07:56 +0000] "GET /robots933456.txt HTTP/1.1" 404 415 "-" "-"
```

You can safely ignore this message. `/robots933456.txt` is a dummy URL path that App Service uses to check if the container is capable of serving requests. A 404 response simply indicates that the path doesn't exist, but it lets App Service know that the container is healthy and ready to respond to requests.

Next steps

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

[Azure App Service on Linux FAQ](#)

Or, see additional resources:

[Environment variables and app settings reference](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

Configure a PHP app for Azure App Service

Article • 08/31/2023

Show PHP version

This guide shows you how to configure your PHP web apps, mobile back ends, and API apps in Azure App Service.

This guide provides key concepts and instructions for PHP developers who deploy apps to App Service. If you've never used Azure App Service, follow the [PHP quickstart](#) and [PHP with MySQL tutorial](#) first.

To show the current PHP version, run the following command in the [Cloud Shell](#):

Azure CLI

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

ⓘ Note

To address a development slot, include the parameter `--slot` followed by the name of the slot.

To show all supported PHP versions, run the following command in the [Cloud Shell](#):

Azure CLI

```
az webapp list-runtimes --os linux | grep PHP
```

Set PHP version

Run the following command in the [Cloud Shell](#) to set the PHP version to 8.1:

Azure CLI

```
az webapp config set --resource-group <resource-group-name> --name <app-name> --linux-fx-version "PHP|8.1"
```

Customize build automation

If you deploy your app using Git, or using zip packages [with build automation enabled](#), the App Service build automation steps through the following sequence:

1. Run custom script if specified by `PRE_BUILD_SCRIPT_PATH`.
2. Run `php composer.phar install`.
3. Run custom script if specified by `POST_BUILD_SCRIPT_PATH`.

`PRE_BUILD_COMMAND` and `POST_BUILD_COMMAND` are environment variables that are empty by default. To run pre-build commands, define `PRE_BUILD_COMMAND`. To run post-build commands, define `POST_BUILD_COMMAND`.

The following example specifies the two variables to a series of commands, separated by commas.

Azure CLI

```
az webapp config appsettings set --name <app-name> --resource-group
<resource-group-name> --settings PRE_BUILD_COMMAND="echo foo,
scripts/prebuild.sh"
az webapp config appsettings set --name <app-name> --resource-group
<resource-group-name> --settings POST_BUILD_COMMAND="echo foo,
scripts/postbuild.sh"
```

For additional environment variables to customize build automation, see [Oryx configuration](#).

For more information on how App Service runs and builds PHP apps in Linux, see [Oryx documentation: How PHP apps are detected and built](#).

Customize start-up

If you want, you can run a custom command at the container start-up time, by running the following command in the [Cloud Shell](#):

Azure CLI

```
az webapp config set --resource-group <resource-group-name> --name <app-
name> --startup-file "<custom-command>"
```

Access environment variables

In App Service, you can [set app settings](#) outside of your app code. Then you can access them using the standard [getenv\(\)](#) pattern. For example, to access an app setting called `DB_HOST`, use the following code:

```
PHP
```

```
getenv("DB_HOST")
```

Change site root

The web framework of your choice may use a subdirectory as the site root. For example, [Laravel](#), uses the `public/` subdirectory as the site root.

The default PHP image for App Service uses Nginx, and you change the site root by [configuring the Nginx server with the root directive](#). This [example configuration file](#) contains the following snippets that changes the `root` directive:

```
server {
    #proxy_cache cache;
    #proxy_cache_valid 200 1s;
    listen 8080;
    listen [::]:8080;
    root /home/site/wwwroot/public; # Changed for Laravel

    location / {
        index index.php index.html index.htm hostingstart.html;
        try_files $uri $uri/ /index.php?$args; # Changed for Laravel
    }
}
```

The default container uses the configuration file found at `/etc/nginx/sites-available/default`. Keep in mind that any edit you make to this file is erased when the app restarts. To make a change that is effective across app restarts, [add a custom start-up command](#) like this example:

```
cp /home/site/wwwroot/default /etc/nginx/sites-available/default && service nginx reload
```

This command replaces the default Nginx configuration file with a file named `default` in your repository root and restarts Nginx.

Detect HTTPS session

In App Service, [TLS/SSL termination](#) happens at the network load balancers, so all HTTPS requests reach your app as unencrypted HTTP requests. If your app logic needs to check if the user requests are encrypted or not, inspect the `X-Forwarded-Proto` header.

PHP

```
if (isset($_SERVER['HTTP_X_FORWARDED_PROTO']) &&
$_SERVER['HTTP_X_FORWARDED_PROTO'] === 'https') {
// Do something when HTTPS is used
}
```

Popular web frameworks let you access the `X-Forwarded-*` information in your standard app pattern. In [CodeIgniter](#), the `is_https()` checks the value of `X_FORWARDED_PROTO` by default.

Customize php.ini settings

If you need to make changes to your PHP installation, you can change any of the [php.ini directives](#) by following these steps.

ⓘ Note

The best way to see the PHP version and the current *php.ini* configuration is to call `phpinfo()` in your app.

Customize-non-PHP_INI_SYSTEM directives

To customize PHP_INI_USER, PHP_INI_PERDIR, and PHP_INI_ALL directives for linux web apps, such as `upload_max_filesize` and `expose_php`, use a custom "ini" file. You can create it in an [SSH session](#).

1. Go to your KUDU site `https://<sitename>.scm.azurewebsites.net`.
2. Select Bash or SSH from the top menu.
3. In Bash/SSH, go to your `/home/site/wwwroot` directory.
4. Create a directory called "ini" (for example, `mkdir ini`).
5. Change the current working directory to the "ini" folder you just created.

You need to create an "ini" file to add your settings to. In this example, we use "extensions.ini." There are no file editors such as Vi, Vim, or Nano so you'll use echo to add the settings to the file. Change the "upload_max_filesize" from 2M to 50M. Use the following command to add the setting and create an "extensions.ini" file if one doesn't already exist.

```
/home/site/wwwroot/ini>echo "upload_max_filesize=50M" >> extensions.ini  
/home/site/wwwroot/ini>cat extensions.ini  
  
upload_max_filesize=50M  
  
/home/site/wwwroot/ini>
```

Then, go to the Azure portal and add an Application Setting to scan the "ini" directory that you just created to apply the change for upload_max_filesize.

1. Go to the [Azure portal](#) and select your App Service Linux PHP application.
2. Select Application Settings for the app.
3. Under the Application settings section, select **+ Add new setting**.
4. For the App Setting Name, enter "PHP_INI_SCAN_DIR" and for value, enter "/home/site/wwwroot/ini."
5. Select the save button.

 **Note**

If you recompiled a PHP extension, such as GD, follow the steps at [Recompiling PHP Extensions at Azure App Service - Adding PHP Extensions](#)

Customize PHP_INI_SYSTEM directives

To customize PHP_INI_SYSTEM directives (see [php.ini directives](#)), you can't use the .htaccess approach. App Service provides a separate mechanism using the `PHP_INI_SCAN_DIR` app setting.

First, run the following command in the [Cloud Shell](#) to add an app setting called `PHP_INI_SCAN_DIR`:

Azure CLI

```
az webapp config appsettings set --name <app-name> --resource-group  
<resource-group-name> --settings
```

```
PHP_INI_SCAN_DIR="/usr/local/etc/php/conf.d:/home/site/ini"
```

`/usr/local/etc/php/conf.d` is the default directory where `php.ini` exists. `/home/site/ini` is the custom directory in which you'll add a custom `.ini` file. You separate the values with a `:`.

Navigate to the web SSH session with your Linux container (<https://<app-name>.scm.azurewebsites.net/webssh/host>).

Create a directory in `/home/site` called `ini`, then create an `.ini` file in the `/home/site/ini` directory (for example, `settings.ini`) with the directives you want to customize. Use the same syntax you would use in a `php.ini` file.

💡 Tip

In the built-in Linux containers in App Service, `/home` is used as persisted shared storage.

For example, to change the value of [expose_php](#) run the following commands:

Bash

```
cd /home/site
mkdir ini
echo "expose_php = Off" >> ini/setting.ini
```

For the changes to take effect, restart the app.

Enable PHP extensions

The built-in PHP installations contain the most commonly used extensions. You can enable additional extensions in the same way that you [customize php.ini directives](#).

ⓘ Note

The best way to see the PHP version and the current `php.ini` configuration is to call `phpinfo()` in your app.

To enable additional extensions, by following these steps:

Add a `bin` directory to the root directory of your app and put the `.so` extension files in it (for example, `mongodb.so`). Make sure that the extensions are compatible with the PHP version in Azure and are VC9 and non-thread-safe (nts) compatible.

Deploy your changes.

Follow the steps in [Customize PHP_INI_SYSTEM directives](#), add the extensions into the custom `.ini` file with the `extension` or `zend_extension` directives.

```
ini  
  
extension=/home/site/wwwroot/bin/mongodb.so  
zend_extension=/home/site/wwwroot/bin/xdebug.so
```

For the changes to take effect, restart the app.

Access diagnostic logs

You can access the console logs generated from inside the container.

First, turn on container logging by running the following command:

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

Replace `<app-name>` and `<resource-group-name>` with the names appropriate for your web app.

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

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

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

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

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

Troubleshooting

When a working PHP app behaves differently in App Service or has errors, try the following:

- [Access the log stream](#).
- Test the app locally in production mode. App Service runs your app in production mode, so you need to make sure that your project works as expected in production mode locally. For example:
 - Depending on your `composer.json`, different packages may be installed for production mode (`require` vs. `require-dev`).
 - Certain web frameworks may deploy static files differently in production mode.
 - Certain web frameworks may use custom startup scripts when running in production mode.
- Run your app in App Service in debug mode. For example, in [Laravel](#), you can configure your app to output debug messages in production by [setting the APP_DEBUG app setting to true](#).

robots933456 in logs

You may see the following message in the container logs:

```
2019-04-08T14:07:56.641002476Z "-" - - [08/Apr/2019:14:07:56 +0000] "GET /robots933456.txt HTTP/1.1" 404 415 "-" "-"
```

You can safely ignore this message. `/robots933456.txt` is a dummy URL path that App Service uses to check if the container is capable of serving requests. A 404 response simply indicates that the path doesn't exist, but it lets App Service know that the container is healthy and ready to respond to requests.

Next steps

[Tutorial: PHP app with MySQL](#)

[App Service Linux FAQ](#)

Or, see additional resources:

[Environment variables and app settings reference](#)

Configure a Linux Python app for Azure App Service

Article • 08/29/2024

This article describes how [Azure App Service](#) runs Python apps, how you can migrate existing apps to Azure, and how you can customize the behavior of App Service when you need to. Python apps must be deployed with all the required [pip ↗](#) modules.

The App Service deployment engine automatically activates a virtual environment and runs `pip install -r requirements.txt` for you when you deploy a [Git repository](#), or when you deploy a [zip package with build automation enabled](#).

This guide provides key concepts and instructions for Python developers who use a built-in Linux container in App Service. If you've never used Azure App Service, first follow the [Python quickstart](#) and [Python with PostgreSQL tutorial](#).

You can use either the [Azure portal ↗](#) or the Azure CLI for configuration:

- **Azure portal**, use the app's **Settings > Configuration** page as described in [Configure an App Service app in the Azure portal](#).
- **Azure CLI**: you have two options.
 - Run commands in the [Azure Cloud Shell](#).
 - Run commands locally by installing the latest version of the [Azure CLI](#), then sign in to Azure using [az login](#).

ⓘ Note

Linux is the only operating system option for running Python apps in App Service. Python on Windows is no longer supported. You can however build your own custom Windows container image and run that in App Service. For more information, see [use a custom Docker image](#).

Configure Python version

- **Azure portal**: use the **General settings** tab on the **Configuration** page as described in [Configure general settings](#) for Linux containers.
- **Azure CLI**:

- Show the current Python version with [az webapp config show](#):

```
Azure CLI
```

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

Replace `<resource-group-name>` and `<app-name>` with the names appropriate for your web app.

- Set the Python version with [az webapp config set](#)

```
Azure CLI
```

```
az webapp config set --resource-group <resource-group-name> --name <app-name> --linux-fx-version "PYTHON|3.11"
```

- Show all Python versions that are supported in Azure App Service with [az webapp list-runtimes](#):

```
Azure CLI
```

```
az webapp list-runtimes --os linux | grep PYTHON
```

You can run an unsupported version of Python by building your own container image instead. For more information, see [use a custom Docker image](#).

Customize build automation

App Service's build system, called Oryx, performs the following steps when you deploy your app, if the app setting `SCM_DO_BUILD_DURING_DEPLOYMENT` is set to `1`:

1. Run a custom pre-build script, if that step is specified by the `PRE_BUILD_COMMAND` setting. (The script can itself run other Python and Node.js scripts, pip and npm commands, and Node-based tools like yarn, for example, `yarn install` and `yarn build`.)
2. Run `pip install -r requirements.txt`. The `requirements.txt` file must be present in the project's root folder. Otherwise, the build process reports the error: "Could not find setup.py or requirements.txt; Not running pip install."
3. If `manage.py` is found in the root of the repository (indicating a Django app), run `manage.py collectstatic`. However, if the `DISABLE_COLLECTSTATIC` setting is `true`, this

step is skipped.

4. Run custom post-build script, if that step is specified by the `POST_BUILD_COMMAND` setting. (Again, the script can run other Python and Node.js scripts, pip and npm commands, and Node-based tools.)

By default, the `PRE_BUILD_COMMAND`, `POST_BUILD_COMMAND`, and `DISABLE_COLLECTSTATIC` settings are empty.

- To disable running collectstatic when building Django apps, set the `DISABLE_COLLECTSTATIC` setting to `true`.
- To run pre-build commands, set the `PRE_BUILD_COMMAND` setting to contain either a command, such as `echo Pre-build command`, or a path to a script file, relative to your project root, such as `scripts/prebuild.sh`. All commands must use relative paths to the project root folder.
- To run post-build commands, set the `POST_BUILD_COMMAND` setting to contain either a command, such as `echo Post-build command`, or a path to a script file, relative to your project root, such as `scripts/postbuild.sh`. All commands must use relative paths to the project root folder.

For other settings that customize build automation, see [Oryx configuration](#).

To access the build and deployment logs, see [Access deployment logs](#).

For more information on how App Service runs and builds Python apps in Linux, see [How Oryx detects and builds Python apps](#).

ⓘ Note

The `PRE_BUILD_SCRIPT_PATH` and `POST_BUILD_SCRIPT_PATH` settings are identical to `PRE_BUILD_COMMAND` and `POST_BUILD_COMMAND` and are supported for legacy purposes.

A setting named `SCM_DO_BUILD_DURING_DEPLOYMENT`, if it contains `true` or `1`, triggers an Oryx build that happens during deployment. The setting is `true` when you deploy by using Git, the Azure CLI command `az webapp up`, and Visual Studio Code.

ⓘ Note

Always use relative paths in all pre- and post-build scripts because the build container in which Oryx runs is different from the runtime container in which the

app runs. Never rely on the exact placement of your app project folder within the container (for example, that it's placed under `site/wwwroot`).

Migrate existing applications to Azure

Existing web applications can be redeployed to Azure as follows:

- 1. Source repository:** Maintain your source code in a suitable repository like GitHub, which enables you to set up continuous deployment later in this process.
 - Your `requirements.txt` file must be at the root of your repository for App Service to automatically install the necessary packages.
- 2. Database:** If your app depends on a database, create the necessary resources on Azure as well.
- 3. App service resources:** Create a resource group, App Service plan, and App Service web app to host your application. You can do this easily by running the Azure CLI command `az webapp up`. Or, you can create and deploy resources as shown in [Tutorial: Deploy a Python \(Django or Flask\) web app with PostgreSQL](#). Replace the names of the resource group, App Service plan, and web app to be more suitable for your application.
- 4. Environment variables:** If your application requires any environment variables, create equivalent [App Service application settings](#). These App Service settings appear to your code as environment variables, as described in [Access environment variables](#).
 - Database connections, for example, are often managed through such settings, as shown in [Tutorial: Deploy a Django web app with PostgreSQL - verify connection settings](#).
 - See [Production settings for Django apps](#) for specific settings for typical Django apps.
- 5. App startup:** Review the section [Container startup process](#) later in this article to understand how App Service attempts to run your app. App Service uses the Gunicorn web server by default, which must be able to find your app object or `wsgi.py` folder. If you need to, you can [Customize the startup command](#).
- 6. Continuous deployment:** Set up continuous deployment from GitHub Actions, Bitbucket, or Azure Repos as described in the article [Continuous deployment to Azure App Service](#). Or, set up continuous deployment from Local Git as described in the article [Local Git deployment to Azure App Service](#).

7. Custom actions: To perform actions within the App Service container that hosts your app, such as Django database migrations, you can [connect to the container through SSH](#). For an example of running Django database migrations, see [Tutorial: Deploy a Django web app with PostgreSQL - generate database schema](#).

- When using continuous deployment, you can perform those actions using post-build commands as described earlier under [Customize build automation](#).

With these steps completed, you should be able to commit changes to your source repository and have those updates automatically deployed to App Service.

Production settings for Django apps

For a production environment like Azure App Service, Django apps should follow Django's [Deployment checklist](#).

The following table describes the production settings that are relevant to Azure. These settings are defined in the app's *setting.py* file.

[+] [Expand table](#)

Django setting	Instructions for Azure
<code>SECRET_KEY</code>	Store the value in an App Service setting as described on Access app settings as environment variables . You can alternatively store the value as a secret in Azure Key Vault .
<code>DEBUG</code>	Create a <code>DEBUG</code> setting on App Service with the value 0 (false), then load the value as an environment variable. In your development environment, create a <code>DEBUG</code> environment variable with the value 1 (true).
<code>ALLOWED_HOSTS</code>	In production, Django requires that you include the app's URL in the <code>ALLOWED_HOSTS</code> array of <i>settings.py</i> . You can retrieve this URL at runtime with the code <code>os.environ['WEBSITE_HOSTNAME']</code> . App Service automatically sets the <code>WEBSITE_HOSTNAME</code> environment variable to the app's URL.
<code>DATABASES</code>	Define settings in App Service for the database connection and load them as environment variables to populate the <code>DATABASES</code> dictionary. You can alternatively store the values (especially the username and password) as Azure Key Vault secrets .

Serve static files for Django apps

If your Django web app includes static front-end files, first follow the instructions on [managing static files](#) in the Django documentation.

For App Service, you then make the following modifications:

1. Consider using environment variables (for local development) and App Settings (when deploying to the cloud) to dynamically set the Django `STATIC_URL` and `STATIC_ROOT` variables. For example:

Python

```
STATIC_URL = os.environ.get("DJANGO_STATIC_URL", "/static/")
STATIC_ROOT = os.environ.get("DJANGO_STATIC_ROOT", "./static/")
```

`DJANGO_STATIC_URL` and `DJANGO_STATIC_ROOT` can be changed as necessary for your local and cloud environments. For example, if the build process for your static files places them in a folder named `django-static`, then you can set `DJANGO_STATIC_URL` to `/django-static/` to avoid using the default.

2. If you have a pre-build script that generates static files in a different folder, include that folder in the Django `STATICFILES_DIRS` variable so that Django's `collectstatic` process finds them. For example, if you run `yarn build` in your front-end folder, and `yarn` generates a `build/static` folder containing static files, then include that folder as follows:

Python

```
FRONTEND_DIR = "path-to-frontend-folder"
STATICFILES_DIRS = [os.path.join(FRONTEND_DIR, 'build', 'static')]
```

Here, `FRONTEND_DIR` is used to build a path to where a build tool like `yarn` is run. You can again use an environment variable and App Setting as desired.

3. Add `whitenoise` to your `requirements.txt` file. [WhiteNoise](#) (whitenoise.evans.io) is a Python package that makes it simple for a production Django app to serve its own static files. WhiteNoise specifically serves those files that are found in the folder specified by the Django `STATIC_ROOT` variable.
4. In your `settings.py` file, add the following line for WhiteNoise:

Python

```
STATICFILES_STORAGE =
```

```
('whitenoise.storage.CompressedManifestStaticFilesStorage')
```

5. Also modify the `MIDDLEWARE` and `INSTALLED_APPS` lists to include WhiteNoise:

Python

```
MIDDLEWARE = [
    'django.middleware.security.SecurityMiddleware',
    # Add whitenoise middleware after the security middleware
    'whitenoise.middleware.WhiteNoiseMiddleware',
    # Other values follow
]

INSTALLED_APPS = [
    "whitenoise.runserver_nostatic",
    # Other values follow
]
```

Serve static files for Flask apps

If your Flask web app includes static front-end files, first follow the instructions on [managing static files](#) in the Flask documentation. For an example of serving static files in a Flask application, see the [sample Flask application](#) on GitHub.

To serve static files directly from a route on your application, you can use the [send_from_directory](#) method:

Python

```
from flask import send_from_directory

@app.route('/reports/<path:path>')
def send_report(path):
    return send_from_directory('reports', path)
```

Container characteristics

When deployed to App Service, Python apps run within a Linux Docker container that's defined in the [App Service Python GitHub repository](#). You can find the image configurations inside the version-specific directories.

This container has the following characteristics:

- Apps are run using the [Gunicorn WSGI HTTP Server](#), using the extra arguments `-bind=0.0.0.0 --timeout 600`.
 - You can provide configuration settings for Gunicorn by [customizing the startup command](#).
 - To protect your web app from accidental or deliberate DDOS attacks, Gunicorn is run behind an Nginx reverse proxy as described in [Deploying Gunicorn](#).
- By default, the base container image includes only the Flask web framework, but the container supports other frameworks that are WSGI-compliant and compatible with Python 3.6+, such as Django.
- To install other packages, such as Django, create a [requirements.txt](#) file in the root of your project that specifies your direct dependencies. App Service then installs those dependencies automatically when you deploy your project.

The `requirements.txt` file *must* be in the project root for dependencies to be installed. Otherwise, the build process reports the error: "Could not find setup.py or requirements.txt; Not running pip install." If you encounter this error, check the location of your requirements file.

- App Service automatically defines an environment variable named `WEBSITE_HOSTNAME` with the web app's URL, such as `msdocs-hello-world.azurewebsites.net`. It also defines `WEBSITE_SITE_NAME` with the name of your app, such as `msdocs-hello-world`.
- npm and Node.js are installed in the container so you can run Node-based build tools, such as yarn.

Container startup process

During startup, the App Service on Linux container runs the following steps:

1. Use a [custom startup command](#), if one is provided.
2. Check for the existence of a [Django app](#), and launch Gunicorn for it if one is detected.
3. Check for the existence of a [Flask app](#), and launch Gunicorn for it if one is detected.
4. If no other app is found, start a default app that's built into the container.

The following sections provide extra details for each option.

Django app

For Django apps, App Service looks for a file named `wsgi.py` within your app code, and then runs Gunicorn using the following command:

Bash

```
# <module> is the name of the folder that contains wsgi.py
gunicorn --bind=0.0.0.0 --timeout 600 <module>.wsgi
```

If you want more specific control over the startup command, use a [custom startup command](#), replace `<module>` with the name of folder that contains `wsgi.py`, and add a `--chdir` argument if that module isn't in the project root. For example, if your `wsgi.py` is located under `knboard/backend/config` from your project root, use the arguments `--chdir knboard/backend config.wsgi`.

To enable production logging, add the `--access-logfile` and `--error-logfile` parameters as shown in the examples for [custom startup commands](#).

Flask app

For Flask, App Service looks for a file named `application.py` or `app.py` and starts Gunicorn as follows:

Bash

```
# If application.py
gunicorn --bind=0.0.0.0 --timeout 600 application:app

# If app.py
gunicorn --bind=0.0.0.0 --timeout 600 app:app
```

If your main app module is contained in a different file, use a different name for the app object. If you want to provide other arguments to Gunicorn, use a [custom startup command](#).

Default behavior

If the App Service doesn't find a custom command, a Django app, or a Flask app, then it runs a default read-only app, located in the `opt/defaultsite` folder and shown in the following image.

If you deployed code and still see the default app, see [Troubleshooting - App doesn't appear](#).

The screenshot shows the Microsoft Azure portal interface. At the top left is the Microsoft Azure logo. Below it, a large text box contains the message: "Your web app is running and waiting for your content". To the right of this text is a 3D-style illustration of a computer monitor displaying a globe icon, with three blue circles containing code snippets (less than/greater than symbols and curly braces) floating around it. Below the main message, a smaller text box says: "Your web app is live, but we don't have your content yet. If you've already deployed, it could take up to 5 minutes for your content to show up, so come back soon." Underneath this, there are two sections: "Built with Python" (with a Python logo icon) and two buttons: "Deployment center" and "Quickstart".

Customize startup command

You can control the container's startup behavior by providing either a custom startup command or multiple commands in a startup command file. A startup command file can use whatever name you choose, such as `startup.sh`, `startup.cmd`, `startup.txt`, and so on.

All commands must use relative paths to the project root folder.

To specify a startup command or command file:

- **Azure portal:** select the app's **Configuration** page, then select **General settings**. In the **Startup Command** field, place either the full text of your startup command or the name of your startup command file. Then select **Save** to apply the changes. See [Configure general settings](#) for Linux containers.
- **Azure CLI:** use the `az webapp config set` command with the `--startup-file` parameter to set the startup command or file:

Azure CLI

```
az webapp config set --resource-group <resource-group-name> --name <app-name> --startup-file "<custom-command>"
```

Replace `<custom-command>` with either the full text of your startup command or the name of your startup command file.

App Service ignores any errors that occur when processing a custom startup command or file, then continues its startup process by looking for Django and Flask apps. If you don't see the behavior you expect, check that your startup command or file is error-free, and that a startup command file is deployed to App Service along with your app code. You can also check the [diagnostic logs](#) for more information. Also check the app's [Diagnose and solve problems](#) page on the [Azure portal](#).

Example startup commands

- **Added Gunicorn arguments:** The following example adds the `--workers=4` argument to a Gunicorn command line for starting a Django app:

Bash

```
# <module-path> is the relative path to the folder that contains the module
# that contains wsgi.py; <module> is the name of the folder containing wsgi.py.
gunicorn --bind=0.0.0.0 --timeout 600 --workers=4 --chdir <module_path>
<module>.wsgi
```

For more information, see [Running Gunicorn](#). If you're using auto-scale rules to scale your web app up and down, you should also dynamically set the number of Gunicorn workers using the `NUM_CORES` environment variable in your startup command, for example: `--workers $((($NUM_CORES*2)+1))`. For more information on setting the recommended number of Gunicorn workers, see [the Gunicorn FAQ](#).

- **Enable production logging for Django:** Add the `--access-logfile '-'` and `--error-logfile '-'` arguments to the command line:

Bash

```
# '-' for the log files means stdout for --access-logfile and stderr for --error-logfile.
gunicorn --bind=0.0.0.0 --timeout 600 --workers=4 --chdir <module_path>
<module>.wsgi --access-logfile '-' --error-logfile '-'
```

These logs will appear in the [App Service log stream](#).

For more information, see [Gunicorn logging](#).

- **Custom Flask main module:** By default, App Service assumes that a Flask app's main module is *application.py* or *app.py*. If your main module uses a different name, then you must customize the startup command. For example, if you have a Flask app whose main module is *hello.py* and the Flask app object in that file is named `myapp`, then the command is as follows:

```
Bash
```

```
gunicorn --bind=0.0.0.0 --timeout 600 hello:myapp
```

If your main module is in a subfolder, such as `website`, specify that folder with the `--chdir` argument:

```
Bash
```

```
gunicorn --bind=0.0.0.0 --timeout 600 --chdir website hello:myapp
```

- **Use a non-Gunicorn server:** To use a different web server, such as [aiohttp](#), use the appropriate command as the startup command or in the startup command file:

```
Bash
```

```
python3.7 -m aiohttp.web -H localhost -P 8080 package.module:init_func
```

Access app settings as environment variables

App settings are values stored in the cloud specifically for your app, as described in [Configure app settings](#). These settings are available to your app code as environment variables and accessed using the standard [os.environ](#) pattern.

For example, if you've created an app setting called `DATABASE_SERVER`, the following code retrieves that setting's value:

```
Python
```

```
db_server = os.environ['DATABASE_SERVER']
```

Detect HTTPS session

In App Service, [TLS/SSL termination](#) happens at the network load balancers, so all HTTPS requests reach your app as unencrypted HTTP requests. If your app logic needs to check if the user requests are encrypted or not, inspect the `X-Forwarded-Proto` header.

Python

```
if 'X-Forwarded-Proto' in request.headers and request.headers['X-Forwarded-Proto'] == 'https':  
    # Do something when HTTPS is used
```

Popular web frameworks let you access the `X-Forwarded-*` information in your standard app pattern. For example, in Django you can use the [SECURE_PROXY_SSL_HEADER](#) to tell Django to use the `X-Forwarded-Proto` header.

Access diagnostic logs

You can access the console logs generated from inside the container.

First, turn on container logging by running the following command:

Azure CLI

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

Replace `<app-name>` and `<resource-group-name>` with the names appropriate for your web app.

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

Azure CLI

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

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

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

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

To access logs through the Azure portal, select **Monitoring > Log stream** on the left side menu for your app.

Access deployment logs

When you deploy your code, App Service performs the build process described earlier in the section [Customize build automation](#). Because the build runs in its own container, build logs are stored separately from the app's diagnostic logs.

Use the following steps to access the deployment logs:

1. On the Azure portal for your web app, select **Deployment > Deployment Center** on the left menu.
2. On the **Logs** tab, select the **Commit ID** for the most recent commit.
3. On the **Log details** page that appears, select the **Show Logs** link that appears next to "Running oryx build...".

Build issues such as incorrect dependencies in *requirements.txt* and errors in pre- or post-build scripts will appear in these logs. Errors also appear if your requirements file isn't named *requirements.txt* or doesn't appear in the root folder of your project.

Open SSH session in browser

To make open a direct SSH session with your container, your app should be running.

Paste the following URL into your browser and replace `<app-name>` with your app name:

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

If you're not yet authenticated, you're required to authenticate with your Azure subscription to connect. Once authenticated, you see an in-browser shell, where you can run commands inside your container.

```
root@9e933156516f:~# service --status-all
[ + ]  apache2
[ - ]  bootlogs
[ - ]  bootmisc.sh
[ - ]  checkfs.sh
[ - ]  checkroot-bootclean.sh
[ - ]  checkroot.sh
[ - ]  hostname.sh
[ ? ]  hwclock.sh
[ - ]  killprocs
[ - ]  motd
[ - ]  mountall-bootclean.sh
[ - ]  mountall.sh
[ - ]  mountdevsubfs.sh
[ - ]  mountkernfs.sh
[ - ]  mountnfs-bootclean.sh
[ - ]  mountnfs.sh
[ - ]  mysql
[ - ]  procps
[ - ]  rc.local
[ - ]  rmmnlogin
[ - ]  sendsigs
[ + ]  ssh
[ + ]  udev
[ ? ]  udev-finish
[ - ]  umountfs
[ - ]  umountnfs.sh
[ - ]  umountroot
[ - ]  urandom
root@9e933156516f:~#
```

ssh://root@...:2222 | SSH CONNECTION ESTABLISHED |

ⓘ Note

Any changes you make outside the `/home` directory are stored in the container itself and don't persist beyond an app restart.

To open a remote SSH session from your local machine, see [Open SSH session from remote shell](#).

When you're successfully connected to the SSH session, you should see the message "SSH CONNECTION ESTABLISHED" at the bottom of the window. If you see errors such as "SSH_CONNECTION_CLOSED" or a message that the container is restarting, an error might be preventing the app container from starting. See [Troubleshooting](#) for steps to investigate possible issues.

URL rewrites

When deploying Python applications on Azure App Service for Linux, you might need to handle URL rewrites within your application. This is particularly useful for ensuring specific URL patterns are redirected to the correct endpoints without relying on external web server configurations. For Flask applications, [URL processors](#) and custom middleware can be used to achieve this. In Django applications, the robust [URL dispatcher](#) allows for efficient management of URL rewrites.

Troubleshooting

In general, the first step in troubleshooting is to use App Service diagnostics:

1. In the Azure portal for your web app, select **Diagnose and solve problems** from the left menu.
2. Select **Availability and Performance**.
3. Examine the information in the **Application Logs**, **Container Crash**, and **Container Issues** options, where the most common issues will appear.

Next, examine both the [deployment logs](#) and the [app logs](#) for any error messages.

These logs often identify specific issues that can prevent app deployment or app startup. For example, the build can fail if your *requirements.txt* file has the wrong filename or isn't present in your project root folder.

The following sections provide guidance for specific issues.

- [App doesn't appear - default app shows](#)
- [App doesn't appear - "service unavailable" message](#)
- [Could not find setup.py or requirements.txt](#)
- [ModuleNotFoundError on startup](#)
- [Database is locked](#)
- [Passwords don't appear in SSH session when typed](#)
- [Commands in the SSH session appear to be cut off](#)
- [Static assets don't appear in a Django app](#)
- [Fatal SSL Connection is Required](#)

App doesn't appear

- You see the default app after deploying your own app code. The [default app](#) appears because you either haven't deployed your app code to App Service, or App Service failed to find your app code and ran the default app instead.
 - Restart the App Service, wait 15-20 seconds, and check the app again.
 - Use [SSH](#) to connect directly to the App Service container and verify that your files exist under *site/wwwroot*. If your files don't exist, use the following steps:
 1. Create an app setting named `SCM_DO_BUILD_DURING_DEPLOYMENT` with the value of 1, redeploy your code, wait a few minutes, then try to access the app again. For more information on creating app settings, see [Configure an App Service app in the Azure portal](#).

2. Review your deployment process, [check the deployment logs](#), correct any errors, and redeploy the app.
 - If your files exist, then App Service wasn't able to identify your specific startup file. Check that your app is structured as App Service expects for [Django](#) or [Flask](#), or use a [custom startup command](#).
 - You see the message "Service Unavailable" in the browser. The browser has timed out waiting for a response from App Service, which indicates that App Service started the Gunicorn server, but the app itself didn't start. This condition could indicate that the Gunicorn arguments are incorrect, or that there's an error in the app code.
 - Refresh the browser, especially if you're using the lowest pricing tiers in your App Service plan. The app might take longer to start up when you use free tiers, for example, and becomes responsive after you refresh the browser.
 - Check that your app is structured as App Service expects for [Django](#) or [Flask](#), or use a [custom startup command](#).
 - Examine the [app log stream](#) for any error messages. The logs will show any errors in the app code.

Could not find setup.py or requirements.txt

- The log stream shows "Could not find setup.py or requirements.txt; Not running pip install.": The Oryx build process failed to find your *requirements.txt* file.
 - Connect to the web app's container via [SSH](#) and verify that *requirements.txt* is named correctly and exists directly under *site/wwwroot*. If it doesn't exist, make sure the file exists in your repository and is included in your deployment. If it exists in a separate folder, move it to the root.

ModuleNotFoundError when app starts

If you see an error like `ModuleNotFoundError: No module named 'example'`, then Python couldn't find one or more of your modules when the application started. This error most often occurs if you deploy your virtual environment with your code. Virtual environments aren't portable, so a virtual environment shouldn't be deployed with your application code. Instead, let Oryx create a virtual environment and install your packages on the web app by creating an app setting, `SCM_DO_BUILD_DURING_DEPLOYMENT`, and setting it to `1`. This setting will force Oryx to install your packages whenever you deploy

to App Service. For more information, see [this article on virtual environment portability](#).

Database is locked

When attempting to run database migrations with a Django app, you might see "sqlite3.OperationalError: database is locked." The error indicates that your application is using a SQLite database, for which Django is configured by default, rather than using a cloud database such as Azure Database for PostgreSQL.

Check the `DATABASES` variable in the app's `settings.py` file to ensure that your app is using a cloud database instead of SQLite.

If you're encountering this error with the sample in [Tutorial: Deploy a Django web app with PostgreSQL](#), check that you completed the steps in [Verify connection settings](#).

Other issues

- **Passwords don't appear in the SSH session when typed:** For security reasons, the SSH session keeps your password hidden when you type. The characters are being recorded, however, so type your password as usual and select **Enter** when done.
- **Commands in the SSH session appear to be cut off:** The editor might not be word-wrapping commands, but they should still run correctly.
- **Static assets don't appear in a Django app:** Ensure that you've enabled the [WhiteNoise module](#).
- **You see the message, "Fatal SSL Connection is Required":** Check any usernames and passwords used to access resources (such as databases) from within the app.

Related content

- [Tutorial: Python app with PostgreSQL](#)
- [Tutorial: Deploy from private container repository](#)
- [App Service on Linux FAQ](#)
- [Environment variables and app settings reference](#)

Feedback

Was this page helpful?

 Yes

 No

Provide product feedback  | Get help at Microsoft Q&A

Deploy and configure a Tomcat, JBoss, or Java SE app in Azure App Service

Article • 01/28/2025

This article shows you the most common deployment and runtime configuration for Java apps in App Service. If you've never used Azure App Service, you should read through the [Java quickstart](#) first. General questions about using App Service that aren't specific to Java development are answered in the [App Service FAQ](#).

Azure App Service runs Java web applications on a fully managed service in three variants:

- Java SE - Can run an app deployed as a JAR package that contains an embedded server (such as Spring Boot, Dropwizard, Quarkus, or one with an embedded Tomcat or Jetty server).
- Tomcat - The built-in Tomcat server can run an app deployed as a WAR package.
- JBoss EAP - Supported for Linux apps in the Free, Premium v3, and Isolated v2 pricing tiers only. The built-in JBoss EAP server can run an app deployed as a WAR or EAR package.

ⓘ Note

For Spring applications, we recommend using Azure Spring Apps. However, you can still use Azure App Service as a destination. See [Java Workload Destination Guidance](#) for advice.

Show Java version

Windows

To show the current Java version, run the following command in the [Cloud Shell](#):

Azure CLI

```
az webapp config show --name <app-name> --resource-group <resource-group-name> --query "[javaVersion, javaContainer, javaContainerVersion]"
```

To show all supported Java versions, run the following command in the [Cloud Shell](#):

Azure CLI

```
az webapp list-runtimes --os windows | grep java
```

For more information on version support, see [App Service language runtime support policy](#).

Deploying your app

Build Tools

Maven

With the [Maven Plugin for Azure Web Apps](#), you can prepare your Maven Java project for Azure Web App easily with one command in your project root:

shell

```
mvn com.microsoft.azure:azure-webapp-maven-plugin:2.13.0:config
```

This command adds an `azure-webapp-maven-plugin` plugin and related configuration by prompting you to select an existing Azure Web App or create a new one. During configuration, it attempts to detect whether your application should be deployed to Java SE, Tomcat, or (Linux only) JBoss EAP. Then you can deploy your Java app to Azure using the following command:

shell

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

Here's a sample configuration in `pom.xml`:

XML

```
<plugin>
  <groupId>com.microsoft.azure</groupId>
  <artifactId>azure-webapp-maven-plugin</artifactId>
  <version>2.11.0</version>
  <configuration>
    <subscriptionId>111111-11111-11111111</subscriptionId>
    <resourceGroup>spring-boot-xxxxxxxxxx-rg</resourceGroup>
    <appName>spring-boot-xxxxxxxxxx</appName>
```

```
<pricingTier>B2</pricingTier>
<region>westus</region>
<runtime>
  <os>Linux</os>
  <webContainer>Java SE</webContainer>
  <javaVersion>Java 17</javaVersion>
</runtime>
<deployment>
  <resources>
    <resource>
      <type>jar</type>
      <directory>${project.basedir}/target</directory>
      <includes>
        <include>*.jar</include>
      </includes>
    </resource>
  </resources>
</deployment>
</configuration>
</plugin>
```

Gradle

1. Set up the [Gradle Plugin for Azure Web Apps](#) by adding the plugin to your

`build.gradle`:

```
groovy

plugins {
  id "com.microsoft.azure.azurewebapp" version "1.10.0"
}
```

2. Configure your web app details. The corresponding Azure resources are created if they don't exist. Here's a sample configuration, for details, refer to this [document](#).

```
groovy

azurewebapp {
  subscription = '<your subscription id>'
  resourceGroup = '<your resource group>'
  appName = '<your app name>'
  pricingTier = '<price tier like 'P1v2'>'
  region = '<region like 'westus'>'
  runtime {
    os = 'Linux'
    webContainer = 'Tomcat 10.0' // or 'Java SE' if you want to run
    an executable jar
    javaVersion = 'Java 17'
```

```
    }
    appSettings {
        <key> = <value>
    }
    auth {
        type = 'azure_cli' // support azure_cli, oauth2, device_code
        and service_principal
    }
}
```

3. Deploy with one command.

```
shell
gradle azureWebAppDeploy
```

IDEs

Azure provides seamless Java App Service development experience in popular Java IDEs, including:

- *VS Code*: [Java Web Apps with Visual Studio Code](#) ↗
- *IntelliJ IDEA*: [Create a Hello World web app for Azure App Service using IntelliJ](#)
- *Eclipse*: [Create a Hello World web app for Azure App Service using Eclipse](#)

Kudu API

To deploy jar files to Java SE, use the `/api/publish` endpoint of the Kudu site. For more information on this API, see [this documentation](#).

ⓘ Note

Your .jar application must be named `app.jar` for App Service to identify and run your application. The [Maven plugin](#) does this for you automatically during deployment. If you don't wish to rename your JAR to `app.jar`, you can upload a shell script with the command to run your .jar app. Paste the absolute path to this script in the [Startup File](#) textbox in the Configuration section of the portal. The startup script doesn't run from the directory into which it's placed. Therefore, always use absolute paths to reference files in your startup script (for example: `java -jar /home/myapp/myapp.jar`).

Don't deploy your .war or .jar using FTP. The FTP tool is designed to upload startup scripts, dependencies, or other runtime files. It's not the optimal choice for deploying web apps.

Rewrite or redirect URL

To rewrite or redirect URL, use one of the available URL rewriters, such as [UrlRewriteFilter](#).

Logging and debugging apps

Performance reports, traffic visualizations, and health checkups are available for each app through the Azure portal. For more information, see [Azure App Service diagnostics overview](#).

Stream diagnostic logs

Windows

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](#):

Azure CLI

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

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

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

Azure CLI

```
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.

① Note

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

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

For more information, see [Stream logs in Cloud Shell](#).

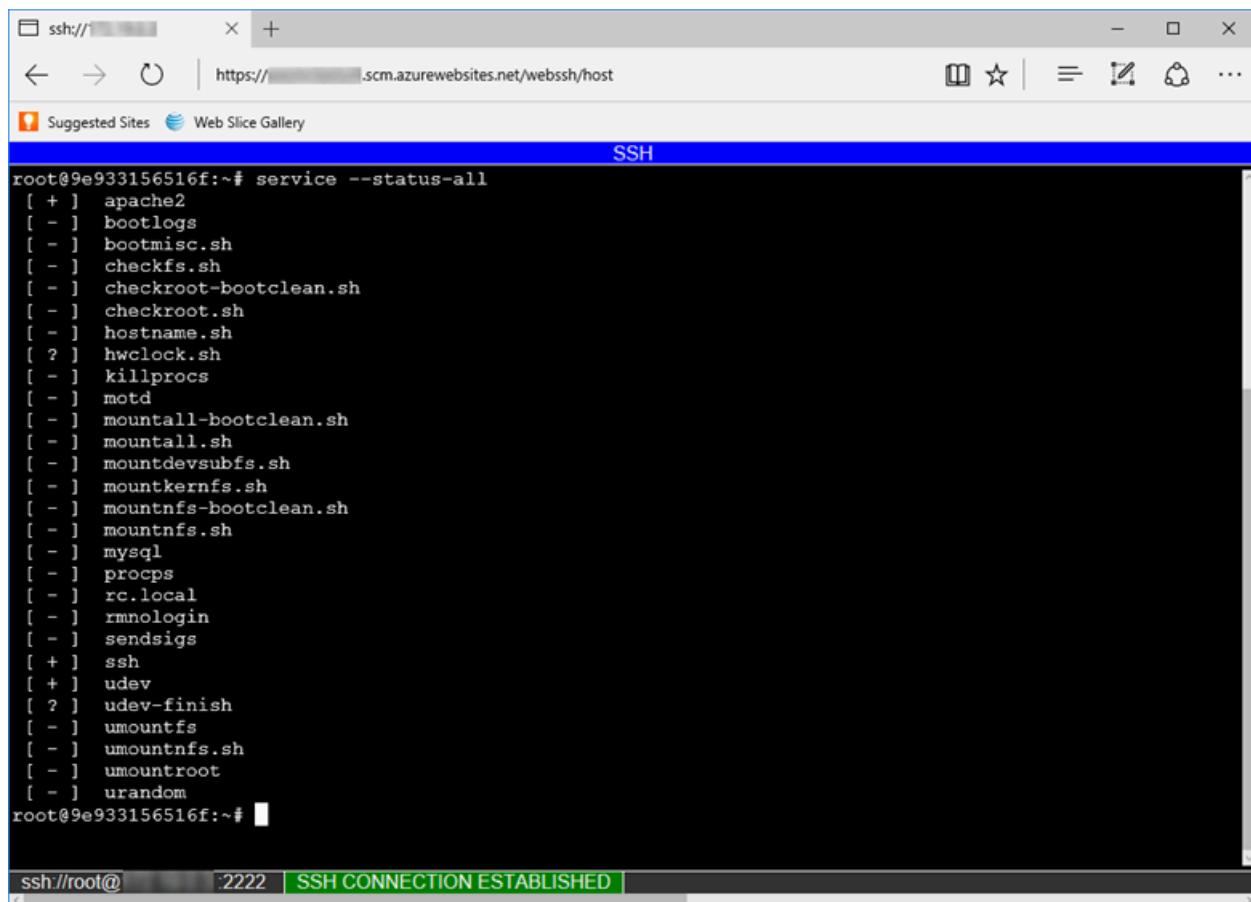
SSH console access in Linux

To make open a direct SSH session with your container, your app should be running.

Paste the following URL into your browser and replace `<app-name>` with your app name:

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

If you're not yet authenticated, you're required to authenticate with your Azure subscription to connect. Once authenticated, you see an in-browser shell, where you can run commands inside your container.



The screenshot shows a web browser window with the title bar "ssh://". The address bar contains the URL "https://<app-name>.scm.azurewebsites.net/webssh/host". Below the address bar, there are standard browser controls like back, forward, and search. A toolbar includes "Suggested Sites" and "Web Slice Gallery". The main content area is titled "SSH" and displays a terminal session. The terminal output shows the command "service --status-all" being run, listing various system services with their status (e.g., apache2 [+], bootlogs [-], etc.). At the bottom of the terminal window, the prompt "root@9e933156516f:~# " is visible. The status bar at the bottom of the browser window shows "ssh//root@ 2222 | SSH CONNECTION ESTABLISHED |".

```
root@9e933156516f:~# service --status-all
[ + ]  apache2
[ - ]  bootlogs
[ - ]  bootmisc.sh
[ - ]  checkfs.sh
[ - ]  checkroot-bootclean.sh
[ - ]  checkroot.sh
[ - ]  hostname.sh
[ ? ]  hwclock.sh
[ - ]  killprocs
[ - ]  motd
[ - ]  mountall-bootclean.sh
[ - ]  mountall.sh
[ - ]  mountdevsubfs.sh
[ - ]  mountkernfs.sh
[ - ]  mountnfs-bootclean.sh
[ - ]  mountnfs.sh
[ - ]  mysql
[ - ]  procps
[ - ]  rc.local
[ - ]  rmmnologin
[ - ]  sendsigs
[ + ]  ssh
[ + ]  udev
[ ? ]  udev-finish
[ - ]  umountfs
[ - ]  umountnfs.sh
[ - ]  umountroot
[ - ]  urandom
root@9e933156516f:~#
```

ⓘ Note

Any changes you make outside the `/home` directory are stored in the container itself and don't persist beyond an app restart.

To open a remote SSH session from your local machine, see [Open SSH session from remote shell](#).

Linux troubleshooting tools

The built-in Java images are based on the [Alpine Linux](#) operating system. Use the `apk` package manager to install any troubleshooting tools or commands.

Java Profiler

All Java runtimes on Azure App Service come with the JDK Flight Recorder for profiling Java workloads. You can use it to record JVM, system, and application events and troubleshoot problems in your applications.

To learn more about the Java Profiler, visit the [Azure Application Insights documentation](#).

Flight Recorder

All Java runtimes on App Service come with the Java Flight Recorder. You can use it to record JVM, system, and application events and troubleshoot problems in your Java applications.

Windows

Timed Recording

To take a timed recording, you need the PID (Process ID) of the Java application. To find the PID, open a browser to your web app's SCM site at `https://<your-site-name>.scm.azurewebsites.net/ProcessExplorer/`. This page shows the running processes in your web app. Find the process named "java" in the table and copy the corresponding PID (Process ID).

Next, open the **Debug Console** in the top toolbar of the SCM site and run the following command. Replace `<pid>` with the process ID you copied earlier. This command starts a 30-second profiler recording of your Java application and generates a file named `timed_recording_example.jfr` in the `C:\home` directory.

```
jccmd <pid> JFR.start name=TimedRecording settings=profile duration=30s  
filename="C:\home\timed_recording_example.JFR"
```

Analyze .jfr files

Use [FTPS](#) to download your JFR file to your local machine. To analyze the JFR file, download and install [Java Mission Control](#). For instructions on Java Mission Control, see the [JMC documentation](#) and the [installation instructions](#).

App logging

Windows

Enable [application logging](#) through the Azure portal or [Azure CLI](#) to configure App Service to write your application's standard console output and standard console error streams to the local filesystem or Azure Blob Storage. Logging to the local App Service filesystem instance is disabled 12 hours after you enable it. If you need longer retention, configure the application to write output to a Blob storage container.

Your Java and Tomcat app logs can be found in the `/home/LogFiles/Application/` directory.

If your application uses [Logback](#) or [Log4j](#) for tracing, you can forward these traces for review into Azure Application Insights using the logging framework configuration instructions in [Explore Java trace logs in Application Insights](#).

ⓘ Note

Due to known vulnerability [CVE-2021-44228](#), be sure to use Log4j version 2.16 or later.

Customization and tuning

Azure App Service supports out of the box tuning and customization through the Azure portal and CLI. Review the following articles for non-Java-specific web app configuration:

- [Configure app settings](#)
- [Set up a custom domain](#)
- [Configure TLS/SSL bindings](#)
- [Add a CDN](#)
- [Configure the Kudu site ↗](#)

Copy App Content Locally

Set the app setting `JAVA_COPY_ALL` to `true` to copy your app contents to the local worker from the shared file system. This setting helps address file-locking issues.

Set Java runtime options

To set allocated memory or other JVM runtime options, create an [app setting](#) named `JAVA_OPTS` with the options. App Service passes this setting as an environment variable to the Java runtime when it starts.

In the Azure portal, under **Application Settings** for the web app, create a new app setting named `JAVA_OPTS` that includes other settings, such as `-Xms512m -Xmx1204m`.

To configure the app setting from the Maven plugin, add setting/value tags in the Azure plugin section. The following example sets a specific minimum and maximum Java heap size:

```
XML

<appSettings>
  <property>
    <name>JAVA_OPTS</name>
    <value>-Xms1024m -Xmx1024m</value>
  </property>
</appSettings>
```

Developers running a single application with one deployment slot in their App Service plan can use the following options:

- B1 and S1 instances: `-Xms1024m -Xmx1024m`

- B2 and S2 instances: -Xms3072m -Xmx3072m
- B3 and S3 instances: -Xms6144m -Xmx6144m
- P1v2 instances: -Xms3072m -Xmx3072m
- P2v2 instances: -Xms6144m -Xmx6144m
- P3v2 instances: -Xms12800m -Xmx12800m
- P1v3 instances: -Xms6656m -Xmx6656m
- P2v3 instances: -Xms14848m -Xmx14848m
- P3v3 instances: -Xms30720m -Xmx30720m
- I1 instances: -Xms3072m -Xmx3072m
- I2 instances: -Xms6144m -Xmx6144m
- I3 instances: -Xms12800m -Xmx12800m
- I1v2 instances: -Xms6656m -Xmx6656m
- I2v2 instances: -Xms14848m -Xmx14848m
- I3v2 instances: -Xms30720m -Xmx30720m

When tuning application heap settings, review your App Service plan details and take into account multiple applications and deployment slot needs to find the optimal allocation of memory.

Turn on web sockets

Turn on support for web sockets in the Azure portal in the **Application settings** for the application. You need to restart the application for the setting to take effect.

Turn on web socket support using the Azure CLI with the following command:

```
Azure CLI
az webapp config set --name <app-name> --resource-group <resource-group-name> --web-sockets-enabled true
```

Then restart your application:

```
Azure CLI
az webapp stop --name <app-name> --resource-group <resource-group-name>
az webapp start --name <app-name> --resource-group <resource-group-name>
```

Set default character encoding

In the Azure portal, under **Application Settings** for the web app, create a new app setting named `JAVA_OPTS` with value `-Dfile.encoding=UTF-8`.

Alternatively, you can configure the app setting using the App Service Maven plugin. Add the setting name and value tags in the plugin configuration:

XML

```
<appSettings>
  <property>
    <name>JAVA_OPTS</name>
    <value>-Dfile.encoding=UTF-8</value>
  </property>
</appSettings>
```

robots933456 in logs

You may see the following message in the container logs:

```
2019-04-08T14:07:56.641002476Z "-" - - [08/Apr/2019:14:07:56 +0000] "GET
/robots933456.txt HTTP/1.1" 404 415 "-" "-"
```

You can safely ignore this message. `/robots933456.txt` is a dummy URL path that App Service uses to check if the container is capable of serving requests. A 404 response simply indicates that the path doesn't exist, but it lets App Service know that the container is healthy and ready to respond to requests.

Choosing a Java runtime version

App Service allows users to choose the major version of the JVM, such as Java 8 or Java 11, and the patch version, such as 1.8.0_232 or 11.0.5. You can also choose to have the patch version automatically updated as new minor versions become available. In most cases, production apps should use pinned patch JVM versions. This prevents unanticipated outages during a patch version autoupdate. All Java web apps use 64-bit JVMs, and it's not configurable.

If you choose to pin the minor version, you need to periodically update the JVM minor version on the app. To ensure that your application runs on the newer minor version, create a staging slot and increment the minor version on the staging slot. Once you

confirm the application runs correctly on the new minor version, you can swap the staging and production slots.

Next steps

Visit the [Azure for Java Developers](#) center to find Azure quickstarts, tutorials, and Java reference documentation.

- [App Service Linux FAQ](#)
 - [Environment variables and app settings reference](#)
-

Feedback

Was this page helpful?



[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

Configure data sources for a Tomcat, JBoss, or Java SE app in Azure App Service

Article • 11/11/2024

This article shows how to configure data sources in a Java SE, Tomcat, or JBoss app in App Service.

Azure App Service runs Java web applications on a fully managed service in three variants:

- Java SE - Can run an app deployed as a JAR package that contains an embedded server (such as Spring Boot, Dropwizard, Quarkus, or one with an embedded Tomcat or Jetty server).
- Tomcat - The built-in Tomcat server can run an app deployed as a WAR package.
- JBoss EAP - Supported for Linux apps in the Free, Premium v3, and Isolated v2 pricing tiers only. The built-in JBoss EAP server can run an app deployed as a WAR or EAR package.

ⓘ Note

For Spring applications, we recommend using Azure Spring Apps. However, you can still use Azure App Service as a destination. See [Java Workload Destination Guidance](#) for advice.

Configure the data source

To connect to data sources in Spring Boot applications, we suggest creating connection strings and injecting them into your *application.properties* file.

1. In the "Configuration" section of the App Service page, set a name for the string, paste your JDBC connection string into the value field, and set the type to "Custom". You can optionally set this connection string as slot setting.

This connection string is accessible to our application as an environment variable named `CUSTOMCONNSTR_<your-string-name>`. For example, `CUSTOMCONNSTR_exampledb`.

2. In your *application.properties* file, reference this connection string with the environment variable name. For our example, we would use the following code:

```
yml
```

```
app.datasource.url=${CUSTOMCONNSTR_exampledbs}
```

For more information, see the [Spring Boot documentation on data access](#) and [externalized configurations](#).

Next steps

Visit the [Azure for Java Developers center](#) to find Azure quickstarts, tutorials, and Java reference documentation.

- [App Service Linux FAQ](#)
- [Environment variables and app settings reference](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

Configure APM platforms for Tomcat, JBoss, or Java SE apps in Azure App Service

Article • 07/22/2024

This article shows how to connect Java applications deployed on Azure App Service with Azure Monitor Application Insights, NewRelic, and AppDynamics application performance monitoring (APM) platforms.

Azure App Service runs Java web applications on a fully managed service in three variants:

- Java SE - Can run an app deployed as a JAR package that contains an embedded server (such as Spring Boot, Dropwizard, Quarkus, or one with an embedded Tomcat or Jetty server).
- Tomcat - The built-in Tomcat server can run an app deployed as a WAR package.
- JBoss EAP - Supported for Linux apps in the Free, Premium v3, and Isolated v2 pricing tiers only. The built-in JBoss EAP server can run an app deployed as a WAR or EAR package.

ⓘ Note

For Spring applications, we recommend using Azure Spring Apps. However, you can still use Azure App Service as a destination. See [Java Workload Destination Guidance](#) for advice.

Configure Application Insights

Azure Monitor Application Insights is a cloud native application monitoring service that enables customers to observe failures, bottlenecks, and usage patterns to improve application performance and reduce mean time to resolution (MTTR). With a few clicks or CLI commands, you can enable monitoring for your Node.js or Java apps, autocollecting logs, metrics, and distributed traces, eliminating the need for including an SDK in your app. For more information about the available app settings for configuring the agent, see the [Application Insights documentation](#).

Azure portal

To enable Application Insights from the Azure portal, go to **Application Insights** on the left-side menu and select **Turn on Application Insights**. By default, a new application insights resource of the same name as your web app is used. You can choose to use an existing application insights resource, or change the name. Select **Apply** at the bottom.

Configure New Relic

Windows

1. Create a NewRelic account at NewRelic.com ↗
2. Download the Java agent from NewRelic. It has a file name similar to *newrelic-java-x.x.x.zip*.
3. Copy your license key, you need it to configure the agent later.
4. [SSH into your App Service instance](#) and create a new directory `/home/site/wwwroot/apm`.
5. Upload the unpacked NewRelic Java agent files into a directory under `/home/site/wwwroot/apm`. The files for your agent should be in `/home/site/wwwroot/apm/newrelic`.
6. Modify the YAML file at `/home/site/wwwroot/apm/newrelic/newrelic.yml` and replace the placeholder license value with your own license key.
7. In the Azure portal, browse to your application in App Service and create a new Application Setting.

Create an environment variable named `JAVA_OPTS` with the value -
`javaagent:/home/site/wwwroot/apm/newrelic/newrelic.jar`.

ⓘ Note

If you already have an environment variable for `JAVA_OPTS`, append the -
`javaagent:/...` option to the end of the current value.

Configure AppDynamics

Windows

1. Create an AppDynamics account at [AppDynamics.com](#)
2. Download the Java agent from the AppDynamics website. The file name is similar to *AppServerAgent-x.x.x.xxxxx.zip*
3. Use the [Kudu console](#) to create a new directory */home/site/wwwroot/apm*.
4. Upload the Java agent files into a directory under */home/site/wwwroot/apm*. The files for your agent should be in */home/site/wwwroot/apm/appdynamics*.
5. In the Azure portal, browse to your application in App Service and create a new Application Setting.

Create an environment variable named `JAVA_OPTS` with the value -
`javaagent:/home/site/wwwroot/apm/appdynamics/javaagent.jar -`
`Dappdynamics.agent.applicationName=<app-name>` where `<app-name>` is your
App Service name. If you already have an environment variable for `JAVA_OPTS`,
append the `-javaagent:/...` option to the end of the current value.

Configure Datadog

Windows

The configuration options are different depending on which Datadog site your organization is using. See the official [Datadog Integration for Azure Documentation](#)

Configure Dynatrace

Windows

Dynatrace provides an [Azure Native Dynatrace Service](#). To monitor Azure App Services using Dynatrace, see the official [Dynatrace for Azure documentation](#)

Next steps

Visit the [Azure for Java Developers](#) center to find Azure quickstarts, tutorials, and Java reference documentation.

- [App Service Linux FAQ](#)
 - [Environment variables and app settings reference](#)
-

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

Configure security for a Tomcat, JBoss, or Java SE app in Azure App Service

Article • 07/22/2024

This article shows how to configure Java-specific security settings in App Service. Java applications running in App Service have the same set of [security best practices](#) as other applications.

Azure App Service runs Java web applications on a fully managed service in three variants:

- Java SE - Can run an app deployed as a JAR package that contains an embedded server (such as Spring Boot, Dropwizard, Quarkus, or one with an embedded Tomcat or Jetty server).
- Tomcat - The built-in Tomcat server can run an app deployed as a WAR package.
- JBoss EAP - Supported for Linux apps in the Premium v3 and Isolated v2 pricing tiers only. The built-in JBoss EAP server can run an app deployed as a WAR or EAR package.

Note

For Spring applications, we recommend using Azure Spring Apps. However, you can still use Azure App Service as a destination. See [Java Workload Destination Guidance](#) for advice.

Authenticate users (Easy Auth)

Set up app authentication in the Azure portal with the **Authentication and Authorization** option. From there, you can enable authentication using Microsoft Entra ID or social sign-ins like Facebook, Google, or GitHub. Azure portal configuration only works when configuring a single authentication provider. For more information, see [Configure your App Service app to use Microsoft Entra sign-in](#) and the related articles for other identity providers. If you need to enable multiple sign-in providers, follow the instructions in [Customize sign-ins and sign-outs](#).

Spring Boot developers can use the [Microsoft Entra Spring Boot starter](#) to secure applications using familiar Spring Security annotations and APIs. Be sure to increase the maximum header size in your *application.properties* file. We suggest a value of `16384`.

Configure TLS/SSL

To upload an existing TLS/SSL certificate and bind it to your application's domain name, follow the instructions in [Secure a custom DNS name with an TLS/SSL binding in Azure App Service](#). You can also configure the app to enforce TLS/SSL.

Use KeyVault References

Azure [KeyVault](#) provides centralized secret management with access policies and audit history. You can store secrets (such as passwords or connection strings) in KeyVault and access these secrets in your application through environment variables.

First, follow the instructions for [granting your app access to a key vault](#) and [making a KeyVault reference to your secret in an Application Setting](#). You can validate that the reference resolves to the secret by printing the environment variable while remotely accessing the App Service terminal.

For Spring configuration files, see this documentation on [externalized configurations](#).

To inject these secrets in your Spring configuration file, use environment variable injection syntax (`${MY_ENV_VAR}`).

Use the Java key store in Linux

By default, any public or private certificates [uploaded to App Service Linux](#) are loaded into the respective Java key stores as the container starts. After uploading your certificate, you'll need to restart your App Service for it to be loaded into the Java key store. Public certificates are loaded into the key store at

`$JRE_HOME/lib/security/cacerts`, and private certificates are stored in
`$JRE_HOME/lib/security/client.jks`.

More configuration might be necessary for encrypting your JDBC connection with certificates in the Java key store. Refer to the documentation for your chosen JDBC driver.

- [PostgreSQL](#)
- [SQL Server](#)
- [MongoDB](#)
- [Cassandra](#)

Initialize the Java key store in Linux

To initialize the `import java.security.KeyStore` object, load the keystore file with the password. The default password for both key stores is `changeit`.

Java

```
KeyStore keyStore = KeyStore.getInstance("jks");
keyStore.load(
    new FileInputStream(System.getenv("JRE_HOME")+"/lib/security/cacerts"),
    "changeit".toCharArray());

KeyStore keyStore = KeyStore.getInstance("pkcs12");
keyStore.load(
    new
FileInputStream(System.getenv("JRE_HOME")+"/lib/security/client.jks"),
    "changeit".toCharArray());
```

Manually load the key store in Linux

You can load certificates manually to the key store. Create an app setting, `SKIP_JAVA_KEYSTORE_LOAD`, with a value of `1` to disable App Service from loading the certificates into the key store automatically. All public certificates uploaded to App Service via the Azure portal are stored under `/var/ssl/certs/`. Private certificates are stored under `/var/ssl/private/`.

You can interact or debug the Java Key Tool by [opening an SSH connection](#) to your App Service and running the command `keytool`. See the [Key Tool documentation](#) for a list of commands. For more information on the KeyStore API, see [the official documentation](#).

Next steps

Visit the [Azure for Java Developers](#) center to find Azure quickstarts, tutorials, and Java reference documentation.

- [App Service Linux FAQ](#)
- [Environment variables and app settings reference](#)

Feedback

Was this page helpful?

 Yes

 No

Deploy files to App Service

Article • 05/31/2024

ⓘ Note

Beginning June 1, 2024, all newly created App Service apps will have the option to create a unique default hostname with a naming convention of `<app-name> - <random-hash>`.
`<region>.azurewebsites.net`. The names of existing apps will not change.

Example: myapp-ds27dh7271ah175.westus-01.azurewebsites.net

For more information, refer to [Unique Default Hostname for App Service Resource](#).

This article shows you how to deploy your code as a ZIP, WAR, JAR, or EAR package to [Azure App Service](#). It also shows how to deploy individual files to App Service, separate from your application package.

Prerequisites

To complete the steps in this article, [create an App Service app](#), or use an app that you created for another tutorial.

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

Create a project ZIP package

ⓘ Important

When creating the ZIP package for deployment, don't include the root directory, but only the files and directories in it. If you download a GitHub repository as a ZIP file, you cannot deploy that file as-is to App Service. GitHub adds additional nested directories at the top level, which do not work with App Service.

In a local terminal window, navigate to the root directory of your app project.

This directory should contain the entry file to your web app, such as `index.html`, `index.php`, and `app.js`. It can also contain package management files like `project.json`, `composer.json`, `package.json`, `bower.json`, and `requirements.txt`.

Unless you want App Service to run deployment automation for you, run all the build tasks (for example, `npm`, `bower`, `gulp`, `composer`, and `pip`) and make sure that you have all the files you need to run the app. This step is required if you want to [run your package directly](#).

Create a ZIP archive of everything in your project. For `dotnet` projects, this is everything in the output directory of the `dotnet publish` command (excluding the output directory itself). For example, the following command in your terminal to create a ZIP package of the contents of the current directory:

```
# Bash  
zip -r <file-name>.zip .  
  
# PowerShell  
Compress-Archive -Path * -DestinationPath <file-name>.zip
```

Deploy a ZIP package

When you deploy a ZIP package, App Service unpacks its contents in the default path for your app (`D:\home\site\wwwroot` for Windows, `/home/site/wwwroot` for Linux).

This ZIP package deployment uses the same Kudu service that powers continuous integration-based deployments. Kudu supports the following functionality for ZIP package deployment:

- Deletion of files left over from a previous deployment.
- Option to turn on the default build process, which includes package restore.
- Deployment customization, including running deployment scripts.
- Deployment logs.
- A package size limit of 2048 MB.

⚠ Note

Files in the ZIP package are copied only if their timestamps don't match what is already deployed.

With zip deploy UI in Kudu

In the browser, navigate to `https://<app_name>.scm.azurewebsites.net/ZipDeployUI` (see [note at top](#)).

Upload the ZIP package you created in [Create a project ZIP package](#) by dragging it to the file explorer area on the web page.

When deployment is in progress, an icon in the top right corner shows you the progress in percentage. The page also shows verbose messages for the operation below the explorer area. When deployment completes, the last message should say `Deployment successful`.

The above endpoint doesn't work for Linux App Services at this time. Consider using FTP or the [ZIP deploy API](#) instead.

Without zip deploy UI in Kudu

Azure CLI

Deploy a ZIP package to your web app by using the [az webapp deploy](#) command. The CLI command uses the [Kudu publish API](#) to deploy the files and can be fully customized.

The following example pushes a ZIP package to your site. Specify the path to your local ZIP package for `--src-path`.

Azure CLI

```
az webapp deploy --resource-group <group-name> --name <app-name> --src-path <zip-package-path>
```

This command restarts the app after deploying the ZIP package.

Enable build automation for zip deploy

By default, the deployment engine assumes that a ZIP package is ready to run as-is and doesn't run any build automation. To enable the same build automation as in a [Git deployment](#), set the `SCM_DO_BUILD_DURING_DEPLOYMENT` app setting by running the following command in the [Cloud Shell](#):

Azure CLI

```
az webapp config appsettings set --resource-group <group-name> --name <app-name> --settings SCM_DO_BUILD_DURING_DEPLOYMENT=true
```

For more information, see [Kudu documentation](#).

Deploy WAR/JAR/EAR packages

You can deploy your [WAR](#), [JAR](#), or [EAR](#) package to App Service to run your Java web app using the Azure CLI, PowerShell, or the Kudu publish API.

The deployment process shown here puts the package on the app's content share with the right naming convention and directory structure (see [Kudu publish API reference](#)), and it's the recommended approach. If you deploy WAR/JAR/EAR packages using [FTP](#) or WebDeploy instead, you might see unknown failures due to mistakes in the naming or structure.

Azure CLI

Deploy a WAR package to Tomcat or JBoss EAP by using the `az webapp deploy` command. Specify the path to your local Java package for `--src-path`.

Azure CLI

```
az webapp deploy --resource-group <group-name> --name <app-name> --src-path ./<package-name>.war
```

The CLI command uses the [Kudu publish API](#) to deploy the package and can be fully customized.

Deploy individual files

Azure CLI

Deploy a startup script, library, and static file to your web app by using the [az webapp deploy](#) command with the `--type` parameter.

If you deploy a startup script this way, App Service automatically uses your script to start your app.

The CLI command uses the [Kudu publish API](#) to deploy the files and can be fully customized.

Deploy a startup script

Azure CLI

```
az webapp deploy --resource-group <group-name> --name <app-name> --src-path  
scripts/startup.sh --type=startup
```

Deploy a library file

Azure CLI

```
az webapp deploy --resource-group <group-name> --name <app-name> --src-path driver.jar  
--type=lib
```

Deploy a static file

Azure CLI

```
az webapp deploy --resource-group <group-name> --name <app-name> --src-path  
config.json --type=static
```

Deploy to network-secured apps

Depending on your web app's networking configuration, direct access to the app from your development environment might be blocked (see [Deploying to Network-secured sites](#) and [Deploying to Network-secured sites, Part 2](#)). Instead of pushing the package or file to the web app directly, you can publish it to a storage system accessible from the web app and trigger the app to pull the ZIP from the storage location.

The remote URL can be any publicly accessible location, but it's best to use a blob storage container with a SAS key to protect it.

Azure CLI

Use the `az webapp deploy` command like you would in the other sections, but use `--src-url` instead of `--src-path`. The following example uses the `--src-url` parameter to specify the URL of a ZIP file hosted in an Azure Storage account.

Azure CLI

```
az webapp deploy --resource-group <group-name> --name <app-name> --src-url  
"https://storagesample.blob.core.windows.net/sample-container/myapp.zip?sv=2021-10-  
01&sb&sig=s1k22f3UrS823n4kSh8Skjpa7Naj4CG3" --type zip
```

Kudu publish API reference

The `publish` Kudu API allows you to specify the same parameters from the CLI command as URL query parameters. To authenticate with the Kudu REST API, it's best to use token authentication, but you can also use basic authentication with your app's [deployment credentials](#).

The following table shows the available query parameters, their allowed values, and descriptions.

[+] Expand table

Key	Allowed values	Description	Required	Type
type	war jar ear lib startup static zip	<p>The type of the artifact being deployed, this sets the default target path and informs the web app how the deployment should be handled.</p> <ul style="list-style-type: none">- <code>type=zip</code>: Deploy a ZIP package by unzipping the content to <code>/home/site/wwwroot</code>. <code>target-path</code> parameter is optional.- <code>type=war</code>: Deploy a WAR package. By default, the WAR package is deployed to <code>/home/site/wwwroot/app.war</code>. The target path can be specified with <code>target-path</code>.- <code>type=jar</code>: Deploy a JAR package to <code>/home/site/wwwroot/app.jar</code>. The <code>target-path</code> parameter is ignored- <code>type=ear</code>: Deploy an EAR package to <code>/home/site/wwwroot/app.ear</code>. The <code>target-path</code> parameter is ignored- <code>type=lib</code>: Deploy a JAR library file. By default, the file is deployed to <code>/home/site/libs</code>. The target path can be specified with <code>target-path</code>.- <code>type=static</code>: Deploy a static file (such as a script). By default, the file is deployed to <code>/home/site/wwwroot</code>.- <code>type=startup</code>: Deploy a script that App Service automatically uses as the startup script for your app. By default, the script is deployed to <code>D:\home\site\scripts\<name-of-source></code> for Windows and <code>home/site/wwwroot/startup.sh</code> for Linux. The target path can be specified with <code>target-path</code>.	Yes	String

Key	Allowed values	Description	Required	Type
restart	true false	By default, the API restarts the app following the deployment operation (<code>restart=true</code>). To deploy multiple artifacts, prevent restarts on the all but the final deployment by setting <code>restart=false</code> .	No	Boolean
clean	true false	Specifies whether to clean (delete) the target deployment before deploying the artifact there.	No	Boolean
ignorestack	true false	The publish API uses the <code>WEBSITE_STACK</code> environment variable to choose safe defaults depending on your site's language stack. Setting this parameter to <code>false</code> disables any language-specific defaults.	No	Boolean
target-path	An absolute path	The absolute path to deploy the artifact to. For example, <code>"/home/site/deployments/tools/driver.jar"</code> , <code>"/home/site/scripts/helper.sh"</code> .	No	String

Next steps

For more advanced deployment scenarios, try [deploying to Azure with Git](#). Git-based deployment to Azure enables version control, package restore, MSBuild, and more.

More resources

- [Kudu: Deploying from a zip file](#)
- [Environment variables and app settings reference](#)

Deploy your app to Azure App Service using FTP/S

Article • 02/29/2024

This article shows you how to use FTP or FTPS to deploy your web app, mobile app backend, or API app to [Azure App Service](#).

The FTP/S endpoint for your app is already active. No configuration is necessary to enable FTP/S deployment.

ⓘ Note

When [FTP basic authentication is disabled](#), FTP/S deployment doesn't work, and you can't view or configure FTP credentials in the app's Deployment Center.

Get deployment credentials

1. Follow the instructions at [Configure deployment credentials for Azure App Service](#) to copy the application-scope credentials or set the user-scope credentials. You can connect to the FTP/S endpoint of your app using either credentials.
2. Craft the FTP username in the following format, depending on your choice of credential scope:

⋮ Expand table

Application-scope	User-scope
<code><app-name>\\$<app-name></code>	<code><app-name>\<deployment-user></code>

In App Service, the FTP/S endpoint is shared among apps. Because the user-scope credentials aren't linked to a specific resource, you need to prepend the user-scope username with the app name as shown above.

Get FTP/S endpoint

Azure portal

In the same management page for your app where you copied the deployment credentials (**Deployment Center > FTP Credentials**), copy the **FTPS endpoint**.

Deploy files to Azure

1. From your FTP client (for example, [Visual Studio](#), [Cyberduck](#), or [WinSCP](#)), use the connection information you gathered to connect to your app.
2. Copy your files and their respective directory structure to the [/site/wwwroot directory](#) in Azure (or the `/site/wwwroot/App_Data/Jobs/` directory for WebJobs).
3. Browse to your app's URL to verify the app is running properly.

Note

Unlike [Git-based deployments](#) and [Zip deployment](#), FTP deployment doesn't support build automation, such as:

- dependency restores (such as NuGet, NPM, PIP, and Composer automations)
- compilation of .NET binaries
- generation of web.config (here is a [Node.js example](#))

Generate these necessary files manually on your local machine, and then deploy them together with your app.

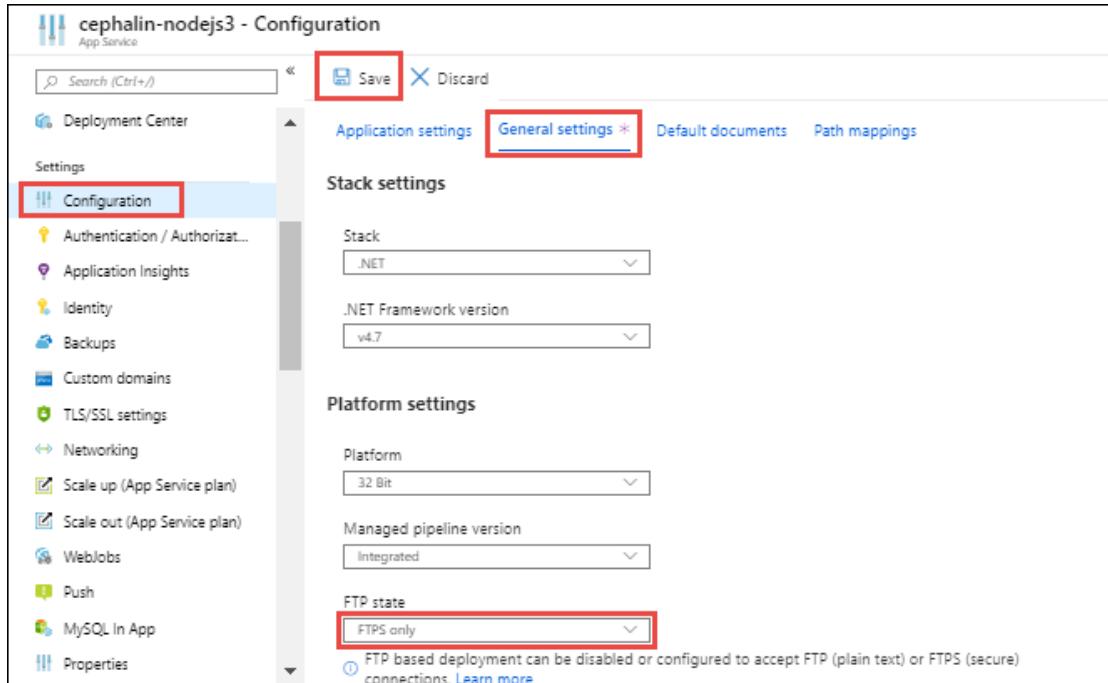
Enforce FTPS

For enhanced security, you should allow FTP over TLS/SSL only. You can also disable both FTP and FTPS if you don't use FTP deployment.

Azure portal

1. In your app's resource page in [Azure portal](#), select **Configuration > General settings** from the left navigation.
2. To disable unencrypted FTP, select **FTPS Only** in **FTP state**. To disable both FTP and FTPS entirely, select **Disabled**. When finished, select **Save**. If using **FTPS Only**, you must enforce TLS 1.2 or higher by navigating to the **TLS/SSL**

settings page of your web app. TLS 1.0 and 1.1 aren't supported with **FTPS Only**.



What happens to my app during deployment?

All the officially supported deployment methods make changes to the files in the `/home/site/wwwroot` folder of your app. These files are used to run your app. So the deployment can fail because of locked files. The app might also behave unpredictably during deployment because the files aren't all updated at the same time. This behavior is undesirable for a customer-facing app. There are a few ways to avoid these issues:

- Run your app directly from the [ZIP package](#), without unpacking it.
- Stop your app or enable offline mode for it during deployment. For more information, see [Deal with locked files during deployment](#).
- Deploy to a [staging slot](#) with [auto swap](#) turned on.

Troubleshoot FTP deployment

- How can I troubleshoot FTP deployment?
- I'm not able to FTP and publish my code. How can I resolve the issue?
- How can I connect to FTP in Azure App Service via passive mode?
- Why is my connection failing when attempting to connect over FTPS using explicit encryption?
- How can I determine the method that was used to deploy my Azure App Service?

How can I troubleshoot FTP deployment?

The first step for troubleshooting FTP deployment is isolating a deployment issue from a runtime application issue.

A deployment issue typically results in no files or wrong files deployed to your app. You can troubleshoot by investigating your FTP deployment or selecting an alternate deployment path (such as source control).

A runtime application issue typically results in the right set of files deployed to your app but incorrect app behavior. You can troubleshoot by focusing on code behavior at runtime and investigating specific failure paths.

To determine a deployment or runtime issue, see [Deployment vs. runtime issues](#).

I'm not able to FTP and publish my code. How can I resolve the issue?

Check that you entered the correct [hostname](#) and [credentials](#). Check also that the following FTP ports on your machine aren't blocked by a firewall:

- FTP control connection port: 21, 990
- FTP data connection port: 989, 10001-10300

How can I connect to FTP in Azure App Service via passive mode?

Azure App Service supports connecting via both Active and Passive mode. Passive mode is preferred because your deployment machines are usually behind a firewall (in the operating system or as part of a home or business network). See an [example from the WinSCP documentation](#).

Why is my connection failing when attempting to connect over FTPS using explicit encryption?

FTPS allows establishing the TLS secure connection in either an Explicit or Implicit way.

- If you connect with Implicit encryption, the connection is established via port 990.
- If you connect with Explicit encryption, the connection is established via port 21.

The URL format you use can affect your connection success, and it also depends on the client application you use. The portal shows the URL as `ftps://`, but note:

- If the URL you connect with starts with `ftp://`, the connection is implied to be on port 21.
- If it starts with `ftps://`, the connection is implied to be Implicit and on port 990.

Make sure not to mix both, such as attempting to connect to `ftps://` and using port 21, as it will fail to connect, even if you wish to do Explicit encryption. This is due to an Explicit connection starting as a plain FTP connection before the AUTH method.

How can I determine the method that was used to deploy my Azure App Service?

You can find out how an app was deployed by checking the application settings. If the app was deployed using an external package URL, you should see the `WEBSITE_RUN_FROM_PACKAGE` setting in the application settings with a URL value. Or if it was deployed using zip deploy, you should see the `WEBSITE_RUN_FROM_PACKAGE` setting with a value of `1`. If the app was deployed using Azure DevOps, you should see the deployment history in the Azure DevOps portal. If Azure Functions Core Tools is used, you should see the deployment history in the Azure portal.

More resources

- [Local Git deployment to Azure App Service](#)
- [Azure App Service Deployment Credentials](#)
- [Sample: Create a web app and deploy files with FTP \(Azure CLI\).](#)
- [Sample: Upload files to a web app using FTP \(PowerShell\).](#)

Continuous deployment to Azure App Service

Article • 05/31/2024

ⓘ Note

Beginning June 1, 2024, all newly created App Service apps will have the option to create a unique default hostname with a naming convention of `<app-name>-<random-hash>. <region>.azurewebsites.net`. The names of existing apps will not change.

Example: myapp-ds27dh7271aah175.westus-01.azurewebsites.net

For more information, refer to [Unique Default Hostname for App Service Resource ↗](#).

Azure App Service enables continuous deployment from [GitHub ↗](#), [Bitbucket ↗](#), and [Azure Repos](#) repositories by pulling in the latest updates.

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.

ⓘ Expand table

Runtime	Root directory files
ASP.NET (Windows only)	<code>*.sln</code> , <code>*.csproj</code> , or <code>default.aspx</code>
ASP.NET Core	<code>*.sln</code> or <code>*.csproj</code>
PHP	<code>index.php</code>
Ruby (Linux only)	<code>Gemfile</code>
Node.js	<code>server.js</code> , <code>app.js</code> , or <code>package.json</code> with a start script
Python	<code>*.py</code> , <code>requirements.txt</code> , or <code>runtime.txt</code>

Runtime	Root directory files
HTML	<code>default.htm</code> , <code>default.html</code> , <code>default.asp</code> , <code>index.htm</code> , <code>index.html</code> , or <code>iisstart.htm</code>
WebJobs	<code><job_name>/run.<extension></code> under <code>App_Data/jobs/continuous</code> for continuous WebJobs, or <code>App_Data/jobs/triggered</code> for triggered WebJobs. For more information, see Kudu WebJobs documentation .
Functions	See Continuous deployment for Azure Functions .

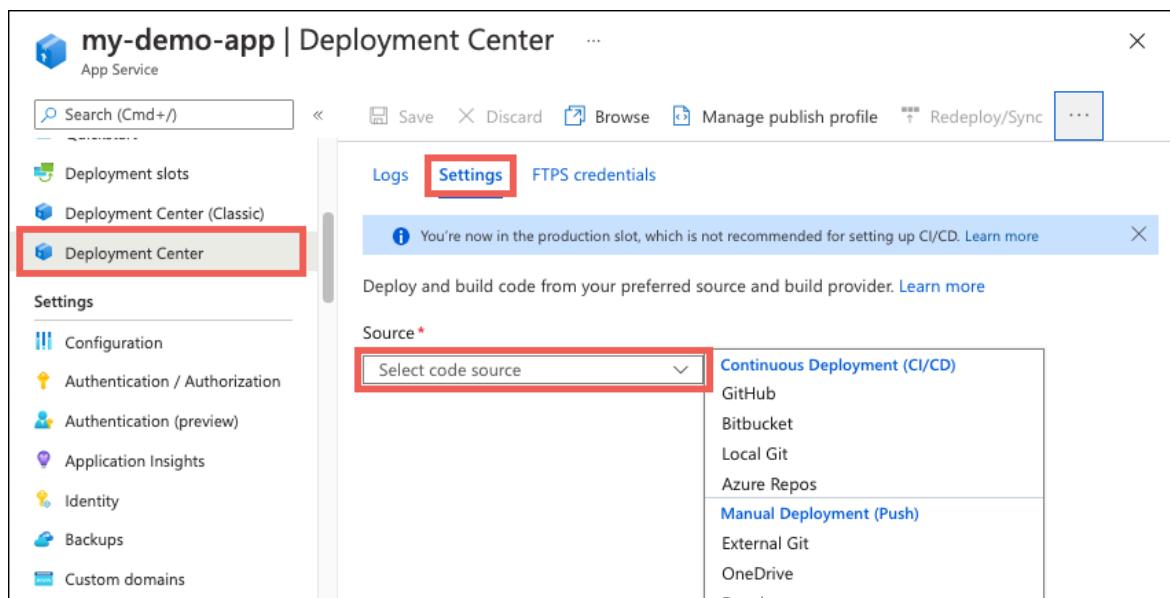
To customize your deployment, include a `.deployment` file in the repository root. For more information, see [Customize deployments](#) and [Custom deployment script](#).

ⓘ Note

If you use Visual Studio, let [Visual Studio create a repository for you](#). Your project will immediately be ready for deployment via Git.

Configure the deployment source

1. In the [Azure portal](#), go to the management page for your App Service app.
2. In the left pane, select **Deployment Center**. Then select **Settings**.
3. In the **Source** box, select one of the CI/CD options:



Select the tab that corresponds to your build provider to continue.



4. [GitHub Actions](#) is the default build provider. To change the provider, select **Change provider > App Service Build Service > OK**.
5. If you're deploying from GitHub for the first time, select **Authorize** and follow the authorization prompts. If you want to deploy from a different user's repository, select **Change Account**.
6. After you authorize your Azure account with GitHub, select the **Organization**, **Repository**, and **Branch** you want.
If you can't find an organization or repository, you might need to enable more permissions on GitHub. For more information, see [Managing access to your organization's repositories](#).
7. Under **Authentication type**, select **User-assigned identity** for better security. For more information, see [frequently asked questions](#).

 **Note**

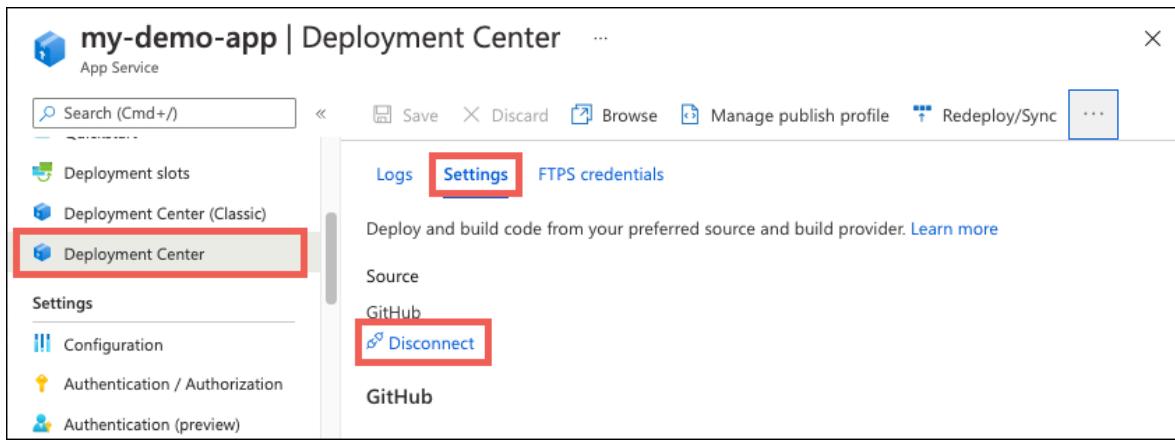
If your Azure account has the [required permissions](#) for the **User-assigned identity** option, Azure creates a [user-assigned managed identity](#) for you. If you don't, work with your Azure administrator to create an [identity with the required role on your app](#), then select it here in the dropdown.

8. (Optional) To see the file before saving your changes, select **Preview file**. App Service selects a workflow template based on the [language stack setting](#) of your app and commits it into your selected GitHub repository.
9. Select **Save**.

New commits in the selected repository and branch now deploy continuously into your App Service app. You can track the commits and deployments on the **Logs** tab.

Disable continuous deployment

1. In the [Azure portal](#), go to the management page for your App Service app.
2. In the left pane, select **Deployment Center**. Then select **Settings > Disconnect**:



3. By default, the GitHub Actions workflow file is preserved in your repository, but it continues to trigger deployment to your app. To delete the file from your repository, select **Delete workflow file**.
4. Select **OK**.

What are the build providers?

Depending on your deployment source in the Deployment Center, you might see a few options to select for build providers. Build providers help you build a CI/CD solution with Azure App Service by automating build, test, and deployment.

You're not limited to the build provider options found in the Deployment Center, but App Service lets you set them up quickly and offers some integrated deployment logging experience.

GitHub Actions

The GitHub Actions build provider is available only for [GitHub deployment](#). When configured from the app's Deployment Center, it completes these actions to set up CI/CD:

- Deposits a GitHub Actions workflow file into your GitHub repository to handle build and deploy tasks to App Service.
- For basic authentication, adds the publish profile for your app as a GitHub secret. The workflow file uses this secret to authenticate with App Service.
- For user-assigned identity, see [What does the user-assigned identity option do for GitHub Actions?](#)
- Captures information from the [workflow run logs](#) and displays it on the **Logs** tab in the Deployment Center.

You can customize the GitHub Actions build provider in these ways:

- Customize the workflow file after it's generated in your GitHub repository. For more information, see [Workflow syntax for GitHub Actions](#). Just make sure that the workflow deploys to App Service with the [azure/webapps-deploy](#) action.
- If the selected branch is protected, you can still preview the workflow file without saving the configuration and then manually add it into your repository. This method doesn't give you log integration with the Azure portal.
- Instead of using basic authentication or a user-assigned identity, you can also deploy by using a [service principal](#) in Microsoft Entra ID. This can't be configured in the portal.

What happens to my app during deployment?

All the officially supported deployment methods make changes to the files in the `/home/site/wwwroot` folder of your app. These files are used to run your app. So the deployment can fail because of locked files. The app might also behave unpredictably during deployment because the files aren't all updated at the same time. This behavior is undesirable for a customer-facing app. There are a few ways to avoid these issues:

- Run your app directly from the ZIP package, without unpacking it.
- Stop your app or enable offline mode for it during deployment. For more information, see [Deal with locked files during deployment](#).
- Deploy to a [staging slot](#) with [auto swap](#) turned on.

Frequently asked questions

- Does the GitHub Actions build provider work with basic authentication if basic authentication is disabled?
- What does the user-assigned identity option do for GitHub Actions?
- Why do I see the error, "This identity does not have write permissions on this app. Please select a different identity, or work with your admin to grant the Website Contributor role to your identity on this app"?
- Why do I see the error, "This identity does not have write permissions on this app. Please select a different identity, or work with your admin to grant the Website Contributor role to your identity on this app"?

Does the GitHub Actions build provider work with basic authentication if basic authentication is disabled?

No. Try using GitHub Actions with the **user-assigned identity** option.

For more information, see [Deployment without basic authentication](#).

What does the user-assigned identity option do for GitHub Actions?

When you select **user-assigned identity** under the **GitHub Actions** source, App Service configures all the necessary resources in Azure and in GitHub to enable the recommended OpenID Connect authentication with GitHub Actions.

Specifically, App Service does the following operations:

- Creates a federated credential between a user-assigned managed identity in Azure and your selected repository and branch in GitHub.
- Creates the secrets `AZURE_CLIENT_ID`, `AZURE_TENANT_ID`, and `AZURE_SUBSCRIPTION_ID` from the federated credential in your selected GitHub repository.
- Assigns the identity to your app.

In a GitHub Actions workflow in your GitHub repository, you can then use the [Azure/login](#) action to authenticate with your app by using OpenID Connect. For examples, see [Add the workflow file to your GitHub repository](#).

If your Azure account has the [required permissions](#), App Service creates a user-assigned managed identity and configures it for you. This identity isn't shown in the **Identities** page of your app. If your Azure account doesn't have the required permissions, you must select an [existing identity with the required role](#).

Why do I see the error, "You do not have sufficient permissions on this app to assign role-based access to a managed identity and configure federated credentials"?

The message indicates that your Azure account doesn't have the required permissions to create a user-assigned managed identity for the GitHub Actions. The required permissions (scoped to your app) are:

- `Microsoft.Authorization/roleAssignments/write`
- `Microsoft.ManagedIdentity/userAssignedIdentities/write`

By default, the **User Access Administrator** role and **Owner** role have these permissions already, but the **Contributor** role doesn't. If you don't have the required permissions, work with your Azure administrator to create a user-assigned managed identity with the

[Website Contributor role](#). In the Deployment Center, you can then select the identity in the GitHub > Identity dropdown.

For more information on the alternative steps, see [Deploy to App Service using GitHub Actions](#).

Why do I see the error, "This identity does not have write permissions on this app. Please select a different identity, or work with your admin to grant the Website Contributor role to your identity on this app"?

The message indicates that the selected user-assigned managed identity doesn't have the required role [to enable OpenID Connect](#) between the GitHub repository and the App Service app. The identity must have one of the following roles on the app: **Owner**, **Contributor**, **Websites Contributor**. The least privileged role that the identity needs is **Websites Contributor**.

More resources

- [Use Azure PowerShell](#)

Local Git deployment to Azure App Service

Article • 02/29/2024

This how-to guide shows you how to deploy your app to [Azure App Service](#) from a Git repository on your local computer.

ⓘ Note

When [SCM basic authentication is disabled](#), Local Git deployment doesn't work, and you can't configure Local Git deployment in the app's Deployment Center.

Prerequisites

To follow the steps in this how-to guide:

- If you don't have an [Azure subscription](#), create an [Azure free account](#) before you begin.
- [Install Git](#).
- Have a local Git repository with code you want to deploy. To download a sample repository, run the following command in your local terminal window:

Bash

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

Prepare your repository

To get automated builds from Azure App Service build server, make sure that your repository root has the correct files in your project.

[+] [Expand table](#)

Runtime	Root directory files
ASP.NET (Windows only)	<code>*.sln, *.csproj, or default.aspx</code>

Runtime	Root directory files
ASP.NET Core	<code>*.sln</code> or <code>*.csproj</code>
PHP	<code>index.php</code>
Ruby (Linux only)	<code>Gemfile</code>
Node.js	<code>server.js</code> , <code>app.js</code> , or <code>package.json</code> with a start script
Python	<code>*.py</code> , <code>requirements.txt</code> , or <code>runtime.txt</code>
HTML	<code>default.htm</code> , <code>default.html</code> , <code>default.asp</code> , <code>index.htm</code> , <code>index.html</code> , or <code>iisstart.htm</code>
WebJobs	<code><job_name>/run.<extension></code> under <code>App_Data/jobs/continuous</code> for continuous WebJobs, or <code>App_Data/jobs/triggered</code> for triggered WebJobs. For more information, see Kudu WebJobs documentation .
Functions	See Continuous deployment for Azure Functions .

To customize your deployment, include a `.deployment` file in the repository root. For more information, see [Customize deployments](#) and [Custom deployment script](#).

ⓘ Note

If you use Visual Studio, let [Visual Studio create a repository for you](#). Your project will immediately be ready for deployment via Git.

Configure a deployment user

See [Configure deployment credentials for Azure App Service](#). You can use either user-scope credentials or application-scope credentials.

Create a Git enabled app

If you already have an App Service app and want to configure local Git deployment for it, see [Configure an existing app](#) instead.

Azure CLI

Run `az webapp create` with the `--deployment-local-git` option. For example:

Azure CLI

```
az webapp create --resource-group <group-name> --plan <plan-name> --name <app-name> --runtime "<runtime-flag>" --deployment-local-git
```

The output contains a URL like: `https://<deployment-username>@<app-name>.scm.azurewebsites.net/<app-name>.git`. Use this URL to deploy your app in the next step.

Configure an existing app

If you don't have an app yet, see [Create a Git enabled app](#) instead.

Azure CLI

Run `az webapp deployment source config-local-git`. For example:

Azure CLI

```
az webapp deployment source config-local-git --name <app-name> --resource-group <group-name>
```

The output contains a URL like: `https://<deployment-username>@<app-name>.scm.azurewebsites.net/<app-name>.git`. Use this URL to deploy your app in the next step.

💡 Tip

This URL contains the user-scope deployment username. If you like, you can [use the application-scope credentials](#) instead.

Deploy the web app

1. In a local terminal window, change the directory to the root of your Git repository, and add a Git remote using the URL you got from your app. If your chosen method doesn't give you a URL, use `https://<app-name>.scm.azurewebsites.net/<app-name>.git` with your app name in `<app-name>`.

Bash

```
git remote add azure <url>
```

(!) Note

If you [created a Git-enabled app in PowerShell using New-AzWebApp](#), the remote is already created for you.

2. Push to the Azure remote with `git push azure master` (see [Change deployment branch](#)).
3. In the **Git Credential Manager** window, enter your [user-scope or application-scope credentials](#), not your Azure sign-in credentials.
If your Git remote URL already contains the username and password, you won't be prompted.
4. Review the output. You might see runtime-specific automation, such as MSBuild for ASP.NET, `npm install` for Node.js, and `pip install` for Python.
5. Browse to your app in the Azure portal to verify that the content is deployed.

Change deployment branch

When you push commits to your App Service repository, App Service deploys the files in the `master` branch by default. Because many Git repositories are moving away from `master` to `main`, you need to make sure that you push to the right branch in the App Service repository in one of two ways:

- Deploy to `master` explicitly with a command like:

Bash

```
git push azure main:master
```

- Change the deployment branch by setting the `DEPLOYMENT_BRANCH` app setting, then push commits to the custom branch. To do it with Azure CLI:

Azure CLI

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

```
git push azure main
```

You can also change the `DEPLOYMENT_BRANCH` app setting in the Azure portal, by selecting **Configuration** under **Settings** and adding a new Application Setting with a name of `DEPLOYMENT_BRANCH` and value of `main`.

Troubleshoot deployment

You might see the following common error messages when you use Git to publish to an App Service app in Azure:

 Expand table

Message	Cause	Resolution
<code>Unable to access '[siteURL]': Failed to connect to [scmAddress]</code>	The app isn't up and running.	Start the app in the Azure portal. Git deployment isn't available when the web app is stopped.
<code>Couldn't resolve host 'hostname'</code>	The address information for the <code>azure</code> remote is incorrect.	Use the <code>git remote -v</code> command to list all remotes, along with the associated URL. Verify that the URL for the <code>azure</code> remote is correct. If needed, remove and recreate this remote using the correct URL.
<code>No refs in common and none specified; doing nothing. Perhaps you should specify a branch such as 'main'.</code>	You didn't specify a branch during <code>git push</code> , or you haven't set the <code>push.default</code> value in <code>.gitconfig</code> .	Run <code>git push</code> again, specifying the main branch: <code>git push azure main</code> .
<code>Error - Changes committed to remote repository but deployment to website failed.</code>	You pushed a local branch that doesn't match the app deployment branch on <code>azure</code> .	Verify that current branch is <code>master</code> . To change the default branch, use <code>DEPLOYMENT_BRANCH</code> application setting (see Change deployment branch).
<code>src refspec [branchname] does not match any.</code>	You tried to push to a branch other than main on the <code>azure</code> remote.	Run <code>git push</code> again, specifying the main branch: <code>git push azure main</code> .
<code>RPC failed; result=22, HTTP code = 5xx.</code>	This error can happen if you try to push a large git repository over HTTPS.	Change the git configuration on the local machine to make the <code>postBuffer</code> bigger. For example: <code>git config --global http.postBuffer 524288000</code> .

Message	Cause	Resolution
Error - Changes committed to remote repository but your web app not updated.	You deployed a Node.js app with a <i>package.json</i> file that specifies additional required modules.	<p>Review the <code>npm ERR!</code> error messages before this error for more context on the failure. The following are the known causes of this error, and the corresponding <code>npm ERR!</code> messages:</p> <p>Malformed package.json file: <code>npm ERR!</code> Couldn't read dependencies.</p> <p>Native module doesn't have a binary distribution for Windows:</p> <pre><code>npm ERR! \cmd "/c" "node-gyp rebuild"\ failed with 1</code></pre> <p>or</p> <pre><code>npm ERR! [modulename@version] preinstall: \make gmake\</code></pre>

More resources

- [App Service build server \(Project Kudu documentation\)](#) ↗
- [Continuous deployment to Azure App Service](#)
- [Sample: Create a web app and deploy code from a local Git repository \(Azure CLI\)](#)
- [Sample: Create a web app and deploy code from a local Git repository \(PowerShell\)](#)

Deploy to App Service using Azure Pipelines

Article • 01/08/2025

Azure DevOps Services | Azure DevOps Server 2020 | Azure DevOps Server 2019

ⓘ Note

Starting June 1, 2024, all newly created App Service apps will have the option to generate a unique default hostname using the naming convention `<app-name>-<random-hash>. <region>.azurewebsites.net`. Existing app names will remain unchanged.

Example: `myapp-ds27dh7271aah175.westus-01.azurewebsites.net`

For further details, refer to [Unique Default Hostname for App Service Resource ↗](#).

Use [Azure Pipelines](#) to automatically deploy your web app to [Azure App Service](#) on every successful build. Azure Pipelines lets you build, test, and deploy with continuous integration (CI) and continuous delivery (CD) using [Azure DevOps](#).

YAML pipelines are defined using a YAML file in your repository. A step is the smallest building block of a pipeline and can be a script or task (prepackaged script). [Learn about the key concepts and components that make up a pipeline](#).

You'll use the [Azure Web App task \(AzureWebApp\)](#) to deploy to Azure App Service in your pipeline. For more complicated scenarios such as needing to use XML parameters in your deploy, you can use the [Azure App Service deploy task \(AzureRmWebAppDeployment\)](#).

Prerequisites

- An Azure account with an active subscription. [Create an account for free ↗](#).
- An Azure DevOps organization. [Create one for free](#).
- An ability to run pipelines on Microsoft-hosted agents. You can either purchase a [parallel job](#) or you can request a free tier.
- A working Azure App Service app with code hosted on [GitHub ↗](#) or [Azure Repos ↗](#).
 - .NET: [Create an ASP.NET Core web app in Azure](#)
 - ASP.NET: [Create an ASP.NET Framework web app in Azure](#)
 - JavaScript: [Create a Node.js web app in Azure App Service](#)

- Java: [Create a Java app on Azure App Service](#)
- Python: [Create a Python app in Azure App Service](#)

1. Create a pipeline for your stack

The code examples in this section assume you're deploying an ASP.NET web app. You can adapt the instructions for other frameworks.

Learn more about [Azure Pipelines ecosystem support](#).

YAML

1. Sign in to your Azure DevOps organization and navigate to your project.
2. Go to **Pipelines**, and then select **New Pipeline**.
3. When prompted, select the location of your source code: either **Azure Repos Git** or **GitHub**.

You might be redirected to GitHub to sign in. If so, enter your GitHub credentials.

4. When the list of repositories appears, select your repository.
5. You might be redirected to GitHub to install the Azure Pipelines app. If so, select **Approve & install**.
6. When the **Configure** tab appears, select **ASP.NET Core**.
7. When your new pipeline appears, take a look at the YAML to see what it does. When you're ready, select **Save and run**.

2. Add the deployment task

YAML

1. Click the end of the YAML file, then select **Show assistant**.'
2. Use the Task assistant to add the [Azure Web App](#) task.

A screenshot of the Azure DevOps Marketplace interface. At the top, there's a search bar and a filter section with dropdowns for 'Category' (selected 'Build & Release'), 'Type' (selected 'Task'), and 'Search' (empty). Below the search bar, there are four cards:

- Azure Functions**: Update a function app with .NET, Python, JavaScript...
- Azure Web App**: Deploy an Azure Web App for Linux or Windows. This card has a red border around it.
- Azure Web App for Containers**: Deploy containers to Azure App Service
- Deploy Azure Static Web App**: [PREVIEW] Build and deploy an Azure Static Web ...

Alternatively, you can add the [Azure App Service deploy \(AzureRmWebAppDeployment\)](#) task.

3. Choose your **Azure subscription**. Make sure to **Authorize** your connection.
The authorization creates the required service connection.
4. Select the **App type**, **App name**, and **Runtime stack** based on your App Service app. Your complete YAML should look similar to the following code.

YAML

```
variables:  
  buildConfiguration: 'Release'  
  
steps:  
- task: DotNetCoreCLI@2  
  inputs:  
    command: 'publish'  
    publishWebProjects: true  
- task: AzureWebApp@1  
  inputs:  
    azureSubscription: '<service-connection-name>'  
    appType: 'webAppLinux'  
    appName: '<app-name>'  
    package: '$(System.DefaultWorkingDirectory)/**/*.zip'
```

- **azureSubscription**: Name of the authorized service connection to your Azure subscription.
- **appName**: Name of your existing app.
- **package**: File path to the package or a folder containing your app service contents. Wildcards are supported.

Example: Deploy a .NET app

YAML

To deploy a .zip web package (for example, from an ASP.NET web app) to an Azure Web App, use the following snippet to deploy the build to an app.

YAML

```
variables:
  buildConfiguration: 'Release'

steps:
- task: DotNetCoreCLI@2
  inputs:
    command: 'publish'
    publishWebProjects: true
- task: AzureWebApp@1
  inputs:
    azureSubscription: '<service-connection-name>'
    appType: 'webAppLinux'
    appName: '<app-name>'
    package: '$(System.DefaultWorkingDirectory)/**/*.zip'
```

- **azureSubscription**: your Azure subscription.
- **appType**: your Web App type.
- **appName**: the name of your existing app service.
- **package**: the file path to the package or a folder containing your app service contents. Wildcards are supported.

Example: deploy to a virtual application

YAML

By default, your deployment happens to the root application in the Azure Web App. You can deploy to a specific virtual application by using the `VirtualApplication` property of the Azure App Service deploy (`AzureRmWebAppDeployment`) task:

YAML

```
- task: AzureRmWebAppDeployment@4
  inputs:
    VirtualApplication: '<name of virtual application>'
```

- **VirtualApplication**: the name of the Virtual Application that's configured in the Azure portal. For more information, see [Configure an App Service app in the Azure portal](#).

Example: Deploy to a slot

YAML

The following example shows how to deploy to a staging slot, and then swap to a production slot:

YAML

```
- task: AzureWebApp@1
  inputs:
    azureSubscription: '<service-connection-name>'
    appType: webAppLinux
    appName: '<app-name>'
    deployToSlotOrASE: true
    resourceGroupName: '<name of resource group>'
    slotName: staging
    package: '$(Build.ArtifactStagingDirectory)/**/*.zip'

- task: AzureAppServiceManage@0
  inputs:
    azureSubscription: '<service-connection-name>'
    appType: webAppLinux
    WebAppName: '<app-name>'
    ResourceGroupName: '<name of resource group>'
    SourceSlot: staging
    SwapWithProduction: true
```

- **azureSubscription**: your Azure subscription.
- **appType**: (optional) Use `webAppLinux` to deploy to a Web App on Linux.
- **appName**: the name of your existing app service.
- **deployToSlotOrASE**: Boolean. Deploy to an existing deployment slot or Azure App Service Environment.
- **resourceGroupName**: Name of the resource group. Required if `deployToSlotOrASE` is true.
- **slotName**: Name of the slot, which defaults to `production`. Required if `deployToSlotOrASE` is true.
- **package**: the file path to the package or a folder containing your app service contents. Wildcards are supported.

- **SourceSlot**: Slot sent to production when `SwapWithProduction` is true.
- **SwapWithProduction**: Boolean. Swap the traffic of source slot with production.

Example: Deploy to multiple web apps

YAML

You can use `jobs` in your YAML file to set up a pipeline of deployments. By using `jobs`, you can control the order of deployment to multiple web apps.

YAML

```
jobs:  
- job: buildandtest  
  pool:  
    vmImage: ubuntu-latest  
  
  steps:  
    # publish an artifact called drop  
    - task: PublishPipelineArtifact@1  
      inputs:  
        targetPath: '$(Build.ArtifactStagingDirectory)'  
        artifactName: drop  
  
    # deploy to Azure Web App staging  
    - task: AzureWebApp@1  
      inputs:  
        azureSubscription: '<service-connection-name>'  
        appType: <app type>  
        appName: '<staging-app-name>'  
        deployToSlotOrASE: true  
        resourceGroupName: <group-name>  
        slotName: 'staging'  
        package: '$(Build.ArtifactStagingDirectory)/**/*.{zip}'  
  
  - job: deploy  
    dependsOn: buildandtest  
    condition: succeeded()  
  
  pool:  
    vmImage: ubuntu-latest  
  
  steps:  
    # download the artifact drop from the previous job  
    - task: DownloadPipelineArtifact@2  
      inputs:  
        source: 'current'
```

```
artifact: 'drop'
path: '$(Pipeline.Workspace)'

- task: AzureWebApp@1
  inputs:
    azureSubscription: '<service-connection-name>'
    appType: <app type>
    appName: '<production-app-name>'
    resourceGroupName: <group-name>
    package: '$(Pipeline.Workspace)/**/*.zip'
```

Example: Deploy conditionally

YAML

To do this in YAML, you can use one of the following techniques:

- Isolate the deployment steps into a separate job, and add a condition to that job.
- Add a condition to the step.

The following example shows how to use step conditions to deploy only builds that originate from the main branch:

YAML

```
- task: AzureWebApp@1
  condition: and(succeeded(), eq(variables['Build.SourceBranch'],
'refs/heads/main'))
  inputs:
    azureSubscription: '<service-connection-name>'
    appName: '<app-name>'
```

To learn more about conditions, see [Specify conditions](#).

Example: deploy using Web Deploy

The Azure App Service deploy (`AzureRmWebAppDeployment`) task can deploy to App Service using Web Deploy.

YAML

yml

```
trigger:
- main

pool:
  vmImage: windows-latest

variables:
  buildConfiguration: 'Release'

steps:
- task: DotNetCoreCLI@2
  inputs:
    command: 'publish'
    publishWebProjects: true
    arguments: '--configuration $(buildConfiguration)'
    zipAfterPublish: true
- task: AzureRmWebAppDeployment@4
  inputs:
    ConnectionType: 'AzureRM'
    azureSubscription: '<service-connection-name>'
    appType: 'webApp'
    WebAppName: '<app-name>'
    packageForLinux: '$(System.DefaultWorkingDirectory)/**/*.zip'
    enableCustomDeployment: true
    DeploymentType: 'webDeploy'
```

Frequently asked questions

What's the difference between the `AzureWebApp` and `AzureRmWebAppDeployment` tasks?

The Azure Web App task (`AzureWebApp`) is the simplest way to deploy to an Azure Web App. By default, your deployment happens to the root application in the Azure Web App.

The [Azure App Service Deploy task \(AzureRmWebAppDeployment\)](#) can handle more custom scenarios, such as:

- Deploy with [Web Deploy](#), if you're used to the IIS deployment process.
- Deploy to [virtual applications](#).
- Deploy to other app types, like Container apps, Function apps, WebJobs, or API and Mobile apps.

Note

File transforms and variable substitution are also supported by the separate [File Transform task](#) for use in Azure Pipelines. You can use the File Transform task to apply file transformations and variable substitutions on any configuration and parameters files.

I get the message "Invalid App Service package or folder path provided."

In YAML pipelines, depending on your pipeline, there may be a mismatch between where your built web package is saved and where the deploy task is looking for it. For example, the `AzureWebApp` task picks up the web package for deployment. For example, the `AzureWebApp` task looks in `$(System.DefaultWorkingDirectory)/**/*.zip`. If the web package is deposited elsewhere, modify the value of `package`.

I get the message "Publish using webdeploy options are supported only when using Windows agent."

This error occurs in the `AzureRmWebAppDeployment` task when you configure the task to deploy using Web Deploy, but your agent isn't running Windows. Verify that your YAML has something similar to the following code:

```
yml  
  
pool:  
  vmImage: windows-latest
```

Web Deploy doesn't work when I disable basic authentication

For troubleshooting information on getting Microsoft Entra ID authentication to work with the `AzureRmWebAppDeployment` task, see [I can't Web Deploy to my Azure App Service using Microsoft Entra ID authentication from my Windows agent](#)

Next steps

- Customize your [Azure DevOps pipeline](#).

Feedback

Was this page helpful?

 Yes

 No

Provide product feedback  | Get help at Microsoft Q&A

Deploy to App Service using GitHub Actions

Article • 01/17/2025

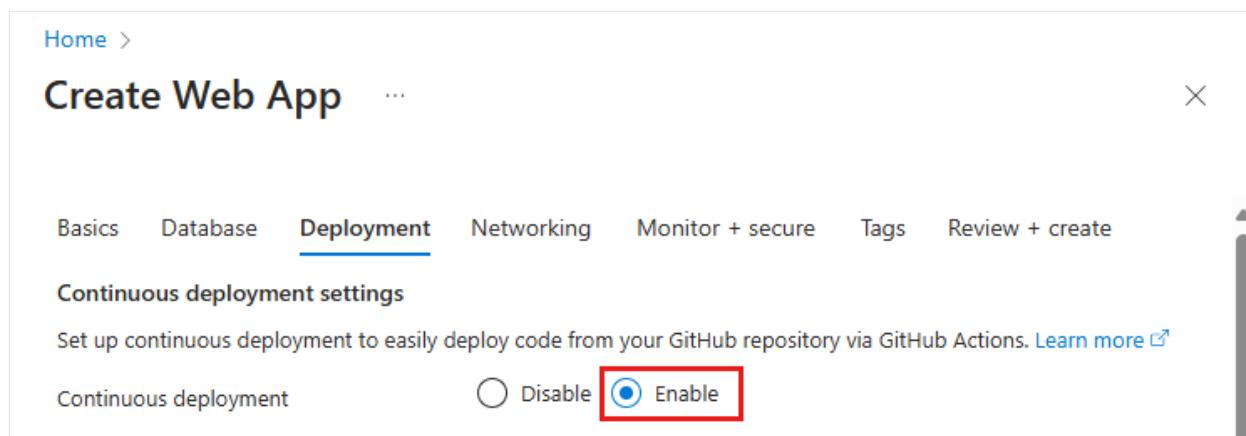
Get started with [GitHub Actions](#) to automate your workflow and deploy to [Azure App Service](#) from GitHub.

Prerequisites

- An Azure account with an active subscription. [Create an account for free](#).
- A GitHub account. If you don't have one, sign up for [free](#).

Set up GitHub Actions deployment when creating the app

GitHub Actions deployment is integrated into the default [Create Web App process](#). Set **Continuous deployment** to **Enable** in the Deployment tab, and configure the organization, repository, and branch that you want.



When you enable continuous deployment, app creation automatically picks the authentication method based on the basic authentication selection and configures your app and your GitHub repository accordingly:

[] [Expand table](#)

Basic authentication selection	Authentication method
Disable	User-assigned identity (OpenID Connect) (recommended)
Enable	Basic authentication

Note

You might receive an error when you create the app that your Azure account doesn't have certain permissions. Your account might need [the required permissions to create and configure the user-assigned identity](#). For an alternative, see [Set up GitHub Actions deployment from the Deployment Center](#).

Set up GitHub Actions deployment from the Deployment Center

For an existing app, you can get started quickly with GitHub Actions by using the App Service Deployment Center. This turn-key method generates a GitHub Actions workflow file based on your application stack and commits it to your GitHub repository.

The Deployment Center also lets you easily configure the more secure OpenID Connect authentication with a *user-assigned identity*. For more information, see [the user-assigned identity option](#).

If your Azure account has the [needed permissions](#), you can create a user-assigned identity. Otherwise, you can select an existing user-assigned managed identity in the **Identity** dropdown menu. You can work with your Azure administrator to create a user-assigned managed identity with the [Website Contributor role](#).

For more information, see [Continuous deployment to Azure App Service](#).

Set up a GitHub Actions workflow manually

You can deploy a workflow without using the Deployment Center. In that case you need to perform three steps:

1. [Generate deployment credentials](#)
2. [Configure the GitHub secret](#)
3. [Add the workflow file to your GitHub repository](#)

Generate deployment credentials

The recommended way to authenticate with Azure App Services for GitHub Actions is with OpenID Connect. This approach is an authentication method that uses short-lived tokens. Setting up [OpenID Connect with GitHub Actions](#) is more complex but offers hardened security.

Alternatively, you can authenticate with a User-assigned Managed Identity, a service principal, or a publish profile.

OpenID Connect

The following procedure describes the steps for creating an active directory application, service principal, and federated credentials using Azure CLI statements. To learn how to create an active directory application, service principal, and federated credentials in Azure portal, see [Connect GitHub and Azure](#).

1. If you don't have an existing application, register a [new Active Directory application and service principal that can access resources](#). Create the Active Directory application.

Azure CLI

```
az ad app create --display-name myApp
```

This command returns a JSON with an `appId` that is your `client-id`. Save the value to use as the `AZURE_CLIENT_ID` GitHub secret later.

You use the `objectId` value when creating federated credentials with Graph API and reference it as the `APPLICATION-OBJECT-ID`.

2. Create a service principal. Replace the `$appId` with the `appId` from your JSON output.

This command generates JSON output with a different `objectId` to use in the next step. The new `objectId` is the `assignee-object-id`.

Copy the `appOwnerTenantId` to use as a GitHub secret for `AZURE_TENANT_ID` later.

Azure CLI

```
az ad sp create --id $appId
```

3. Create a new role assignment by subscription and object. By default, the role assignment is tied to your default subscription. Replace `$subscriptionId` with your subscription ID, `$resourceGroupName` with your resource group name, `$webappName` with your web app name, and `$assigneeObjectId` with the generated `id`. Learn [how to manage Azure subscriptions with the Azure CLI](#).

Azure CLI

```
az role assignment create --role contributor --subscription $subscriptionId --assignee-object-id $assigneeObjectId --scope /subscriptions/$subscriptionId/resourceGroups/$resourceGroupName/providers/Microsoft.Web/sites/$webappName --assignee-principal-type ServicePrincipal
```

4. Run the following command to [create a new federated identity credential](#) for your active directory application.

- Replace `APPLICATION-OBJECT-ID` with the **appId (generated while creating app)** for your Active Directory application.
- Set a value for `CREDENTIAL-NAME` to reference later.
- Set the `subject`. GitHub defines its value depending on your workflow:
 - For jobs in your GitHub Actions environment: `repo:<Organization/Repository>:environment:< Name >`
 - For Jobs not tied to an environment, include the ref path for branch/tag based on the ref path used for triggering the workflow: `repo:< Organization/Repository >:ref:< ref path >`. For example, `repo:n-username/ node_express:ref:refs/heads/my-branch` or `repo:n-username/ node_express:ref:refs/tags/my-tag`.
 - For workflows triggered by a pull request event: `repo:<Organization/Repository >:pull_request`.

Azure CLI

```
az ad app federated-credential create --id <APPLICATION-OBJECT-ID> --parameters credential.json  
("credential.json" contains the following content)  
{  
    "name": "<CREDENTIAL-NAME>",  
    "issuer": "https://token.actions.githubusercontent.com",  
    "subject": "repo:organization/repository:ref:refs/heads/main",  
    "description": "Testing",  
    "audiences": [  
        "api://AzureADTokenExchange"  
    ]  
}
```

Configure the GitHub secret

OpenID Connect

You need to provide your application's **Client ID**, **Tenant ID**, and **Subscription ID** to the [Azure/login](#) action. These values can either be provided directly in the workflow or can be stored in GitHub secrets and referenced in your workflow. Saving the values as GitHub secrets is the more secure option.

1. Open your GitHub repository and go to **Settings > Security > Secrets and variables > Actions > New repository secret**.
2. Create secrets for `AZURE_CLIENT_ID`, `AZURE_TENANT_ID`, and `AZURE_SUBSCRIPTION_ID`. Use these values from your Active Directory application for your GitHub secrets:

 Expand table

GitHub Secret	Active Directory Application
<code>AZURE_CLIENT_ID</code>	Application (client) ID
<code>AZURE_TENANT_ID</code>	Directory (tenant) ID
<code>AZURE_SUBSCRIPTION_ID</code>	Subscription ID

3. Save each secret by selecting **Add secret**.

Add the workflow file to your GitHub repository

A YAML (.yml) file in the `/.github/workflows/` path in your GitHub repository defines a workflow. This definition contains the various steps and parameters that make up the workflow.

At a minimum, the workflow file would have the following distinct steps:

1. Authenticate with App Service using the GitHub secret you created.
2. Build the web app.
3. Deploy the web app.

To deploy your code to an App Service app, use the [azure/webapps-deploy@v3](#) action. The action requires the name of your web app in `app-name` and, depending on your language stack, the path of a *.zip, *.war, *.jar, or folder to deploy in `package`. For a

complete list of possible inputs for the `azure/webapps-deploy@v3` action, see [action.yml](#).

The following examples show the part of the workflow that builds the web app, in different supported languages.

OpenID Connect

To deploy with OpenID Connect using the managed identity you configured, use the `azure/login@v1` action with the `client-id`, `tenant-id`, and `subscription-id` keys. Reference the GitHub secrets that you created earlier.

ASP.NET Core

YAML

```
name: .NET Core

on: [push]

permissions:
  id-token: write
  contents: read

env:
  AZURE_WEBAPP_NAME: my-app      # set this to your application's name
  AZURE_WEBAPP_PACKAGE_PATH: '.'    # set this to the path to your
  web app project, defaults to the repository root
  DOTNET_VERSION: '6.0.x'          # set this to the dot net
  version to use

jobs:
  build:
    runs-on: ubuntu-latest

    steps:
      # Checkout the repo
      - uses: actions/checkout@main
      - uses: azure/login@v1
        with:
          client-id: ${{ secrets.AZURE_CLIENT_ID }}
          tenant-id: ${{ secrets.AZURE_TENANT_ID }}
          subscription-id: ${{ secrets.AZURE_SUBSCRIPTION_ID }}

      # Setup .NET Core SDK
      - name: Setup .NET Core
        uses: actions/setup-dotnet@v3
        with:
          dotnet-version: ${{ env.DOTNET_VERSION }}
```

```

# Run dotnet build and publish
- name: dotnet build and publish
  run: |
    dotnet restore
    dotnet build --configuration Release
    dotnet publish -c Release --property:PublishDir='${{ env.AZURE_WEBAPP_PACKAGE_PATH }}/myapp'

# Deploy to Azure Web apps
- name: 'Run Azure webapp deploy action using publish profile credentials'
  uses: azure/webapps-deploy@v3
  with:
    app-name: '${{ env.AZURE_WEBAPP_NAME }}' # Replace with your app name
    package: '${{ env.AZURE_WEBAPP_PACKAGE_PATH }}/myapp'

- name: logout
  run: |
    az logout

```

Frequently Asked Questions

- How do I deploy a WAR file through Maven plugin?
- How do I deploy a WAR file through Az CLI?
- How do I deploy a startup file?
- How do I deploy to a Container?
- How do I update the Tomcat configuration after deployment?

How do I deploy a WAR file through Maven plugin?

In case you configured your Java Tomcat project with the [Maven plugin](#), you can also deploy to Azure App Service through this plugin. If you use the [Azure CLI GitHub action](#), it makes use of your Azure credentials.

YAML

```

- name: Azure CLI script file
  uses: azure/cli@v2
  with:
    inlineScript: |
      mvn package azure-webapp:deploy

```

For more information on the Maven plugin and how to use and configure it, see [Maven plugin wiki for Azure App Service](#).

How do I deploy a WAR file through Az CLI?

If you prefer the Azure CLI to deploy to App Service, you can use the GitHub Action for Azure CLI.

YAML

```
- name: Azure CLI script
  uses: azure/cli@v2
  with:
    inlineScript: |
      az webapp deploy --src-path '${{ github.workspace }}'/target/yourpackage.war' --name ${{ env.AZURE_WEBAPP_NAME }} --resource-group ${{ env.RESOURCE_GROUP }} --async true --type war
```

For more information on the GitHub Action for CLI and how to use and configure it, see [Azure CLI GitHub action](#).

For more information on the az webapp deploy command, how to use it, and the parameter details, see [az webapp deploy documentation](#).

How do I deploy a startup file?

Use the GitHub Action for CLI. For example:

YAML

```
- name: Deploy startup script
  uses: azure/cli@v2
  with:
    inlineScript: |
      az webapp deploy --src-path '${{ github.workspace }}'/src/main/azure/createPasswordlessDataSource.sh --name ${{
        env.AZURE_WEBAPP_NAME }} --resource-group ${{
        env.RESOURCE_GROUP }} --type
        startup --track-status false
```

How do I deploy to a Container?

With the Azure Web Deploy action, you can automate your workflow to deploy custom containers to App Service using GitHub Actions. For more information about the steps to deploy using GitHub Actions, see [Deploy to a Container](#).

How do I update the Tomcat configuration after deployment?

In case you would like to update any of your web apps settings after deployment, you can use the [App Service Settings](#) action.

YAML

```
- uses: azure/appservice-settings@v1
  with:
    app-name: 'my-app'
    slot-name: 'staging' # Optional and needed only if the settings
have to be configured on the specific deployment slot
    app-settings-json: '[{"name": "CATALINA_OPTS", "value": "-Dfoo=bar"}]'
    connection-strings-json: '${{ secrets.CONNECTION_STRINGS }}'
    general-settings-json: '{"alwaysOn": "false", "webSocketsEnabled": "true"}' #'General configuration settings as Key Value pairs'
    id: settings
```

For more information on this action and how to use and configure it, see the [App Service Settings](#) repository.

Related content

Check out references on Azure GitHub Actions and workflows:

- [Azure/login action](#)
- [Azure/webapps-deploy action](#)
- [Docker/login action](#)
- [Azure/k8s-deploy action](#)
- [Actions workflows to deploy to Azure](#)
- [Starter Workflows](#)
- [Events that trigger workflows](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | Get help at Microsoft Q&A

Run your app in Azure App Service directly from a ZIP package

Article • 09/30/2024

ⓘ Note

Run from package is not supported for Python apps. When deploying a ZIP file of your Python code, you need to set a flag to enable Azure build automation. The build automation will create the Python virtual environment for your app and install any necessary requirements and package needed. See [build automation](#) for more details.

In [Azure App Service](#), you can run your apps directly from a deployment ZIP package file. This article shows how to enable this functionality in your app.

All other deployment methods in App Service have something in common: your files are deployed to `D:\home\site\wwwroot` in your app (or `/home/site/wwwroot` for Linux apps). Since the same directory is used by your app at runtime, it's possible for deployment to fail because of file lock conflicts, and for the app to behave unpredictably because some of the files are not yet updated.

In contrast, when you run directly from a package, the files in the package are not copied to the `wwwroot` directory. Instead, the ZIP package itself gets mounted directly as the read-only `wwwroot` directory. There are several benefits to running directly from a package:

- Eliminates file lock conflicts between deployment and runtime.
- Ensures only full-deployed apps are running at any time.
- Can be deployed to a production app (with restart).
- Improves the performance of Azure Resource Manager deployments.
- May reduce cold-start times, particularly for JavaScript functions with large npm package trees.

ⓘ Note

Currently, only ZIP package files are supported.

Create a project ZIP package

Important

When creating the ZIP package for deployment, don't include the root directory, but only the files and directories in it. If you download a GitHub repository as a ZIP file, you cannot deploy that file as-is to App Service. GitHub adds additional nested directories at the top level, which do not work with App Service.

In a local terminal window, navigate to the root directory of your app project.

This directory should contain the entry file to your web app, such as *index.html*, *index.php*, and *app.js*. It can also contain package management files like *project.json*, *composer.json*, *package.json*, *bower.json*, and *requirements.txt*.

Unless you want App Service to run deployment automation for you, run all the build tasks (for example, `npm`, `bower`, `gulp`, `composer`, and `pip`) and make sure that you have all the files you need to run the app. This step is required if you want to [run your package directly](#).

Create a ZIP archive of everything in your project. For `dotnet` projects, this is everything in the output directory of the `dotnet publish` command (excluding the output directory itself). For example, the following command in your terminal to create a ZIP package of the contents of the current directory:

```
# Bash  
zip -r <file-name>.zip .  
  
# PowerShell  
Compress-Archive -Path * -DestinationPath <file-name>.zip
```

Enable running from package

The `WEBSITE_RUN_FROM_PACKAGE` app setting enables running from a package. To set it, run the following command with Azure CLI.

Azure CLI

```
az webapp config appsettings set --resource-group <group-name> --name <app-name> --settings WEBSITE_RUN_FROM_PACKAGE="1"
```

`WEBSITE_RUN_FROM_PACKAGE="1"` lets you run your app from a package local to your app.

You can also [run from a remote package](#).

Run the package

The easiest way to run a package in your App Service is with the Azure CLI [az webapp deployment source config-zip](#) command. For example:

Azure CLI

```
az webapp deploy --resource-group <group-name> --name <app-name> --src-path  
<filename>.zip
```

Because the `WEBSITE_RUN_FROM_PACKAGE` app setting is set, this command doesn't extract the package content to the `D:\home\site\wwwroot` directory of your app. Instead, it uploads the ZIP file as-is to `D:\home\data\SitePackages`, and creates a `packagename.txt` in the same directory, that contains the name of the ZIP package to load at runtime. If you upload your ZIP package in a different way (such as [FTP](#)), you need to create the `D:\home\data\SitePackages` directory and the `packagename.txt` file manually.

The command also restarts the app. Because `WEBSITE_RUN_FROM_PACKAGE` is set, App Service mounts the uploaded package as the read-only `wwwroot` directory and runs the app directly from that mounted directory.

Run from external URL instead

You can also run a package from an external URL, such as Azure Blob Storage. You can use the [Azure Storage Explorer](#) to upload package files to your Blob storage account. You should use a private storage container with a [Shared Access Signature \(SAS\)](#) or use a [managed identity](#) to enable the App Service runtime to access the package securely.

Note

Currently, an existing App Service resource that runs a local package cannot be migrated to run from a remote package. You will have to create a new App Service resource configured to run from an external URL.

Once you upload your file to Blob storage and have an SAS URL for the file, set the `WEBSITE_RUN_FROM_PACKAGE` app setting to the URL. The following example does it by using Azure CLI:

Azure CLI

```
az webapp config appsettings set --name <app-name> --resource-group  
<resource-group-name> --settings  
WEBSITE_RUN_FROM_PACKAGE="https://myblobstorage.blob.core.windows.net/conten  
t/SampleCoreMVCApp.zip?st=2018-02-13T09%3A48%3A00Z&se=2044-06-  
14T09%3A48%3A00Z&sp=r&sv=2017-04-  
17&sr=b&sig=bNrVrEFzRHQB17GFJ7boEanetyJ9DGwBSV80M3Mdh%2FM%3D"
```

If you publish an updated package with the same name to Blob storage, you need to restart your app so that the updated package is loaded into App Service.

Access a package in Azure Blob Storage using a managed identity

You can configure Azure Blob Storage to [authorize requests with Microsoft Entra ID](#). This configuration means that instead of generating a SAS key with an expiration, you can instead rely on the application's [managed identity](#). By default, the app's system-assigned identity is used. If you wish to specify a user-assigned identity, you can set the `WEBSITE_RUN_FROM_PACKAGE_BLOB_MI_RESOURCE_ID` app setting to the resource ID of that identity. The setting can also accept `SystemAssigned` as a value, which is equivalent to omitting the setting.

To enable the package to be fetched using the identity:

1. Ensure that the blob is [configured for private access](#).
2. Grant the identity the [Storage Blob Data Reader](#) role with scope over the package blob. See [Assign an Azure role for access to blob data](#) for details on creating the role assignment.
3. Set the `WEBSITE_RUN_FROM_PACKAGE` application setting to the blob URL of the package. This URL is usually of the form `https://<storage-account-name>.blob.core.windows.net/<container-name>/<path-to-package>` or similar.

Deploy WebJob files when running from package

There are two ways to deploy [WebJob](#) files when you [enable running an app from package](#):

- Deploy in the same ZIP package as your app: include them as you normally would in `<project-root>\app_data\jobs\...` (which maps to the deployment path `\site\wwwroot\app_data\jobs\...` as specified in the [WebJobs quickstart](#)).
- Deploy separately from the ZIP package of your app: Since the usual deployment path `\site\wwwroot\app_data\jobs\...` is now read-only, you can't deploy WebJob files there. Instead, deploy WebJob files to `\site\jobs\...`, which is not read only. WebJobs deployed to `\site\wwwroot\app_data\jobs\...` and `\site\jobs\...` both run.

ⓘ Note

When `\site\wwwroot` becomes read-only, operations like the creation of the `disable.job` will fail.

Troubleshooting

- Running directly from a package makes `wwwroot` read-only. Your app will receive an error if it tries to write files to this directory.
- TAR and GZIP formats are not supported.
- The ZIP file can be at most 1GB
- This feature is not compatible with [local cache](#).
- For improved cold-start performance, use the local Zip option (`WEBSITE_RUN_FROM_PACKAGE = 1`).

More resources

- [Continuous deployment for Azure App Service](#)
- [Deploy code with a ZIP or WAR file](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

Authentication types by deployment methods in Azure App Service

Article • 01/24/2025

Azure App Service lets you deploy your web application code and configuration by using multiple options. These deployment options support one or more authentication mechanisms. This article provides details about various authentication mechanisms supported by different deployment methods.

ⓘ Note

To disable basic authentication for your App Service app, see [Disable basic authentication in App Service deployments](#).

[+] Expand table

Deployment method	Authentication	Reference Documents
Azure CLI	Microsoft Entra ID	In Azure CLI, version 2.48.1 or higher, the following commands use Microsoft Entra if basic authentication is turned off for your web app or function app: - az webapp up - az webapp deploy - az webapp log deployment show - az webapp log deployment list - az webapp log download - az webapp log tail - az webapp browse - az webapp create-remote-connection - az webapp ssh - az functionapp deploy - az functionapp log deployment list - az functionapp log deployment show - az functionapp deployment source config-zip For more information, see az appservice and az webapp .

Deployment method	Authentication	Reference Documents
Azure PowerShell	Microsoft Entra	In Azure PowerShell, version 9.7.1 or above, Microsoft Entra is available for App Service. For more information, see PowerShell samples for Azure App Service .
SCM/Kudu/OneDeploy REST endpoint	Basic authentication Microsoft Entra	Deploy files to App Service
Kudu UI	Basic authentication Microsoft Entra	Deploy files to App Service
FTP\FTPS	Basic authentication	Deploy your app to Azure App Service using FTP/S
Visual Studio	Basic authentication	Quickstart: Deploy an ASP.NET web app Develop and deploy WebJobs using Visual Studio Troubleshoot an app in Azure App Service using Visual Studio GitHub Actions integration in Visual Studio Deploy your application to Azure using GitHub Actions workflows created by Visual Studio
Visual Studio Code	Microsoft Entra	Quickstart: Deploy an ASP.NET web app Working with GitHub in VS Code
GitHub with GitHub Actions	Publish profile (basic authentication) Service principal (Microsoft Entra) OpenID Connect (Microsoft Entra)	Deploy to App Service using GitHub Actions
GitHub with App Service build service as build engine	Basic authentication	Continuous deployment to Azure App Service
GitHub with Azure Pipelines as build engine	Publish profile (basic authentication) Azure DevOps service connection	Deploy to App Service using Azure Pipelines

Deployment method	Authentication	Reference Documents
Azure Repos with App Service build service as build engine	Basic authentication	Continuous deployment to Azure App Service
Azure Repos with Azure Pipelines as build engine	Publish profile (basic authentication) Azure DevOps service connection	Deploy to App Service using GitHub Actions
Bitbucket	Basic authentication	Continuous deployment to Azure App Service
Local Git	Basic authentication	Local Git deployment to Azure App Service
External Git repository	Basic authentication	Setting up continuous deployment using manual steps
Run directly from an uploaded ZIP file	Microsoft Entra	Run your app in Azure App Service directly from a ZIP package
Run directly from external URL	Not applicable (outbound connection)	Run from external URL instead
Azure Web app plugin for Maven (Java)	Microsoft Entra	Quickstart: Create a Java app on Azure App Service
Azure WebApp Plugin for Gradle (Java)	Microsoft Entra	Configure a Java app for Azure App Service
Webhooks	Basic authentication	Web hooks
App Service migration assistant	Basic authentication	Azure App Service migration tools
App Service migration assistant for PowerShell scripts	Basic authentication	Azure App Service migration tools
Azure Migrate App Service discovery/assessment/migration	Microsoft Entra	Tutorial: Assess ASP.NET web apps for migration to Azure App Service Modernize ASP.NET web apps to Azure App Service code

Feedback

Was this page helpful?

 Yes

 No

Provide product feedback  | Get help at Microsoft Q&A

Configure deployment credentials for Azure App Service

Article • 01/30/2024

To secure app deployment from a local computer, [Azure App Service](#) supports two types of credentials for [local Git deployment](#) and [FTP/S deployment](#). These credentials are not the same as your Azure subscription credentials.

- **User-level credentials:** one set of credentials for the entire Azure account. It can be used to deploy to App Service for any app, in any subscription, that the Azure account has permission to access. It's the default set that's surfaced in the portal GUI (such as the [Overview](#) and [Properties](#) of the app's [resource page](#)). When a user is granted app access via Role-Based Access Control (RBAC) or coadmin permissions, that user can use their own user-level credentials until the access is revoked. Do not share these credentials with other Azure users.
- **App-level credentials:** one set of credentials for each app. It can be used to deploy to that app only. The credentials for each app are generated automatically at app creation. They can't be configured manually, but can be reset anytime. For a user to be granted access to app-level credentials via (RBAC), that user must be contributor or higher on the app (including Website Contributor built-in role). Readers are not allowed to publish, and can't access those credentials.

ⓘ Note

When [basic authentication is disabled](#), you can't view or configure deployment credentials in the Deployment Center.

Configure user-scope credentials

Azure CLI

Run the [az webapp deployment user set](#) command. Replace <username> and <password> with a deployment user username and password.

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

Azure CLI

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

The JSON output shows the password as `null`.

Use user-scope credentials with FTP/FTPS

Authenticating to an FTP/FTPS endpoint using user-scope credentials requires a username in the following format: `<app-name>\<user-name>`

Since user-scope credentials are linked to the user and not a specific resource, the username must be in this format to direct the sign-in action to the right app endpoint.

Get application-scope credentials

Azure CLI

Get the application-scope credentials using the [az webapp deployment list-publishing-profiles](#) command. For example:

Azure CLI

```
az webapp deployment list-publishing-profiles --resource-group <group-name> --name <app-name>
```

For [local Git deployment](#), you can also use the [az webapp deployment list-publishing-credentials](#) command to get a Git remote URI for your app, with the application-scope credentials already embedded. For example:

Azure CLI

```
az webapp deployment list-publishing-credentials --resource-group <group-name> --name <app-name> --query scmUri
```

Note that the returned Git remote URI doesn't contain `/<app-name>.git` at the end. When you add the remote URI, make sure to append `/<app-name>.git` to avoid an error 22 with `git-http-push`. Additionally, when using `git remote add ...` via shells that use the dollar sign for variable interpolation (such as bash), escape any

dollar signs (\\$) in the username or password. Failure to escape this character can result in authentication errors.

Reset application-scope credentials

Azure CLI

Reset the application-scope credentials using the [az resource invoke-action](#) command:

Azure CLI

```
az resource invoke-action --action newpassword --resource-group <group-name> --name <app-name> --resource-type Microsoft.Web/sites
```

Disable basic authentication

See [Disable basic authentication in App Service deployments](#).

Next steps

Find out how to use these credentials to deploy your app from [local Git](#) or using [FTP/S](#).

Set up staging environments in Azure App Service

Article • 07/30/2024

ⓘ Note

Starting June 1, 2024, all newly created App Service apps will have the option to generate a unique default hostname using the naming convention <app-name>-<random-hash>. <region>.azurewebsites.net. Existing app names will remain unchanged.

Example: myapp-ds27dh7271aah175.westus-01.azurewebsites.net

For further details, refer to [Unique Default Hostname for App Service Resource ↗](#).

When you deploy your web app, web app on Linux, mobile back end, or API app to [Azure App Service](#), you can use a separate deployment slot instead of the default production slot when you're running in the **Standard**, **Premium**, or **Isolated** App Service plan tier. Deployment slots are live apps with their own host names. App content and configurations elements can be swapped between two deployment slots, including the production slot.

Deploying your application to a nonproduction slot has the following benefits:

- You can validate app changes in a staging deployment slot before swapping it with the production slot.
- Deploying an app to a slot first and swapping it into production makes sure that all instances of the slot are warmed up before being swapped into production. This eliminates downtime when you deploy your app. The traffic redirection is seamless, and no requests are dropped because of swap operations. You can automate this entire workflow by configuring [auto swap](#) when pre-swap validation isn't needed.
- After a swap, the slot with previously staged app now has the previous production app. If the changes swapped into the production slot aren't as you expect, you can perform the same swap immediately to get your "last known good site" back.

Each App Service plan tier supports a different number of deployment slots. There's no extra charge for using deployment slots. To find out the number of slots your app's tier supports, see [App Service limits](#).

To scale your app to a different tier, make sure that the target tier supports the number of slots your app already uses. For example, if your app has more than five slots, you can't scale it down to the **Standard** tier, because the **Standard** tier supports only five deployment slots.

This video shows you how to set up staging environments in Azure App Service.

<https://learn-video.azurefd.net/vod/player?id=99aaff5e-fd3a-4568-b03a-a65745807d0f&locale=en-us&embedUrl=%2Fazure%2Fapp-service%2Fdeploy-staging-slots>

The steps in the video are also described in the following sections.

Prerequisites

For information on the permissions you need to perform the slot operation you want, see [Resource provider operations](#) (search for *slot*, for example).

Add a slot

The app must be running in the **Standard**, **Premium**, or **Isolated** tier in order for you to enable multiple deployment slots.

Azure portal

1. In the [Azure portal](#), navigate to your app's management page.
2. In the left pane, select **Deployment slots** > **Add Slot**.

ⓘ Note

If the app isn't already in the **Standard**, **Premium**, or **Isolated** tier, select **Upgrade** and go to the **Scale** tab of your app before continuing.

3. In the **Add a slot** dialog box, give the slot a name, and select whether to clone an app configuration from another deployment slot. Select **Add** to continue.

The screenshot shows the Azure portal interface for managing deployment slots. On the left, the navigation bar includes links for Overview, Activity log, Access control (IAM), Tags, Diagnose and solve problems, Microsoft Defender for Cloud, Events (preview), Deployment, Deployment slots (which is selected and highlighted with a red box), Deployment Center, Settings (Configuration, Authentication, Application Insights, Identity, Backups, Custom domains), and a search bar.

In the center, the 'Deployment Slots' section displays a table with one row:

NAME	STATUS
my-demo-app	PRODUCTION

The status is listed as 'Running'. At the top right of this section is a blue '+ Add Slot' button, also highlighted with a red box.

To the right, a modal dialog box titled 'Add a slot' is open. It contains fields for 'Name' (set to 'staging') and 'Clone settings from:' (set to 'Do not clone settings'). At the bottom of the dialog are 'Add' and 'Close' buttons, both highlighted with red boxes.

You can clone a configuration from any existing slot. Settings that can be cloned include app settings, connection strings, language framework versions, web sockets, HTTP version, and platform bitness.

Note

Currently, a private endpoint isn't cloned across slots.

4. After the slot is added, select **Close** to close the dialog box. The new slot is now shown on the **Deployment slots** page. By default, **Traffic %** is set to 0 for the new slot, with all customer traffic routed to the production slot.
5. Select the new deployment slot to open that slot's resource page.

Save Discard Add Slot Swap Logs Refresh

Deployment Slots

Deployment slots are live apps with their own hostnames. App content and configurations elements can be swapped between two deployment slots, including the production slot.

NAME	STATUS	APP SERVICE PLAN	TRAFFIC %
my-demo-app PRODUCTION	Running	myAppServicePlan	100
my-demo-app-staging	Running	myAppServicePlan	0

The staging slot has a management page just like any other App Service app. You can change the slot's configuration. To remind you that you're viewing the deployment slot, the app name is shown as <app-name>/<slot-name>, and the app type is **App Service (Slot)**. You can also see the slot as a separate app in your resource group, with the same designations.

6. Select the app URL on the slot's resource page. The deployment slot has its own host name and is also a live app. To limit public access to the deployment slot, see [Azure App Service IP restrictions](#).

The new deployment slot has no content, even if you clone the settings from a different slot. For example, you can [publish to this slot with Git](#). You can deploy to the slot from a different repository branch or a different repository. Get publish profile [from Azure App Service](#) can provide required information to deploy to the slot. The profile can be imported by Visual Studio to deploy contents to the slot.

The slot's URL has the format `http://sitename-slotname.azurewebsites.net`. To keep the URL length within necessary DNS limits, the site name will be truncated at 40 characters, and the combined site name and slot name must be fewer than 59 characters.

What happens during a swap

Swap operation steps

When you swap two slots (usually from a staging slot *as the source* into the production slot *as the target*), App Service does the following to ensure that the target slot doesn't experience downtime:

1. Apply the following settings from the target slot (for example, the production slot) to all instances of the source slot:

- [Slot-specific](#) app settings and connection strings, if applicable.
- [Continuous deployment](#) settings, if enabled.
- [App Service authentication](#) settings, if enabled.

When any of the settings is applied to the source slot, the change triggers all instances in the source slot to restart. During [swap with preview](#), this marks the end of the first phase. The swap operation is paused, and you can validate that the source slot works correctly with the target slot's settings.

2. Wait for every instance in the source slot to complete its restart. If any instance fails to restart, the swap operation reverts all changes to the source slot and stops the operation.
3. If [local cache](#) is enabled, trigger local cache initialization by making an HTTP request to the application root ("/") on each instance of the source slot. Wait until each instance returns any HTTP response. Local cache initialization causes another restart on each instance.
4. If [auto swap](#) is enabled with [custom warm-up](#), trigger the custom [Application Initiation](#) on each instance of the source slot.

If `applicationInitialization` isn't specified, trigger an HTTP request to the application root of the source slot on each instance.

If an instance returns any HTTP response, it's considered to be warmed up.

5. If all instances on the source slot are warmed up successfully, swap the two slots by switching the routing rules for the two slots. After this step, the target slot (for example, the production slot) has the app that's previously warmed up in the source slot.
6. Now that the source slot has the pre-swap app previously in the target slot, perform the same operation by applying all settings and restarting the instances.

At any point of the swap operation, all work of initializing the swapped apps happens on the source slot. The target slot remains online while the source slot is being prepared and warmed up, regardless of whether the swap succeeds or fails. To swap a staging slot

with the production slot, make sure that the production slot is always the target slot. This way, the swap operation doesn't affect your production app.

① Note

The instances in your former production instances (those that will be swapped into staging after this swap operation) will be recycled quickly in the last step of the swap process. In case you have any long running operations in your application, they will be abandoned, when the workers recycle. This also applies to function apps. Therefore your application code should be written in a fault tolerant way.

Which settings are swapped?

When you clone configuration from another deployment slot, the cloned configuration is editable. Some configuration elements follow the content across a swap (not slot specific), whereas other configuration elements stay in the same slot after a swap (slot specific). The following lists show the settings that change when you swap slots.

Settings that are swapped:

- Language stack and version, 32/64-bit
- App settings (can be configured to stick to a slot)
- Connection strings (can be configured to stick to a slot)
- Mounted storage accounts (can be configured to stick to a slot)
- Handler mappings
- Public certificates
- WebJobs content
- Hybrid connections *
- Service endpoints *
- Azure Content Delivery Network *
- Path mappings

Features marked with an asterisk (*) are planned to be unswapped.

Settings that aren't swapped:

- General settings not mentioned in **Settings that are swapped**
- Publishing endpoints
- Custom domain names
- Non-public certificates and TLS/SSL settings
- Scale settings
- WebJobs schedulers

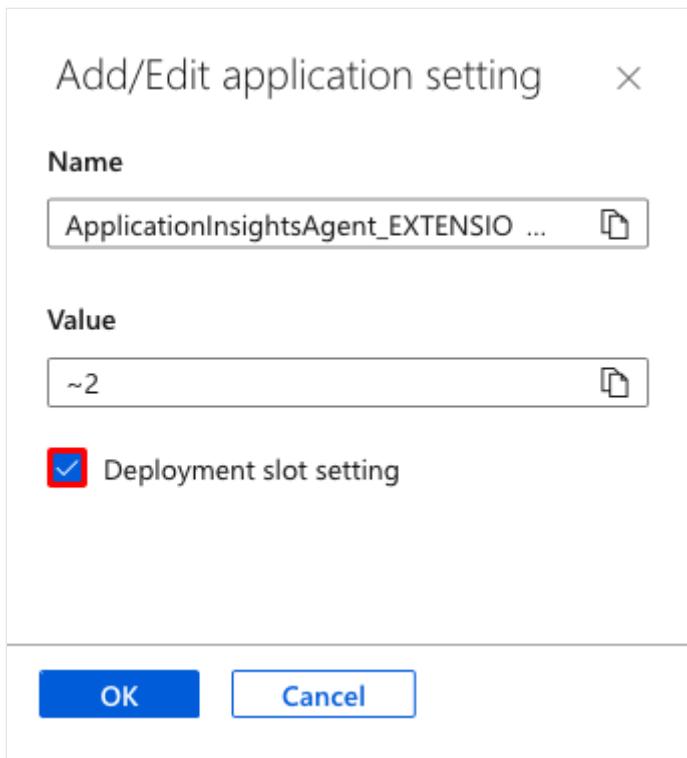
- IP restrictions
- Always On
- Diagnostic settings
- Cross-origin resource sharing (CORS)
- Virtual network integration
- Managed identities and related settings
- Settings that end with the suffix _EXTENSION_VERSION
- Settings that created by [Service Connector](#)

 **Note**

To make aforementioned settings swappable, add the app setting `WEBSITE_OVERRIDE_PRESERVE_DEFAULT_STICKY_SLOT_SETTINGS` in every slot of the app and set its value to `0` or `false`. These settings are either all swappable or not at all. You can't make just some settings swappable and not the others. Managed identities are never swapped and are not affected by this override app setting.

Certain app settings that apply to unswapped settings are also not swapped. For example, since diagnostic settings are not swapped, related app settings like `WEBSITE_HTTPLOGGING_RETENTION_DAYS` and `DIAGNOSTICS_AZUREBLOBRETENTIONDAYS` are also not swapped, even if they don't show up as slot settings.

To configure an app setting or connection string to stick to a specific slot (not swapped), go to the **Configuration** page for that slot. Add or edit a setting, and then select **deployment slot setting**. Selecting this check box tells App Service that the setting isn't swappable.



Swap two slots

You can swap deployment slots on your app's **Deployment slots** page and the **Overview** page. For technical details on the slot swap, see [What happens during swap](#).

i Important

Before you swap an app from a deployment slot into production, make sure that production is your target slot and that all settings in the source slot are configured exactly as you want to have them in production.

Azure portal

To swap deployment slots:

1. Go to your app's **Deployment slots** page and select **Swap**.

The screenshot shows the 'Deployment Slots' section of the Azure App Service blade. The 'Swap' button at the top right is highlighted with a red box. The left sidebar has 'Deployment slots' selected, also highlighted with a red box. The main area displays a table with two rows:

NAME	STATUS	APP SERVICE PLAN	TRAFFIC %
my-demo-app PRODUCTION	Running	myAppServicePlan	100
my-demo-app-staging	Running	myAppServicePlan	0

The **Swap** dialog box shows settings in the selected source and target slots that will be changed.

2. Select the desired **Source** and **Target** slots. Usually, the target is the production slot. Also, select the **Source Changes** and **Target Changes** tabs and verify that the configuration changes are expected. When you're finished, you can swap the slots immediately by selecting **Swap**.

Swap

X

● Source

my-demo-app-staging

● Target PRODUCTION

my-demo-app

i Swap with preview can only be used with sites that have deployment slot settings enabled

Perform swap with preview

Config Changes

This is a summary of the final set of configuration changes on the source and target deployment slots after the swap has completed.

● Source Changes		● Target Changes	
SETTING	TYPE	OLD VALUE	NEW VALUE
NetFrameworkVersion	General	v4.0	v7.0
PhpVersion	General	5.6	
WEBSITE_NODE_DEF...	AppSetting	6.9.1	Not set
APPINSIGHTS_INSTR...	AppSetting	Not set	10c4ca05-b90f-451f-8...
APPLICATIONINSIGH...	AppSetting	Not set	InstrumentationKey=...
ApplicationInsightsAg...	AppSetting	Not set	~2
XDT_MicrosoftApplica...	AppSetting	Not set	default

Swap

Close



To see how your target slot would run with the new settings before the swap actually happens, don't select **Swap**, but follow the instructions in [Swap with preview](#).

3. When you're finished, close the dialog box by selecting **Close**.

If you have any problems, see [Troubleshoot swaps](#).

Swap with preview (multi-phase swap)

Before you swap into production as the target slot, validate that the app runs with the swapped settings. The source slot is also warmed up before the swap completion, which

is desirable for mission-critical applications.

When you perform a swap with preview, App Service performs the same [swap operation](#) but pauses after the first step. You can then verify the result on the staging slot before completing the swap.

If you cancel the swap, App Service reapplies configuration elements to the source slot.

 **Note**

Swap with preview can't be used when one of the slots has site authentication enabled.

Azure portal

To swap with preview:

1. Follow the steps in [Swap deployment slots](#) but select **Perform swap with preview**.

Swap

X

Source

my-demo-app-staging

Target PRODUCTION

my-demo-app

Perform swap with preview

Swap with preview breaks down a normal swap into two phases. In phase one, any slot-specific application settings and connection strings on the destination will be temporarily copied to the source slot. This allows you to test the slot with its final configuration values. From here, you may choose to either cancel phase one to revert to your normal configuration, or proceed to phase two, which would remove the temporary config changes and complete swapping the source to destination slot.

Phase One Changes

These are the deployment slot specific application settings and connection strings from the target deployment slot which will be temporarily applied to the source deployment slot in the first phase of the swap.

Source Changes			
Setting	Type	Current Value	Temp Value
APPINSIGHTS_INSTR...	AppSetting	Not Set	10c4ca05-b90f-451f-8...
APPLICATIONINSIGH...	AppSetting	Not Set	InstrumentationKey=...
ApplicationInsightsAg...	AppSetting	Not Set	~2
XDT_MicrosoftApplica...	AppSetting	Not Set	default

Phase Two Changes

This is a summary of the final set of configuration changes on the source and target deployment slots after the swap has completed.

Source Changes		Target Changes	
Setting	Type	Old Value	New Value
NetFrameworkVersion	General	v4.0	v7.0
PhpVersion	General	5.6	
WEBSITE_NODE_DEF...	AppSetting	6.9.1	Not set



The dialog box shows you how the configuration in the source slot changes in phase 1, and how the source and target slot change in phase 2.

- When you're ready to start the swap, select Start Swap.

When phase 1 finishes, you're notified in the dialog box. Preview the swap in the source slot by going to `https://<app_name>-<source-slot-name>.azurewebsites.net`.

3. When you're ready to complete the pending swap, select **Complete Swap** in **Swap action** and select **Complete Swap**.

To cancel a pending swap, select **Cancel Swap** instead, and then select **Cancel Swap** at the bottom.

4. When you're finished, close the dialog box by selecting **Close**.

If you have any problems, see [Troubleshoot swaps](#).

Roll back a swap

If any errors occur in the target slot (for example, the production slot) after a slot swap, restore the slots to their pre-swap states by swapping the same two slots immediately.

Configure auto swap

ⓘ Note

Auto swap isn't supported in web apps on Linux and Web App for Containers.

Auto swap streamlines Azure DevOps scenarios where you want to deploy your app continuously with zero cold starts and zero downtime for customers of the app. When auto swap is enabled from a slot into production, every time you push your code changes to that slot, App Service automatically [swaps the app into production](#) after it's warmed up in the source slot.

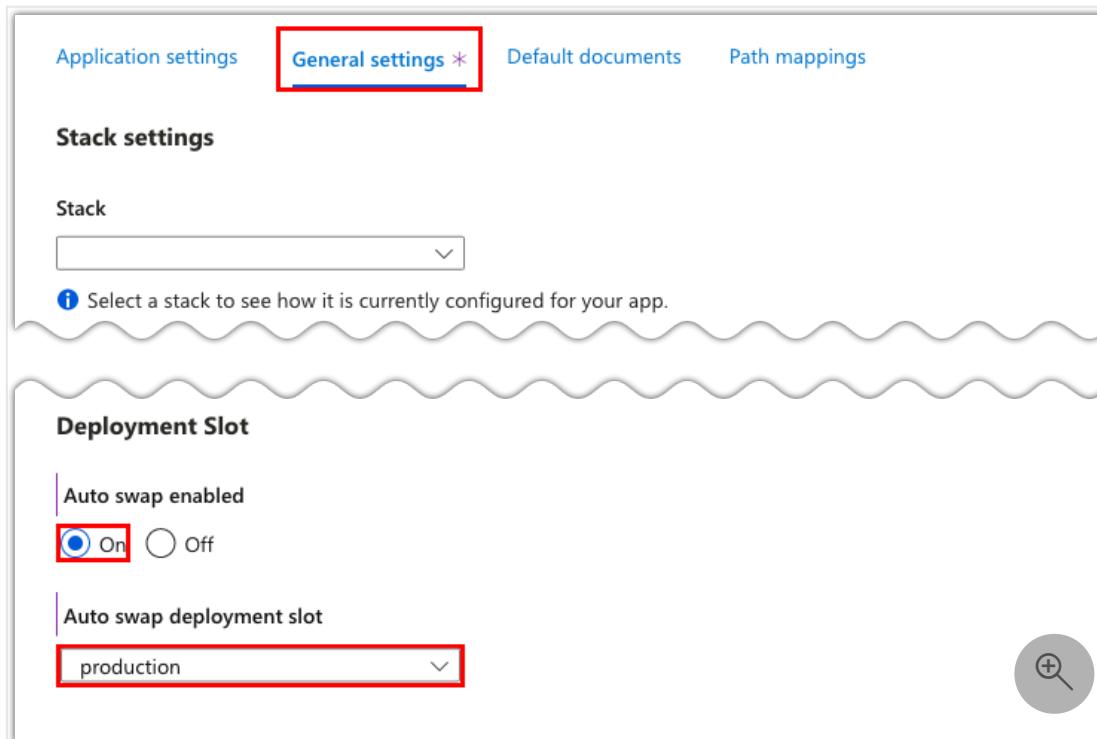
ⓘ Note

Before you configure auto swap for the production slot, consider testing auto swap on a nonproduction target slot.

Azure portal

To configure auto swap:

1. Go to your app's resource page. Select **Deployment slots** > *<desired source slot>* > **Configuration** > **General settings**.
2. For **Auto swap enabled**, select **On**. Then select the desired target slot for **Auto swap deployment slot**, and select **Save** on the command bar.



3. Execute a code push to the source slot. Auto swap happens after a short time, and the update is reflected at your target slot's URL.

If you have any problems, see [Troubleshoot swaps](#).

Specify custom warm-up

Some apps might require custom warm-up actions before the swap. The `applicationInitialization` configuration element in `web.config` lets you specify custom initialization actions. The [swap operation](#) waits for this custom warm-up to finish before swapping with the target slot. Here's a sample `web.config` fragment.

```
XML

<system.webServer>
  <applicationInitialization>
    <add initializationPage="/" hostName="[app hostname]" />
    <add initializationPage="/Home/About" hostName="[app hostname]" />
  </applicationInitialization>
</system.webServer>
```

For more information on customizing the `<applicationInitialization>` element, see [Most common deployment slot swap failures and how to fix them](#).

You can also customize the warm-up behavior with one or both of the following [app settings](#):

- `WEBSITE_SWAP_WARMUP_PING_PATH`: The path to ping over HTTP to warm up your site. Add this app setting by specifying a custom path that begins with a slash as the value. An example is `/statuscheck`. The default value is `/`.
- `WEBSITE_SWAP_WARMUP_PING_STATUSES`: Valid HTTP response codes for the warm-up operation. Add this app setting with a comma-separated list of HTTP codes. An example is `200,202`. If the returned status code isn't in the list, the warmup and swap operations are stopped. By default, all response codes are valid.
- `WEBSITE_WARMUP_PATH`: A relative path on the site that should be pinged whenever the site restarts (not only during slot swaps). Example values include `/statuscheck` or the root path, `/`.

Note

The `<applicationInitialization>` configuration element is part of each app startup, whereas the two warm-up behavior app settings apply only to slot swaps.

If you have any problems, see [Troubleshoot swaps](#).

Monitor a swap

If the [swap operation](#) takes a long time to complete, you can get information on the swap operation in the [activity log](#).

Azure portal

On your app's resource page in the portal, in the left pane, select **Activity log**.

A swap operation appears in the log query as `Swap Web App Slots`. You can expand it and select one of the suboperations or errors to see the details.

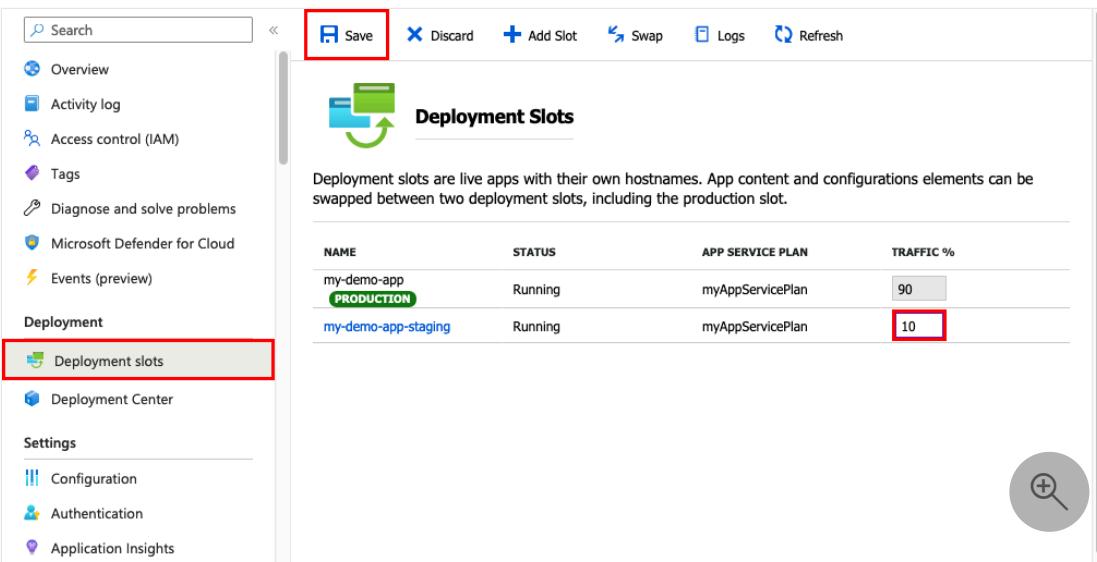
Route production traffic automatically

By default, all client requests to the app's production URL (`http://<app_name>.azurewebsites.net`) are routed to the production slot. You can route a portion of the traffic to another slot. This feature is useful if you need user feedback for a new update, but you're not ready to release it to production.

Azure portal

To route production traffic automatically:

1. Go to your app's resource page and select **Deployment slots**.
2. In the **Traffic %** column of the slot you want to route to, specify a percentage (between 0 and 100) to represent the amount of total traffic you want to route. Select **Save**.



NAME	STATUS	APP SERVICE PLAN	TRAFFIC %
my-demo-app PRODUCTION	Running	myAppServicePlan	90
my-demo-app-staging	Running	myAppServicePlan	10

After the setting is saved, the specified percentage of clients is randomly routed to the nonproduction slot.

After a client is automatically routed to a specific slot, it's "pinned" to that slot for one hour or until the cookies are deleted. On the client browser, you can see which slot your session is pinned to by looking at the `x-ms-routing-name` cookie in your HTTP headers. A request that's routed to the "staging" slot has the cookie `x-ms-routing-name=staging`. A request that's routed to the production slot has the cookie `x-ms-routing-name=self`.

Route production traffic manually

In addition to automatic traffic routing, App Service can route requests to a specific slot. This is useful when you want your users to be able to opt in to or opt out of your beta

app. To route production traffic manually, you use the `x-ms-routing-name` query parameter.

To let users opt out of your beta app, for example, you can put this link on your webpage:

HTML

```
<a href="<webappname>.azurewebsites.net/?x-ms-routing-name=self">Go back to  
production app</a>
```

The string `x-ms-routing-name=self` specifies the production slot. After the client browser accesses the link, it's redirected to the production slot. Every subsequent request has the `x-ms-routing-name=self` cookie that pins the session to the production slot.

To let users opt in to your beta app, set the same query parameter to the name of the nonproduction slot. Here's an example:

```
<webappname>.azurewebsites.net/?x-ms-routing-name=staging
```

By default, new slots are given a routing rule of `0%`, shown in grey. When you explicitly set this value to `0%` (shown in black text), your users can access the staging slot manually by using the `x-ms-routing-name` query parameter. But they won't be routed to the slot automatically because the routing percentage is set to 0. This is an advanced scenario where you can "hide" your staging slot from the public while allowing internal teams to test changes on the slot.

Delete a slot

Azure portal

Search for and select your app. Select **Deployment slots** > `<slot to delete>` > **Overview**. The app type is shown as **App Service (Slot)** to remind you that you're viewing a deployment slot. Before deleting a slot, make sure to stop the slot and set the traffic in the slot to zero. Select **Delete** on the command bar.

The screenshot shows the Azure portal interface for an app service slot named 'staging (my-demo-app/staging)'. The 'Delete' button in the top right corner is highlighted with a red box. The 'Overview' section displays various configuration details:

Resource	Value
Resource group	myresourcegroup
Status	Running
Location	West Europe
Subscription	Antares-Demo
App Service Plan	myAppServicePlan (S1: 1)
Default domain	my-demo-app4-staging.azureweeb...
Operating System	Windows
Health Check	Not Configured
Subscription ID	[redacted]

Automate with Resource Manager templates

Azure Resource Manager templates are declarative JSON files used to automate the deployment and configuration of Azure resources. To swap slots by using Resource Manager templates, you set two properties on the *Microsoft.Web/sites/slots* and *Microsoft.Web/sites* resources:

- `buildVersion`: this is a string property that represents the current version of the app deployed in the slot. For example: "v1", "1.0.0.1", or "2019-09-20T11:53:25.2887393-07:00".
- `targetBuildVersion`: this is a string property that specifies what `buildVersion` the slot should have. If the `targetBuildVersion` doesn't equal the current `buildVersion`, it triggers the swap operation by finding the slot with the specified `buildVersion`.

Example Resource Manager template

The following Resource Manager template swap two slots by updating the `buildVersion` of the `staging` slot and setting the `targetBuildVersion` on the production slot. It assumes you've created a slot called `staging`.

```
JSON

{
    "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",
    "contentVersion": "1.0.0.0",
    "parameters": {
```

```

    "my_site_name": {
        "defaultValue": "SwapAPIDemo",
        "type": "String"
    },
    "sites_buildVersion": {
        "defaultValue": "v1",
        "type": "String"
    }
},
"resources": [
{
    "type": "Microsoft.Web/sites/slots",
    "apiVersion": "2018-02-01",
    "name": "[concat(parameters('my_site_name'), '/staging')]",
    "location": "East US",
    "kind": "app",
    "properties": {
        "buildVersion": "[parameters('sites_buildVersion')]"
    }
},
{
    "type": "Microsoft.Web/sites",
    "apiVersion": "2018-02-01",
    "name": "[parameters('my_site_name')]",
    "location": "East US",
    "kind": "app",
    "dependsOn": [
        "[resourceId('Microsoft.Web/sites/slots',
parameters('my_site_name'), 'staging')]"
    ],
    "properties": {
        "targetBuildVersion": "[parameters('sites_buildVersion')]"
    }
}
]
}

```

This Resource Manager template is idempotent, meaning that it can be executed repeatedly and produce the same state of the slots. Without any change to the template, subsequent runs of the same template don't trigger any slot swap because the slots are already in the desired state.

Troubleshoot swaps

If any error occurs during a [slot swap](#), it's logged in `D:\home\LogFiles\eventlog.xml`. It's also logged in the application-specific error log.

Here are some common swap errors:

- An HTTP request to the application root is timed. The swap operation waits for 90 seconds for each HTTP request, and retries up to five times. If all retries are timed out, the swap operation is stopped.
- Local cache initialization might fail when the app content exceeds the local disk quota specified for the local cache. For more information, see [Local cache overview](#).
- During a site update operation, the following error may occur "*The slot cannot be changed because its configuration settings have been prepared for swap*". This can occur if either [swap with preview \(multi-phase swap\)](#) phase 1 has been completed but phase 2 has not yet been performed, or a swap has failed. There are two ways to resolve this issue:
 1. Cancel the swap operation which will reset the site back to the old state
 2. Complete the swap operation which will update site to the desired new state

Refer to [swap with preview \(multi-phase swap\)](#) to learn how to cancel or complete the swap operation.

- During [custom warm-up](#), the HTTP requests are made internally (without going through the external URL). They can fail with certain URL rewrite rules in *Web.config*. For example, rules for redirecting domain names or enforcing HTTPS can prevent warm-up requests from reaching the app code. To work around this issue, modify your rewrite rules by adding the following two conditions:

XML

```
<conditions>
  <add input="{WARMUP_REQUEST}" pattern="1" negate="true" />
  <add input="{REMOTE_ADDR}" pattern="^100?\." negate="true" />
  ...
</conditions>
```

- Without a custom warm-up, the URL rewrite rules can still block HTTP requests. To work around this issue, modify your rewrite rules by adding the following condition:

XML

```
<conditions>
  <add input="{REMOTE_ADDR}" pattern="^100?\." negate="true" />
  ...
</conditions>
```

- After slot swaps, the app may experience unexpected restarts. This is because after a swap, the hostname binding configuration goes out of sync, which by itself doesn't cause restarts. However, certain underlying storage events (such as storage volume failovers) may detect these discrepancies and force all worker processes to restart. To minimize these types of restarts, set the [WEBSITE_ADD_SITENAME_BINDINGS_IN_APPHOST_CONFIG=1 app setting](#) on *all slots*. However, this app setting does *not* work with Windows Communication Foundation (WCF) apps.

Next steps

[Block access to nonproduction slots](#)

ⓘ **Note:** The author created this article with assistance from AI. [Learn more](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

Tutorial: Configure a sidecar container for a Linux app in Azure App Service

Article • 11/19/2024

In this tutorial, you add an OpenTelemetry collector as a sidecar container to a Linux (bring-your-own-code) app in Azure App Service. For custom containers, see [Tutorial: Configure a sidecar container for custom container in Azure App Service](#).

In Azure App Service, you can add up to nine sidecar containers for each Linux app. Sidecar containers let you deploy extra services and features to your Linux apps without making them tightly coupled to the main (built-in) container. For example, you can add monitoring, logging, configuration, and networking services as sidecar containers. An OpenTelemetry collector sidecar is one such monitoring example.

For more information about side container in App Service, see:

- [Introducing Sidecars for Azure App Service for Linux: Now Generally Available ↗](#)
- [Announcing the general availability of sidecar extensibility in Azure App Service ↗](#)

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

1. Set up the needed resources

First you create the resources that the tutorial uses. They're used for this particular scenario and aren't required for sidecar containers in general.

1. In the [Azure Cloud Shell ↗](#), run the following commands. Be sure to supply the

`<environment-name>`.

```
Azure CLI  
  
git clone https://github.com/Azure-Samples/app-service-sidecar-tutorial-prereqs  
cd app-service-sidecar-tutorial-prereqs  
azd env new <environment-name>  
azd provision
```

2. When prompted, supply the subscription and region of your choice. For example:

- Subscription: Your subscription.
- Region: *(Europe) West Europe*.

When deployment completes, you should see the following output:

```
APPLICATIONINSIGHTS_CONNECTION_STRING =
InstrumentationKey=...;IngestionEndpoint=...;LiveEndpoint=...
Azure container registry name = <registry-name>
Managed identity resource ID = <managed-identity-resource-id>
Managed identity client ID = <managed-identity-client-id>

Open resource group in the portal:
https://portal.azure.com/#@/resource/subscriptions/<subscription-
id>/resourceGroups/<group-name>
```

3. Copy these output values for later. You can also find them in the portal, in the management pages of the respective resources.

 **Note**

`azd provision` uses the included templates to create the following Azure resources:

- A resource group based on the environment name.
- A container registry with two images deployed:
 - An Nginx image with the OpenTelemetry module.
 - An OpenTelemetry collector image, configured to export to Azure Monitor.
- A user-assigned managed identity with the `AcrPull` permission on the resource group (to pull images from the registry).
- A log analytics workspace.
- An Application Insights component.

2. Create a web app

In this step, you deploy a template ASP.NET Core application. Back in the [Azure Cloud Shell](#), run the following commands. Replace `<app-name>` with a unique app name.

.NET CLI

```
cd ~
dotnet new webapp -n MyFirstAzureWebApp --framework net8.0
```

```
cd MyFirstAzureWebApp  
az webapp up --name <app-name> --os-type linux
```

After a few minutes, this .NET web application is deployed as MyFirstAzureWebApp.dll to a new App Service app.

3. Add a sidecar container

In this section, you add a sidecar container to your Linux app. The portal experience is still being rolled out. If it's not available to you yet, continue with the **Use ARM template** tab below.

Use portal UI

1. In the [Azure portal](#), navigate to the app's management page
2. In the app's management page, from the left menu, select **Deployment Center**.
3. Select the banner **Interested in adding containers to run alongside your app?** [Click here to give it a try.](#)

If you can't see the banner, then the portal UI isn't rolled out for your subscription yet. Select the **Use ARM template** tab here instead and continue.

4. When the page reloads, select the **Containers (new)** tab.
5. Select **Add** and configure the new container as follows:
 - **Name:** *otel-collector*
 - **Image source:** Azure Container Registry
 - **Authentication:** Admin Credentials
 - **Registry:** The registry created by `azd provision`
 - **Image:** *otel-collector*
 - **Tag:** latest
6. Select **Apply**.

The screenshot shows the Azure App Service management interface for a web app named 'my-web-app'. On the left, the 'Deployment Center' is selected in the navigation menu. On the right, a modal window titled 'Add container' is open, showing the configuration for a new container. The container is set to be a 'Sidecar' type, named 'otel-collector', using an 'Azure Container Registry' image source. The authentication method is set to 'Admin Credentials'. The registry name is 'acrccrnsnqbezs2', the image name is 'otel-collector', and the image tag is 'latest'. The port is set to 8080, and the startup command is left blank. Under 'Environment variables', there is a note about selecting app settings to customize the deployment or runtime environment, and a checkbox for 'Allow access to all app settings' is checked. At the bottom of the modal are 'Apply' and 'Discard' buttons, with 'Apply' being highlighted.

4. Configure environment variables

For the sample scenario, the otel-collector sidecar is configured to export the OpenTelemetry data to Azure Monitor, but it needs the connection string as an environment variable (see the [OpenTelemetry configuration file for the otel-collector image](#)).

You configure environment variables for the containers like any App Service app, by configuring [app settings](#). The app settings are accessible to all the containers in the app.

1. Navigate to the App Service app's management page.
2. From the left menu, select **Environment variables**.
3. Add an app setting by selecting **Add** and configure it as follows:
 - **Name:** `APPLICATIONINSIGHTS_CONNECTION_STRING`
 - **Value:** The connection string in the output of `azd provision`. If you lost the Cloud Shell session, you can also find it in the **Overview** page of the Application Insight resource, under **Connection String**.
4. Select **Apply**, then **Apply**, then **Confirm**.

5. Configure instrumentation at startup

In this step, you create the instrumentation for your app according to the steps outlined in the [OpenTelemetry .NET zero-code instrumentation](#).

1. Back in the Cloud Shell, create *startup.sh* with the following lines.

```
Azure CLI

cat > startup.sh << 'EOF'
#!/bin/bash

# Download the bash script
curl -sSfL https://github.com/open-telemetry/opentelemetry-dotnet-
instrumentation/releases/latest/download/otel-dotnet-auto-install.sh -O

# Install core files
sh ./otel-dotnet-auto-install.sh

# Enable execution for the instrumentation script
chmod +x $HOME/.otel-dotnet-auto/instrument.sh

# Setup the instrumentation for the current shell session
. $HOME/.otel-dotnet-auto/instrument.sh

export OTEL_SERVICE_NAME="MyFirstAzureWebApp-Azure"
export OTEL_EXPORTER_OTLP_ENDPOINT="http://localhost:4318"
export OTEL_TRACES_EXPORTER="otlp"
export OTEL_METRICS_EXPORTER="otlp"
export OTEL_LOGS_EXPORTER="otlp"

# Run your application with instrumentation
OTEL_SERVICE_NAME=myapp
OTEL_RESOURCE_ATTRIBUTES=deployment.environment=staging,service.version
=1.0.0 dotnet /home/site/wwwroot/MyFirstAzureWebApp.dll
EOF
```

2. Deploy this file to your app with the following Azure CLI command. If you're still in the `~/MyFirstAzureWebApp` directory, then no other parameters are necessary because `az webapp up` already set defaults for the resource group and the app name.

Azure CLI

```
az webapp deploy --src-path startup.sh --target-path  
/home/site/startup.sh --type static
```

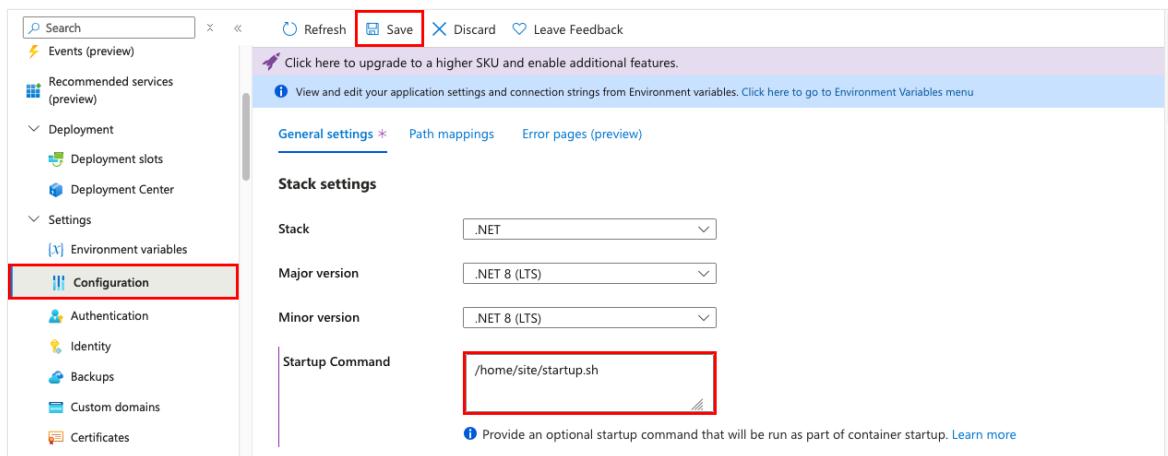
💡 Tip

This approach deploys the `startup.sh` file separately from your application. That way, the instrumentation configuration is separate from your application code. However, you can use other deployment methods to deploy the script together with your application.

3. Back in the app's management page, from the left menu, select **Configuration**.

4. Set **Startup Command** to `/home/site/startup.sh`. It's the same path that you deployed to in the previous step.

5. Select **Save**, then **Continue**.

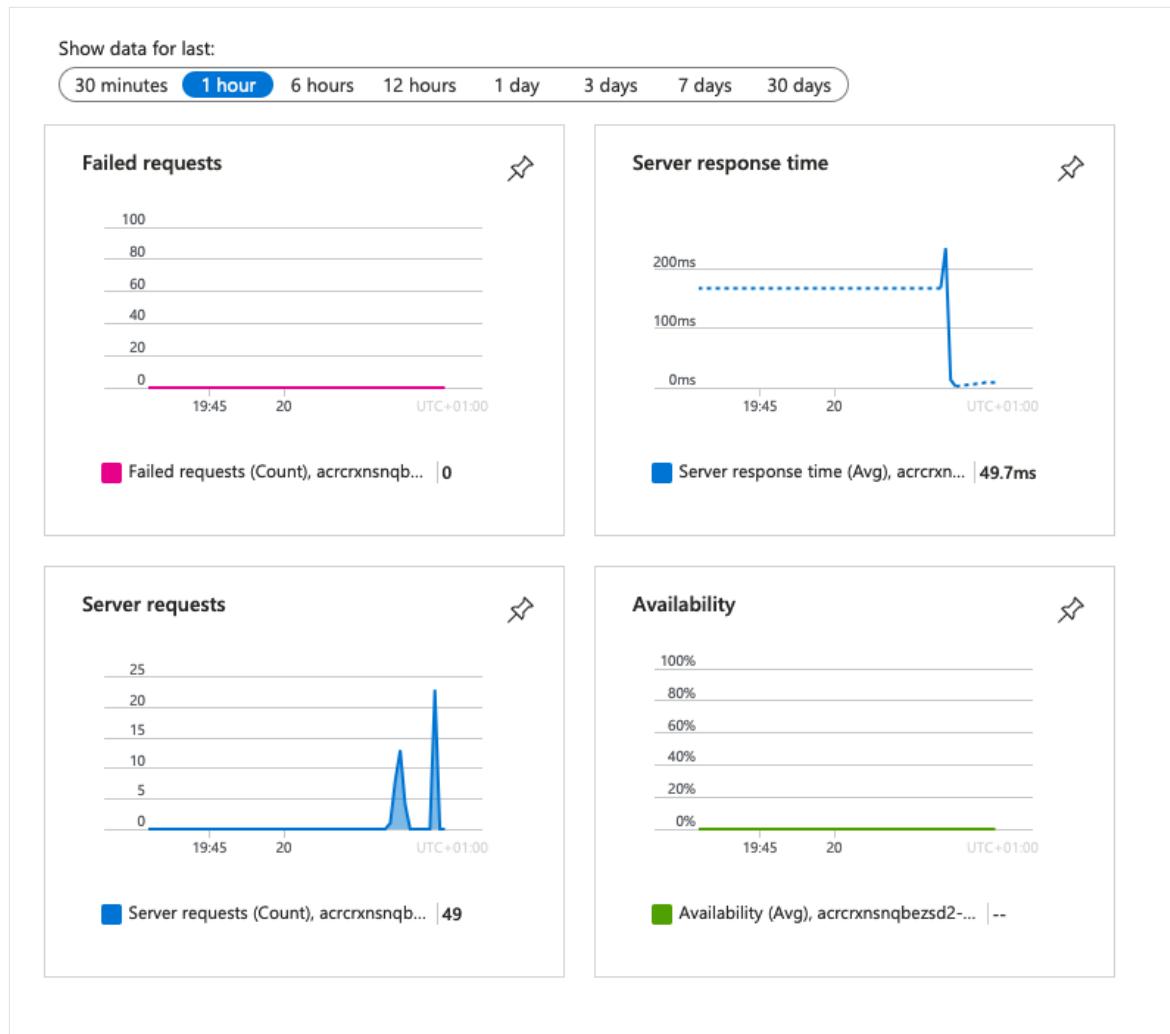


5. Verify in Application Insights

The otel-collector sidecar should export data to Application Insights now.

1. Back in the browser tab for `https://<app-name>.azurewebsites.net`, refresh the page a few times to generate some web requests.

2. Go back to the resource group overview page, then select the Application Insights resource that `azd up` created. You should now see some data in the default charts.



ⓘ Note

In this very common monitoring scenario, Application Insights is just one of the OpenTelemetry targets you can use, such as Jaeger, Prometheus, and Zipkin.

6. Clean up resources

When you no longer need the environment, you can delete the resource groups and all related resources. Just run these commands in the Cloud Shell:

Azure CLI

```
cd ~/MyFirstAzureWebApp
az group delete --yes
cd ~/app-service-sidecar-tutorial-prereqs
azd down
```

Frequently asked questions

- [How do sidecar containers handle internal communication?](#)
- [How do I instrument other language stacks?](#)

How do sidecar containers handle internal communication?

Sidecar containers share the same network host as the main container, so the main container (and other sidecar containers) can reach any port on the sidecar with `localhost:<port>`. The example `startup.sh` uses `localhost:4318` to access port 4318 on the `otel-collector` sidecar.

In the **Edit container** dialog, the **Port** box isn't currently used by App Service. You can use it as part of the sidecar metadata, such as to indicate which port the sidecar is listening to.

How do I instrument other language stacks?

You can use a similar approach to instrument apps in other language stacks. For more information, see OpenTelemetry documentation:

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

More resources

- [Deploy to App Service using GitHub Actions](#)
- [OpenTelemetry ↗](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

Mount Azure Storage as a local share in App Service

Article • 01/05/2024

ⓘ Note

NFS support is now available for App Service on Linux.

This guide shows how to mount Azure Storage as a network share in a built-in Linux container or a custom Linux container in App Service. Azure Storage is Microsoft's cloud storage solution for modern data storage scenarios. Azure Storage offers highly available, massively scalable, durable, and secure storage for a variety of data objects in the cloud. Azure Storage is non-default storage for App Service and billed separately. You can also [configure Azure Storage in an ARM template ↗](#).

Benefits

The benefits of custom-mounted storage include:

- Configure persistent storage for your App Service app and manage the storage separately.
- Make static content like video and images readily available for your App Service app.
- Write application log files or archive older application log to Azure File shares.
- Share content across multiple apps or with other Azure services.
- Azure Files [NFS](#) and Azure Files [SMB](#) are supported.
- Azure Blobs (read-only) are supported.
- Up to five mount points per app are supported.

Limitations

The limitations of custom-mounted storage include:

- [Storage firewall](#) is supported only through [service endpoints](#) and [private endpoints](#) (when [VNET integration](#) is used).
- FTP/FTPS access to custom-mounted storage isn't supported (use [Azure Storage Explorer ↗](#)).
- Azure CLI, Azure PowerShell, and Azure SDK support is in preview.
- Mapping `/` or `/home` to custom-mounted storage isn't supported.

- Don't map the storage mount to `/tmp` or its subdirectories as this action may cause a timeout during app startup.
- Azure Storage isn't supported with [Docker Compose](#) scenarios.
- Storage mounts aren't included in [backups](#). Be sure to follow best practices to back up the Azure Storage accounts.
- NFS support is only available for App Service on Linux. NFS isn't supported for Windows code and Windows containers. The web app and storage account need to be configured on the same VNET for NFS. The storage account used for file share should have "Premium" performance tier and "Filestorage" as the Account Kind. Azure Key Vault is not applicable when using the NFS protocol.
- With VNET integration on your app, the mounted drive uses an RFC1918 IP address and not an IP address from your VNET.

Mounting options

You first need to mount the storage to the app. Here are three mounting options for Azure storage:

[\[+\] Expand table](#)

Mounting option	Usage
Basic	Choose this option when mounting storage using the Azure portal. You can use the basic option as long as the storage account isn't using service endpoints , private endpoints , or Azure Key Vault . In this case, the portal gets and stores the access key for you.
Access Key	If you plan to mount storage using the Azure CLI, you need to obtain an access key. Choose this option storage account isn't using service endpoints , private endpoints , or Azure Key Vault .
Key Vault	Also use this option when you plan to mount storage using the Azure CLI, which requires the access key. Choose this option when using Azure Key Vault to securely store and retrieve access keys. Azure Key Vault has the benefits of storing application secrets centrally and securely with the ability to monitor, administer, and integrate with other Azure services like Azure App Service.

Prerequisites

Basic

- An existing [App Service on Linux](#) app.

- An [Azure Storage account](#).
- An [Azure file share and directory](#).

Prepare for mounting

Basic

No extra steps are required because the portal gets and stores the access key for you.

Mount storage to Linux container

The way that you mount storage depends on your storage access option and whether you are using the portal or the Azure CLI.

Azure portal

1. In the [Azure portal](#), navigate to the app.
2. From the left navigation, click **Configuration > Path Mappings > New Azure Storage Mount**.
3. Configure the storage mount according to the following table. When finished, click **OK**.

[\[\]](#) Expand table

Setting	Description
Name	Name of the mount configuration. Spaces aren't allowed.
Configuration options	Select Basic , if the storage account isn't using service endpoints , private endpoints , or Azure Key Vault . Otherwise, select Advanced .
Storage accounts	Azure Storage account.
Storage type	Select the type based on the storage you want to mount. Azure Blobs only supports read-only access.
Storage container or	Files share or Blobs container to mount.

Setting	Description
Share name	
Mount path	Directory inside the Linux container to mount to Azure Storage. Don't use <code>/</code> or <code>/home</code> .
Deployment slot setting	When checked, the storage mount settings also apply to deployment slots.

ⓘ Note

Adding, editing, or deleting a storage mount causes the app to be restarted.

Validate the mounted storage

To validate that the Azure Storage is mounted successfully for the app:

1. [Open an SSH session](#) into the container.
2. In the SSH terminal, execute the following command:

```
Bash
```

```
df -h
```

3. Check if the storage share is mounted. If it's not present, there's an issue with mounting the storage share.
4. Check latency or general reachability of the storage mount with the following command:

```
Bash
```

```
tcpping Storageaccount.file.core.windows.net
```

Best practices

Performance

- To avoid latency issues, place the app and the Azure Storage account in the same region. If you grant access from App Service IP addresses in the [Azure Storage firewall configuration](#) when the app and Azure Storage account are in the same region, then these IP restrictions aren't honored.
- The mounted Azure Storage account can be either Standard or Premium performance tier. Based on the app capacity and throughput requirements, choose the appropriate performance tier for the storage account. See the scalability and performance targets that correspond to the storage type: [Files](#) and [Blobs](#).
- If your app [scales to multiple instances](#), all the instances connect to the same mounted Azure Storage account. To avoid performance bottlenecks and throughput issues, choose the appropriate performance tier for the storage account.

Security

- In the Azure Storage account, avoid [regenerating the access key](#) that's used to mount the storage in the app. The storage account contains two different keys. Azure App Services stores Azure storage account key. Use a stepwise approach to ensure that the storage mount remains available to the app during key regeneration. For example, assuming that you used **key1** to configure storage mount in your app:
 1. Regenerate **key2**.
 2. In the storage mount configuration, update the access the key to use the regenerated **key2**.
 3. Regenerate **key1**.

Troubleshooting

- The mount directory in the custom container should be empty. Any content stored at this path is deleted when the Azure Storage is mounted (if you specify a directory under `/home`, for example). If you are migrating files for an existing app, make a backup of the app and its content before you begin.
- If you delete an Azure Storage account, container, or share, remove the corresponding storage mount configuration in the app to avoid possible error scenarios.
- It isn't recommended to use storage mounts for local databases (such as SQLite) or for any other applications and components that rely on file handles and locks.

- Ensure the following ports are open when using VNET integration: Azure Files: 80 and 445. Azure Blobs: 80 and 443.
- If you [initiate a storage failover](#) when the storage account is mounted to the app, the mount won't connect until the app is restarted or the storage mount is removed and re-added.

Next steps

- [Configure a custom container](#).
- [Video: How to mount Azure Storage as a local share](#) ↗.

Create App Service app using Bicep

Article • 04/09/2023

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

Bicep is a domain-specific language (DSL) that uses declarative syntax to deploy Azure resources. It provides concise syntax, reliable type safety, and support for code reuse. You can use Bicep instead of JSON to develop your Azure Resource Manager templates ([ARM templates](#)). The JSON syntax to create an ARM template can be verbose and require complicated expressions. Bicep syntax reduces that complexity and improves the development experience. Bicep is a transparent abstraction over ARM template JSON and doesn't lose any of the JSON template capabilities. During deployment, Bicep CLI transpiles a Bicep file into ARM template JSON.

Prerequisites

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

To effectively create resources with Bicep, you'll need to set up a Bicep [development environment](#). The Bicep extension for [Visual Studio Code](#) provides language support and resource autocompletion. The extension helps you create and validate Bicep files and is recommended for those developers that will create resources using Bicep after completing this quickstart.

Review the template

The template used in this quickstart is shown below. It deploys an App Service plan and an App Service app on Linux and a sample Node.js "Hello World" app from the [Azure Samples](#) repo.

Bicep

```
param webAppName string = uniqueString(resourceGroup().id) // Generate unique String for web app name
param sku string = 'F1' // The SKU of App Service Plan
param linuxFxVersion string = 'node|14-lts' // The runtime stack of web app
param location string = resourceGroup().location // Location for all resources
param repositoryUrl string = 'https://github.com/Azure-Samples/nodejs-docs-
```

```

hello-world'
param branch string = 'main'
var appServicePlanName = toLower('AppServicePlan-${webAppName}')
var webSiteName = toLower('wapp-${webAppName}')

resource appServicePlan 'Microsoft.Web/serverfarms@2020-06-01' = {
    name: appServicePlanName
    location: location
    properties: {
        reserved: true
    }
    sku: {
        name: sku
    }
    kind: 'linux'
}

resource appService 'Microsoft.Web/sites@2020-06-01' = {
    name: webSiteName
    location: location
    properties: {
        serverFarmId: appServicePlan.id
        siteConfig: {
            linuxFxVersion: linuxFxVersion
        }
    }
}

resource srcControls 'Microsoft.Web/sites/sourcecontrols@2021-01-01' = {
    name: '${appService.name}/web'
    properties: {
        repoUrl: repositoryUrl
        branch: branch
        isManualIntegration: true
    }
}

```

Three Azure resources are defined in the template:

- **Microsoft.Web/serverfarms**: create an App Service plan.
- **Microsoft.Web/sites**: create an App Service app.
- **Microsoft.Web/sites/sourcecontrols**: create an external git deployment configuration.

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

Parameters	Type	Default value	Description
webAppName	string	"webApp-<uniqueString>"	App name

Parameters	Type	Default value	Description
location	string	"[resourceGroup().location]"	App region
sku	string	"F1"	Instance size
linuxFxVersion	string	"NODE 14-LTS"	"Programming language stack Version"
repositoryUrl	string	"https://github.com/Azure-Samples/nodejs-docs-hello-world"	External Git repo (optional)
branch	string	"master"	Default branch for code sample

Deploy the template

Copy and paste the template to your preferred editor/IDE and save the file to your local working directory.

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

The following code creates a resource group, an App Service plan, and a web app. A default resource group, App Service plan, and location have been set for you. Replace `<app-name>` with a globally unique app name (valid characters are `a-z`, `0-9`, and `-`).

Open up a terminal where the Azure CLI is installed and run the code below to create a Node.js app on Linux.

Azure CLI

```
az group create --name myResourceGroup --location "southcentralus" &&
az deployment group create --resource-group myResourceGroup --template-file
<path-to-template>
```

To deploy a different language stack, update `linuxFxVersion` with appropriate values. Samples are shown below. To show current versions, run the following command in the Cloud Shell: `az webapp config show --resource-group myResourceGroup --name <app-name> --query linuxFxVersion`

Language	Example
.NET	<code>linuxFxVersion="DOTNETCORE 3.0"</code>

Language	Example
PHP	linuxFxVersion="PHP 7.4"
Node.js	linuxFxVersion="NODE 10.15"
Java	linuxFxVersion="JAVA 1.8 TOMCAT 9.0"
Python	linuxFxVersion="PYTHON 3.7"
Ruby	linuxFxVersion="RUBY 2.6"

Validate the deployment

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

Clean up resources

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

Next steps

[Bicep documentation](#)

[Bicep samples for Azure App Service](#)

Create App Service app using a Terraform template

Article • 04/09/2023

Get started with [Azure App Service](#) by deploying an app to the cloud using [Terraform](#). Because you use a free App Service tier, you incur no costs to complete this quickstart.

Terraform allows you to define and create complete infrastructure deployments in Azure. You build Terraform templates in a human-readable format that create and configure Azure resources in a consistent, reproducible manner. This article shows you how to create a Windows app with Terraform.

Prerequisites

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

Configure Terraform: If you haven't already done so, configure Terraform using one of the following options:

- [Configure Terraform in Azure Cloud Shell with Bash](#)
- [Configure Terraform in Azure Cloud Shell with PowerShell](#)
- [Configure Terraform in Windows with Bash](#)
- [Configure Terraform in Windows with PowerShell](#)

The Azure Terraform Visual Studio Code extension enables you to work with Terraform from the editor. With this extension, you can author, test, and run Terraform configurations. The extension also supports resource graph visualization. See [this guide](#) for configuring the Azure Terraform Visual Studio Code extension.

Review the template

The template used in this quickstart is shown below. It deploys an App Service plan and an App Service app on Linux and a sample Node.js "Hello World" app from the [Azure Samples](#) repo.

HashiCorp Configuration Language

```
# Configure the Azure provider
terraform {
    required_providers {
```

```

azurerm = {
    source  = "hashicorp/azurerm"
    version = "~> 3.0.0"
}
}
required_version = ">= 0.14.9"
}
provider "azurerm" {
    features {}
}

# Generate a random integer to create a globally unique name
resource "random_integer" "ri" {
    min = 10000
    max = 99999
}

# Create the resource group
resource "azurerm_resource_group" "rg" {
    name      = "myResourceGroup-${random_integer.ri.result}"
    location = "eastus"
}

# Create the Linux App Service Plan
resource "azurerm_service_plan" "appserviceplan" {
    name          = "webapp-asp-${random_integer.ri.result}"
    location      = azurerm_resource_group.rg.location
    resource_group_name = azurerm_resource_group.rg.name
    os_type       = "Linux"
    sku_name      = "B1"
}

# Create the web app, pass in the App Service Plan ID
resource "azurerm_linux_web_app" "webapp" {
    name          = "webapp-${random_integer.ri.result}"
    location      = azurerm_resource_group.rg.location
    resource_group_name = azurerm_resource_group.rg.name
    service_plan_id = azurerm_service_plan.appserviceplan.id
    https_only     = true
    site_config {
        minimum_tls_version = "1.2"
    }
}

# Deploy code from a public GitHub repo
resource "azurerm_app_service_source_control" "sourcecontrol" {
    app_id          = azurerm_linux_web_app.webapp.id
    repo_url        = "https://github.com/Azure-Samples/nodejs-docs-hello-world"
    branch         = "master"
    use_manual_integration = true
    use_mercurial   = false
}

```

Four Azure resources are defined in the template. Links to the [Azure Provider Terraform Registry](#) are given below for further details and usage information:

- **Microsoft.Resources/resourcegroups**: create a Resource Group if one doesn't already exist.
 - [azurerm_resource_group](#)
- **Microsoft.Web/serverfarms**: create an App Service plan.
 - [azurerm_service_plan](#)
- **Microsoft.Web/sites**: create a Linux App Service app.
 - [azurerm_linux_web_app](#)
- **Microsoft.Web/sites/sourcecontrols**: create an external git deployment configuration.
 - [azurerm_app_service_source_control](#)

For further information on how to construct Terraform templates, have a look at the [Terraform Learn documentation](#).

Implement the Terraform code

Terraform provides many features for managing, building, deploying, and updating infrastructure. The steps below will just guide you through deploying and destroying your resources. The [Terraform Learn documentation](#) and [Terraform on Azure documentation](#) go into more detail and should be reviewed if Terraform is part of your Azure infrastructure strategy.

1. Create a directory in which to test and run the sample Terraform code and make it the current directory.

```
Bash
```

```
mkdir appservice_tf_quickstart  
cd appservice_tf_quickstart
```

2. Create a file named `main.tf` and insert the above code.

```
Bash
```

```
code main.tf
```

3. Initialize Terraform.

```
Bash
```

```
terraform init
```

4. Create the Terraform plan.

```
Bash
```

```
terraform plan
```

5. Provision the resources that are defined in the `main.tf` configuration file (Confirm the action by entering `yes` at the prompt).

```
Bash
```

```
terraform apply
```

Validate the deployment

1. On the main menu of the Azure portal, select **Resource groups** and navigate to the resource group you created with the above template. It will be named "myResourceGroup-" followed by a string of random integers.
2. You now see all the resources that Terraform has created (an App Service and an App Service Plan).
3. Select the **App Service** and navigate to the url to verify your site has been created properly. Instead, you can just browse to `http://<app_name>.azurewebsites.net/` where app name is "webapp-" followed by that same string of random integers from the resource group.

Clean up resources

When no longer needed, either [delete the resource group](#) or head back to your terminal/command line and execute `terraform destroy` to delete all resources associated with this quickstart.

Note

You can find more Azure App Service Terraform samples [here](#). You can find even more Terraform samples across all of the Azure services [here](#).

Next steps

[Learn more about using Terraform in Azure](#)

[Terraform samples for Azure App Service](#)

Microsoft.Web resource types

Article • 12/09/2024

This article lists the available versions for each resource type.

For a list of changes in each API version, see [change log](#)

Resource types and versions

[] Expand table

Types	Versions
Microsoft.Web/certificates	2015-08-01 2016-03-01 2018-02-01 2018-11-01 2019-08-01 2020-06-01 2020-09-01 2020-10-01 2020-12-01 2021-01-01 2021-01-15 2021-02-01 2021-03-01 2022-03-01 2022-09-01 2023-01-01 2023-12-01

Types	Versions
	2024-04-01
Microsoft.Web/classicMobileServices	2015-08-01
Microsoft.Web/connectionGateways	2016-06-01
Microsoft.Web/connections	2015-08-01-preview 2016-06-01
Microsoft.Web/containerApps	2021-03-01 2022-03-01 2022-09-01 2023-01-01 2023-12-01 2024-04-01
Microsoft.Web/containerApps/revisions	2021-03-01 2022-03-01 2022-09-01 2023-01-01 2023-12-01 2024-04-01
Microsoft.Web/csrs	2015-08-01
Microsoft.Web/customApis	2016-06-01
Microsoft.Web/deletedSites	2020-12-01 2021-01-01 2021-

Types	Versions
	01-15
	2021-
	02-01
	2021-
	03-01
	2022-
	03-01
	2022-
	09-01
	2023-
	01-01
	2023-
	12-01
	2024-
	04-01
Microsoft.Web/hostingEnvironments	2015-
	08-01
	2016-
	09-01
	2018-
	02-01
	2019-
	08-01
	2020-
	06-01
	2020-
	09-01
	2020-
	10-01
	2020-
	12-01
	2021-
	01-01
	2021-
	01-15
	2021-
	02-01
	2021-
	03-01
	2022-
	03-01
	2022-
	09-01
	2023-
	01-01
	2023-
	12-01
	2024-
	04-01

Types	Versions
Microsoft.Web/hostingEnvironments/capacities	2019-08-01 2020-06-01 2020-09-01 2020-10-01 2020-12-01 2021-01-01 2021-01-15 2021-02-01 2021-03-01 2022-03-01 2022-09-01 2023-01-01 2023-12-01 2024-04-01
Microsoft.Web/hostingEnvironments/configurations	2020-12-01 2021-01-01 2021-01-15 2021-02-01 2021-03-01 2022-03-01 2022-09-01 2023-01-01 2023-12-01 2024-04-01

Types	Versions
Microsoft.Web/hostingEnvironments/detectors	2016-03-01 2018-02-01 2019-08-01 2020-06-01 2020-09-01 2020-10-01 2020-12-01 2021-01-01 2021-01-15 2021-02-01 2021-03-01 2022-03-01 2022-09-01 2023-01-01 2023-12-01 2024-04-01
Microsoft.Web/hostingEnvironments/multiRolePools	2015-08-01 2016-09-01 2018-02-01 2019-08-01 2020-06-01 2020-09-01 2020-10-01 2020-12-01 2021-

Types	Versions
	01-01
	2021-
	01-15
	2021-
	02-01
	2021-
	03-01
	2022-
	03-01
	2022-
	09-01
	2023-
	01-01
	2023-
	12-01
	2024-
	04-01
Microsoft.Web/hostingEnvironments/privateEndpointConnections	2020-
	12-01
	2021-
	01-01
	2021-
	01-15
	2021-
	02-01
	2021-
	03-01
	2022-
	03-01
	2022-
	09-01
	2023-
	01-01
	2023-
	12-01
	2024-
	04-01
Microsoft.Web/hostingEnvironments/recommendations	2018-
	02-01
	2019-
	08-01
	2020-
	06-01
	2020-
	09-01
	2020-
	10-01
	2020-
	12-01

Types	Versions
	2021-
	01-01
	2021-
	01-15
	2021-
	02-01
	2021-
	03-01
	2022-
	03-01
	2022-
	09-01
	2023-
	01-01
	2023-
	12-01
	2024-
	04-01
Microsoft.Web/hostingEnvironments/workerPools	2015-
	08-01
	2016-
	09-01
	2018-
	02-01
	2019-
	08-01
	2020-
	06-01
	2020-
	09-01
	2020-
	10-01
	2020-
	12-01
	2021-
	01-01
	2021-
	01-15
	2021-
	02-01
	2021-
	03-01
	2022-
	03-01
	2022-
	09-01
	2023-
	01-01
	2023-
	12-01

Types	Versions
	2024-04-01
Microsoft.Web/kubeEnvironments	2021-01-01 2021-01-15 2021-02-01 2021-03-01 2022-03-01 2022-09-01 2023-01-01 2023-12-01 2024-04-01
Microsoft.Web/locations/connectionGatewayInstallations	2016-06-01
Microsoft.Web/locations/deletedSites	2018-02-01 2019-08-01 2020-06-01 2020-09-01 2020-10-01 2020-12-01 2021-01-01 2021-01-15 2021-02-01 2021-03-01 2022-03-01 2022-09-01 2023-01-01

Types	Versions
	2023-12-01 2024-04-01
Microsoft.Web/locations/managedApis	2015-08-01-preview 2016-06-01
Microsoft.Web/managedHostingEnvironments	2015-08-01
Microsoft.Web/publishingUsers	2015-08-01 2016-03-01 2018-02-01 2019-08-01 2020-06-01 2020-09-01 2020-10-01 2020-12-01 2021-01-01 2021-01-15 2021-02-01 2021-03-01 2022-03-01 2022-09-01 2023-01-01 2023-01-01 2023-12-01 2024-04-01
Microsoft.Web/serverfarms	2015-08-01

Types	Versions
	2016-
	09-01
	2018-
	02-01
	2019-
	08-01
	2020-
	06-01
	2020-
	09-01
	2020-
	10-01
	2020-
	12-01
	2021-
	01-01
	2021-
	01-15
	2021-
	02-01
	2021-
	03-01
	2022-
	03-01
	2022-
	09-01
	2023-
	01-01
	2023-
	12-01
	2024-
	04-01
Microsoft.Web/serverfarms/hybridConnectionNamespaces/relays	2016-
	09-01
	2018-
	02-01
	2019-
	08-01
	2020-
	06-01
	2020-
	09-01
	2020-
	10-01
	2020-
	12-01
	2021-
	01-01
	2021-
	01-15

Types	Versions
	2021-02-01 2021-03-01 2022-03-01 2022-09-01 2023-01-01 2023-12-01 2024-04-01
Microsoft.Web/serverfarms/hybridConnectionPlanLimits	2016-09-01 2018-02-01 2019-08-01 2020-06-01 2020-09-01 2020-10-01 2020-12-01 2021-01-01 2021-01-15 2021-02-01 2021-03-01 2022-03-01 2022-09-01 2023-01-01 2023-12-01 2024-04-01
Microsoft.Web/serverfarms/operationresults	2015-08-01

Types	Versions
Microsoft.Web/serverfarms/virtualNetworkConnections	2015-08-01
	2016-09-01
	2018-02-01
	2019-08-01
	2020-06-01
	2020-09-01
	2020-10-01
	2020-12-01
	2021-01-01
	2021-01-15
	2021-02-01
	2021-03-01
	2022-03-01
	2022-09-01
	2023-01-01
	2023-12-01
	2024-04-01
Microsoft.Web/serverfarms/virtualNetworkConnections/gateways	2015-08-01
	2016-09-01
	2018-02-01
	2019-08-01
	2020-06-01
	2020-09-01
	2020-10-01
	2020-

Types	Versions
	12-01
	2021-
	01-01
	2021-
	01-15
	2021-
	02-01
	2021-
	03-01
	2022-
	03-01
	2022-
	09-01
	2023-
	01-01
	2023-
	12-01
	2024-
	04-01
Microsoft.Web/serverfarms/virtualNetworkConnections/routes	2015-
	08-01
	2016-
	09-01
	2018-
	02-01
	2019-
	08-01
	2020-
	06-01
	2020-
	09-01
	2020-
	10-01
	2020-
	12-01
	2021-
	01-01
	2021-
	01-15
	2021-
	02-01
	2021-
	03-01
	2022-
	03-01
	2022-
	09-01
	2023-
	01-01
	2023-

Types	Versions
	12-01 2024-04-01
Microsoft.Web/sites	2015-08-01 2016-08-01 2018-02-01 2018-11-01 2019-08-01 2020-06-01 2020-09-01 2020-10-01 2020-12-01 2021-01-01 2021-01-15 2021-02-01 2021-03-01 2022-03-01 2022-09-01 2023-01-01 2023-12-01 2024-04-01
Microsoft.Web/sites/backups	2015-08-01 2016-08-01 2018-02-01 2018-11-01 2019-08-01

Types	Versions
	2020-
	06-01
	2020-
	09-01
	2020-
	10-01
	2020-
	12-01
	2021-
	01-01
	2021-
	01-15
	2021-
	02-01
	2021-
	03-01
	2022-
	03-01
	2022-
	09-01
	2023-
	01-01
	2023-
	12-01
	2024-
	04-01
Microsoft.Web/sites/basicPublishingCredentialsPolicies	2019-
	08-01
	2020-
	06-01
	2020-
	09-01
	2020-
	10-01
	2020-
	12-01
	2021-
	01-01
	2021-
	01-15
	2021-
	02-01
	2021-
	03-01
	2022-
	03-01
	2022-
	09-01
	2023-
	01-01

Types	Versions
	2023- 12-01 2024- 04-01
Microsoft.Web/sites/config	2015- 08-01 2016- 08-01 2018- 02-01 2018- 11-01 2019- 08-01 2020- 06-01 2020- 09-01 2020- 10-01 2020- 12-01 2021- 01-01 2021- 01-15 2021- 02-01 2021- 03-01 2022- 03-01 2022- 09-01 2023- 01-01 2023- 12-01 2024- 04-01
Microsoft.Web/sites/config/appsettings	2019- 08-01 2020- 12-01 2021- 01-01 2021- 01-15 2021-

Types	Versions
	02-01 2021- 03-01 2022- 03-01 2022- 09-01 2023- 01-01 2023- 12-01 2024- 04-01
Microsoft.Web/sites/config/connectionstrings	2020- 12-01 2021- 01-01 2021- 01-15 2021- 02-01 2021- 03-01 2022- 03-01 2022- 09-01 2023- 01-01 2023- 12-01 2024- 04-01
Microsoft.Web/sites/config/snapshots	2016- 08-01 2018- 02-01 2018- 11-01 2019- 08-01 2020- 06-01 2020- 09-01 2020- 10-01 2020- 12-01

Types	Versions
	2021-
	01-01
	2021-
	01-15
	2021-
	02-01
	2021-
	03-01
	2022-
	03-01
	2022-
	09-01
	2023-
	01-01
	2023-
	12-01
	2024-
	04-01
Microsoft.Web/sites/continuouswebjobs	2016-
	08-01
	2018-
	02-01
	2018-
	11-01
	2019-
	08-01
	2020-
	06-01
	2020-
	09-01
	2020-
	10-01
	2020-
	12-01
	2021-
	01-01
	2021-
	01-15
	2021-
	02-01
	2021-
	03-01
	2022-
	03-01
	2022-
	09-01
	2023-
	01-01
	2023-
	12-01

Types	Versions
	2024-04-01
Microsoft.Web/sites/deployments	2015-08-01
	2016-08-01
	2018-02-01
	2018-11-01
	2019-08-01
	2020-06-01
	2020-09-01
	2020-10-01
	2020-12-01
	2021-01-01
	2021-01-15
	2021-02-01
	2021-03-01
	2022-03-01
	2022-09-01
	2023-01-01
	2023-12-01
	2024-04-01
Microsoft.Web/sites/deploymentStatus	2022-03-01
	2022-09-01
	2023-01-01
	2023-12-01
	2024-04-01

Types	Versions
Microsoft.Web/sites/detectors	2016-03-01 2018-02-01 2019-08-01 2020-06-01 2020-09-01 2020-10-01 2020-12-01 2021-01-01 2021-01-15 2021-02-01 2021-03-01 2022-03-01 2022-09-01 2023-01-01 2023-12-01 2024-04-01
Microsoft.Web/sites/diagnostics	2016-03-01 2018-02-01 2019-08-01 2020-06-01 2020-09-01 2020-10-01 2020-12-01 2021-01-01 2021-01-15 2021-02-01 2021-03-01 2022-03-01 2022-09-01 2023-01-01 2023-12-01 2024-04-01

Types	Versions
	01-15 2021- 02-01 2021- 03-01 2022- 03-01 2022- 09-01 2023- 01-01 2023- 12-01 2024- 04-01
Microsoft.Web/sites/diagnostics/analyses	2016- 03-01 2018- 02-01 2019- 08-01 2020- 06-01 2020- 09-01 2020- 10-01 2020- 12-01 2021- 01-01 2021- 01-15 2021- 02-01 2021- 03-01 2022- 03-01 2022- 09-01 2023- 01-01 2023- 12-01 2024- 04-01
Microsoft.Web/sites/diagnostics/detectors	2019- 08-01

Types	Versions
	2020-
	06-01
	2020-
	09-01
	2020-
	10-01
	2020-
	12-01
	2021-
	01-01
	2021-
	01-15
	2021-
	02-01
	2021-
	03-01
	2022-
	03-01
	2022-
	09-01
	2023-
	01-01
	2023-
	12-01
	2024-
	04-01
Microsoft.Web/sites/domainOwnershipIdentifiers	2016-
	08-01
	2018-
	02-01
	2018-
	11-01
	2019-
	08-01
	2020-
	06-01
	2020-
	09-01
	2020-
	10-01
	2020-
	12-01
	2021-
	01-01
	2021-
	01-15
	2021-
	02-01
	2021-
	03-01

Types	Versions
	2022-03-01 2022-09-01 2023-01-01 2023-02-01 2024-01-01
Microsoft.Web/sites/extensions	2016-08-01 2018-02-01 2018-11-01 2019-08-01 2020-06-01 2020-09-01 2020-10-01 2020-12-01 2021-01-01 2021-01-15 2021-02-01 2021-03-01 2022-03-01 2022-09-01 2023-01-01 2023-12-01 2024-01-01
Microsoft.Web/sites/functions	2016-08-01 2018-02-01 2018-

Types	Versions
	11-01
	2019-
	08-01
	2020-
	06-01
	2020-
	09-01
	2020-
	10-01
	2020-
	12-01
	2021-
	01-01
	2021-
	01-15
	2021-
	02-01
	2021-
	03-01
	2022-
	03-01
	2022-
	09-01
	2023-
	01-01
	2023-
	12-01
	2024-
	04-01
Microsoft.Web/sites/functions/keys	2018-
	02-01
	2019-
	08-01
	2020-
	06-01
	2020-
	09-01
	2020-
	10-01
	2020-
	12-01
	2021-
	01-01
	2021-
	01-15
	2021-
	02-01
	2021-
	03-01
	2022-

Types	Versions
	03-01 2022- 09-01 2023- 01-01 2023- 12-01 2024- 04-01
Microsoft.Web/sites/hostNameBindings	2015- 08-01 2016- 08-01 2018- 02-01 2018- 11-01 2019- 08-01 2020- 06-01 2020- 09-01 2020- 10-01 2020- 12-01 2021- 01-01 2021- 01-15 2021- 02-01 2021- 03-01 2022- 03-01 2022- 09-01 2023- 01-01 2023- 12-01 2024- 04-01
Microsoft.Web/sites/hostruntime/webhooks/api/workflows/runs	2022- 03-01 2022- 09-01

Types	Versions
	2023-01-01 2023-12-01 2024-04-01
Microsoft.Web/sites/hostruntime/webhooks/api/workflows/runs/actions	2022-03-01 2022-09-01 2023-01-01 2023-12-01 2024-04-01
Microsoft.Web/sites/hostruntime/webhooks/api/workflows/runs/actions/repetitions	2022-03-01 2022-09-01 2023-01-01 2023-12-01 2024-04-01
Microsoft.Web/sites/hostruntime/webhooks/api/workflows/runs/actions/repetitions/requestHistories	2022-03-01 2022-09-01 2023-01-01 2023-12-01 2024-04-01
Microsoft.Web/sites/hostruntime/webhooks/api/workflows/runs/actions/scopeRepetitions	2022-03-01 2022-09-01 2023-01-01 2023-12-01 2024-04-01

Types	Versions
Microsoft.Web/sites/hostruntime/webhooks/api/workflows/triggers	2022-03-01 2022-09-01 2023-01-01 2023-04-01 2023-12-01 2024-01-01 2024-04-01
Microsoft.Web/sites/hostruntime/webhooks/api/workflows/triggers/histories	2022-03-01 2022-09-01 2023-01-01 2023-04-01 2023-12-01 2024-01-01 2024-04-01
Microsoft.Web/sites/hostruntime/webhooks/api/workflows/versions	2022-03-01 2022-09-01 2023-01-01 2023-04-01 2023-12-01 2024-01-01 2024-04-01
Microsoft.Web/sites/hybridconnection	2015-08-01 2016-08-01 2018-02-01 2018-04-01 2018-11-01 2019-08-01 2020-06-01 2020-09-01 2020-10-01 2020-12-01

Types	Versions
	2021-
	01-01
	2021-
	01-15
	2021-
	02-01
	2021-
	03-01
	2022-
	03-01
	2022-
	09-01
	2023-
	01-01
	2023-
	12-01
	2024-
	04-01
Microsoft.Web/sites/hybridConnectionNamespaces/relays	2016-
	08-01
	2018-
	02-01
	2018-
	11-01
	2019-
	08-01
	2020-
	06-01
	2020-
	09-01
	2020-
	10-01
	2020-
	12-01
	2021-
	01-01
	2021-
	01-15
	2021-
	02-01
	2021-
	03-01
	2022-
	03-01
	2022-
	09-01
	2023-
	01-01
	2023-
	12-01

Types	Versions
	2024-04-01
Microsoft.Web/sites/instances	2019-08-01 2020-06-01 2020-09-01 2020-10-01 2020-12-01 2021-01-01 2021-01-15 2021-02-01 2021-03-01 2022-03-01 2022-09-01 2023-01-01 2023-12-01 2024-04-01
Microsoft.Web/sites/instances/deployments	2015-08-01
Microsoft.Web/sites/instances/extensions	2016-08-01 2018-02-01 2018-11-01 2019-08-01 2020-06-01 2020-09-01 2020-10-01 2020-12-01

Types	Versions
	2021-
	01-01
	2021-
	01-15
	2021-
	02-01
	2021-
	03-01
	2022-
	03-01
	2022-
	09-01
	2023-
	01-01
	2023-
	12-01
	2024-
	04-01
Microsoft.Web/sites/instances/processes	2016-
	08-01
	2018-
	02-01
	2018-
	11-01
	2019-
	08-01
	2020-
	06-01
	2020-
	09-01
	2020-
	10-01
	2020-
	12-01
	2021-
	01-01
	2021-
	01-15
	2021-
	02-01
	2021-
	03-01
	2022-
	03-01
	2022-
	09-01
	2023-
	01-01
	2023-
	12-01

Types	Versions
	2024-04-01
Microsoft.Web/sites/instances/processes/modules	2016-08-01 2018-02-01 2018-11-01 2019-08-01 2020-06-01 2020-09-01 2020-10-01 2020-12-01 2021-01-01 2021-01-15 2021-02-01 2021-03-01 2022-03-01 2022-09-01 2023-01-01 2023-12-01 2024-04-01
Microsoft.Web/sites/instances/processes/threads	2016-08-01 2018-02-01 2018-11-01
Microsoft.Web/sites/migratemyql	2016-08-01 2018-02-01 2018-11-01

Types	Versions
	2019-
	08-01
	2020-
	06-01
	2020-
	09-01
	2020-
	10-01
	2020-
	12-01
	2021-
	01-01
	2021-
	01-15
	2021-
	02-01
	2021-
	03-01
	2022-
	03-01
	2022-
	09-01
	2023-
	01-01
	2023-
	12-01
	2024-
	04-01
Microsoft.Web/sites/networkConfig	2018-
	02-01
	2018-
	11-01
	2019-
	08-01
	2020-
	06-01
	2020-
	09-01
	2020-
	10-01
	2020-
	12-01
	2021-
	01-01
	2021-
	01-15
	2021-
	02-01
	2021-
	03-01

Types	Versions
	2022-03-01 2022-09-01 2023-01-01 2023-02-01 2023-12-01 2024-01-01 2024-04-01
Microsoft.Web/sites/networkFeatures	2015-08-01 2016-08-01 2018-02-01 2018-11-01 2019-08-01 2020-06-01 2020-09-01 2020-10-01 2020-12-01 2021-01-01 2021-01-15 2021-02-01 2021-03-01 2022-03-01 2022-09-01 2023-01-01 2023-12-01 2024-01-01 2024-04-01
Microsoft.Web/sites/premieraddons	2015-08-01 2016-

Types	Versions
	08-01
	2018-
	02-01
	2018-
	11-01
	2019-
	08-01
	2020-
	06-01
	2020-
	09-01
	2020-
	10-01
	2020-
	12-01
	2021-
	01-01
	2021-
	01-15
	2021-
	02-01
	2021-
	03-01
	2022-
	03-01
	2022-
	09-01
	2023-
	01-01
	2023-
	12-01
	2024-
	04-01
Microsoft.Web/sites/privateAccess	2018-
	02-01
	2018-
	11-01
	2019-
	08-01
	2020-
	06-01
	2020-
	09-01
	2020-
	10-01
	2020-
	12-01
	2021-
	01-01
	2021-

Types	Versions
	01-15 2021- 02-01 2021- 03-01 2022- 03-01 2022- 09-01 2023- 01-01 2023- 12-01 2024- 04-01
Microsoft.Web/sites/privateEndpointConnections	2019- 08-01 2020- 06-01 2020- 09-01 2020- 10-01 2020- 12-01 2021- 01-01 2021- 01-15 2021- 02-01 2021- 03-01 2022- 03-01 2022- 09-01 2023- 01-01 2023- 12-01 2024- 04-01
Microsoft.Web/sites/processes	2016- 08-01 2018- 02-01 2018- 11-01

Types	Versions
	2019-
	08-01
	2020-
	06-01
	2020-
	09-01
	2020-
	10-01
	2020-
	12-01
	2021-
	01-01
	2021-
	01-15
	2021-
	02-01
	2021-
	03-01
	2022-
	03-01
	2022-
	09-01
	2023-
	01-01
	2023-
	12-01
	2024-
	04-01
Microsoft.Web/sites/processes/modules	2016-
	08-01
	2018-
	02-01
	2018-
	11-01
	2019-
	08-01
	2020-
	06-01
	2020-
	09-01
	2020-
	10-01
	2020-
	12-01
	2021-
	01-01
	2021-
	01-15
	2021-
	02-01

Types	Versions
	2021-03-01 2022-03-01 2022-09-01 2023-01-01 2023-12-01 2024-04-01
Microsoft.Web/sites/processes/threads	2016-08-01 2018-02-01 2018-11-01
Microsoft.Web/sites/publicCertificates	2016-08-01 2018-02-01 2018-11-01 2019-08-01 2020-06-01 2020-09-01 2020-10-01 2020-12-01 2021-01-01 2021-01-15 2021-02-01 2021-03-01 2022-03-01 2022-09-01 2023-01-01 2023-04-01

Types	Versions
	12-01 2024-04-01
Microsoft.Web/sites/recommendations	2016-03-01 2018-02-01 2019-08-01 2020-06-01 2020-09-01 2020-10-01 2020-12-01 2021-01-01 2021-01-15 2021-02-01 2021-03-01 2022-03-01 2022-09-01 2023-01-01 2023-12-01 2024-04-01
Microsoft.Web/sites/resourceHealthMetadata	2016-03-01 2018-02-01 2019-08-01 2020-06-01 2020-09-01 2020-10-01 2020-12-01

Types	Versions
	2021-
	01-01
	2021-
	01-15
	2021-
	02-01
	2021-
	03-01
	2022-
	03-01
	2022-
	09-01
	2023-
	01-01
	2023-
	12-01
	2024-
	04-01
Microsoft.Web/sites/sitecontainers	2023-
	12-01
	2024-
	04-01
Microsoft.Web/sites/siteextensions	2016-
	08-01
	2018-
	02-01
	2018-
	11-01
	2019-
	08-01
	2020-
	06-01
	2020-
	09-01
	2020-
	10-01
	2020-
	12-01
	2021-
	01-01
	2021-
	01-15
	2021-
	02-01
	2021-
	03-01
	2022-
	03-01
	2022-

Types	Versions
	09-01 2023- 01-01 2023- 12-01 2024- 04-01
Microsoft.Web/sites/slots	2015- 08-01 2016- 08-01 2018- 02-01 2018- 11-01 2019- 08-01 2020- 06-01 2020- 09-01 2020- 10-01 2020- 12-01 2021- 01-01 2021- 01-15 2021- 02-01 2021- 03-01 2022- 03-01 2022- 09-01 2023- 01-01 2023- 12-01 2024- 04-01
Microsoft.Web/sites/slots/backups	2015- 08-01 2016- 08-01 2018- 02-01

Types	Versions
	2018-
	11-01
	2019-
	08-01
	2020-
	06-01
	2020-
	09-01
	2020-
	10-01
	2020-
	12-01
	2021-
	01-01
	2021-
	01-15
	2021-
	02-01
	2021-
	03-01
	2022-
	03-01
	2022-
	09-01
	2023-
	01-01
	2023-
	12-01
	2024-
	04-01
Microsoft.Web/sites/slots/basicPublishingCredentialsPolicies	2020-
	12-01
	2021-
	01-01
	2021-
	01-15
	2021-
	02-01
	2021-
	03-01
	2022-
	03-01
	2022-
	09-01
	2023-
	01-01
	2023-
	12-01

Types	Versions
	2024-04-01
Microsoft.Web/sites/slots/config	2015-08-01 2016-08-01 2018-02-01 2018-11-01 2019-08-01 2020-06-01 2020-09-01 2020-10-01 2020-12-01 2021-01-01 2021-01-15 2021-02-01 2021-03-01 2022-03-01 2022-09-01 2023-01-01 2023-12-01 2024-04-01
Microsoft.Web/sites/slots/config/appsettings	2020-12-01 2021-01-01 2021-01-15 2021-02-01 2021-03-01 2022-04-01

Types	Versions
	03-01 2022- 09-01 2023- 01-01 2023- 12-01 2024- 04-01
Microsoft.Web/sites/slots/config/connectionstrings	2020- 12-01 2021- 01-01 2021- 01-15 2021- 02-01 2021- 03-01 2022- 03-01 2022- 09-01 2023- 01-01 2023- 12-01 2024- 04-01
Microsoft.Web/sites/slots/config/snapshots	2016- 08-01 2018- 02-01 2018- 11-01 2019- 08-01 2020- 06-01 2020- 09-01 2020- 10-01 2020- 12-01 2021- 01-01 2021- 01-15

Types	Versions
	2021-
	02-01
	2021-
	03-01
	2022-
	03-01
	2022-
	09-01
	2023-
	01-01
	2023-
	12-01
	2024-
	04-01
Microsoft.Web/sites/slots/continuouswebjobs	2016-
	08-01
	2018-
	02-01
	2018-
	11-01
	2019-
	08-01
	2020-
	06-01
	2020-
	09-01
	2020-
	10-01
	2020-
	12-01
	2021-
	01-01
	2021-
	01-15
	2021-
	02-01
	2021-
	03-01
	2022-
	03-01
	2022-
	09-01
	2023-
	01-01
	2023-
	12-01
	2024-
	04-01

Types	Versions
Microsoft.Web/sites/slots/deployments	2015-08-01 2016-08-01 2018-02-01 2018-11-01 2019-08-01 2020-06-01 2020-09-01 2020-10-01 2020-12-01 2021-01-01 2021-01-15 2021-02-01 2021-03-01 2022-03-01 2022-09-01 2023-01-01 2023-12-01 2024-04-01
Microsoft.Web/sites/slots/deploymentStatus	2022-03-01 2022-09-01 2023-01-01 2023-12-01 2024-04-01
Microsoft.Web/sites/slots/detectors	2016-03-01

Types	Versions
	2018-
	02-01
	2019-
	08-01
	2020-
	06-01
	2020-
	09-01
	2020-
	10-01
	2020-
	12-01
	2021-
	01-01
	2021-
	01-15
	2021-
	02-01
	2021-
	03-01
	2022-
	03-01
	2022-
	09-01
	2023-
	01-01
	2023-
	12-01
	2024-
	04-01
Microsoft.Web/sites/slots/diagnostics	2016-
	03-01
	2018-
	02-01
	2019-
	08-01
	2020-
	06-01
	2020-
	09-01
	2020-
	10-01
	2020-
	12-01
	2021-
	01-01
	2021-
	01-15
	2021-
	02-01

Types	Versions
	2021-03-01 2022-03-01 2022-09-01 2023-01-01 2023-12-01 2024-04-01
Microsoft.Web/sites/slots/diagnostics/analyses	2016-03-01 2018-02-01 2019-08-01 2020-06-01 2020-09-01 2020-10-01 2020-12-01 2021-01-01 2021-01-15 2021-02-01 2021-03-01 2022-03-01 2022-09-01 2023-01-01 2023-12-01 2024-04-01
Microsoft.Web/sites/slots/diagnostics/detectors	2019-08-01 2020-06-01 2020-

Types	Versions
	09-01
	2020-
	10-01
	2020-
	12-01
	2021-
	01-01
	2021-
	01-15
	2021-
	02-01
	2021-
	03-01
	2022-
	03-01
	2022-
	09-01
	2023-
	01-01
	2023-
	12-01
	2024-
	04-01
Microsoft.Web/sites/slots/domainOwnershipIdentifiers	2016-
	08-01
	2018-
	02-01
	2018-
	11-01
	2019-
	08-01
	2020-
	06-01
	2020-
	09-01
	2020-
	10-01
	2020-
	12-01
	2021-
	01-01
	2021-
	01-15
	2021-
	02-01
	2021-
	03-01
	2022-
	03-01
	2022-

Types	Versions
	09-01 2023- 01-01 2023- 12-01 2024- 04-01
Microsoft.Web/sites/slots/extensions	2016- 08-01 2018- 02-01 2018- 11-01 2019- 08-01 2020- 06-01 2020- 09-01 2020- 10-01 2020- 12-01 2021- 01-01 2021- 01-15 2021- 02-01 2021- 03-01 2022- 03-01 2022- 09-01 2023- 01-01 2023- 12-01 2024- 04-01
Microsoft.Web/sites/slots/functions	2016- 08-01 2018- 02-01 2018- 11-01 2019- 08-01

Types	Versions
	2020-
	06-01
	2020-
	09-01
	2020-
	10-01
	2020-
	12-01
	2021-
	01-01
	2021-
	01-15
	2021-
	02-01
	2021-
	03-01
	2022-
	03-01
	2022-
	09-01
	2023-
	01-01
	2023-
	12-01
	2024-
	04-01
Microsoft.Web/sites/slots/functions/keys	2018-
	02-01
	2019-
	08-01
	2020-
	06-01
	2020-
	09-01
	2020-
	10-01
	2020-
	12-01
	2021-
	01-01
	2021-
	01-15
	2021-
	02-01
	2021-
	03-01
	2022-
	03-01
	2022-
	09-01

Types	Versions
	2023-01-01 2023-12-01 2024-04-01
Microsoft.Web/sites/slots/hostNameBindings	2015-08-01 2016-08-01 2018-02-01 2018-11-01 2019-08-01 2020-06-01 2020-09-01 2020-10-01 2020-12-01 2021-01-01 2021-01-15 2021-02-01 2021-03-01 2022-03-01 2022-09-01 2023-01-01 2023-12-01 2024-04-01
Microsoft.Web/sites/slots/hybridconnection	2015-08-01 2016-08-01 2018-02-01 2018-

Types	Versions
	11-01
	2019-
	08-01
	2020-
	06-01
	2020-
	09-01
	2020-
	10-01
	2020-
	12-01
	2021-
	01-01
	2021-
	01-15
	2021-
	02-01
	2021-
	03-01
	2022-
	03-01
	2022-
	09-01
	2023-
	01-01
	2023-
	12-01
	2024-
	04-01
Microsoft.Web/sites/slots/hybridConnectionNamespaces/relays	2016-
	08-01
	2018-
	02-01
	2018-
	11-01
	2019-
	08-01
	2020-
	06-01
	2020-
	09-01
	2020-
	10-01
	2020-
	12-01
	2021-
	01-01
	2021-
	01-15
	2021-

Types	Versions
	02-01 2021- 03-01 2022- 03-01 2022- 09-01 2023- 01-01 2023- 12-01 2024- 04-01
Microsoft.Web/sites/slots/instances	2019- 08-01 2020- 06-01 2020- 09-01 2020- 10-01 2020- 12-01 2021- 01-01 2021- 01-15 2021- 02-01 2021- 03-01 2022- 03-01 2022- 09-01 2023- 01-01 2023- 12-01 2024- 04-01
Microsoft.Web/sites/slots/instances/deployments	2015- 08-01
Microsoft.Web/sites/slots/instances/extensions	2016- 08-01 2018- 02-01 2018-

Types	Versions
	11-01
	2019-
	08-01
	2020-
	06-01
	2020-
	09-01
	2020-
	10-01
	2020-
	12-01
	2021-
	01-01
	2021-
	01-15
	2021-
	02-01
	2021-
	03-01
	2022-
	03-01
	2022-
	09-01
	2023-
	01-01
	2023-
	12-01
	2024-
	04-01
Microsoft.Web/sites/slots/instances/processes	2016-
	08-01
	2018-
	02-01
	2018-
	11-01
	2019-
	08-01
	2020-
	06-01
	2020-
	09-01
	2020-
	10-01
	2020-
	12-01
	2021-
	01-01
	2021-
	01-15
	2021-

Types	Versions
	02-01 2021- 03-01 2022- 03-01 2022- 09-01 2023- 01-01 2023- 12-01 2024- 04-01
Microsoft.Web/sites/slots/instances/processes/modules	2016- 08-01 2018- 02-01 2018- 11-01 2019- 08-01 2020- 06-01 2020- 09-01 2020- 10-01 2020- 12-01 2021- 01-01 2021- 01-15 2021- 02-01 2021- 03-01 2022- 03-01 2022- 09-01 2023- 01-01 2023- 12-01 2024- 04-01
Microsoft.Web/sites/slots/instances/processes/threads	2016- 08-01

Types	Versions
	2018-02-01 2018-11-01
Microsoft.Web/sites/slots/migratemysql	2016-08-01 2018-02-01 2018-11-01 2019-08-01 2020-06-01 2020-09-01 2020-10-01 2020-12-01 2021-01-01 2021-01-15 2021-02-01 2021-03-01 2022-03-01 2022-09-01 2023-01-01 2023-12-01 2024-04-01
Microsoft.Web/sites/slots/networkConfig	2018-02-01 2018-11-01 2019-08-01 2020-06-01 2020-09-01 2020-

Types	Versions
	10-01
	2020-
	12-01
	2021-
	01-01
	2021-
	01-15
	2021-
	02-01
	2021-
	03-01
	2022-
	03-01
	2022-
	09-01
	2023-
	01-01
	2023-
	12-01
	2024-
	04-01
Microsoft.Web/sites/slots/networkFeatures	2015-
	08-01
	2016-
	08-01
	2018-
	02-01
	2018-
	11-01
	2019-
	08-01
	2020-
	06-01
	2020-
	09-01
	2020-
	10-01
	2020-
	12-01
	2021-
	01-01
	2021-
	01-15
	2021-
	02-01
	2021-
	03-01
	2022-
	03-01
	2022-

Types	Versions
	09-01 2023- 01-01 2023- 12-01 2024- 04-01
Microsoft.Web/sites/slots/premieraddons	2015- 08-01 2016- 08-01 2018- 02-01 2018- 11-01 2019- 08-01 2020- 06-01 2020- 09-01 2020- 10-01 2020- 12-01 2021- 01-01 2021- 01-15 2021- 02-01 2021- 03-01 2022- 03-01 2022- 09-01 2023- 01-01 2023- 12-01 2024- 04-01
Microsoft.Web/sites/slots/privateAccess	2018- 02-01 2018- 11-01 2019- 08-01

Types	Versions
	2020-06-01 2020-09-01 2020-10-01 2020-12-01 2021-01-01 2021-01-15 2021-02-01 2021-03-01 2022-03-01 2022-09-01 2023-01-01 2023-12-01 2024-04-01
Microsoft.Web/sites/slots/privateEndpointConnections	2020-12-01 2021-01-01 2021-01-15 2021-02-01 2021-03-01 2022-03-01 2022-09-01 2023-01-01 2023-12-01 2024-04-01
Microsoft.Web/sites/slots/processes	2016-08-01 2018-

Types	Versions
	02-01
	2018-
	11-01
	2019-
	08-01
	2020-
	06-01
	2020-
	09-01
	2020-
	10-01
	2020-
	12-01
	2021-
	01-01
	2021-
	01-15
	2021-
	02-01
	2021-
	03-01
	2022-
	03-01
	2022-
	09-01
	2023-
	01-01
	2023-
	12-01
	2024-
	04-01
Microsoft.Web/sites/slots/processes/modules	2016-
	08-01
	2018-
	02-01
	2018-
	11-01
	2019-
	08-01
	2020-
	06-01
	2020-
	09-01
	2020-
	10-01
	2020-
	12-01
	2021-
	01-01
	2021-

Types	Versions
	01-15
	2021-
	02-01
	2021-
	03-01
	2022-
	03-01
	2022-
	09-01
	2023-
	01-01
	2023-
	12-01
	2024-
	04-01
Microsoft.Web/sites/slots/processes/threads	2016-
	08-01
	2018-
	02-01
	2018-
	11-01
Microsoft.Web/sites/slots/publicCertificates	2016-
	08-01
	2018-
	02-01
	2018-
	11-01
	2019-
	08-01
	2020-
	06-01
	2020-
	09-01
	2020-
	10-01
	2020-
	12-01
	2021-
	01-01
	2021-
	01-15
	2021-
	02-01
	2021-
	03-01
	2022-
	03-01
	2022-
	09-01

Types	Versions
	2023-01-01 2023-12-01 2024-04-01
Microsoft.Web/sites/slots/resourceHealthMetadata	2016-03-01 2018-02-01 2019-08-01 2020-06-01 2020-09-01 2020-10-01 2020-12-01 2021-01-01 2021-01-15 2021-02-01 2021-03-01 2022-03-01 2022-09-01 2023-01-01 2023-12-01 2024-04-01
Microsoft.Web/sites/slots/sitecontainers	2023-12-01 2024-04-01
Microsoft.Web/sites/slotsextensions	2016-08-01 2018-02-01 2018-11-01

Types	Versions
	2019-
	08-01
	2020-
	06-01
	2020-
	09-01
	2020-
	10-01
	2020-
	12-01
	2021-
	01-01
	2021-
	01-15
	2021-
	02-01
	2021-
	03-01
	2022-
	03-01
	2022-
	09-01
	2023-
	01-01
	2023-
	12-01
	2024-
	04-01
Microsoft.Web/sites/slots/sourcecontrols	2015-
	08-01
	2016-
	08-01
	2018-
	02-01
	2018-
	11-01
	2019-
	08-01
	2020-
	06-01
	2020-
	09-01
	2020-
	10-01
	2020-
	12-01
	2021-
	01-01
	2021-
	01-15

Types	Versions
	2021-
	02-01
	2021-
	03-01
	2022-
	03-01
	2022-
	09-01
	2023-
	01-01
	2023-
	12-01
	2024-
	04-01
Microsoft.Web/sites/slots/triggeredwebjobs	2016-
	08-01
	2018-
	02-01
	2018-
	11-01
	2019-
	08-01
	2020-
	06-01
	2020-
	09-01
	2020-
	10-01
	2020-
	12-01
	2021-
	01-01
	2021-
	01-15
	2021-
	02-01
	2021-
	03-01
	2022-
	03-01
	2022-
	09-01
	2023-
	01-01
	2023-
	12-01
	2024-
	04-01

Types	Versions
Microsoft.Web/sites/slots/triggeredwebjobs/history	2016-08-01 2018-02-01 2018-11-01 2019-08-01 2020-06-01 2020-09-01 2020-10-01 2020-12-01 2021-01-01 2021-01-15 2021-02-01 2021-03-01 2022-03-01 2022-09-01 2023-01-01 2023-12-01 2024-04-01
Microsoft.Web/sites/slots/virtualNetworkConnections	2015-08-01 2016-08-01 2018-02-01 2018-11-01 2019-08-01 2020-06-01 2020-09-01 2020-

Types	Versions
	10-01
	2020-
	12-01
	2021-
	01-01
	2021-
	01-15
	2021-
	02-01
	2021-
	03-01
	2022-
	03-01
	2022-
	09-01
	2023-
	01-01
	2023-
	12-01
	2024-
	04-01
Microsoft.Web/sites/slots/virtualNetworkConnections/gateways	2015-
	08-01
	2016-
	08-01
	2018-
	02-01
	2018-
	11-01
	2019-
	08-01
	2020-
	06-01
	2020-
	09-01
	2020-
	10-01
	2020-
	12-01
	2021-
	01-01
	2021-
	01-15
	2021-
	02-01
	2021-
	03-01
	2022-
	03-01
	2022-

Types	Versions
	09-01 2023- 01-01 2023- 12-01 2024- 04-01
Microsoft.Web/sites/slots/webjobs	2016- 08-01 2018- 02-01 2018- 11-01 2019- 08-01 2020- 06-01 2020- 09-01 2020- 10-01 2020- 12-01 2021- 01-01 2021- 01-15 2021- 02-01 2021- 03-01 2022- 03-01 2022- 09-01 2023- 01-01 2023- 12-01 2024- 04-01
Microsoft.Web/sites/sourcecontrols	2015- 08-01 2016- 08-01 2018- 02-01 2018- 11-01

Types	Versions
	2019-
	08-01
	2020-
	06-01
	2020-
	09-01
	2020-
	10-01
	2020-
	12-01
	2021-
	01-01
	2021-
	01-15
	2021-
	02-01
	2021-
	03-01
	2022-
	03-01
	2022-
	09-01
	2023-
	01-01
	2023-
	12-01
	2024-
	04-01
Microsoft.Web/sites/triggeredwebjobs	2016-
	08-01
	2018-
	02-01
	2018-
	11-01
	2019-
	08-01
	2020-
	06-01
	2020-
	09-01
	2020-
	10-01
	2020-
	12-01
	2021-
	01-01
	2021-
	01-15
	2021-
	02-01

Types	Versions
	2021-03-01 2022-03-01 2022-09-01 2023-01-01 2023-12-01 2024-04-01
Microsoft.Web/sites/triggeredwebjobs/history	2016-08-01 2018-02-01 2018-11-01 2019-08-01 2020-06-01 2020-09-01 2020-10-01 2020-12-01 2021-01-01 2021-01-15 2021-02-01 2021-03-01 2022-03-01 2022-09-01 2023-01-01 2023-12-01 2024-04-01
Microsoft.Web/sites/virtualNetworkConnections	2015-08-01 2016-

Types	Versions
	08-01
	2018-
	02-01
	2018-
	11-01
	2019-
	08-01
	2020-
	06-01
	2020-
	09-01
	2020-
	10-01
	2020-
	12-01
	2021-
	01-01
	2021-
	01-15
	2021-
	02-01
	2021-
	03-01
	2022-
	03-01
	2022-
	09-01
	2023-
	01-01
	2023-
	12-01
	2024-
	04-01
Microsoft.Web/sites/virtualNetworkConnections/gateways	2015-
	08-01
	2016-
	08-01
	2018-
	02-01
	2018-
	11-01
	2019-
	08-01
	2020-
	06-01
	2020-
	09-01
	2020-
	10-01
	2020-

Types	Versions
	12-01
	2021-
	01-01
	2021-
	01-15
	2021-
	02-01
	2021-
	03-01
	2022-
	03-01
	2022-
	09-01
	2023-
	01-01
	2023-
	12-01
	2024-
	04-01
Microsoft.Web/sites/webjobs	2016-
	08-01
	2018-
	02-01
	2018-
	11-01
	2019-
	08-01
	2020-
	06-01
	2020-
	09-01
	2020-
	10-01
	2020-
	12-01
	2021-
	01-01
	2021-
	01-15
	2021-
	02-01
	2021-
	03-01
	2022-
	03-01
	2022-
	09-01
	2023-
	01-01
	2023-

Types	Versions
	12-01 2024-04-01
Microsoft.Web/sourcecontrols	2015-08-01 2016-03-01 2018-02-01 2019-08-01 2020-06-01 2020-09-01 2020-10-01 2020-12-01 2021-01-01 2021-01-15 2021-02-01 2021-03-01 2022-03-01 2022-09-01 2023-01-01 2023-12-01 2024-04-01
Microsoft.Web/staticSites	2019-08-01 2020-06-01 2020-09-01 2020-10-01 2020-12-01 2021-01-01

Types	Versions
	2021-01-15 2021-02-01 2021-03-01 2022-03-01 2022-09-01 2023-01-01 2023-12-01 2024-04-01
Microsoft.Web/staticSites/basicAuth	2022-09-01 2023-01-01 2023-12-01 2024-04-01
Microsoft.Web/staticSites/builds	2019-08-01 2020-06-01 2020-09-01 2020-10-01 2020-12-01 2021-01-01 2021-01-15 2021-02-01 2021-03-01 2022-03-01 2022-09-01 2023-01-01 2023-04-01

Types	Versions
	12-01 2024-04-01
Microsoft.Web/staticSites/builds/config	2019-08-01 2020-06-01 2020-09-01 2020-10-01 2020-12-01 2021-01-01 2021-01-15 2021-02-01 2021-03-01 2022-03-01 2022-09-01 2023-01-01 2023-12-01 2024-04-01
Microsoft.Web/staticSites/builds/databaseConnections	2022-09-01 2023-01-01 2023-12-01 2024-04-01
Microsoft.Web/staticSites/builds/linkedBackends	2022-03-01 2022-09-01 2023-01-01 2023-12-01

Types	Versions
	2024-04-01
Microsoft.Web/staticSites/builds/userProvidedFunctionApps	2020-12-01 2021-01-01 2021-01-15 2021-02-01 2021-03-01 2022-03-01 2022-09-01 2023-01-01 2023-12-01 2024-04-01
Microsoft.Web/staticSites/config	2019-08-01 2020-06-01 2020-09-01 2020-10-01 2020-12-01 2021-01-01 2021-01-15 2021-02-01 2021-03-01 2022-03-01 2022-09-01 2023-01-01 2023-12-01

Types	Versions
	2024-04-01
Microsoft.Web/staticSites/customDomains	2019-08-01 2020-06-01 2020-09-01 2020-10-01 2020-12-01 2021-01-01 2021-01-15 2021-02-01 2021-03-01 2022-03-01 2022-09-01 2023-01-01 2023-12-01 2024-04-01
Microsoft.Web/staticSites/databaseConnections	2022-09-01 2023-01-01 2023-12-01 2024-04-01
Microsoft.Web/staticSites/linkedBackends	2022-03-01 2022-09-01 2023-01-01 2023-12-01 2024-04-01

Types	Versions
Microsoft.Web/staticSites/privateEndpointConnections	2020-12-01 2021-01-01 2021-01-15 2021-02-01 2021-03-01 2022-03-01 2022-09-01 2023-01-01 2023-01-15 2023-02-01 2023-03-01 2023-09-01 2024-01-01 2024-04-01
Microsoft.Web/staticSites/userProvidedFunctionApps	2020-12-01 2021-01-01 2021-01-15 2021-02-01 2021-03-01 2022-03-01 2022-09-01 2023-01-01 2023-01-15 2023-02-01 2023-03-01 2023-09-01 2024-01-01 2024-04-01

Feedback

Was this page helpful?

 Yes

 No

Monitor Azure App Service

Article • 07/10/2024

This article describes:

- The types of monitoring data you can collect for this service.
- Ways to analyze that data.

ⓘ Note

If you're already familiar with this service and/or Azure Monitor and just want to know how to analyze monitoring data, see the [Analyze](#) section near the end of this article.

When you have critical applications and business processes that rely on Azure resources, you need to monitor and get alerts for your system. The Azure Monitor service collects and aggregates metrics and logs from every component of your system. Azure Monitor provides you with a view of availability, performance, and resilience, and notifies you of issues. You can use the Azure portal, PowerShell, Azure CLI, REST API, or client libraries to set up and view monitoring data.

- For more information on Azure Monitor, see the [Azure Monitor overview](#).
- For more information on how to monitor Azure resources in general, see [Monitor Azure resources with Azure Monitor](#).

App Service monitoring

Azure App Service provides several monitoring options for monitoring resources for availability, performance, and operation. Options include Diagnostic Settings, Application Insights, log stream, metrics, quotas and alerts, and activity logs.

On the Azure portal page for your web app, you can select **Diagnose and solve problems** from the left navigation to access complete App Service diagnostics for your app. For more information about the App Service diagnostics tool, see [Azure App Service diagnostics overview](#).

App Service provides built-in diagnostics logging to assist with debugging apps. For more information about the built-in logs, see [Stream diagnostics logs](#).

You can also use Azure Health check to monitor App Service instances. For more information, see [Monitor App Service instances using Health check](#).

If you're using ASP.NET Core, ASP.NET, Java, Node.js, or Python, we recommend [enabling observability with Application Insights](#). To learn more about observability experiences offered by Application Insights, see [Application Insights overview](#).

Monitoring scenarios

The following table lists monitoring methods to use for different scenarios.

[] [Expand table](#)

Scenario	Monitoring method
I want to monitor platform metrics and logs	Azure Monitor platform metrics
I want to monitor application performance and usage	(Azure Monitor) Application Insights
I want to monitor built-in logs for testing and development	Log stream
I want to monitor resource limits and configure alerts	Quotas and alerts
I want to monitor web app resource events	Activity logs
I want to monitor metrics visually	Metrics

Insights

Some services in Azure have a built-in monitoring dashboard in the Azure portal that provides a starting point for monitoring your service. These dashboards are called *insights*, and you can find them in the **Insights Hub** of Azure Monitor in the Azure portal.

Application Insights

Application Insights uses the powerful data analysis platform in Azure Monitor to provide you with deep insights into your application's operations. Application Insights monitors the availability, performance, and usage of your web applications, so you can identify and diagnose errors without waiting for a user to report them.

Application Insights includes connection points to various development tools and integrates with Visual Studio to support your DevOps processes. For more information, see [Application monitoring for App Service](#).

Resource types

Azure uses the concept of resource types and IDs to identify everything in a subscription. Azure Monitor similarly organizes core monitoring data into metrics and logs based on resource types, also called *namespaces*. Different metrics and logs are available for different resource types. Your service might be associated with more than one resource type.

Resource types are also part of the resource IDs for every resource running in Azure. For example, one resource type for a virtual machine is `Microsoft.Compute/virtualMachines`. For a list of services and their associated resource types, see [Resource providers](#).

For more information about the resource types for App Service, see [App Service monitoring data reference](#).

Data storage

For Azure Monitor:

- Metrics data is stored in the Azure Monitor metrics database.
- Log data is stored in the Azure Monitor logs store. Log Analytics is a tool in the Azure portal that can query this store.
- The Azure activity log is a separate store with its own interface in the Azure portal.

You can optionally route metric and activity log data to the Azure Monitor logs store. You can then use Log Analytics to query the data and correlate it with other log data.

Many services can use diagnostic settings to send metric and log data to other storage locations outside Azure Monitor. Examples include Azure Storage, [hosted partner systems](#), and [non-Azure partner systems](#), by using Event Hubs.

For detailed information on how Azure Monitor stores data, see [Azure Monitor data platform](#).

Azure Monitor platform metrics

Azure Monitor provides platform metrics for most services. These metrics are:

- Individually defined for each namespace.
- Stored in the Azure Monitor time-series metrics database.
- Lightweight and capable of supporting near real-time alerting.
- Used to track the performance of a resource over time.

Collection: Azure Monitor collects platform metrics automatically. No configuration is required.

Routing: You can also usually route platform metrics to Azure Monitor Logs / Log Analytics so you can query them with other log data. For more information, see the [Metrics diagnostic setting](#). For how to configure diagnostic settings for a service, see [Create diagnostic settings in Azure Monitor](#).

For a list of all metrics it's possible to gather for all resources in Azure Monitor, see [Supported metrics in Azure Monitor](#).

For a list of available metrics for App Service, see [App Service monitoring data reference](#).

For help understanding metrics in App Service, see [Understand metrics](#). Metrics can be viewed by aggregates on data (ie. average, max, min, etc.), instances, time range, and other filters. Metrics can monitor performance, memory, CPU, and other attributes.

Azure Monitor resource logs

Resource logs provide insight into operations that were done by an Azure resource. Logs are generated automatically, but you must route them to Azure Monitor logs to save or query them. Logs are organized in categories. A given namespace might have multiple resource log categories.

Collection: Resource logs aren't collected and stored until you create a *diagnostic setting* and route the logs to one or more locations. When you create a diagnostic setting, you specify which categories of logs to collect. There are multiple ways to create and maintain diagnostic settings, including the Azure portal, programmatically, and through Azure Policy.

Routing: The suggested default is to route resource logs to Azure Monitor Logs so you can query them with other log data. Other locations such as Azure Storage, Azure Event Hubs, and certain Microsoft monitoring partners are also available. For more information, see [Azure resource logs](#) and [Resource log destinations](#).

For detailed information about collecting, storing, and routing resource logs, see [Diagnostic settings in Azure Monitor](#).

For a list of all available resource log categories in Azure Monitor, see [Supported resource logs in Azure Monitor](#).

All resource logs in Azure Monitor have the same header fields, followed by service-specific fields. The common schema is outlined in [Azure Monitor resource log schema](#).

For the available resource log categories, their associated Log Analytics tables, and the logs schemas for App Service, see [App Service monitoring data reference](#).

💡 Tip

Logs are grouped into Category groups. Category groups are a collection of different logs to help you achieve different monitoring goals.

The **audit** category group allows you to select the resource logs that are necessary for auditing your resource. For more information, see [Diagnostic settings in Azure Monitor Resource logs](#).

Azure activity log

The activity log contains subscription-level events that track operations for each Azure resource as seen from outside that resource; for example, creating a new resource or starting a virtual machine.

Collection: Activity log events are automatically generated and collected in a separate store for viewing in the Azure portal.

Routing: You can send activity log data to Azure Monitor Logs so you can analyze it alongside other log data. Other locations such as Azure Storage, Azure Event Hubs, and certain Microsoft monitoring partners are also available. For more information on how to route the activity log, see [Overview of the Azure activity log](#).

Azure activity logs for App Service

Azure activity logs for App Service include details such as:

- What operations were taken on the resources (ex: App Service Plans)
- Who started the operation
- When the operation occurred
- Status of the operation
- Property values to help you research the operation

Azure activity logs can be queried using the Azure portal, PowerShell, REST API, or CLI.

Ship activity logs to Event Grid

While activity logs are user-based, there's a new [Azure Event Grid](#) integration with App Service (preview) that logs both user actions and automated events. With Event Grid, you can configure a handler to react to the said events. For example, use Event Grid to instantly trigger a serverless function to run image analysis each time a new photo is added to a blob storage container.

Alternatively, you can use Event Grid with Logic Apps to process data anywhere, without writing code. Event Grid connects data sources and event handlers.

To view the properties and schema for App Service events, see [Azure App Service as an Event Grid source](#).

Log stream (via App Service Logs)

Azure provides built-in diagnostics to assist during testing and development to debug an App Service app. [Log stream](#) can be used to get quick access to output and errors written by your application, and logs from the web server. These are standard output/error logs in addition to web server logs.

Analyze monitoring data

There are many tools for analyzing monitoring data.

Azure Monitor tools

Azure Monitor supports the following basic tools:

- [Metrics explorer](#), a tool in the Azure portal that allows you to view and analyze metrics for Azure resources. For more information, see [Analyze metrics with Azure Monitor metrics explorer](#).
- [Log Analytics](#), a tool in the Azure portal that allows you to query and analyze log data by using the [Kusto query language \(KQL\)](#). For more information, see [Get started with log queries in Azure Monitor](#).
- The [activity log](#), which has a user interface in the Azure portal for viewing and basic searches. To do more in-depth analysis, you have to route the data to Azure Monitor logs and run more complex queries in Log Analytics.

Tools that allow more complex visualization include:

- [Dashboards](#) that let you combine different kinds of data into a single pane in the Azure portal.
- [Workbooks](#), customizable reports that you can create in the Azure portal. Workbooks can include text, metrics, and log queries.
- [Grafana](#), an open platform tool that excels in operational dashboards. You can use Grafana to create dashboards that include data from multiple sources other than Azure Monitor.

- **Power BI**, a business analytics service that provides interactive visualizations across various data sources. You can configure Power BI to automatically import log data from Azure Monitor to take advantage of these visualizations.

Azure Monitor export tools

You can get data out of Azure Monitor into other tools by using the following methods:

- **Metrics**: Use the [REST API for metrics](#) to extract metric data from the Azure Monitor metrics database. The API supports filter expressions to refine the data retrieved. For more information, see [Azure Monitor REST API reference](#).
- **Logs**: Use the REST API or the [associated client libraries](#).
- Another option is the [workspace data export](#).

To get started with the REST API for Azure Monitor, see [Azure monitoring REST API walkthrough](#).

Kusto queries

You can analyze monitoring data in the Azure Monitor Logs / Log Analytics store by using the Kusto query language (KQL).

ⓘ Important

When you select **Logs** from the service's menu in the portal, Log Analytics opens with the query scope set to the current service. This scope means that log queries will only include data from that type of resource. If you want to run a query that includes data from other Azure services, select **Logs** from the **Azure Monitor** menu. See [Log query scope and time range in Azure Monitor Log Analytics](#) for details.

For a list of common queries for any service, see the [Log Analytics queries interface](#).

The following sample query can help you monitor app logs using `AppServiceAppLogs`:

Kusto

```
AppServiceAppLogs
| project CustomLevel, _ResourceId
| summarize count() by CustomLevel, _ResourceId
```

The following sample query can help you monitor HTTP logs using `AppServiceHTTPLogs` where the `HTTP response code` is `500` or higher:

Kusto

```
AppServiceHTTPLogs
//| where ResourceId = "MyResourceId" // Uncomment to get results for a
// specific resource Id when querying over a group of Apps
| where ScStatus >= 500
| reduce by strcat(CsMethod, ':\\\', CsUriStem)
```

The following sample query can help you monitor HTTP 500 errors by joining `AppServiceConsoleLogs` and `AppserviceHTTPLogs`:

Kusto

```
let myHttp = AppServiceHTTPLogs | where ScStatus == 500 | project
TimeGen=substring(TimeGenerated, 0, 19), CsUriStem, ScStatus;

let myConsole = AppServiceConsoleLogs | project
TimeGen=substring(TimeGenerated, 0, 19), ResultDescription;

myHttp | join myConsole on TimeGen | project TimeGen, CsUriStem, ScStatus,
ResultDescription;
```

See [Azure Monitor queries for App Service](#) for more sample queries.

Alerts

Azure Monitor alerts proactively notify you when specific conditions are found in your monitoring data. Alerts allow you to identify and address issues in your system before your customers notice them. For more information, see [Azure Monitor alerts](#).

There are many sources of common alerts for Azure resources. For examples of common alerts for Azure resources, see [Sample log alert queries](#). The [Azure Monitor Baseline Alerts \(AMBA\)](#) site provides a semi-automated method of implementing important platform metric alerts, dashboards, and guidelines. The site applies to a continually expanding subset of Azure services, including all services that are part of the Azure Landing Zone (ALZ).

The common alert schema standardizes the consumption of Azure Monitor alert notifications. For more information, see [Common alert schema](#).

Types of alerts

You can alert on any metric or log data source in the Azure Monitor data platform. There are many different types of alerts depending on the services you're monitoring and the monitoring data you're collecting. Different types of alerts have various benefits and drawbacks. For more information, see [Choose the right monitoring alert type](#).

The following list describes the types of Azure Monitor alerts you can create:

- [Metric alerts](#) evaluate resource metrics at regular intervals. Metrics can be platform metrics, custom metrics, logs from Azure Monitor converted to metrics, or Application Insights metrics. Metric alerts can also apply multiple conditions and dynamic thresholds.
- [Log alerts](#) allow users to use a Log Analytics query to evaluate resource logs at a predefined frequency.
- [Activity log alerts](#) trigger when a new activity log event occurs that matches defined conditions. Resource Health alerts and Service Health alerts are activity log alerts that report on your service and resource health.

Some Azure services also support [smart detection alerts](#), [Prometheus alerts](#), or [recommended alert rules](#).

For some services, you can monitor at scale by applying the same metric alert rule to multiple resources of the same type that exist in the same Azure region. Individual notifications are sent for each monitored resource. For supported Azure services and clouds, see [Monitor multiple resources with one alert rule](#).

 **Note**

If you're creating or running an application that runs on your service, [Azure Monitor application insights](#) might offer more types of alerts.

Quotas and alerts

Apps that are hosted in App Service are subject to certain limits on the resources they can use. [The limits](#) are defined by the App Service plan that's associated with the app. Metrics for an app or an App Service plan can be hooked up to alerts.

App Service alert rules

The following table lists common and recommended alert rules for App Service.

 Expand table

Alert type	Condition	Examples
Metric	Average connections	When number of connections exceed a set value
Metric	HTTP 404	When HTTP 404 responses exceed a set value
Metric	HTTP Server Errors	When HTTP 5xx errors exceed a set value
Activity Log	Create or Update Web App	When app is created or updated
Activity Log	Delete Web App	When app is deleted
Activity Log	Restart Web App	When app is restarted
Activity Log	Stop Web App	When app is stopped

Advisor recommendations

For some services, if critical conditions or imminent changes occur during resource operations, an alert displays on the service [Overview](#) page in the portal. You can find more information and recommended fixes for the alert in [Advisor recommendations](#) under [Monitoring](#) in the left menu. During normal operations, no advisor recommendations display.

For more information on Azure Advisor, see [Azure Advisor overview](#).

Related content

- See [App Service monitoring data reference](#) for a reference of the metrics, logs, and other important values created for App Service.
- See [Monitoring Azure resources with Azure Monitor](#) for general details on monitoring Azure resources.

Feedback

Was this page helpful?



[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

Azure App Service monitoring data reference

Article • 03/21/2024

This article contains all the monitoring reference information for this service.

See [Monitor Azure App Service](#) for details on the data you can collect for Azure App Service and how to use it.

Metrics

This section lists all the automatically collected platform metrics for this service. These metrics are also part of the global list of [all platform metrics supported in Azure Monitor](#).

For information on metric retention, see [Azure Monitor Metrics overview](#).

Supported metrics for Microsoft.Web

The following tables list the automatically collected platform metrics for App Service.

[] [Expand table](#)

Metric Type	Resource Provider / Type Namespace and link to individual metrics
Web apps	Microsoft.Web/sites
App Service Plans	Microsoft.Web/serverfarms
Staging slots	Microsoft.Web/sites/slots
App Service Environment	Microsoft.Web/hostingEnvironments
App Service Environment Front-end	Microsoft.Web/hostingEnvironments/multiRolePools
App Service Environment Worker Pools	Microsoft.Web/hostingEnvironments/workerPools

ⓘ Note

Azure App Service, Functions, and Logic Apps share the Microsoft.Web/sites namespace dating back to when they were a single service. Refer to the **Metric** column in the [Microsoft.Web/sites](#) table to see which metrics apply to which

services. The **Metrics** interface in the Azure portal for each service shows only the metrics that apply to that service.

 **Note**

App Service Plan metrics are available only for plans in *Basic*, *Standard*, and *Premium* tiers.

Metric dimensions

For information about what metric dimensions are, see [Multi-dimensional metrics](#).

This service has the following dimensions associated with its metrics.

Some metrics in the following namespaces have the listed dimensions:

Microsoft.Web/sites

- Instance
- workflowName
- status
- accountName

Microsoft.Web/serverFarms,

Microsoft.Web/sites/slots,

Microsoft.Web/hostingEnvironments,

Microsoft.Web/hostingenvironments/multirolepools,

Microsoft.Web/hostingenvironments/workerpools

- Instance

Resource logs

This section lists the types of resource logs you can collect for this service. The section pulls from the list of [all resource logs category types supported in Azure Monitor](#).

Supported resource logs for Microsoft.Web

- [Microsoft.Web/hostingEnvironments](#)
- [Microsoft.Web/sites](#)
- [Microsoft.Web/sites/slots](#)

The following table lists more information about resource logs you can collect for App Service.

[Expand table](#)

Log type	Windows	Windows Container	Linux	Linux Container	Description
AppServiceConsoleLogs	Java SE & Tomcat	Yes	Yes	Yes	Standard output and standard error
AppServiceHTTPLogs	Yes	Yes	Yes	Yes	Web server logs
AppServiceEnvironmentPlatformLogs	Yes	N/A	Yes	Yes	App Service Environment: scaling, configuration changes, and status logs
AppServiceAuditLogs	Yes	Yes	Yes	Yes	Login activity via FTP and Kudu
AppServiceFileAuditLogs	Yes	Yes	TBA	TBA	File changes made to the site content; only available for Premium tier and above
AppServiceAppLogs	ASP.NET	ASP.NET	Java SE & Tomcat Blessed Images ¹	Java SE & Tomcat Blessed Images ¹	Application logs
AppServiceIPSecAuditLogs	Yes	Yes	Yes	Yes	Requests from IP Rules
AppServicePlatformLogs	TBA	Yes	Yes	Yes	Container operation logs
AppServiceAntivirusScanAuditLogs	Yes	Yes	Yes	Yes	Anti-virus scan logs ↗

Log type	Windows	Windows Container	Linux	Linux Container	Description
					using Microsoft Defender for Cloud; only available for Premium tier

¹ For Java SE apps, add "\$WEBSITE_AZMON_PREVIEW_ENABLED" to the app settings and set it to 1 or to true.

Azure Monitor Logs tables

This section lists the Azure Monitor Logs tables relevant to this service, which are available for query by Log Analytics using Kusto queries. The tables contain resource log data and possibly more depending on what is collected and routed to them.

App Services

Microsoft.Web/sites

- [AzureActivity](#)
- [LogicAppWorkflowRuntime](#)
- [AppServiceAuthenticationLogs](#)
- [AppServiceServerlessSecurityPluginData](#)
- [AzureMetrics](#)
- [AppServiceAppLogs](#)
- [AppServiceAuditLogs](#)
- [AppServiceConsoleLogs](#)
- [AppServiceFileAuditLogs](#)
- [AppServiceHTTPLogs](#)
- [FunctionAppLogs](#)
- [AppServicePlatformLogs](#)
- [AppServiceIPSecAuditLogs](#)

Activity log

The linked table lists the operations that can be recorded in the activity log for this service. These operations are a subset of [all the possible resource provider operations in the activity log](#).

For more information on the schema of activity log entries, see [Activity Log schema](#).

The following table lists common activity log operations related to App Service. This list isn't exhaustive. For all Microsoft.Web resource provider operations, see [Microsoft.Web resource provider operations](#).

[] [Expand table](#)

Operation	Description
Create or Update Web App	App was created or updated
Delete Web App	App was deleted
Create Web App Backup	Backup of app
Get Web App Publishing Profile	Download of publishing profile
Publish Web App	App deployed
Restart Web App	App restarted
Start Web App	App started
Stop Web App	App stopped
Swap Web App Slots	Slots were swapped
Get Web App Slots Differences	Slot differences
Apply Web App Configuration	Applied configuration changes
Reset Web App Configuration	Configuration changes reset
Approve Private Endpoint Connections	Approved private endpoint connections
Network Trace Web Apps	Started network trace
Newpassword Web Apps	New password created
Get Zipped Container Logs for Web App	Get container logs
Restore Web App From Backup Blob	App restored from backup

Related content

- See [Monitor App Service](#) for a description of monitoring App Service.
- See [Monitor Azure resources with Azure Monitor](#) for details on monitoring Azure resources.

Tutorial: Troubleshoot an App Service app with Azure Monitor

Article • 06/29/2023

This tutorial shows how to troubleshoot an [App Service](#) app using [Azure Monitor](#). The sample app includes code meant to exhaust memory and cause HTTP 500 errors, so you can diagnose and fix the problem using Azure Monitor. When you're finished, you have a sample app running on App Service on Linux integrated with [Azure Monitor](#).

[Azure Monitor](#) maximizes the availability and performance of your applications and services by delivering a comprehensive solution for collecting, analyzing, and acting on telemetry from your cloud and on-premises environments.

In this tutorial, you learn how to:

- ✓ Configure a web app with Azure Monitor
- ✓ Send console logs to Log Analytics
- ✓ Use Log queries to identify and troubleshoot web app errors

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

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

Prerequisites

To complete this tutorial, you need:

- [Azure subscription](#)
 - [Git](#)
 - Use the Bash environment in [Azure Cloud Shell](#). For more information, see [Quickstart for Bash in Azure Cloud Shell](#).
- A blue rectangular button with a white 'A' icon on the left, the text 'Launch Cloud Shell' in the center, and a small blue square with a white arrow icon on the right.
- If you prefer to run CLI reference commands locally, [install](#) the Azure CLI. If you're running on Windows or macOS, consider running Azure CLI in a Docker container. For more information, see [How to run the Azure CLI in a Docker container](#).

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

Create Azure resources

First, you run several commands locally to set up a sample app to use with this tutorial. The commands create Azure resources, create a deployment user, and deploy the sample app to Azure. You're prompted for the password supplied as a part of the creation of the deployment user.

```
Azure CLI

az group create --name myResourceGroup --location "South Central US"
az webapp deployment user set --user-name <username> --password <password>
az appservice plan create --name myAppServicePlan --resource-group
myResourceGroup --sku B1 --is-linux
az webapp create --resource-group myResourceGroup --plan myAppServicePlan --
name <app-name> --runtime "PHP|8.1" --deployment-local-git
az webapp config appsettings set --name <app-name> --resource-group
myResourceGroup --settings DEPLOYMENT_BRANCH='main'
git clone https://github.com/Azure-Samples/App-Service-Troubleshoot-Azure-
Monitor
cd App-Service-Troubleshoot-Azure-Monitor
git branch -m main
git remote add azure <url-from-app-webapp-create>
git push azure main
```

Configure Azure Monitor

Create a Log Analytics Workspace

Now that you've deployed the sample app to Azure App Service, you configure monitoring capability to troubleshoot the app when problems arise. Azure Monitor stores log data in a Log Analytics workspace. A workspace is a container that includes data and configuration information.

In this step, you create a Log Analytics workspace to configure Azure Monitor with your app.

Azure CLI

```
az monitor log-analytics workspace create --resource-group myResourceGroup --workspace-name myMonitorWorkspace
```

ⓘ Note

For Azure Monitor Log Analytics, you pay for data ingestion and data retention. ↗

Create a diagnostic setting

Diagnostic settings can be used to collect metrics for certain Azure services into Azure Monitor Logs for analysis with other monitoring data using log queries. For this tutorial, you enable the web server and standard output/error logs. See [supported log types](#) for a complete list of log types and descriptions.

You run the following commands to create diagnostic settings for AppServiceConsoleLogs (standard output/error) and AppServiceHTTPLogs (web server logs). Replace <app-name> and <workspace-name> with your values.

ⓘ Note

The first two commands, `resourceID` and `workspaceID`, are variables to be used in the `az monitor diagnostic-settings create` command. See [Create diagnostic settings using Azure CLI](#) for more information on this command.

Azure CLI

```
resourceID=$(az webapp show -g myResourceGroup -n <app-name> --query id --output tsv)

workspaceID=$(az monitor log-analytics workspace show -g myResourceGroup --workspace-name <workspace-name> --query id --output tsv)

az monitor diagnostic-settings create --resource $resourceID \
--workspace $workspaceID \
-n myMonitorLogs \
--logs '[{"category": "AppServiceConsoleLogs", "enabled": true}, {"category": "AppServiceHTTPLogs", "enabled": true}]'
```

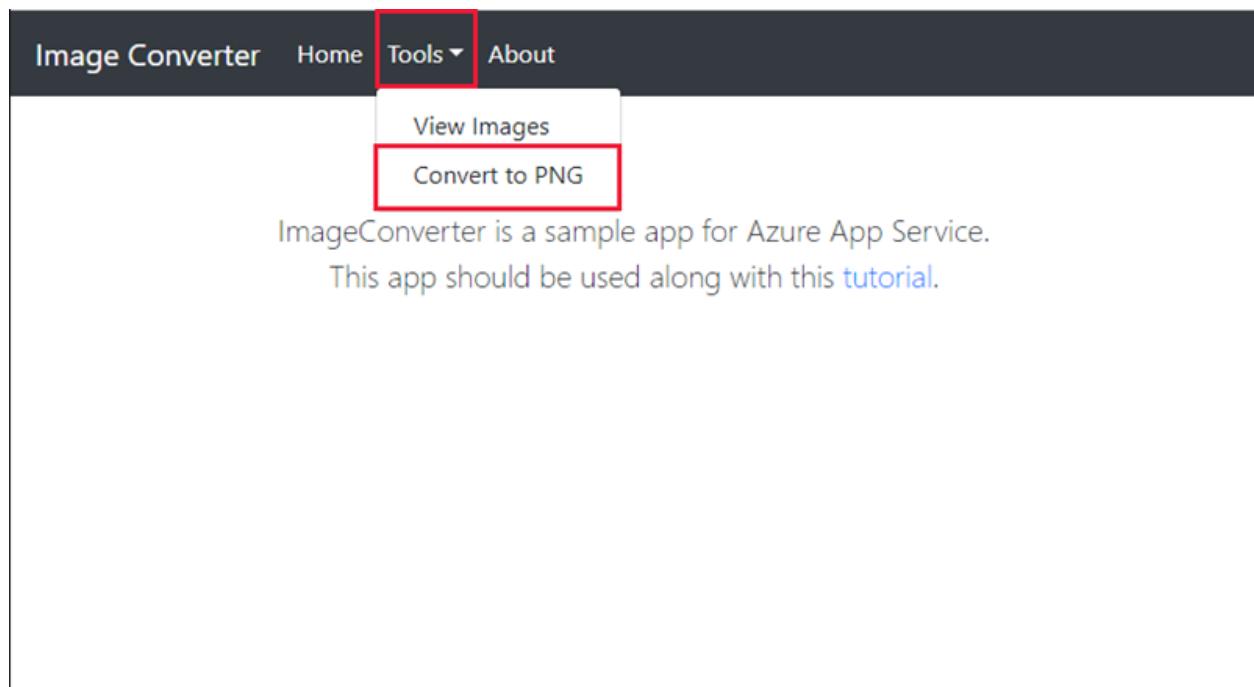
Troubleshoot the app

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

The sample app, ImageConverter, converts included images from [JPG](#) to [PNG](#). A bug has been deliberately placed in the code for this tutorial. If you select enough images, the app produces an HTTP 500 error during image conversion. Imagine this scenario wasn't considered during the development phase. You'll use Azure Monitor to troubleshoot the error.

Verify the app works

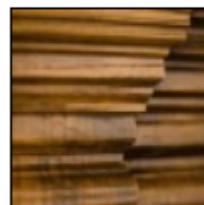
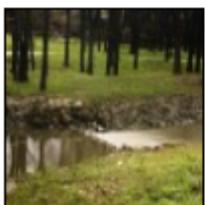
To convert images, click [Tools](#) and select [Convert to PNG](#).



Select the first two images and click [convert](#). This converts successfully.

Select JPGs to convert to PNG

X



2 images selected

Convert

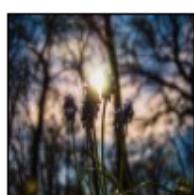
Close

Break the app

Now that you've verified the app by converting two images successfully, we try to convert the first five images.

Select JPGs to convert to PNG

X



5 images selected

Convert

Close

This action fails and produces a `HTTP 500` error that wasn't tested during development.

azmonapp.azurewebsites.net says

error- HTTP 500

OK

Use log query to view Azure Monitor logs

Let's see what logs are available in the Log Analytics workspace.

Click this [Log Analytics workspace link](#) to access your workspace in the Azure portal.

In the Azure portal, select your Log Analytics workspace.

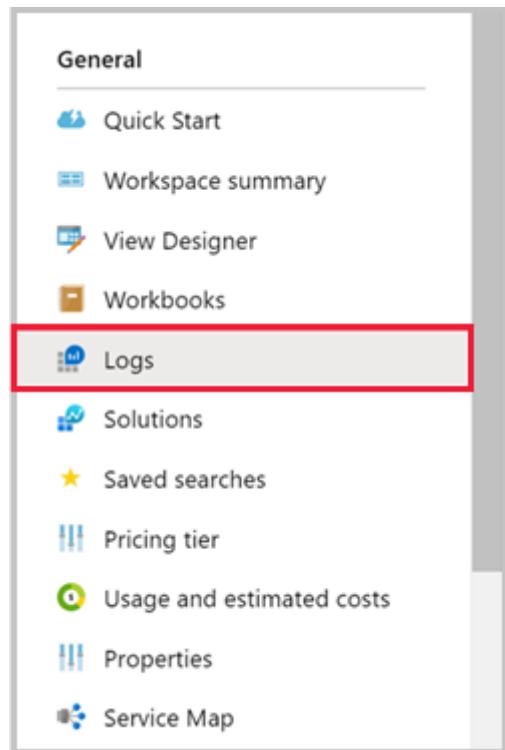
Log queries

Log queries help you to fully apply the value of the data collected in Azure Monitor Logs. You use log queries to identify the logs in both AppServiceHTTPLogs and AppServiceConsoleLogs. See the [log query overview](#) for more information on log queries.

View AppServiceHTTPLogs with log query

Now that we've accessed the app, let's view the data associated with HTTP requests, found in the `AppServiceHTTPLogs`.

1. Click `Logs` from the left-hand navigation.



2. Search for `appservice` and double-click `AppServiceHTTPLogs`.

New Query 1

MangeshWorkspace Select Scope

Tables Filter <<

Search bar: appservice

Group by: Solution Filters: not selected

Favorites

You can add favorites by clicking on the star icon

- LogManagement
 - AppServiceAppLogs
 - AppServiceAuditLogs
 - AppServiceConsoleLogs
 - AppServiceEnvironmentPla...
 - AppServiceFileAuditLogs
 - AppServiceHTTPLogs**

3. Click Run.

Run Time range : Last 24 hours Copy link New alert rule Export Pin to dashboard Prettify

AppServiceHTTPLogs

Completed. Showing results from the last 24 hours. 00:00:00.514 1,277 records Display time (UTC+00:00)

Table Chart Columns Drag a column header and drop it here to group by that column

TimeGenerated [UTC]	Category	CsMethod	CsUriStem	ScStatus	Clp	UserAgent
2/27/2020, 10:08:59.000 PM	AppServiceHTTPLogs	GET	/process.php?images=3...	80	10.0.128.8	Mozilla/5.0 (Windows N...
2/27/2020, 10:08:39.000 PM	AppServiceHTTPLogs	GET	/process.php?images=3...	80	10.0.128.7	Mozilla/5.0 (Windows N...
2/27/2020, 10:07:44.000 PM	AppServiceHTTPLogs	GET	/	80	10.0.128.7	Mozilla/5.0 (Windows N...
2/27/2020, 10:07:42.000 PM	AppServiceHTTPLogs	GET	/	80	10.0.128.8	Mozilla/5.0 (Windows N...
2/27/2020, 10:07:42.000 PM	AppServiceHTTPLogs	GET	/starter-template.css	80	10.0.128.8	Mozilla/5.0 (Windows N...
2/27/2020, 10:07:42.000 PM	AppServiceHTTPLogs	GET	/thumbs/img01.jpg	80	10.0.128.14	Mozilla/5.0 (Windows N...
2/27/2020, 10:07:42.000 PM	AppServiceHTTPLogs	GET	/thumbs/img02.jpg	80	10.0.128.12	Mozilla/5.0 (Windows N...
2/27/2020, 10:07:42.000 PM	AppServiceHTTPLogs	GET	/thumbs/img03.jpg	80	10.0.128.14	Mozilla/5.0 (Windows N...

Page 1 of 26 50 items per page 1 - 50 of 1277 items

The `AppServiceHTTPLogs` query returns all requests in the past 24-hours. The column `ScStatus` contains the HTTP status. To diagnose the `HTTP 500` errors, limit the `ScStatus` to 500 and run the query, as shown below:

Kusto

```
AppServiceHTTPLogs
| where ScStatus == 500
```

View AppServiceConsoleLogs with log query

Now that you've confirmed the HTTP 500s, let's take a look at the standard output/errors from the app. These logs are found in `AppServiceConsoleLogs`.

(1) Click **+** to create a new query.

(2) Double-click the `AppServiceConsoleLogs` table and click **Run**.

Since converting five images results in server errors, you can see if the app is also writing errors by filtering `ResultDescription` for errors, as show below:

Kusto

```
AppServiceConsoleLogs |  
where ResultDescription contains "error"
```

In the `ResultDescription` column, you see the following error:

Output

```
PHP Fatal error: Allowed memory size of 134217728 bytes exhausted  
(tried to allocate 16384 bytes) in /home/site/wwwroot/process.php on line  
20,  
referer: http://<app-name>.azurewebsites.net/
```

Join AppServiceHTTPLogs and AppServiceConsoleLogs

Now that you've identified both HTTP 500s and standard errors, you need to confirm if there's a correlation between these messages. Next, you join the tables together based on the time stamp, `TimeGenerated`.

ⓘ Note

A query has been prepared for you that does the following:

- Filters HTTPLogs for 500 errors
- Queries console logs
- Joins the tables on `TimeGenerated`

Run the following query:

Kusto

```
let myHttp = AppServiceHTTPLogs | where ScStatus == 500 | project  
TimeGen=substring(TimeGenerated, 0, 19), CsUriStem, ScStatus;  
  
let myConsole = AppServiceConsoleLogs | project  
TimeGen=substring(TimeGenerated, 0, 19), ResultDescription;  
  
myHttp | join myConsole on TimeGen | project TimeGen, CsUriStem, ScStatus,  
ResultDescription;
```

In the `ResultDescription` column, you'll see the following error at the same time as web server errors:

Output

```
PHP Fatal error: Allowed memory size of 134217728 bytes exhausted  
(tried to allocate 16384 bytes) in /home/site/wwwroot/process.php on line  
20,  
referer: http://<app-name>.azurewebsites.net/
```

The message states memory has been exhausted on line 20 of `process.php`. You've now confirmed that the application produced an error during the HTTP 500 error. Let's take a look at the code to identify the problem.

Identify the error

In the local directory, open the `process.php` and look at line 20.

PHP

```
imagepng($imgArray[$x], $filename);
```

The first argument, `$imgArray[$x]`, is a variable holding all JPGs (in-memory) needing conversion. However, `imagepng` only needs the image being converted and not all images. Pre-loading images is not necessary and may be causing the memory exhaustion, leading to HTTP 500s. Let's update the code to load images on-demand to see if it resolves the issue. Next, you improve the code to address the memory problem.

Fix the app

Update locally and redeploy the code

You make the following changes to `process.php` to handle the memory exhaustion:

PHP

```
<?php

//Retrieve query parameters
$maxImages = $_GET['images'];
$imgNames = explode(",",$_GET['imgNames']);

//Load JPEGs into an array (in memory)
for ($x=0; $x<$maxImages; $x++){
    $filename = './images/converted_' . substr($imgNames[$x],0,-4) . '.png';
    imagepng(imagecreatefromjpeg("./images/" . $imgNames[$x]), $filename);
}
```

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

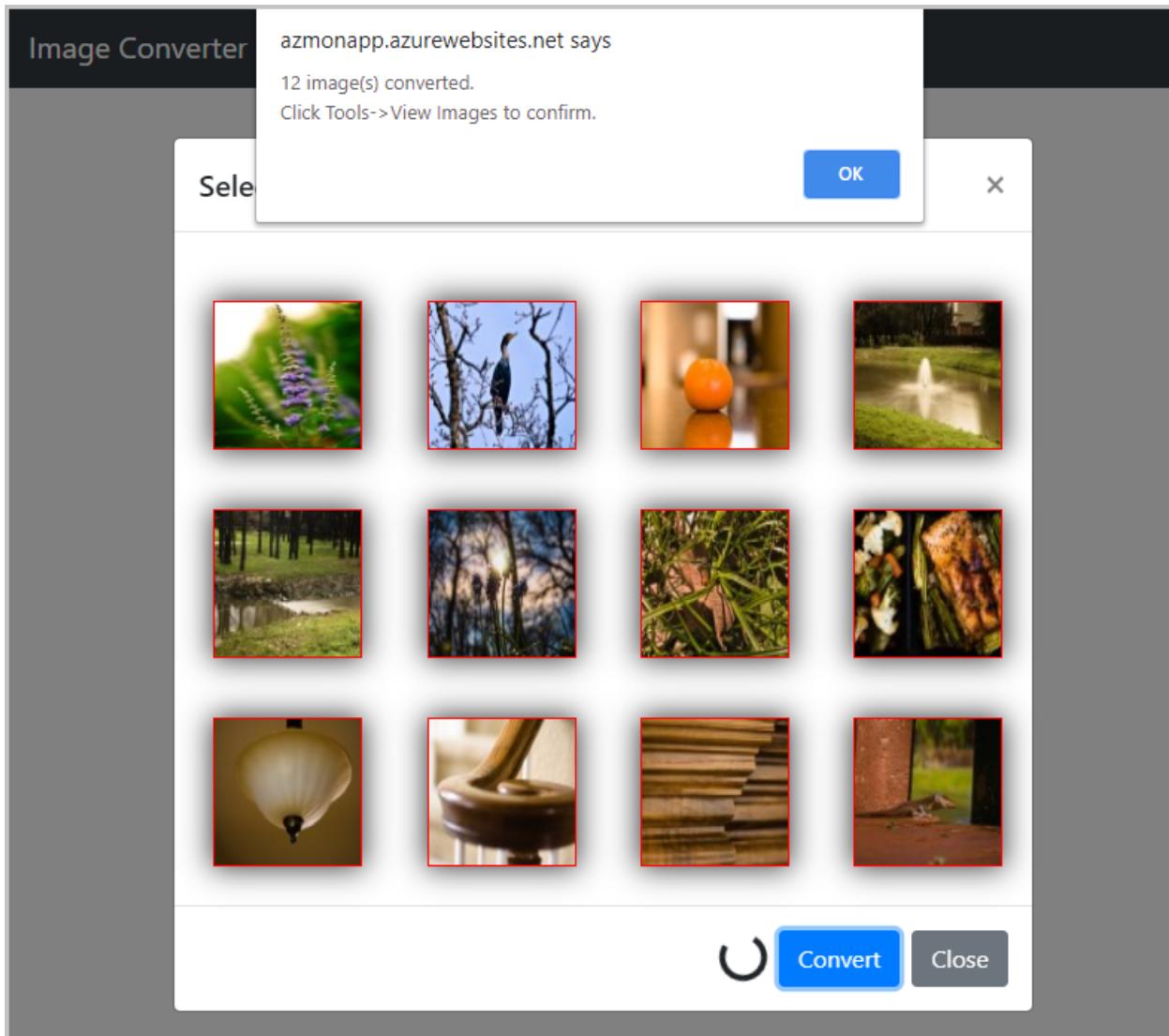
Bash

```
git commit -am "Load images on-demand in process.php"
git push azure main
```

Browse to the Azure app

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

Converting images should not longer produce the HTTP 500 errors.



Clean up resources

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

Azure CLI

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

This command may take a minute to run.

Delete the diagnostic setting with the following command:

Azure CLI

```
az monitor diagnostic-settings delete --resource $resourceID -n myMonitorLogs
```

What you learned:

- ✓ Configured a web app with Azure Monitor
- ✓ Sent logs to Log Analytics
- ✓ Used log queries to identify and troubleshoot web app errors

Next steps

- [Query logs with Azure Monitor](#)
- [Troubleshooting Azure App Service in Visual Studio](#)
- [Analyze app Logs in HDInsight ↗](#)
- [Tutorial: Run a load test to identify performance bottlenecks in a web app](#)

Enable diagnostics logging for apps in Azure App Service

Article • 08/21/2024

ⓘ Note

Starting June 1, 2024, all newly created App Service apps will have the option to generate a unique default hostname using the naming convention <app-name>-<random-hash>. <region>.azurewebsites.net. Existing app names will remain unchanged.

Example: myapp-ds27dh7271ah175.westus-01.azurewebsites.net

For further details, refer to [Unique Default Hostname for App Service Resource ↗](#).

This video shows you how to enable diagnostics logging for apps.

<https://learn-video.azurefd.net/vod/player?id=62f2edbe-1063-4ec3-a76f-faa0bd783f2f&locale=en-us&embedUrl=%2Fazure%2Fapp-service%2Ftroubleshoot-diagnostic-logs> ↗

The steps in the video are also described in the following sections.

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, you can also use the Azure Monitor integrated logging capability. You'll find more on this capability in the [Send logs to Azure Monitor](#) section.

Type	Platform	Log storage 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.
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 higher. 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. This information is useful if you want to improve site performance or isolate a specific HTTP error. One folder is generated for each failed request. The folder 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.

When stored in the App Service file system, logs are subject to the available storage for your pricing tier (see [App Service limits](#)).

ⓘ Note

App Service provides a dedicated, interactive diagnostics tool to help you troubleshoot your application. For more information, see [Azure App Service diagnostics overview](#).

In addition, you can use other Azure services to improve the logging and monitoring capabilities of your app, such as [Azure Monitor](#).

Enable application logging (Windows)

To enable application logging for Windows apps in the [Azure portal](#), navigate to your app and select **App Service logs**.

Select **On** for either **Application Logging (Filesystem)** or **Application Logging (Blob)**, or both.

The **Filesystem** option is for temporary debugging purposes, and turns itself off in 12 hours. The **Blob** option is for long-term logging, and needs a blob storage container to write logs to. The **Blob** option also includes additional information in the log messages, such as the ID of the origin VM instance of the log message (`InstanceId`), thread ID (`Tid`), and a more granular timestamp (`EventTickCount`).

ⓘ Note

Currently only .NET application logs can be written to the blob storage. Java, PHP, Node.js, and 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:

[+] Expand table

Level	Included categories
Disabled	None
Error	Error, Critical
Warning	Warning, Error, Critical
Information	Info, Warning, Error, Critical
Verbose	Trace, Debug, Info, Warning, Error, Critical (all categories)

When finished, select **Save**.

 **Note**

If you write logs to blobs, the retention policy no longer applies if you delete the app but keep the logs in the blobs. For more information, see [Costs that might accrue after resource deletion](#).

Enable application logging (Linux/Container)

To enable application logging for Linux apps or custom containers in the [Azure portal](#), navigate to your app and select **App Service logs**.

In **Application logging**, select **File System**.

In **Quota (MB)**, specify the disk quota for the application logs. In **Retention Period (Days)**, set the number of days the logs should be retained.

When finished, select **Save**.

Enable web server logging

To enable web server logging for Windows apps in the [Azure portal](#), navigate to your app and select **App Service logs**.

For **Web server logging**, select **Storage** to store logs on blob storage, or **File System** to store logs on the App Service file system.

In **Retention Period (Days)**, set the number of days the logs should be retained.

 **Note**

If you [regenerate your storage account's access keys](#), you must reset the respective logging configuration to use the updated keys. To do this:

1. In the **Configure** tab, set the respective logging feature to **Off**. Save your setting.
2. Enable logging to the storage account blob again. Save your setting.

When finished, select **Save**.

 **Note**

If you write logs to blobs, the retention policy no longer applies if you delete the app but keep the logs in the blobs. For more information, see [Costs that might accrue after resource deletion](#).

Log detailed errors

To save the error page or failed request tracing for Windows apps in the [Azure portal](#), navigate to your app and select **App Service logs**.

Under **Detailed Error Logging** or **Failed Request Tracing**, select **On**, and then select **Save**.

Both types of logs are stored in the App Service file system. Up to 50 errors (files or folders) are retained. When the number of HTML files exceeds 50, the oldest error files are automatically deleted.

The Failed Request Tracing feature by default captures a log of requests that failed with HTTP status codes between 400 and 600. To specify custom rules, you can override the `<traceFailedRequests>` section in the `web.config` file.

Add log messages in code

In your application code, you use the usual logging facilities to send log messages to the application logs. For example:

- ASP.NET applications can use the `System.Diagnostics.Trace` class to log information to the application diagnostics log. For example:

C#

```
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](#).

- Python applications can use the [OpenCensus package](#) to send logs to the application diagnostics log.

Stream logs

Before you stream logs in real time, enable the log type that you want. Any information written to the console output or files ending in .txt, .log, or .htm that are stored in the `/home/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 events appearing in the incorrect order 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.

Azure CLI

```
az webapp log tail --name appname --resource-group myResourceGroup
```

To filter specific log types, such as HTTP, use the `--provider` parameter. For example:

Azure CLI

```
az webapp log tail --name appname --resource-group myResourceGroup --provider http
```

In the local terminal

To stream logs in the local console, [install Azure CLI](#) and [sign in to your account](#). After you're signed in, follow the [instructions for Cloud Shell](#).

Access log files

If you configure the Azure Storage blobs option for a log type, you need a client tool that works with Azure Storage. For more information, see [Azure Storage Client Tools](#).

For logs stored in the App Service file system, the easiest way to access the files is to download the ZIP file in the browser at:

- Linux/custom containers: `https://<app-name>.scm.azurewebsites.net/api/logs/docker/zip`
- Windows apps: `https://<app-name>.scm.azurewebsites.net/api/dump`

For Linux/custom containers, 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. Deployment logs are stored in `/site/deployments/`.

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:

 [Expand table](#)

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.

Log type	Directory	Description
Detailed Error Logs	/LogFiles/DetailedErrors/	<p>Contains HTM error files. You can view the HTM files in the browser.</p> <p>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, search for Failed Request Tracing Logs, and then click the icon to browse and view the trace you want.</p>
Web Server Logs	/LogFiles/http/RawLogs/	<p>Contains text files formatted using the W3C extended log file format. You can read these files by using a text editor or a utility like Log Parser.</p> <p>App Service doesn't support the <code>s-computername</code>, <code>s-ip</code>, or <code>cs-version</code> fields.</p>
Deployment logs	/LogFiles/Git/ and /deployments/	Contains logs generated by the internal deployment processes, as well as logs for Git deployments.

Send logs to Azure Monitor

With [Azure Monitor integration](#), you can [create Diagnostic Settings](#) to send logs to storage accounts, event hubs and Log Analytics. When you add a diagnostic setting, App Service adds app settings to your app, which triggers an app restart.

The screenshot shows the Azure App Service Editor interface. On the left, there's a sidebar with various options like Clone App, SSH, Advanced Tools, App Service Editor (Preview), Performance test, Resource explorer, Extensions, API (CORS, Monitoring, Alerts, Metrics), Diagnostic settings (which is highlighted with a red box), Logs, App Service logs, Log stream, Process explorer, Support + troubleshooting (Resource health, App Service Advisor), and New support request.

The main content area has a header with 'Subscription' set to 'Demo Two Subscription' and 'Resource group' set to 'appsvc-azmon-rg'. Below this, it shows 'Diagnostics settings' with a table:

Name	Storage account	Event hub
No diagnostic settings defined		

A button '+ Add diagnostic setting' is highlighted with a red box. Below it, instructions say: 'Click 'Add Diagnostic setting' above to configure the collection of the following data:' followed by a list of items:

- AppServiceHTTPLogs
- AppServiceConsoleLogs
- AppServiceAppLogs
- AppServiceFileAuditLogs
- AppServiceAuditLogs
- AllMetrics

Supported log types

For a list of supported log types and their descriptions, see [Supported resource logs for Microsoft.Web](#).

Networking considerations

For Diagnostic Settings restrictions, refer to the [official Diagnostic Settings documentation](#) regarding destination limits.

Next steps

- [Query logs with Azure Monitor](#)
- [How to Monitor Azure App Service](#)
- [Troubleshooting Azure App Service in Visual Studio](#)
- [Tutorial: Run a load test to identify performance bottlenecks in a web app](#)

ⓘ **Note:** The author created this article with assistance from AI. [Learn more](#)

Feedback

Was this page helpful?

 Yes

 No

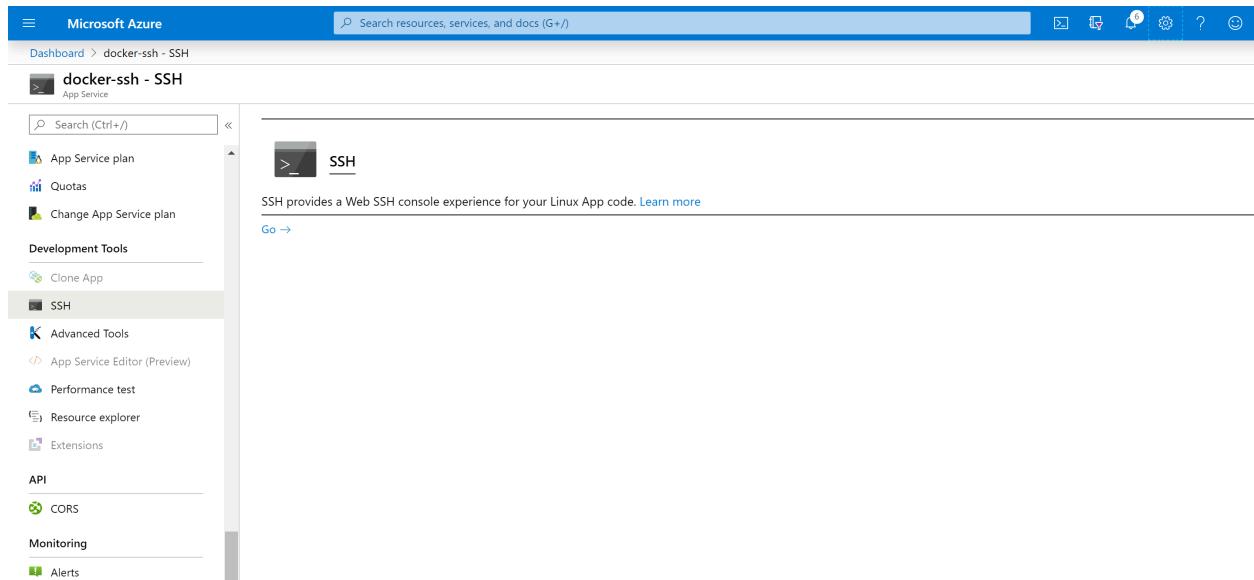
[Provide product feedback ↗](#) | Get help at Microsoft Q&A

Open an SSH session to a container in Azure App Service

Article • 01/29/2025

Secure Shell (SSH) [↗](#) can be used to execute administrative commands remotely to a container. App Service provides SSH support directly into an app hosted in a Linux container (built-in or custom).

The built-in Linux containers already have the necessary configuration to enable SSH sessions. Linux custom containers require additional configurations to enable SSH sessions. See [Enable SSH](#).



The screenshot shows the Microsoft Azure portal interface. At the top, there's a navigation bar with the Microsoft Azure logo, a search bar, and several icons. Below the navigation bar, the URL 'Dashboard > docker-ssh - SSH' is visible. The main content area is titled 'SSH' and contains the text: 'SSH provides a Web SSH console experience for your Linux App code. [Learn more](#)'. A 'Go →' button is located below this text. On the left side, there's a vertical sidebar with a tree view of the app service's configuration. The 'SSH' option is highlighted. Other options in the sidebar include 'App Service plan', 'Quotas', 'Change App Service plan', 'Development Tools' (which includes 'Clone App', 'SSH' which is selected, 'Advanced Tools', 'App Service Editor (Preview)', 'Performance test', 'Resource explorer', and 'Extensions'), 'API' (which includes 'CORS' and 'Monitoring'), and 'Alerts'.

You can also connect to the container directly from your local development machine using SSH and SFTP.

Open SSH session in browser

To make open a direct SSH session with your container, your app should be running.

Paste the following URL into your browser and replace `<app-name>` with your app name:



The screenshot shows a browser window with a single line of text: 'https://<app-name>.scm.azurewebsites.net/webssh/host'. The entire URL is highlighted with a red rectangular selection.

If you're not yet authenticated, you're required to authenticate with your Azure subscription to connect. Once authenticated, you see an in-browser shell, where you can run commands inside your container.

```
root@9e933156516f:~# service --status-all
[ + ]  apache2
[ - ]  bootlogs
[ - ]  bootmisc.sh
[ - ]  checkfs.sh
[ - ]  checkroot-bootclean.sh
[ - ]  checkroot.sh
[ - ]  hostname.sh
[ ? ]  hwclock.sh
[ - ]  killprocs
[ - ]  motd
[ - ]  mountall-bootclean.sh
[ - ]  mountall.sh
[ - ]  mountdevsubfs.sh
[ - ]  mountkernfs.sh
[ - ]  mountnfs-bootclean.sh
[ - ]  mountnfs.sh
[ - ]  mysql
[ - ]  procps
[ - ]  rc.local
[ - ]  rmnlogin
[ - ]  sendsigs
[ + ]  ssh
[ + ]  udev
[ ? ]  udev-finish
[ - ]  umountfs
[ - ]  umountnfs.sh
[ - ]  umountroot
[ - ]  urandom
root@9e933156516f:~#
```

ssh://root@ ...:2222 | SSH CONNECTION ESTABLISHED |

Open SSH session with Azure CLI

Using TCP tunneling you can create a network connection between your development machine and Linux containers over an authenticated WebSocket connection. It enables you to open an SSH session with your container running in App Service from the client of your choice.

To get started, you need to install [Azure CLI](#). To see how it works without installing Azure CLI, open [Azure Cloud Shell](#).

Open a remote connection to your app using the `az webapp create-remote-connection` command. Specify `<subscription-id>`, `<group-name>` and `<app-name>` for your app.

```
Azure CLI

az webapp create-remote-connection --subscription <subscription-id> --
resource-group <resource-group-name> -n <app-name> &
```

💡 Tip

& at the end of the command is just for convenience if you are using Cloud Shell. It runs the process in the background so that you can run the next command in the

same shell.

ⓘ Note

If this command fails, make sure [remote debugging](#) is *disabled* with the following command:

Azure CLI

```
az webapp config set --resource-group <resource-group-name> -n <app-name> --remote-debugging-enabled=false
```

The command output gives you the information you need to open an SSH session.

Output

```
Verifying if app is running....  
App is running. Trying to establish tunnel connection...  
Opening tunnel on addr: 127.0.0.1  
Opening tunnel on port: <port-output>  
SSH is available { username: root, password: Docker! }  
Ctrl + C to close
```

Open an SSH session with your container with the client of your choice, using the local port provided in the output ([`<port-output>`](#)). For example, with the linux [ssh](#) command, you can run a single command like `java -version`:

Bash

```
ssh root@127.0.0.1 -m hmac-sha1 -p <port-output> java -version
```

Or, to enter a full SSH session, just run:

Bash

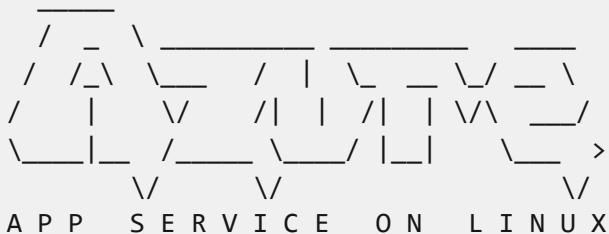
```
ssh root@127.0.0.1 -m hmac-sha1 -p <port-output>
```

When being prompted, type `yes` to continue connecting. You are then prompted for the password. Use `Docker!`, which was shown to you earlier.

```
Warning: Permanently added '[127.0.0.1]:21382' (ECDSA) to the list of known hosts.
```

```
root@127.0.0.1's password:
```

Once you're authenticated, you should see the session welcome screen.



```
0e690efa93e2:~#
```

You are now connected to your connector.

Try running the [top ↗](#) command. You should be able to see your app's process in the process list. In the example output below, it's the one with **PID 263**.

```
Mem: 1578756K used, 127032K free, 8744K shrd, 201592K buff, 341348K cached
CPU: 3% usr 3% sys 0% nic 92% idle 0% io 0% irq 0% sirq
Load average: 0.07 0.04 0.08 4/765 45738
      PID  PPID USER      STAT  VSZ %VSZ CPU %CPU COMMAND
        1    0 root      S    1528  0%   0  0% /sbin/init
      235    1 root      S    632m 38%   0  0% PM2 v2.10.3: God Daemon
(/root/.pm2)
      263   235 root      S    630m 38%   0  0% node /home/site/wwwroot/app.js
      482   291 root      S    7368  0%   0  0% sshd: root@pts/0
    45513   291 root      S    7356  0%   0  0% sshd: root@pts/1
      291    1 root      S    7324  0%   0  0% /usr/sbin/sshd
      490   482 root      S    1540  0%   0  0% -ash
    45539  45513 root      S    1540  0%   0  0% -ash
    45678  45539 root      R    1536  0%   0  0% top
    45733    1 root      Z      0  0%   0  0% [init]
    45734    1 root      Z      0  0%   0  0% [init]
    45735    1 root      Z      0  0%   0  0% [init]
    45736    1 root      Z      0  0%   0  0% [init]
    45737    1 root      Z      0  0%   0  0% [init]
    45738    1 root      Z      0  0%   0  0% [init]
```

Next steps

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

For more information on Web App for Containers, see:

- Introducing remote debugging of Node.js apps on Azure App Service from VS Code ↗
 - Quickstart: Run a custom container on App Service
 - Azure App Service Web App for Containers FAQ
-

Feedback

Was this page helpful?

 Yes

 No

Provide product feedback ↗ | Get help at Microsoft Q&A

Monitor App Service instances by using Health check

Article • 09/15/2024

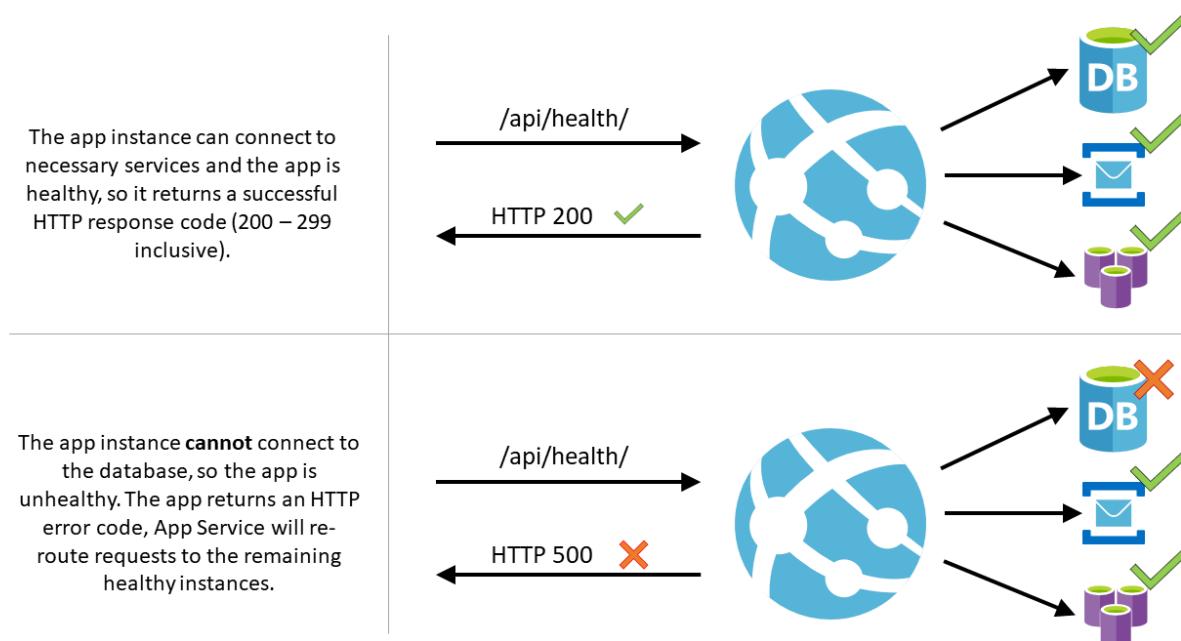
ⓘ Note

Starting June 1, 2024, all newly created App Service apps will have the option to generate a unique default hostname using the naming convention <app-name>-<random-hash>. <region>.azurewebsites.net. Existing app names will remain unchanged.

Example: myapp-ds27dh7271ah175.westus-01.azurewebsites.net

For further details, refer to [Unique Default Hostname for App Service Resource ↗](#).

This article describes how to use Health check in the Azure portal to monitor App Service instances. Health check increases your application's availability by rerouting requests away from unhealthy instances and replacing instances if they remain unhealthy. It does that by pinging your web application every minute, via a path that you choose.



Note that `/api/health` is just an example. There is no default Health check path. You should make sure that the path you choose is a valid path that exists within your application.

How Health check works

- When given a path on your app, Health check pings the path on all instances of your App Service app at 1-minute intervals.
- If a web app that's running on a given instance doesn't respond with a status code between 200 and 299 (inclusive) after 10 requests, App Service determines the instance is unhealthy and removes it from the load balancer for the web app. The required number of failed requests for an instance to be deemed unhealthy is configurable to a minimum of two requests.
- After the instance is removed, Health check continues to ping it. If the instance begins to respond with a healthy status code (200-299), then the instance is returned to the load balancer.
- If the web app that's running on an instance remains unhealthy for one hour, the instance is replaced with a new one.
- When scaling out, App Service pings the Health check path to ensure new instances are ready.

ⓘ Note

- Health check doesn't follow 302 redirects.
- At most, one instance will be replaced per hour, with a maximum of three instances per day per App Service Plan.
- If Health check is sending the status `Waiting for health check response`, then the check is likely failing due to an HTTP status code of 307, which can happen if you have HTTPS redirect enabled but have `HTTPS Only` disabled.

Enable Health check

1. To enable Health check, browse to the Azure portal and select your App Service app.
2. Under **Monitoring**, select **Health check**.
3. Select **Enable** and provide a valid URL path for your application, such as `/health` or `/api/health`.
4. Select **Save**.

ⓘ Note

- Your App Service plan should be scaled to two or more instances to fully utilize Health check.
- The Health check path should check critical components of your application. For example, if your application depends on a database and a messaging system, the Health check endpoint should connect to those components. If the application can't connect to a critical component, then the path should return a 500-level response code to indicate the app is unhealthy. Also, if the path doesn't return a response within one minute, the health check ping is considered unhealthy.
- When selecting the Health check path, make sure you're selecting a path that returns a 200 status code only when the app is fully warmed up.
- In order to use Health check on a function app, you must use a premium or dedicated hosting plan.

- Details about Health check on function apps can be found here: [Monitor function apps using Health check](#).

 **Caution**

Health check configuration changes restart your app. To minimize impact to production apps, we recommend [configuring staging slots](#) and swapping to production.

Configuration

In addition to configuring the Health check options, you can also configure the following [app settings](#):

 [Expand table](#)

App setting name	Allowed values	Description
WEBSITE_HEALTHCHECK_MAXPINGFAILURES	2 - 10	The required number of failed requests for an instance to be deemed unhealthy and removed from the load balancer. For example, when this is set to 2, your instances are removed after 2 failed pings. (The default value is 10.)
WEBSITE_HEALTHCHECK_MAXUNHEALTHYWORKERPERCENT	1 - 100	By default, to avoid overwhelming the remaining healthy instances, no more than half of the instances will be excluded from the load balancer at a time. For example, if an App Service plan is scaled to four instances and three are unhealthy, two are excluded. The other two instances (one healthy and one unhealthy) continue to receive requests. In a scenario where all instances are unhealthy, none are excluded. To override this behavior, set this app setting to a value between 1 and 100. A higher value means

App setting name	Allowed values	Description
		more unhealthy instances are removed. (The default value is 50.).

Authentication and security

Health check integrates with the App Service [authentication and authorization features](#). No other settings are required if these security features are enabled.

If you're using your own authentication system, the Health check path must allow anonymous access. To provide security for the Health check endpoint, you should first use features such as [IP restrictions](#), [client certificates](#), or a virtual network to restrict application access. Once you have those features in place, you can authenticate the Health check request by inspecting the header `x-ms-auth-internal-token` and validating that it matches the SHA256 hash of the environment variable `WEBSITE_AUTH_ENCRYPTION_KEY`. If they match, then the Health check request is valid and originating from App Service.

ⓘ Note

For [Azure Functions authentication](#), the function that serves as the Health check endpoint needs to allow anonymous access.

.NET

C#

```
using System;
using System.Text;

/// <summary>
/// Method <c>HeaderMatchesEnvVar</c> returns true if <c>headerValue</c>
/// matches WEBSITE_AUTH_ENCRYPTION_KEY.
/// </summary>
public Boolean HeaderMatchesEnvVar(string headerValue) {
    var sha = System.Security.Cryptography.SHA256.Create();
    String envVar =
Environment.GetEnvironmentVariable("WEBSITE_AUTH_ENCRYPTION_KEY");
    String hash =
System.Convert.ToBase64String(sha.ComputeHash(Encoding.UTF8.GetBytes(env
Var)));
}
```

```
    return hash == headerValue;  
}
```

ⓘ Note

The `x-ms-auth-internal-token` header is only available on App Service for Windows.

Instances

Once Health check is enabled, you can restart and monitor the status of your application instances from the instances tab. The instances tab shows your instance's name and the status of that application's instance. You can also manually restart the instance from this tab.

If the status of your application instance is "unhealthy," you can restart the instance manually by using the restart button in the table. Keep in mind that any other applications hosted on the same App Service plan as the instance will also be affected by the restart. If there are other applications using the same App Service plan as the instance, they're listed on the opening blade from the restart button.

If you restart the instance and the restart process fails, you'll be given the option to replace the worker. (Only one instance can be replaced per hour.) This will also affect any applications using the same App Service plan.

For Windows applications, you can also view processes via the Process Explorer. This gives you further insight on the instance's processes, including thread count, private memory, and total CPU time.

Diagnostic information collection

For Windows applications, you have the option to collect diagnostic information on the Health Check tab. Enabling diagnostic collection adds an auto-heal rule that creates memory dumps for unhealthy instances and saves them to a designated storage account. Enabling this option changes auto-heal configurations. If there are existing auto-heal rules, we recommend setting this up through App Service diagnostics.

Once diagnostic collection is enabled, you can create a storage account or choose an existing one for your files. You can only select storage accounts in the same region as your application. Keep in mind that saving restarts your application. After saving, if your

site instances are found to be unhealthy after continuous pings, you can go to your storage account resource and view the memory dumps.

Monitoring

After providing your application's Health check path, you can monitor the health of your site using Azure Monitor. From the **Health check** blade in the portal, select **Metrics** in the top toolbar. This opens a new blade where you can see the site's health status history and create a new alert rule. Health check metrics aggregate the successful pings and display failures only when the instance was deemed unhealthy based on the Health check configuration. For more information on monitoring your sites, see [Azure App Service quotas and alerts](#).

Limitations

- Health check can be enabled for **Free** and **Shared** App Service plans, so you can have metrics on the site's health and set up alerts. However, because **Free** and **Shared** sites can't scale out, unhealthy instances won't be replaced. You should scale up to the **Basic** tier or higher so you can scale out to two or more instances and get the full benefit of Health check. This is recommended for production-facing applications as it increases your app's availability and performance.
- An App Service plan can have a maximum of one unhealthy instance replaced per hour and, at most, three instances per day.
- There's a nonconfigurable limit on the total number of instances replaced by Health check per scale unit. If this limit is reached, no unhealthy instances are replaced. This value gets reset every 12 hours.

Frequently asked questions

What happens if my app is running on a single instance?

If your app is only scaled to one instance and becomes unhealthy, it won't be removed from the load balancer because that would take down your application entirely.

However, after one hour of continuous unhealthy pings, the instance is replaced. Scale out to two or more instances to get the rerouting benefit of Health check. If your app is running on a single instance, you can still use the Health check [monitoring](#) feature to keep track of your application's health.

Why are the Health check requests not showing in my web server logs?

The Health check requests are sent to your site internally, so the request won't show in [the web server logs](#). You can add log statements in your Health check code to keep logs of when your Health check path is pinged.

Are Health check requests sent over HTTP or HTTPS?

On App Service for Windows and Linux, the Health check requests are sent via HTTPS when [HTTPS Only](#) is enabled on the site. Otherwise, they're sent over HTTP.

Does Health check follow the application-code configured redirects between the default domain and the custom domain?

No, the Health check feature pings the path of the default domain of the web application. If there's a redirect from the default domain to a custom domain, then the status code that Health check returns won't be a 200. It will be a redirect (301), which marks the worker unhealthy.

What if I have multiple apps on the same App Service plan?

Unhealthy instances will always be removed from the load balancer rotation regardless of other apps on the App Service plan (up to the percentage specified in [WEBSITE_HEALTHCHECK_MAXUNHEALTHYWORKERPERCENT](#)). When an app on an instance remains unhealthy for more than one hour, the instance will only be replaced if all other apps on which Health check is enabled are also unhealthy. Apps that don't have Health check enabled won't be taken into account.

Example

Imagine you have two applications (or one app with a slot) with Health check enabled. They're called App A and App B. They're on the same App Service plan, and the plan is scaled out to four instances. If App A becomes unhealthy on two instances, the load balancer stops sending requests to App A on those two instances. Requests are still routed to App B on those instances, assuming App B is healthy. If App A remains unhealthy for more than an hour on those two instances, the instances are only replaced

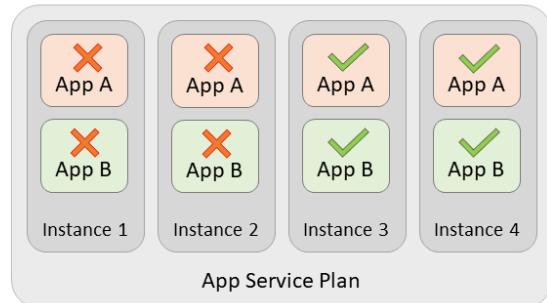
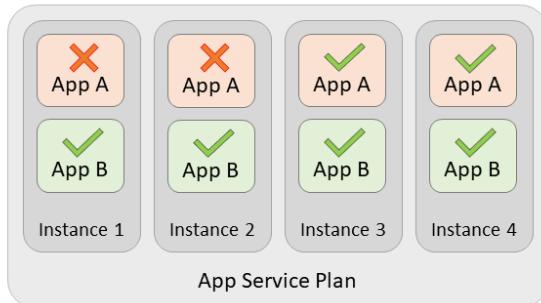
if App B is also unhealthy on those instances. If App B is healthy, the instances aren't replaced.

App A is unhealthy on instances 1 and 2, so requests will **not** be routed to App A on those instances. App B is healthy on instances 1 and 2, so requests will continue going to App B on those instances.

Instances 1 and 2 will **not** be replaced because App B is healthy on those instances.

Apps A and B are both unhealthy on instances 1 and 2, so request will **not** be routed to App A or B on those instances.

Instances 1 and 2 will be replaced because all apps on those instances are unhealthy.



① Note

If there were another site or slot on the plan (App C) without Health check enabled, it wouldn't be taken into consideration for the instance replacement.

What if all my instances are unhealthy?

If all instances of your application are unhealthy, App Service won't remove instances from the load balancer. In this scenario, taking all unhealthy app instances out of the load balancer rotation would effectively cause an outage for your application. However, the instance replacement will still occur.

Does Health check work on App Service Environments?

Yes, health check is available for App Service Environment v3, but not for versions 1 or 2. If you're using the older versions of App Service Environment, you can use the [migration feature](#) to migrate your App Service Environment to version 3.

Next steps

- Create an Activity Log Alert to monitor all Autoscale engine operations on your subscription ↗

- Create an Activity Log Alert to monitor all failed Autoscale scale-in/scale-out operations on your subscription ↗
 - Environment variables and app settings reference
-

Feedback

Was this page helpful?



Yes



No

Provide product feedback ↗ | Get help at Microsoft Q&A

Azure App Service diagnostics overview

Article • 03/03/2023

When you're running a web application, you want to be prepared for any issues that may arise, from 500 errors to your users telling you that your site is down. App Service diagnostics is an intelligent and interactive experience to help you troubleshoot your app with no configuration required. If you do run into issues with your app, App Service diagnostics points out what's wrong to guide you to the right information to more easily and quickly troubleshoot and resolve the issue.

Although this experience is most helpful when you're having issues with your app within the last 24 hours, all the diagnostic graphs are always available for you to analyze.

App Service diagnostics works for not only your app on Windows, but also apps on [Linux/containers](#), [App Service Environment](#), and [Azure Functions](#).

Open App Service diagnostics

To access App Service diagnostics, navigate to your App Service web app or App Service Environment in the [Azure portal](#). In the left navigation, click on **Diagnose and solve problems**.

For Azure Functions, navigate to your function app, and in the top navigation, click on **Platform features**, and select **Diagnose and solve problems** from the **Resource management** section.

In the App Service diagnostics homepage, you can perform a search for a symptom with your app, or choose a diagnostic category that best describes the issue with your app. Next, there is a new feature called Risk Alerts that provides an actionable report to improve your App. Finally, this page is where you can find **Diagnostic Tools**. See [Diagnostic tools](#).

The screenshot shows the Azure App Service Diagnostics page for the 'buggybakery' app. The left sidebar includes links for Overview, Activity log, Access control (IAM), Tags, Diagnose and solve problems (selected), Security, Events (preview), Deployment, Quickstart, Deployment slots, Deployment Center, Settings, Configuration, Authentication, Application Insights, Identity, Backups, Custom domains, TLS/SSL settings, TLS/SSL settings (preview), Networking, Scale up (App Service plan), Scale out (App Service plan), and WebJobs.

The main content area displays the 'App Service Diagnostics' section. It features a search bar, an 'Ask Genie' button, a 'Refresh' button, and a 'Feedback' link. A 'Risk alerts' section shows 3 Critical issues and 1 Success. Below it are 'Troubleshooting categories' including Availability and Performance, Configuration and Management, SSL and Domains, Risk Assessments, Navigator (Preview), and Diagnostic Tools. Each category has a description and a 'Troubleshoot' link. At the bottom is a 'Popular troubleshooting tools' section.

ⓘ Note

If your app is down or performing slow, you can [collect a profiling trace](#) to identify the root cause of the issue. Profiling is light weight and is designed for production scenarios.

Diagnostic Interface

The homepage for App Service diagnostics offers streamlined diagnostics access using four sections:

- Ask Genie search box
- Risk Alerts
- Troubleshooting categories
- Popular troubleshooting tools

Ask Genie search box

The Genie search box is a quick way to find a diagnostic. The same diagnostic can be found through Troubleshooting categories.

[Ask Genie](#)[Feedback](#)

- [Overview](#)
- [Activity log](#)
- [Access control \(IAM\)](#)
- [Tags](#)
- [**Diagnose and solve problems**](#)
- [Security](#)
- [Events \(preview\)](#)

- [Deployment](#)
- [Quickstart](#)
- [Deployment slots](#)
- [Deployment Center](#)

- [Settings](#)
- [Configuration](#)
- [Authentication](#)
- [Application Insights](#)
- [Identity](#)
- [Backups](#)
- [Custom domains](#)
- [TLS/SSL settings](#)
- [TLS/SSL settings \(preview\)](#)

App Service Diagnostics - Investigate how your app is performing, diagnose issues and fix configuration problems.

availability

Best Practices for Availability & Pe...	Risk Assessments
Web App Down	Availability and Perfo...
Web App Slow	Availability and Perfo...
High CPU Analysis	Availability and Perfo...
Memory Analysis	Availability and Perfo...
Web App Restarted	Availability and Perfo...
Application Changes (Preview)	Availability and Perfo...
Application Crashes	Availability and Perfo...
HTTP 4xx Errors	Availability and Perfo...

Found 15 Results, Press Escape to clear search bar

Track changes on your app and its dependencies.
Ex: Change Analysis, SQL, Dependency, ...

Run proactive tools to mitigate the app.
Ex: Auto-Heal

Risk Alerts

The App Service diagnostics homepage performs a series of configuration checks and offers recommendations based on your unique application's configuration.

Risk alerts

Availability

 3 Critical 1 Success
[View more details](#)

Recommendations and checks performed can be reviewed by clicking "View more details" link.

The screenshot shows the Azure portal's 'Diagnose and solve problems' blade for an 'App Service' named 'buggybakery'. The left sidebar includes links for Overview, Activity log, Access control (IAM), Tags, Diagnosis and solve problems (selected), Security, Events (preview), Deployment (Quickstart, Deployment slots, Deployment Center), Settings (Configuration, Authentication, Application Insights, Identity, Backups, Custom domains, TLS/SSL settings, TLS/SSL settings (preview), Networking, Scale up (App Service plan), Scale out (App Service plan), WebJobs), and Popular troubleshooting tools (Web App Down, Web App Slow, High CPU Analysis, Network Troubleshooter).

Risk alerts: Shows 3 Critical and 1 Success.

Troubleshooting categories:

- Availability and Performance**: Checks app health and platform issues. Includes 'Ex: Downtime, 5xx, 4xx, CPU, Memory, SNAT' and a 'Troubleshoot' link.
- Configuration and Management**: Checks app service features. Includes 'Ex: Backups, Slots, Swap, Scaling, IP Config' and a 'Troubleshoot' link.
- SSL and Domains**: Checks certificates and custom domains. Includes 'Ex: 4xx, Permissions, Auth, Binding, Cert Failures' and a 'Troubleshoot' link.
- Navigator (Preview)**: Tracks changes on the app and its dependencies. Includes 'Ex: Change Analysis, SQL Dependency, Storage' and a 'Troubleshoot' link.
- Diagnostic Tools**: Runs proactive tools to mitigate the app. Includes 'Ex: Auto-Heal' and a 'Troubleshoot' link.

Availability risk alerts:

- Auto-Heal is not enabled.**: Describes Auto-Heal as highly recommended for production applications to ensure high availability and resilience. It notes that while Auto-Heal is not an eventful fix, it allows your application to quickly recover from unexpected issues.
- Currently not utilizing Health Check feature.**: Describes the Health Check feature as ping the specified health check path on all instances of your webapp every minute to ensure instances are healthy. It includes a 'Configure and enable health check feature' link.
- Distributing your web app across multiple instances.**: Notes that the webapp is currently configured to run on only one instance. Since you have only one instance you can expect downtime because when the App Service platform is upgraded, the instance on which your web app is running will be upgraded. Therefore, your web app process will be restarted and will experience downtime.
- Scale instance count manually or automatically.**: Describes scaling as the recommended value for production applications. It notes that an App Service Plan does not host more than a certain number of sites. The number may actually be lower depending on how resource intensive the hosted applications are.
- Total active sites on the App Service Plan are within the recommended value.**: Shows a green checkmark.

Troubleshooting categories

Troubleshooting categories group diagnostics for ease of discovery. The following are available:

- **Availability and Performance**
- **Configuration and Management**
- **SSL and Domains**
- **Risk Assessments**
- **Navigator (Preview)**
- **Diagnostic Tools**

Troubleshooting categories



Availability and Performance

Check your app's health and discover app or platform issues.

Ex: Downtime, 5xx, 4xx, CPU, Memory, SNAT

[Troubleshoot](#)



Configuration and Management

Find out if your app service features are misconfigured.

Ex: Backups, Slots, Swaps, Scaling, IP Config

[Troubleshoot](#)



SSL and Domains

Discover issues with certificates and custom domains.

Ex: 4xx, Permissions, Auth, Binding, Cert Failures

[Troubleshoot](#)



Risk Assessments

Analyze your app for optimal performance and configurations.

Ex: Autoscale, AlwaysOn, Density, ARR, Health Check

[Troubleshoot](#)



Navigator (Preview)

Track changes on your app and its dependencies.

Ex: Change Analysis, SQL, Dependency, Storage

[Troubleshoot](#)



Diagnostic Tools

Run proactive tools to automatically mitigate the app.

Ex: Auto-Heal

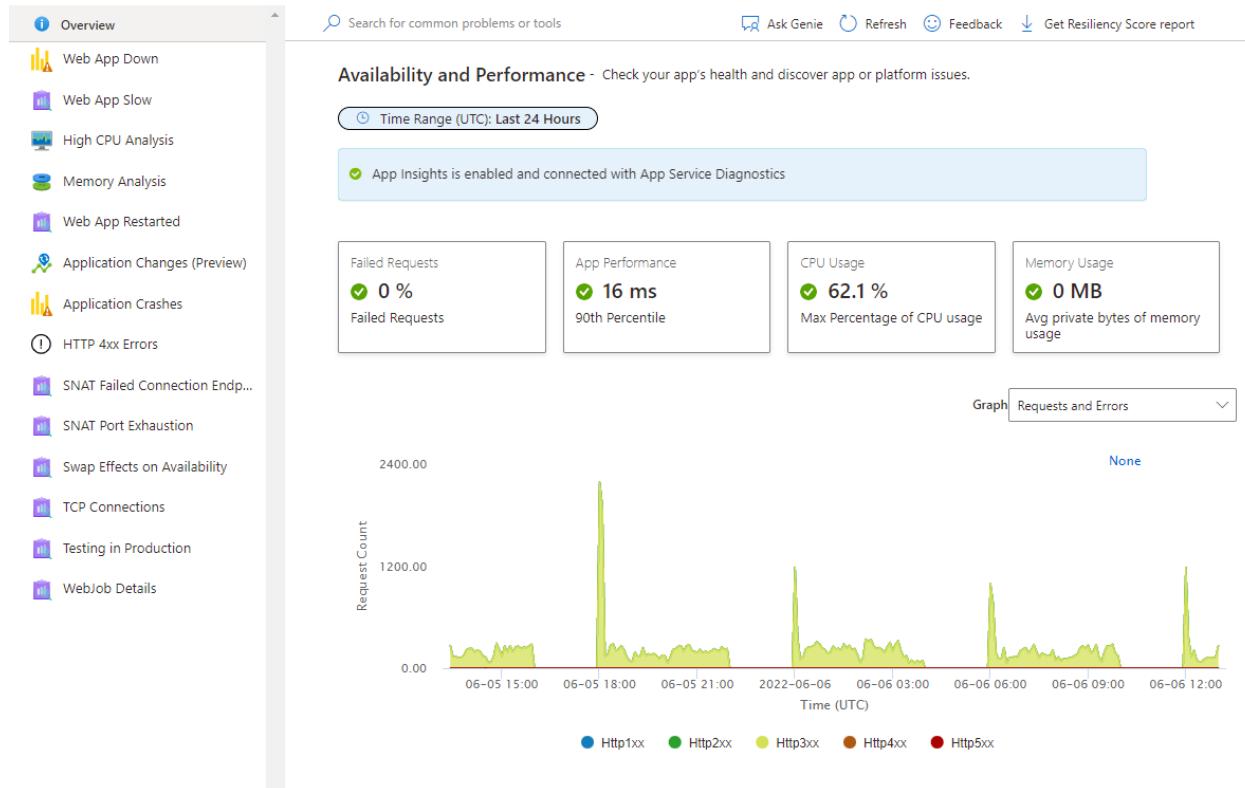
[Troubleshoot](#)

The tiles or the Troubleshoot link show the available diagnostics for the category. If you were interested in investigating Availability and performance the following diagnostics are offered:

- Overview
- Web App Down
- Web App Slow
- High CPU Analysis
- Memory Analysis
- Web App Restarted
- Application Change (Preview)
- Application Crashes
- HTTP 4xx Errors
- SNAT Failed Connection Endpoints
- SWAP Effects on Availability
- TCP Connections
- Testing in Production
- WebJob Details

buggybakery | Availability and Performance ...

X



Diagnostic report

After you choose to investigate the issue further by clicking on a topic, you can view more details about the topic often supplemented with graphs and markdowns. Diagnostic report can be a powerful tool for pinpointing the problem with your app. The following is the Web App Down from Availability and Performance:

buggybakery | Availability and Performance ...

The screenshot shows the Azure Monitor interface for the 'buggybakery' application. On the left, there's a sidebar with various troubleshooting categories like 'Web App Down', 'Web App Slow', etc. The main area is titled 'Web App Down' and includes a search bar and navigation buttons ('Ask Genie', 'Refresh', 'Feedback', '...'). It displays a chart titled 'Troubleshoot App Performance and Availability' showing 'App Availability' (blue line) and 'Platform Availability' (green line) from June 5, 2022, to June 6, 2022. A tooltip indicates a downtime of 96.88% from 02:30 PM UTC to 02:35 PM UTC. Below the chart, it says 'Organic SLA : 99.99%'. A section titled 'Observations and Solutions (1)' lists 'HTTP Server Errors' with a count of 19 errors, all being 'Bad Config Metadata'.

HttpStatus	HttpSubStatus	Errors	Description
500	19	4	Bad Config Metadata

Resiliency Score

To review tailored best practice recommendations, check out the Resiliency Score Report. This is available as a downloadable PDF Report. To get it, simply click on the "Get Resilience Score report" button available on the command bar of any of the Troubleshooting categories.

Azure App Service Resiliency Score report for: **buggybakery**

Report generated on: 2022-06-06T13:36:45.959Z

Your Web App's Resiliency Score



This is a weighted calculation based on which best practices were followed. A score of 80 or above is considered highly resilient and it will be marked as green. A score of 100% doesn't mean that the Web App will never be down but rather that it has implemented 100% of our resiliency best practices.

Contributing factors to your score and how you can improve it

Feature/Site name	buggybakery
Use of multiple instances	✗
Health Check	✗
Auto-Heal	✗
Deploy in Multiple Regions/Zones	✗
Regional Pairing	✗
App density	✓
AlwaysOn check	!
App Service Advisor Recommendations	✓
ARR Affinity Check (Recommendation. Not counted against the score)	✗
Production SKU used	✓

Investigate application code issues (only for Windows app)

Because many app issues are related to issues in your application code, App Service diagnostics integrates with [Application Insights](#) to highlight exceptions and dependency issues to correlate with the selected downtime. Application Insights has to be enabled separately.



Application Insights Analysis

We have run the following application insights analysis for you
Application Exceptions that occurred during this time period

Message	Exception	Count
No Data to Display		
Open Application Insight Failures Blade		

To view Application Insights exceptions and dependencies, select the **web app down** or **web app slow** tile shortcuts.

Troubleshooting steps

If an issue is detected with a specific problem category within the last 24 hours, you can view the full diagnostic report, and App Service diagnostics may prompt you to view more troubleshooting advice and next steps for a more guided experience.

The screenshot shows the Azure portal interface with the following details:

- Header:** Shows the date (06-05) and time (15:00). There are navigation icons for back, forward, and search.
- Left Sidebar:** Includes sections for "Observations and Solutions" (with "HTTP Server Error" expanded), "Application Insights Analysis" (with "Application Exceptions" expanded), and "Successful Checks (19)".
- Central Content:**
 - HTTP Server Errors:** A section titled "HTTP Server Errors" with a status of "Organic SLA : 99.99%".
 - Option -- 1:** A section titled "Option -- 1" with the sub-section "Review Application Insights Data". It includes a note about integrated application insights and a list of three troubleshooting steps:
 - Go to [Application Insights](#) blade for this App.
 - Click on [View Application Insights Data](#).
 - If that doesn't help, use [Azure Application Insights Snapshot Debugger](#) to debug the issue further.
 - Option -- 2:** A section titled "Option -- 2" with the sub-section "Collect .NET Profiler". It includes a note about collecting a trace to identify exceptions and a list of three troubleshooting steps:
 - Go to [Application Insights](#) blade for this App.
 - Click on [View Application Insights Data](#).
 - If that doesn't help, use [Azure Application Insights Snapshot Debugger](#) to debug the issue further.
 - Option -- 3:** A section titled "Option -- 3" with the sub-section "Configure AutoHealing Custom Action". It includes a note about configuring AutoHealing's custom action to collect data and a list of three troubleshooting steps:
 - Go to [Application Insights](#) blade for this App.
 - Click on [View Application Insights Data](#).
 - If that doesn't help, use [Azure Application Insights Snapshot Debugger](#) to debug the issue further.

Diagnostic tools

Diagnostics Tools include more advanced diagnostic tools that help you investigate application code issues, slowness, connection strings, and more. and proactive tools that help you mitigate issues with CPU usage, requests, and memory.

Proactive CPU monitoring (only for Windows app)

Proactive CPU monitoring provides you an easy, proactive way to take an action when your app or child process for your app is consuming high CPU resources. You can set your own CPU threshold rules to temporarily mitigate a high CPU issue until the real cause for the unexpected issue is found. For more information, see [Mitigate your CPU problems before they happen ↗](#).

Proactive CPU Monitoring

Proactive CPU Monitoring provides you with an easy way to take an action when your app or any child process for your app is consuming high CPU resources. The triggers allow you to define CPU thresholds at which you want the actions to be taken. This feature also helps in mitigating the issue by killing the process consuming high CPU. Please note that these mitigations should only be considered a temporary workaround until you find the real cause for the issue causing the unexpected behavior.

Storage account: **buggyb8c1nr6** ([change](#))
Diagnostic data captured via this tool will be stored in this storage account

- 1. Configure

Monitoring Enabled

On Off



CPU Threshold

This is the CPU threshold at which the rule will be triggered



Threshold Seconds

For the rule to trigger, CPU should exceed 75% for this many seconds



Auto-healing

Auto-healing is a mitigation action you can take when your app is having unexpected behavior. You can set your own rules based on request count, slow request, memory limit, and HTTP status code to trigger mitigation actions. Use the tool to temporarily mitigate an unexpected behavior until you find the root cause. The tool is currently available for Windows Web Apps, Linux Web Apps, and Linux Custom Containers. Supported conditions and mitigation vary depending on the type of the web app. For more information, see [Announcing the new auto healing experience in app service diagnostics ↗](#) and [Announcing Auto Heal for Linux ↗](#).

The screenshot shows the 'Configure Mitigation Rules' dialog box. At the top, there are three tabs: 'Configure Mitigation Rules' (selected), 'ProActive Auto-Healing' (disabled), and 'View Auto-Healing History'. Below the tabs, the 'Auto-Healing Enabled' section has a switch set to 'On'. The main area is divided into four sections: 1. Define Conditions (with buttons for Request Duration, Memory Limit, Request Count, and Status Codes), 2. Configure Actions (with buttons for Recycle Process, Log an Event, and Custom Action), 3. Override when Action executes (Optional) (with a Startup Time button), and 4. Review and Save your Settings (showing 'Current Settings: No rule configured!' and 'Save' and 'Cancel' buttons).

Proactive auto-healing (only for Windows app)

Like proactive CPU monitoring, proactive auto-healing is a turn-key solution to mitigating unexpected behavior of your app. Proactive auto-healing restarts your app when App Service determines that your app is in an unrecoverable state. For more information, see [Introducing Proactive Auto Heal](#).

Navigator and change analysis (only for Windows app)

In a large team with continuous integration and where your app has many dependencies, it can be difficult to pinpoint the specific change that causes an unhealthy behavior. Navigator helps get visibility on your app's topology by automatically rendering a dependency map of your app and all the resources in the same subscription. Navigator lets you view a consolidated list of changes made by your app and its dependencies and narrow down on a change causing unhealthy behavior. It can be accessed through the homepage tile **Navigator** and needs to be enabled before you use it the first time. For more information, see [Get visibility into your app's dependencies with Navigator](#).



2 change groups have been detected for servers/navigator-sql

Changes were last scanned on Thu, Aug 8 2019, 9:14:12 am

[Go to Change Analysis Settings](#)

Click the below button to scan your resource and get the latest changes

[Scan changes now](#)

Properties				
Code				
		Thu 1 August 2019	Fri 2	Sat 3
July 2019				

Level	Time	Name	Description	Initiated By
> 📁	Jun 5 2019, 2:45:49 pm	\Areas\HelpPage\Views\Web.config	Application file	someone@microsoft.com
> 📁	Jun 5 2019, 2:45:49 pm	\Views\Web.config	Application file	someone@microsoft.com
▼ ⚠️	Jun 5 2019, 2:45:49 pm	\Web.config	Application file	someone@microsoft.com
<pre> 1 <?xml version="1.0" encoding="utf-8"?> 2 <!-- 3 For more information on how to configure your ASP.NET application, 4 visit https://go.microsoft.com/fwlink/?LinkId=301879 5 --> 6 <configuration> 7 <connectionStrings> 8 - <add name="MyDbConnection" connectionString="Server=(localdb)\MSSQLLocalDB;Database=MyDb;Trusted_Connection=True;" /> 9 <add name="StorageConnection" connectionString="IIS APPPOOL\MyApp;Data Source=(localdb)\MSSQLLocalDB;Initial Catalog=MyStorage;Trusted_Connection=True;" /> 10 </connectionStrings> 11 <appSettings> 12 <add key="webpages:Version" value="3.0.0.0" /> 13 <add key="webpages:Enabled" value="false" /> </pre>		<pre> 1 <?xml version="1.0" encoding="utf-8"?> 2 <!-- 3 For more information on how to configure your ASP.NET application, 4 visit https://go.microsoft.com/fwlink/?LinkId=301879 5 --> 6 <configuration> 7 <connectionStrings> 8 + <add name="MyDbConnection" connectionString="Server=(localdb)\MSSQLLocalDB;Database=MyDb;Trusted_Connection=True;" /> 9 <add name="StorageConnection" connectionString="IIS APPPOOL\MyApp;Data Source=(localdb)\MSSQLLocalDB;Initial Catalog=MyStorage;Trusted_Connection=True;" /> 10 </connectionStrings> 11 <appSettings> 12 <add key="webpages:Version" value="3.0.0.0" /> 13 <add key="webpages:Enabled" value="false" /> </pre>		
> 📁	Jun 5 2019, 2:45:49 pm	\bin\changeanalysis-webapp.dll	Application file	someone@microsoft.com
> 📁	Jun 5 2019, 2:45:49 pm	\bin\changeanalysis-webapp.pdb	Application file	someone@microsoft.com

Change analysis for app changes can be accessed through tile shortcuts, **Application Changes** and **Application Crashes in Availability and Performance** so you can use it concurrently with other metrics. Before using the feature, you must first enable it. For more information, see [Announcing the new change analysis experience in App Service Diagnostics ↗](#).

Post your questions or feedback at [UserVoice ↗](#) by adding "[Diag]" in the title.

More resources

Tutorial: Run a load test to identify performance bottlenecks in a web app

Azure App Service quotas and alerts

Article • 07/10/2024

Azure App Service provides built-in monitoring functionality for web apps, mobile, and API apps in the [Azure portal](#).

In the Azure portal, you can review *quotas* and *metrics* for an app and App Service plan, and set up *alerts* and *autoscaling* rules based metrics.

Understand quotas

Apps that are hosted in App Service are subject to certain limits on the resources they can use. The limits are defined by the App Service plan that's associated with the app.

Note

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

If the app is hosted in a *Free* or *Shared* plan, the limits on the resources that the app can use are defined by quotas.

If the app is hosted in a *Basic*, *Standard*, or *Premium* plan, the limits on the resources that they can use are set by the *size* (Small, Medium, Large) and *instance count* (1, 2, 3, and so on) of the App Service plan.

Quotas for Free or Shared apps are:

 Expand table

Quota	Description
CPU (Short)	The amount of CPU allowed for this app in a 5-minute interval. This quota resets every five minutes.
CPU (Day)	The total amount of CPU allowed for this app in a day. This quota resets every 24 hours at midnight UTC.
Memory	The total amount of memory allowed for this app.

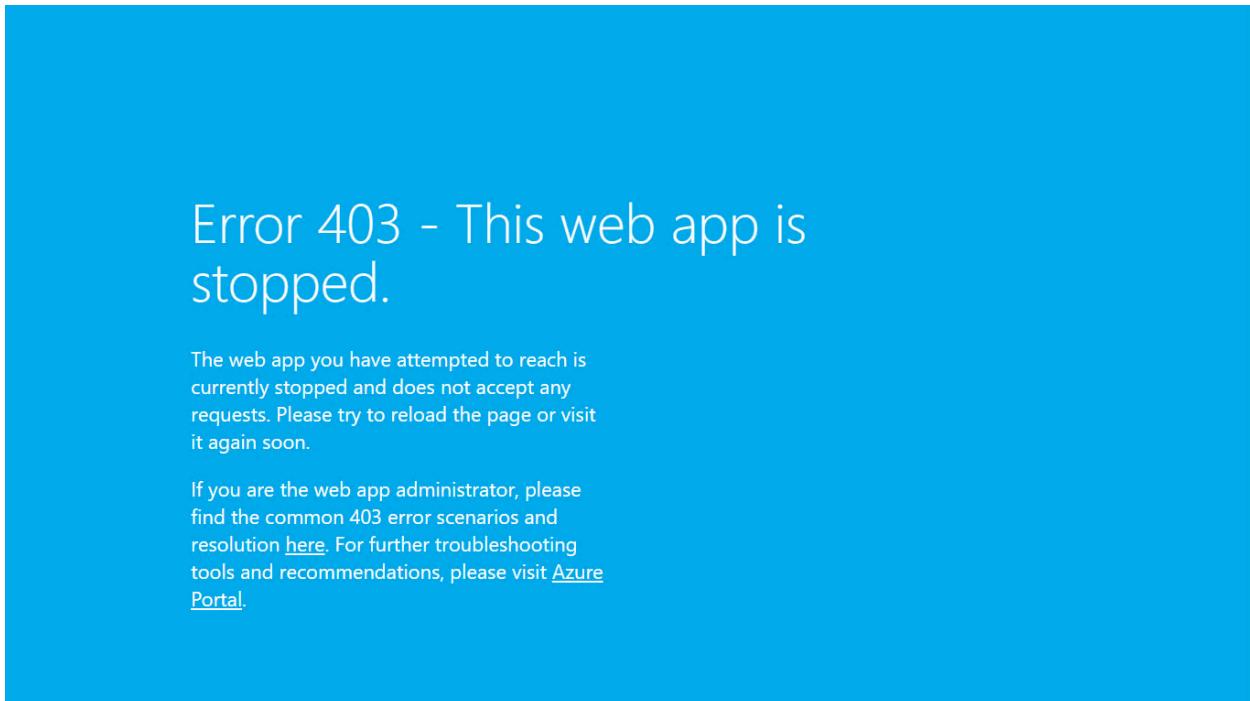
Quota	Description
Bandwidth	The total amount of outgoing bandwidth allowed for this app in a day. This quota resets every 24 hours at midnight UTC.
Filesystem	The total amount of storage allowed.

The only quota applicable to apps that are hosted in *Basic*, *Standard*, and *Premium* is Filesystem.

For more information about the specific quotas, limits, and features available to the various App Service SKUs, see [Azure Subscription service limits](#).

Quota enforcement

If an app exceeds the *CPU (short)*, *CPU (Day)*, or *bandwidth* quota, the app is stopped until the quota resets. During this time, all incoming requests result in an HTTP 403 error.



If the app Memory quota is exceeded, the app is stopped temporarily.

If the Filesystem quota is exceeded, any write operation fails. Write operation failures include any writes to logs.

You can increase or remove quotas from your app by upgrading your App Service plan.

Understand metrics

ⓘ Important

Average Response Time will be deprecated to avoid confusion with metric aggregations. Use Response Time as a replacement.

ⓘ Note

Metrics for an app include the requests to the app's SCM site(Kudu). This includes requests to view the site's logstream using Kudu. Logstream requests may span several minutes, which will affect the Request Time metrics. Users should be aware of this relationship when using these metrics with autoscale logic.

Http Server Errors only records requests that reach the backend service (the worker(s) hosting the app). If the requests are failing at the FrontEnd, they are not recorded as Http Server Errors. The [Health Check feature](#) / Application Insights availability tests can be used for outside in monitoring.

Metrics provide information about the app or the App Service plan's behavior.

For a list of available metrics for apps or for App Service plans, see [Supported metrics for Microsoft.Web](#).

ⓘ Note

App Service plan metrics are available only for plans in *Basic*, *Standard*, *Premium*, and *Isolated* tiers.

CPU time vs CPU percentage

There are two metrics that reflect CPU usage:

CPU Time: Useful for apps hosted in Free or Shared plans, because one of their quotas is defined in CPU minutes used by the app.

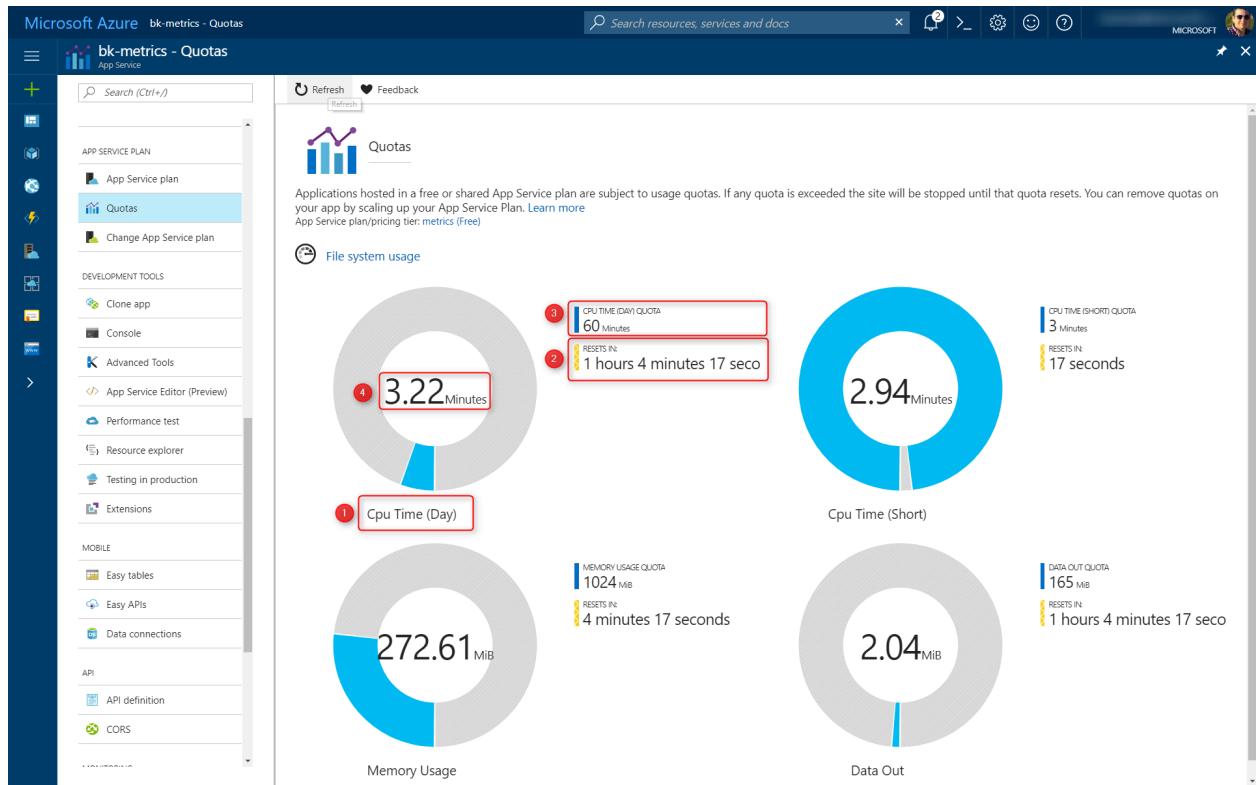
CPU percentage: Useful for apps hosted in Basic, Standard, and Premium plans, because they can be scaled out. CPU percentage is a good indication of the overall usage across all instances.

Metrics granularity and retention policy

Metrics for an app and app service plan are logged and aggregated by the service and retained according to these rules.

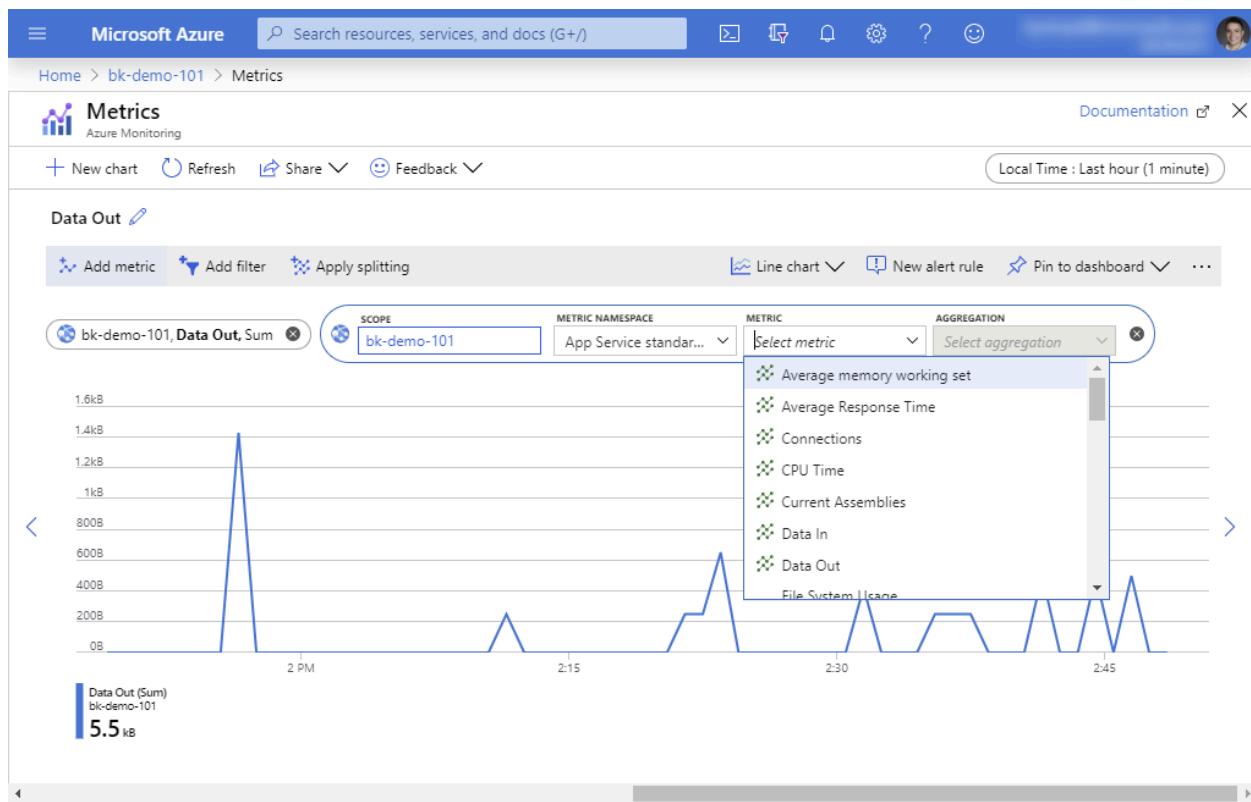
Monitoring quotas and metrics in the Azure portal

To review the status of the various quotas and metrics that affect an app, go to the [Azure portal](#).



To find quotas, select **Settings > Quotas**. On the chart, you can review:

1. The quota name.
2. Its reset interval.
3. Its current limit.
4. Its current value.



You can access metrics directly from the resource [Overview](#) page. Here you'll see charts representing some of the apps metrics.

Clicking on any of those charts will take you to the metrics view where you can create custom charts, query different metrics and much more.

To learn more about metrics, see [Monitor service metrics](#).

Alerts and autoscale

Metrics for an app or an App Service plan can be hooked up to alerts. For more information, see [Alerts](#).

App Service apps hosted in Basic or higher App Service plans support autoscale. With autoscale, you can configure rules that monitor the App Service plan metrics. Rules can increase or decrease the instance count, which can provide additional resources as needed. Rules can also help you save money when the app is over-provisioned.

For more information about autoscale, see [How to scale](#) and [Best practices for Azure Monitor autoscaling](#).

Feedback

Was this page helpful?

Yes

No

Tutorial: Secure your Azure App Service app with a custom domain and a managed certificate

Article • 02/01/2023

The default domain name that comes with your app `<app-name>.azurewebsites.net` may not represent your brand the way you want. In this tutorial, you configure App Service with a `www` domain you own, such as `www.contoso.com`, and secure the custom domain with an App Service managed certificate.

The `<app-name>.azurewebsites.net` name is already secured by a wildcard certificate for all App Service apps, but your custom domain needs to be TLS secured with a separate certificate. The easiest way is to use a managed certificate from App Service. It's free and easy to use, and it provides the basic functionality of securing a custom domain in App Service. For more information, see [Add a TLS certificate to App Service](#).

Scenario prerequisites

- [Create an App Service app](#).
- Make sure you can edit the DNS records for your custom domain. To edit DNS records, you need access to the DNS registry for your domain provider, such as GoDaddy. For example, to add DNS entries for `www.contoso.com`, you must be able to configure the DNS settings for the `contoso.com` root domain. Your custom domains must be in a public DNS zone; private DNS zone is only supported on Internal Load Balancer (ILB) App Service Environment (ASE).
- If you don't have a custom domain yet, you can [purchase an App Service domain](#).

A. Scale up your app

You need to scale your app up to **Basic** tier. **Basic** tier fulfills the minimum pricing tier requirement for custom domains (**Shared**) and certificates (**Basic**).

Step 1. In the Azure portal:

1. Enter the name of your app in the search bar at the top.
2. Select your named resource with the type **App Service**.

Microsoft Azure

my-demo-app2

Azure services

Create a resource

App Services

Subscriptions

Azure Virtual Desktop

Resources

Recent Favorite

Name

All Resources (2)

Documentation (30)

Azure Active Directory (10) Services (0)

Resource Groups (0) Marketplace (0)

my-demo-app2 Application Insights

my-demo-app2 App Service

Documentation See all

Configure apps - Azure App Service | Microsoft ...

Continue searching in Azure Active Directory

Searching 95 of 130 subscriptions. Change Give feedback

Step 2. In your app's management page:

1. In the left navigation, select **Scale up (App Service plan)**.
2. Select the checkbox for **Basic B1**.
3. Select **Select**. When the app update is complete, you see a notification toast.

Search

Backups

Custom domains

Custom domains (classic)

Certificates

TLS/SSL settings (classic)

Networking

Scale up (App Service plan)

Scale out (App Service plan)

WebJobs

Push

MySQL In App

Service Connector

Properties

Hardware view Feature view Showing 17 App Service pricing plans

Name	Cost per hour	Cost per month
Dev/Test (For less demanding workloads)		
Free F1	Free	Free
Shared D1	X.XXX USD	X.XXX USD
Basic B1	X.XXX USD	X.XXX USD
Basic B2	X.XXX USD	X.XXX USD
Basic B3	X.XXX USD	X.XXX USD
Production (For most production workloads)		
Premium v2 P1V2	X.XXX USD	X.XXX USD

Select Learn more about App Service pricing

For more information on app scaling, see [Scale up an app in Azure App Service](#).

B. Configure a custom domain

Step 1. In your app's management page:

1. In the left menu, select **Custom domains**.
2. Select **Add custom domain**.

The screenshot shows the 'Custom domains' section of the Azure App Service management portal. On the left, there's a sidebar with various settings like Configuration, Authentication, Application Insights, Identity, Backups, and Custom domains (which is highlighted with a red box). The main area shows a table with one item: 'contoso.azurewebsites.net' under 'Custom domains', 'Secured' under 'Status', and '-' under 'Solution'. At the top right, there are buttons for 'Add custom domain' (highlighted with a red box), 'Buy App Service domain', and 'Delete'. Below the table are navigation buttons for 'Previous', 'Page 1 of 1', and 'Next'.

Step 2. In the Add custom domain dialog:

1. For **Domain provider**, select **All other domain services**.
2. For **TLS/SSL certificate**, select **App Service Managed Certificate**.
3. For **Domain**, specify a fully qualified domain name you want based on the domain you own. For example, if you own `contoso.com`, you can use www.contoso.com.
4. Don't select **Validate** yet.

Add custom domain

X

Domain provider * ⓘ

App Service Domain

All other domain services

TLS/SSL certificate * ⓘ

App Service Managed Certificate

Add certificate later

TLS/SSL type * ⓘ

SNI SSL

IP based SSL

Enter your custom domain. We'll give you hostname records to register with your domain provider, and we can validate and add your custom domain here. [Learn more ↗](#)

Domain * ⓘ

www.contoso.com

Hostname record type * ⓘ

CNAME (www.example.com or any subdo... ▾)

Domain validation

To validate your domain ownership, copy the hostname records below and enter them with your domain provider. [Learn more ↗](#)

Type	Host	Value	Status
CNAME	www	contoso.azurewebsites.net	
TXT	asuid.www	XXXXXXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXXXXXX XXXXXXXXXXXX	

If you use traffic manager, make sure to point it to the CNAME record.

[Validate](#)

[Add](#)

[Cancel](#)



For each custom domain in App Service, you need two DNS records with your domain provider. The **Domain validation** section shows you two DNS records that you must add with your domain provider. Select the respective **Copy** button to help you with the next step.

C. Create the DNS records

Sign in to the website of your domain provider.

1. Find the page for managing DNS records, **Domain Name, DNS, or Name Server Management** (the exact page differs by domain provider).
2. Select **Add** or the appropriate widget to create a DNS record.
3. Select the DNS record type based on the **Domain validation** section in the Azure portal (CNAME, A, or TXT).
4. Configure the DNS record based on the **Host** and **Value** columns from the **Domain validation** section in the Azure portal.
5. Be sure to add two different records for your custom domain.
6. For certain providers, changes to DNS records don't become effective until you select a separate **Save Changes** link. The screenshot shows what your DNS records should look like for a `www` subdomain after you're finished.

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

D. Validate and complete

Step 1. Back in the **Add custom domain** dialog in the Azure portal, select **Validate**.

Domain validation

To validate your domain ownership, copy the hostname records below and enter them with your domain provider. [Learn more ↗](#)

Type	Host	Value	Status
CNAME	www	contoso.azurewebsites.net	
TXT	asuid.www	XXXXXXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXXXXXX XXXXXXXXXXXX	

If you use traffic manager, make sure to point it to the CNAME record.

Validate

Add

Cancel



Step 2. If the **Domain validation** section shows green check marks next for both domain records, then you've configured them correctly. Select **Add**. If it shows any red X, fix any errors in the DNS record settings in your domain provider's website.

Domain validation

To validate your domain ownership, copy the hostname records below and enter them with your domain provider. [Learn more ↗](#)

Type	Host	Value	Status
CNAME	www	contoso.azurewebsites.net	
TXT	asuid.www	XXXXXXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXXXXXX XXXXXXXXXXXX	

If you use traffic manager, make sure to point it to the CNAME record.

Validation passed. Select **Add** to finish up.

Validate

Add

Cancel



Step 3. You should see the custom domain added to the list. You may also see a red X with **No binding**. Wait a few minutes for App Service to create the managed certificate for your custom domain. When the process is complete, the red X becomes a green check mark with **Secured**.

 Refresh  Troubleshoot

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

IP address: 

Custom Domain Verification ID: 

  Add filter

2 items

 Add custom domain |  Buy App Service domain |  Delete

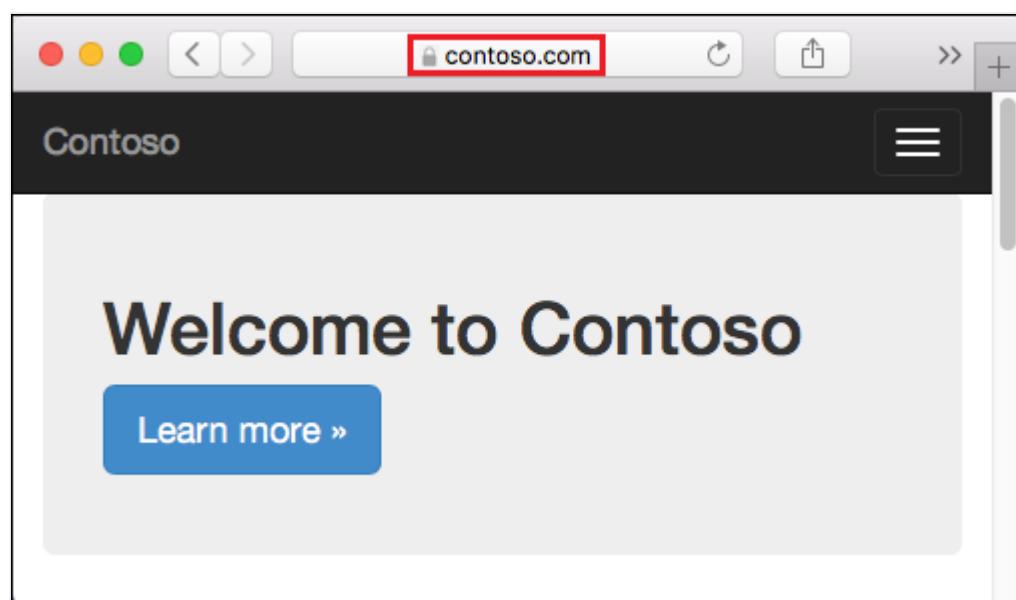
Custom domains	Status	Solution
<input type="checkbox"/> www.contoso.com	 Secured	-
contoso.azurewebsites.net	 Secured	-

[<> Previous](#) Page of 1 [Next >](#)



E. Test in a browser

Browse to the DNS names that you configured earlier (like `www.contoso.com`). The address bar should now show the security lock icon for your app's URL, indicating that it's secured by TLS.



If you receive an HTTP 404 (Not Found) error when you browse to the URL of your custom domain, the browser client may have cached the old IP address of your custom

domain. Clear the cache, and try navigating to the URL again. On a Windows machine, you clear the cache with `ipconfig /flushdns`.

Frequently asked questions

- [What do I do if I don't have a custom domain yet?](#)
- [Does this managed certificate expire?](#)
- [What else can I do with the App Service managed certificate for my app?](#)
- [How do I use a certificate I already have to secure my custom domain?](#)

What do I do if I don't have a custom domain yet?

The `<app-name>.azurewebsites.net` name is always assigned to your app as long as you don't delete it. If you want, you can [purchase an App Service domain](#). An App Service domain is managed by Azure and is integrated with App Service, making it easier to manage together with your apps.

Does this managed certificate expire?

The App Service managed certificate doesn't expire as long as it's configured for a custom domain in an App Service app.

What else can I do with the App Service managed certificate for my app?

The managed certificate is provided for free for the sole purpose of securing your app's configured custom domain. It comes with [a number of limitations](#). To do more, such as download the certificate, or use it in your application code, you can upload your own certificate, purchase an App Service certificate, or import a Key Vault certificate. For more information, see [Add a private certificate to your app](#).

How do I use a certificate I already have to secure my custom domain?

See [Add a private certificate to your app](#) and [Secure a custom DNS name with a TLS/SSL binding in Azure App Service](#).

Next steps

- Map an existing custom DNS name to Azure App Service
- Purchase an App Service domain
- Add a private certificate to your app
- Secure a custom DNS name with a TLS/SSL binding in Azure App Service

Map an existing custom DNS name to Azure App Service

Article • 09/13/2024

ⓘ Note

Starting June 1, 2024, all newly created App Service apps will have the option to generate a unique default hostname using the naming convention <app-name>-<random-hash>. <region>.azurewebsites.net. Existing app names will remain unchanged.

Example: myapp-ds27dh7271ah175.westus-01.azurewebsites.net

For further details, refer to [Unique Default Hostname for App Service Resource ↗](#).

Azure App Service provides a highly scalable, self-patching web hosting service. This guide shows you how to map an existing custom Domain Name System (DNS) name to App Service. To migrate a live site and its DNS domain name to App Service with no downtime, see [Migrate an active DNS name to Azure App Service](#).

The DNS record type you need to add with your domain provider depends on the domain you want to add to App Service.

ⓘ [Expand table](#)

Scenario	Example	Recommended DNS record
Root domain	contoso.com	A record ↗ . Don't use the CNAME record for the root record. (For information, see RFC 1912, section 2.4 ↗ .)
Subdomain	www.contoso.com, my.contoso.com	CNAME record ↗ . You can map a subdomain to the app's IP address directly with an A record, but it's possible for the IP address to change . The CNAME maps to the app's default hostname instead, which is less susceptible to change.
Wildcard	*.contoso.com	CNAME record ↗ .

ⓘ Note

For an end-to-end tutorial that shows you how to configure a subdomain and a managed certificate, see [Tutorial: Secure your Azure App Service app with a](#)

custom domain and a managed certificate.

Prerequisites

- Create an App Service app, or use an app that you created for another tutorial. The web app's App Service plan must be a paid tier, not the Free (F1) tier. See [Scale up an app](#) to update the tier.
- Make sure you can edit the DNS records for your custom domain. To edit DNS records, you need access to the DNS registry for your domain provider, such as GoDaddy. For example, to add DNS entries for `contoso.com` and `www.contoso.com`, you must be able to configure the DNS settings for the `contoso.com` root domain. Your custom domains must be in a public DNS zone; private DNS zones are not supported.
- If you don't have a custom domain yet, you can [purchase an App Service domain](#) instead.

Configure a custom domain

1. In the [Azure portal](#), navigate to your app's management page.
2. In the left menu for your app, select **Custom domains**.
3. Select **Add custom domain**.

The screenshot shows the Azure portal interface for managing custom domains. The left sidebar has a 'Settings' section with various options like Configuration, Authentication, Application Insights, Identity, Backups, and Custom domains. The 'Custom domains' option is highlighted with a red box. The main content area is titled 'Configure and manage custom domains assigned to your app.' It includes fields for 'IP address' and 'Custom Domain Verification ID', both with browse buttons. Below these are filters ('Filter by keywords' and 'Add filter') and a table showing one item: 'contoso.azurewebsites.net' with a status of 'Secured'. At the bottom, there are buttons for '+ Add custom domain', '+ Buy App Service domain', and 'Delete'. Navigation at the bottom includes '< Previous', 'Page 1 of 1', and 'Next >'.

4. For **Domain provider**, select **All other domain services** to configure a third-party domain.

 **Note**

To configure an App Service domain, see [Buy a custom domain name for Azure App Service](#).

5. For **TLS/SSL certificate**, select **App Service Managed Certificate** if your app is in the Basic tier or higher. If you want to remain in the Shared tier, or if you want to use your own certificate, select **Add certificate later**.
6. For **TLS/SSL type**, select the binding type you want.

 [Expand table](#)

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 based SSL. IP SSL is supported only in Standard tier or above.

7. For **Domain**, specify a fully qualified domain name you want based on the domain you own. The **Hostname record type** box defaults to the recommended DNS record to use, depending on whether the domain is a root domain (like `contoso.com`), a subdomain (like `www.contoso.com`), or a wildcard domain (like `*.contoso.com`).
8. Don't select **Validate** yet.
9. For each custom domain in App Service, you need two DNS records with your domain provider. The **Domain validation** section shows you two DNS records that you must add with your domain provider. You can use the copy buttons to copy the value or values that you need in the next section.

The following screenshot shows the default selections for a `www.contoso.com` domain. It shows a CNAME record and a TXT record to add.

Add custom domain

Domain provider * ⓘ App Service Domain All other domain services

TLS/SSL certificate * ⓘ App Service Managed Certificate Add certificate later

TLS/SSL type * ⓘ SNI SSL IP based SSL

Enter your custom domain. We'll give you hostname records to register with your domain provider, and we can validate and add your custom domain here. [Learn more ↗](#)

Domain * ⓘ `www.contoso.com`

Hostname record type * ⓘ `CNAME (www.example.com or any subdo...)`

Domain validation

To validate your domain ownership, copy the hostname records below and enter them with your domain provider. [Learn more ↗](#)

Type	Host	Value	Status
CNAME	www	contoso.azurewebsites.net	
TXT	asuid.www	XXXXXXXXXXXXXXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXXXXXX	

If you use traffic manager, make sure to point it to the CNAME record.

Validate **Add** **Cancel**

⚠ Warning

While it's not absolutely required to add the TXT record, it's highly recommended for security. The TXT record is a *domain verification ID* that helps avoid subdomain takeovers from other App Service apps. For custom

domains you previously configured without this verification ID, you should protect them from the same risk by adding the verification ID (the TXT record) to your DNS configuration. For more information on this common high-severity threat, see [Subdomain takeover](#).

Create the DNS records

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

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

2. Find the page for managing DNS records.

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

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

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

Records

Last updated 6/18/2018 3:40 PM

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

[ADD](#)

3. Select **Add** or the appropriate widget to create a record.

Note

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

Select the type of record to create and follow the instructions. You can use either a [CNAME record](#) or an [A record](#) to map a custom DNS name to App Service. When your function app is hosted in a [Consumption plan](#), only the CNAME option is supported.

Root domain (for example, contoso.com)

Create two records, as described in the following table:

 [Expand table](#)

Record type	Host	Value	Comments
A	@	The app's IP address shown in the Add custom domain dialog .	The domain mapping itself. (@ typically represents the root domain.)
TXT	asuid	The domain verification ID shown in the Add custom domain dialog .	For the root domain, App Service accesses the asuid TXT record to verify your ownership of the custom domain.

Name	Type	TTL	Value
@	A	3600	40.118.102.46
@	NS	172800	ns1-05.azure-dns.com. ns2-05.azure-dns.net. ns3-05.azure-dns.org. ns4-05.azure-dns.info.
@	SOA	3600	Email: azuredns-hostmaster.microsoft.com Host: ns1-05.azure-dns.com. Refresh: 3600 Retry: 300 Expire: 2419200 Minimum TTL: 300 Serial number: 1
asuid	TXT	3600	<domain-verification-id-from-your-app>

Validate domain ownership and complete the mapping

1. Back in the [Add custom domain](#) dialog in the Azure portal, select **Validate**.

Domain validation

To validate your domain ownership, copy the hostname records below and enter them with your domain provider. [Learn more ↗](#)

Type	Host	Value	Status
CNAME	www	contoso.azurewebsites.net	
TXT	asuid.www	XXXXXXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXXXXXX XXXXXXXXXXXX	

If you use traffic manager, make sure to point it to the CNAME record.

Validate

Add

Cancel

2. If the **Domain validation** section shows green check marks next to both domain records, you've configured them correctly. Select **Add**. If you see any errors or warnings, resolve them in the DNS record settings on your domain provider's website.

Domain validation

To validate your domain ownership, copy the hostname records below and enter them with your domain provider. [Learn more ↗](#)

Type	Host	Value	Status
CNAME	www	contoso.azurewebsites.net	
TXT	asuid.www	XXXXXXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXXXXXX XXXXXXXXXXXX	

If you use traffic manager, make sure to point it to the CNAME record.

Validation passed. Select **Add** to finish up.

Validate

Add

Cancel

ⓘ Note

If you configured the TXT record but not the A or CNAME record, App Service treats the change as a **domain migration** scenario and allows the validation to succeed, but you won't see green check marks next to the records.

3. You should see the custom domain added to the list. You might also see a red X and the text **No binding**.

If you selected **App Service Managed Certificate** earlier, wait a few minutes for App Service to create the managed certificate for your custom domain. When the process is complete, the red X becomes a green check mark and you see the word **Secured**. If you selected **Add certificate later**, the red X will remain until you [add a private certificate for the domain](#) and [configure the binding](#).

The screenshot shows the 'Custom Domains' blade in the Azure portal. At the top, there are refresh and troubleshoot buttons. Below that, a message says 'Configure and manage custom domains assigned to your app.' with a 'Learn more' link. There are input fields for 'IP address' and 'Custom Domain Verification ID', each with a copy icon. Below these are filter and add filter buttons. The main area shows a table with two items:

Custom domains	Status	Solution
<input type="checkbox"/> www.contoso.com	✓ Secured	-
contoso.azurewebsites.net	✓ Secured	-

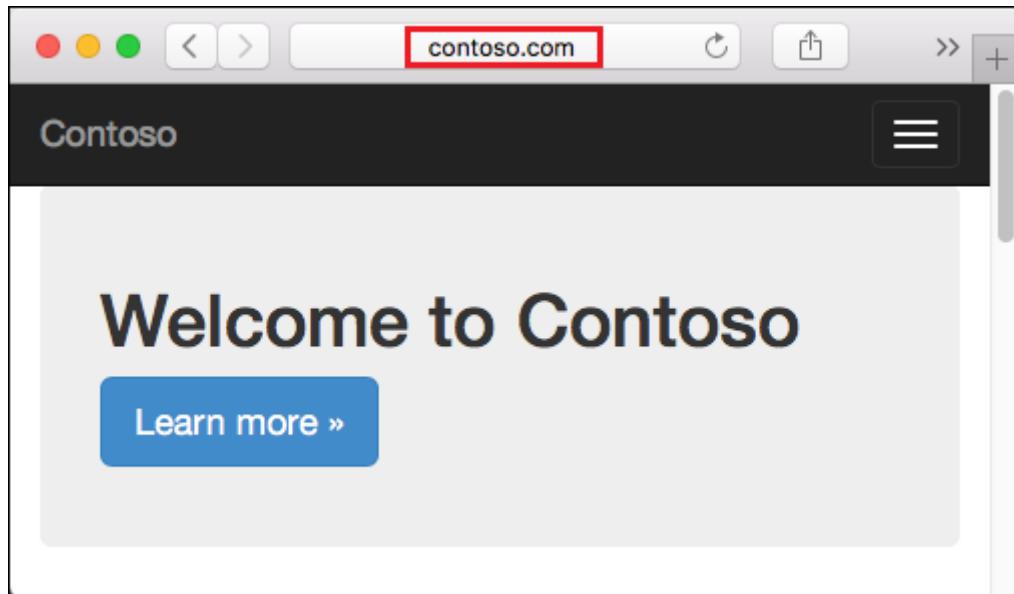
At the bottom, there are navigation buttons for '< Previous', 'Page 1 of 1', and 'Next >'.

ⓘ Note

Unless you configure a certificate binding for your custom domain, any HTTPS request from a browser to the domain will receive an error or warning, depending on the browser.

Test the DNS resolution

Browse to the DNS names that you configured.



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

- The browser client has cached the old IP address of your domain. Clear the cache and test the DNS resolution again. On a Windows machine, you can clear the cache with `ipconfig /flushdns`.
- You configured an IP-based certificate binding, and the app's IP address has changed because of it. [Remap the A record](#) in your DNS entries to the new IP address.

If you receive a `Page not secure` warning or error, it's because your domain doesn't have a certificate binding yet. [Add a private certificate for the domain](#) and [configure the binding](#).

(Optional) Automate with scripts

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

Azure CLI

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

Azure CLI

```
az webapp config hostname add \
--webapp-name <app-name> \
```

```
--resource-group <resource_group_name> \
--hostname <fully_qualified_domain_name>
```

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

Next steps

[Purchase an App Service domain](#)

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

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

Buy an App Service domain and configure an app with it

Article • 09/13/2024

App Service domains are custom domains that are managed directly in Azure. They make it easy to manage custom domains for [Azure App Service](#). This article shows you how to buy an App Service domain and configure an App Service app with it.

Prerequisites

- [Create an App Service app](#), or use an app that you created for another tutorial. The app should be in an Azure Public region. At this time, Azure national clouds are not supported.
- To use an App Service domain, the app's [App Service plan](#) must be a paid tier, not a Free (F1) tier. See [Scale up an app](#) to update the tier.
- [Remove the spending limit on your subscription](#).

Note

For some subscriptions types, before you can create an App Service domain, the subscription needs to have sufficient history on Azure. App Service domains aren't supported on free trial or credit-based subscriptions.

Buy and map an App Service domain

For pricing information on App Service domains, visit the [App Service Pricing page](#) and scroll down to App Service Domain.

1. In the [Azure portal](#), navigate to your app's management page.
2. In the left menu for your app, select **Custom domains**.
3. Select **Buy App Service domain**.

The screenshot shows the Azure portal's 'Custom domains' settings for an app service. On the left, a sidebar lists various configuration options like Configuration, Authentication, Application Insights, Identity, Backups, and Custom domains (which is selected and highlighted with a red box). The main area displays a table of custom domains. At the top of the table, there are buttons for 'Add custom domain' and 'Buy App Service domain' (also highlighted with a red box). The table has columns for 'Custom domains', 'Status', and 'Solution'. One row is shown: 'contoso.azurewebsites.net' with a green checkmark under 'Status' and '-' under 'Solution'. Navigation controls at the bottom include '< Previous', 'Page 1 of 1', and 'Next >'.

➊ Note

You can also create an App Service domain independently of an app by going to the App Service Domains view and selecting **Create**, or by navigating to [the create page directly](#). But since the domain is independent from your app, you won't be able to assign hostnames like `www` to your app, which you can do if you create the domain from your app's **Custom domains** page.

4. In the **Basics** tab, configure the following settings:

[] [Expand table](#)

Setting	Description
Subscription	The subscription to use to buy the domain.
Resource Group	The resource group to put the domain in. For example, the resource group your app is in.
Domain	The domain you want. For example, <code>contoso.com</code> . If the domain you want isn't available, you can select from a list of suggested available domains or try a different domain.

➋ Note

The following [top-level domains](#) are supported by App Service domains: *com, net, co.uk, org, nl, in, biz, org.uk, and co.in*.

5. Select **Next: Contact information** and supply your information as required by [ICANN](#) for the domain registration.

It's important that you fill out all required fields with as much accuracy as possible. Incorrect data for contact information can result in failure to buy the domain.

 **Note**

Make sure you have access to the email address on your contact information. GoDaddy will send emails directly to your contact information from a "@secureserver.net" email; these will only be important transactional emails.

6. Select **Next: Hostname assignment** and verify the default hostnames to map to your app:

 [Expand table](#)

Hostname	Description
root (@)	The root or apex subdomain. If you buy the <code>contoso.com</code> domain, that's the root domain. Select No if you don't want to map the hostname to your app.
'www' subdomain	If you buy the <code>contoso.com</code> domain, the <code>www</code> subdomain would be <code>www.contoso.com</code> . Select No if you don't want to map the hostname to your app.

 **Note**

If you didn't launch the App Service domain wizard from an app's **Custom domains** page, you won't see this tab. You can still add the hostnames later by following the steps at [Map a hostname manually](#).

7. Select **Next: Advanced** and configure the optional settings:

 [Expand table](#)

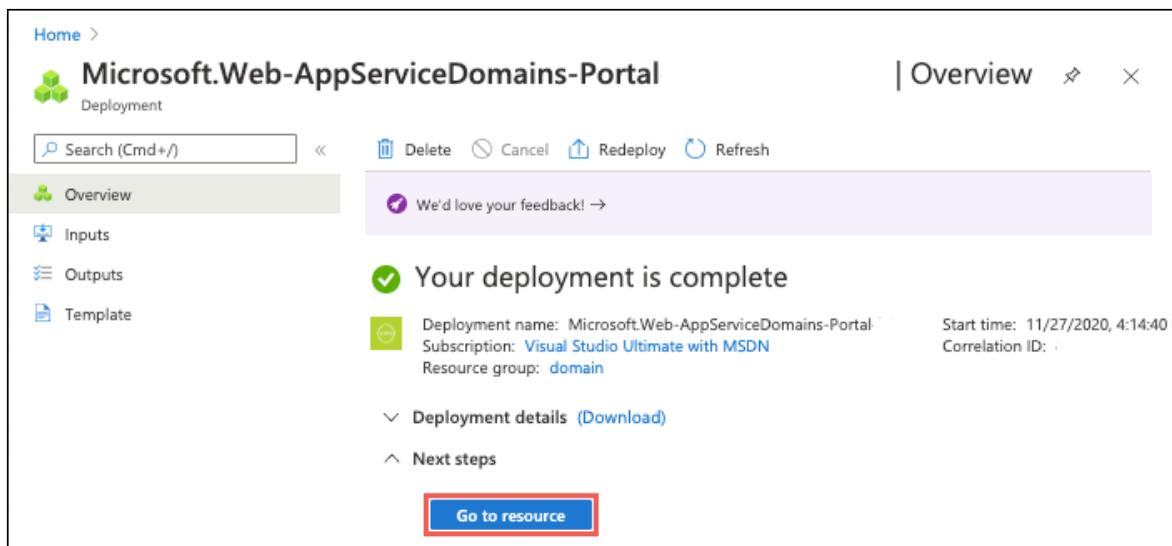
Setting	Description
Auto renewal	Your App Service domain is registered to you at one-year increments. Enable auto renewal so that your domain registration doesn't expire and you retain ownership of the domain. Your Azure subscription is automatically charged the yearly domain registration fee at the time of renewal. If you leave this option disabled, you must renew the domain manually .
Privacy protection	Enabled by default. Privacy protection hides your domain registration contact information from the WHOIS database and is already included in the yearly domain registration fee. To opt out, select Disable . Privacy protection isn't supported in following top-level domains (TLDs): co.uk, in, org.uk, co.in, and nl.

8. Select **Next: Tags** and set the tags you want for your App Service domain. Tagging isn't required. It's a [feature in Azure that helps you manage your resources](#).
9. Select **Next: Review + create** and review your domain order. When finished, select **Create**.

 **Note**

App Service domains use GoDaddy for domain registration and Azure DNS to host the domains. In addition to the yearly domain registration fee, usage charges for Azure DNS apply. For information, see [Azure DNS Pricing](#).

10. When the domain registration is complete, you see a **Go to resource** button. Select it to see the management page.



You're now ready to assign an App Service app to this custom domain.

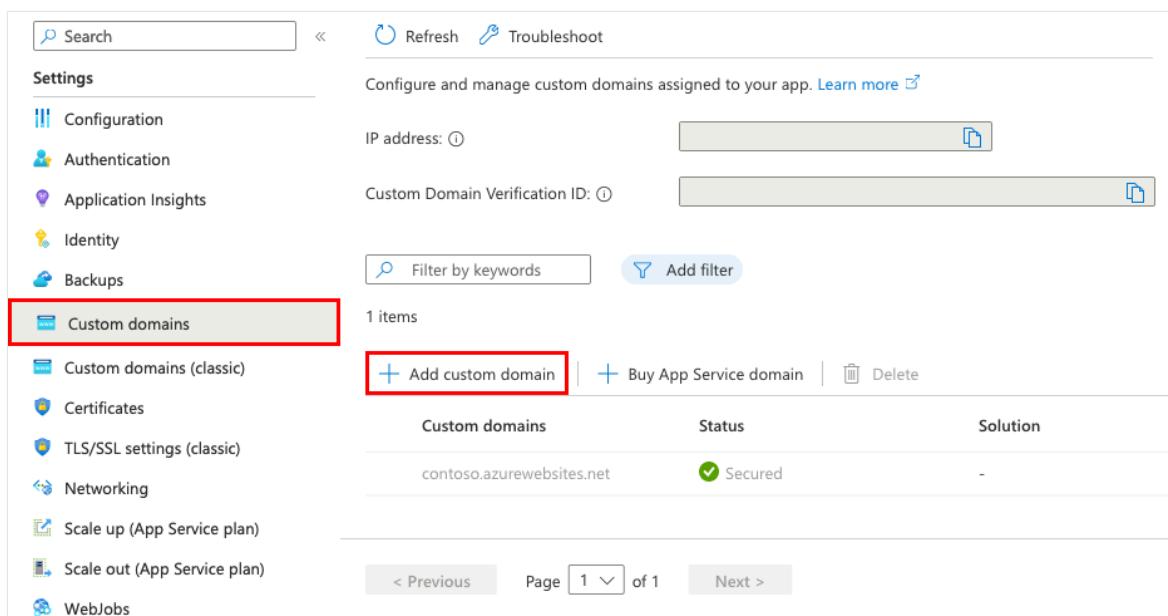
(!) Note

For some subscription types, before you can create an App Service domain, the subscription needs to have sufficient history on Azure. App Service domains aren't supported on free trial or credit-based subscriptions.

Map a hostname manually

If launched from an app's **Custom domains** page, the App Service domain wizard already lets you map the root domain (like `contoso.com`) and the `www` subdomain (like `www.contoso.com`) to your app. You can map any other subdomain to your app, like `shoppingcart` (as in `shoppingcart.contoso.com`).

1. In the [Azure portal](#), navigate to your app's management page.
2. In the left menu for your app, select **Custom domains**.
3. Select **Add custom domain**.



The screenshot shows the 'Custom domains' settings page in the Azure portal. The left sidebar has a 'Custom domains' section highlighted with a red box. The main area shows a table with one item: 'contoso.azurewebsites.net' under 'Custom domains', 'Secured' under 'Status', and '-' under 'Solution'. At the bottom of the table is a 'Delete' button. Below the table are buttons for 'Add custom domain' (highlighted with a red box) and 'Buy App Service domain'. At the very bottom are navigation buttons for 'Previous', 'Page 1 of 1', and 'Next'.

Custom domains	Status	Solution
contoso.azurewebsites.net	Secured	-

4. For **Domain provider**, select **App Service Domain**.
5. For **TLS/SSL certificate**, select **App Service Managed Certificate** if your app is in the Basic tier or higher. If you want to remain in the Shared tier, or if you want to use your own certificate, select **Add certificate later**.
6. For **TLS/SSL type**, select the binding type you want.

[+] Expand table

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 based SSL. IP SSL is supported only in Standard tier or above.

7. In **App Service Domain**, select an App Service domain in your subscription.

 **Note**

To map to an App Service domain in a different subscription, see [Map an externally purchased domain](#). In this case, Azure DNS is the external domain provider, and you need to add the required DNS records manually.

8. In **Domain type**, configure the domain type you want to map:

[\[\] Expand table](#)

Domain type	Description
Root domain	The root or apex subdomain. If you buy the <code>contoso.com</code> domain, that's the root domain.
Subdomain	In the Subdomain box, specify a subdomain like <code>www</code> or <code>shoppingcart</code> .

9. Select **Add**.

Add custom domain

X

Domain provider * ⓘ

App Service Domain

All other domain services

TLS/SSL certificate * ⓘ

App Service Managed Certificate

Add certificate later

TLS/SSL type * ⓘ

SNI SSL

IP based SSL

App Service Domain * ⓘ

contoso.com

▼

[Create new](#)

Domain type * ⓘ

Root domain

Subdomain

Subdomain * ⓘ

shoppingcart

.contoso.com

i Please note that it might take up to 10 minutes for App Service Managed Certificate to be issued.

Validate

Add

Cancel

10. You should see the custom domain added to the list. You might also see a red X and the text **No binding**.

If you selected **App Service Managed Certificate** earlier, wait a few minutes for App Service to create the managed certificate for your custom domain. When the process is complete, the red X becomes a green check mark with the word **Secured**. If you selected **Add certificate later**, the red X will remain until you [add a private certificate for the domain](#) and [configure the binding](#).



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

IP address: [\(i\)](#)



Custom Domain Verification ID: [\(i\)](#)



[Filter by keywords](#)

[Add filter](#)

4 items

[Add custom domain](#)

[Buy App Service domain](#)

[Delete](#)

Custom domains	Status	Solution
<input type="checkbox"/> contoso.com	Secured	-
<input type="checkbox"/> shoppingcart.contoso.com	Secured	-
<input type="checkbox"/> www.contoso.com	Secured	-
contoso.azurewebsites.net	Secured	-

< Previous

Page of 1

Next >

Note

Unless you configure a certificate binding for your custom domain, any HTTPS request from a browser to the domain will receive an error or warning, depending on the browser.

11. Test the mapping by navigating to it in a browser. (For example, go to

`shoppingcart.contoso.com`.)

Renew the domain

The App Service domain you bought is valid for one year from the time of purchase. You can configure your domain to renew automatically, or you can manually renew your domain name up to 90 days ahead of domain expiration. Upon successful automatic or manual renewal, you'll be billed for the cost of the domain, and your domain expiration will be extended for another year.

Note

For .nl domains, you can only manually renew the domain starting 90 days ahead of domain expiration and up to the 20th of the month before the expiration date. You won't be able to renew the domain after this period even if the domain hasn't yet expired.

If you want to configure automatic renewal, or if you want to manually renew your domain, follow these steps:

1. In the search bar, search for and select App Service Domains.

The screenshot shows the Microsoft Azure (Preview) portal interface. At the top, there is a search bar with the text "App Service Domains" and a magnifying glass icon. A red box highlights the search bar. Below the search bar, the results list is displayed under the heading "Services". The "App Service Domains" item is listed and highlighted with a red box. Other items in the list include "Marketplace (1)", "Documentation (27)", "Azure Active Directory (2)", "Resources (0)", and "Resource Groups (0)". At the bottom of the results list, there is a link "Continue searching in Azure Active Dire..." and some status information: "Searching 94 of 137 subscriptions. Change Give feedback".

2. Select the domain you want to configure.
3. From the left navigation of the domain, select **Domain renewal**. To start renewing your domain automatically, select **On**. Otherwise select **Off**. The setting takes effect immediately. If automatic renewal is enabled, on the day after your domain expiration date, Azure attempts to bill you for the domain name renewal.

Search

Diagnose and solve problems

Settings

Properties

Locks

Domain Management

Advanced Management portal

Domain renewal

Automation

Tasks (preview)

Export template

Support + troubleshooting

FAQs

Domain renewal

App Service domains can be set to auto renew to prevent expiration and unexpected domain ownership loss.

Auto renew domain:

On Off

⚠️ Manual domain renewal can only be performed up to 90 days ahead of domain expiration and up to 18 days after domain expiration. You can enable domain auto-renewal policy to reduce the management overhead of this operations.

Expiration date:

October 7, 2023 at 7:00:37 PM GMT+2

Renew domain

ⓘ Note

When navigating away from the page, disregard the "Your unsaved edits will be discarded" error by selecting **OK**.

To manually renew your domain, select **Renew domain**. However, this button isn't active until 90 days before the domain's expiration date.

If your domain renewal is successful, you receive an email notification within 24 hours.

Manage custom DNS records

In Azure, DNS records for an App Service domain are managed using [Azure DNS](#). You can add, remove, and update DNS records just as you would for an externally purchased domain. To manage custom DNS records:

1. In the search bar, search for and select **App Service Domains**.

The screenshot shows the Microsoft Azure (Preview) interface. At the top, there's a search bar with the placeholder "App Service Domains". Below the search bar, the "Azure services" section includes links for "Create a resource", "App Service Domains", "Subscriptions", and "Azure Virtual Desktop". In the "Resources" section, there are tabs for "Recent" (which is selected) and "Favorite", followed by a "Name" filter. To the right, a sidebar lists various services: All (Services (99+)), Marketplace (1), Documentation (27), Azure Active Directory (2), Resources (0), Resource Groups (0), Services (with a "See all" link), App Service Domains (which is highlighted with a red box), Service Health, and Continue searching in Azure Active Dire... Below the sidebar, a message says "Searching 94 of 137 subscriptions. Change Give feedback".

2. Select the domain you want to configure.
3. From the **Overview** page, select **Manage DNS records**.

This screenshot shows the "Manage DNS records" page for a domain named "contoso.com". The page has a header with "Manage DNS records", "FAQs", "Delete", "View Cost", and "JSON View". Under the "Essentials" section, it displays the following information:

Resource group (move)	Domain
myResourceGroup	contoso.com
Location	Expiration date
Global	2023-10-07T19:00:37
Subscription (move)	Auto renew
Antares-Demo	Enabled
Subscription ID	Privacy protection
00000000-0000-0000-0000-000000000000	Enabled
Status	
Active	

For information on how to edit DNS records, see [How to manage DNS Zones in the Azure portal](#).

Update contact information

After you purchase the App Service domain, you can update the domain contact information if you need to. It's important to keep this contact information up to date so

that you can receive notifications about your domain and receive verification emails if you decide to transfer out your domain. To update your contact information:

1. In the search bar, search for and select App Service Domains.

The screenshot shows the Microsoft Azure (Preview) interface. At the top, there is a search bar with the text "App Service Domains" and a red box highlighting it. Below the search bar, the "Azure services" section is visible, featuring icons for "Create a resource", "App Service Domains", "Subscriptions", and "Azure Virtual Desktop". The "Resources" section includes tabs for "Recent" (which is selected) and "Favorite". On the right side, a sidebar lists various services: All (Services (99+)), Marketplace (1), Documentation (27), Azure Active Directory (2), Resources (0), Resource Groups (0), Services (with a "See all" link), and App Service Domains (which is also highlighted with a red box). Below these, there are links for "Service Health" and "Continue searching in Azure Active Dire...". At the bottom of the sidebar, there are links for "Searching 94 of 137 subscriptions", "Change type", "Give feedback", and "Last viewed".

2. Select the domain you want to configure.

3. From the left navigation of the domain, select Advanced domain management (preview). To update your contact information, select Edit contact.

The screenshot shows the "contoso.com | Advanced domain management (preview)" page. The left sidebar has a red box around the "Advanced domain management (preview)" option under the "Domain Management" section. The main content area shows "Contact information" for the domain, with fields for Name, Email (contoso@contoso.com), Phone, Street Address (1 Contoso Way), City/State (Redmond, WA), and Country/Postal Code (98052). Below this, there is a "Domain privacy" section with "Privacy" set to "Enabled" and a "Disable" link. The right side of the screen shows the "Edit contact (preview)" pane, which contains "Personal information" fields for First name (Connie), Last name, and Email (contoso@contoso.com). It also contains "Phone and address" fields for Country code (United States (+1)), Phone, Address 1 (1 Contoso Way), Address 2 (Apartment, suite, unit, building, floor, etc. (optional)), Country (United States), State or territory (WA), City (Redmond), and Postal code (98052). At the bottom of the pane are "Submit" and "Cancel" buttons, and a circular icon with a plus sign and a magnifying glass.

4. In the pane that appears, update the necessary fields and then select Submit.

- Once you submit the request, it might take some time for your contact information to update.

 **Note**

If you have privacy protection disabled and update name or organization information, an email verification is sent to the email address on file for confirmation. Additionally, if you update your email address, a verification email is sent first to the previous email on file for confirmation. Once that's completed, another email is sent to the new email on file for confirmation. The contact information won't update until after you have confirmed via email.

Disable privacy protection

 **Important**

Once you disable privacy protection, you can no longer re-enable privacy protection again.

Privacy protection hides your domain registration contact information from the WHOIS database. If it's enabled during domain creation, privacy protection is already included in the yearly domain registration fee for no additional cost. However, there are some scenarios, such as transferring the domain out, where you need to disable privacy protection, you can do that by:

- In the search bar, search for and select App Service Domains.

The screenshot shows the Microsoft Azure (Preview) dashboard. At the top, there's a search bar with the placeholder "App Service Domains". Below the search bar, there are sections for "Azure services" and "Resources". Under "Azure services", there are icons for "Create a resource", "App Service Domains", "Subscriptions", and "Azure Virtual Desktop". Under "Resources", there are tabs for "Recent" and "Favorite", followed by a "Name" search input field. To the right, there's a sidebar titled "Services" with categories like "All", "Services (99+)", "Marketplace (1)", "Documentation (27)", "Azure Active Directory (2)", "Resources (0)", and "Resource Groups (0)". A red box highlights the "App Service Domains" category in the sidebar.

2. Select the domain you want to configure.
3. From the left navigation of the domain, select **Advanced domain management (preview)**. To disable privacy protection, select **Disable** in the **Domain Privacy** section.

The screenshot shows the "contoso.com | Advanced domain management (preview)" page. The left sidebar has a list of options: Overview, Access control (IAM), Tags, Diagnose and solve problems, Settings (Properties, Locks), Domain Management (Advanced domain management (preview), Domain renewal), Automation (Tasks (preview), Export template), and Help (FAQs, Support + Troubleshooting). The "Advanced domain management (preview)" option is highlighted with a red box. The main content area shows "Contact information" with fields for Name, Email (contoso@contoso.com), Phone, Street Address (1 Contoso Way), City/State (Redmond, WA), and Country/Postal Code (98052). Below this, the "Domain privacy" section shows "Privacy" set to "Enabled" with a "Disable" button, which is also highlighted with a red box. The top of the page includes a search bar, a Copilot icon, and a user profile for "contoso@contoso.com".

Cancel the purchase

After you purchase the App Service domain, you have five days to cancel your purchase and get a full refund. After five days, you can delete the App Service domain but can't

receive a refund.

1. In the search bar, search for and select App Service Domains.

The screenshot shows the Microsoft Azure (Preview) interface. At the top, there's a search bar with the text "App Service Domains" and a magnifying glass icon. To the right of the search bar is a user profile picture. Below the search bar, there's a sidebar titled "Azure services" with icons for "Create a resource", "Subscriptions", and "Azure Virtual Desktop". Under "Resources", there are tabs for "Recent" (which is selected) and "Favorite". A "Name" filter dropdown is also present. On the right side, a search results panel is open, showing a list of services: All (Services (99+)), Marketplace (1), Documentation (27), Azure Active Directory (2), Resources (0), Resource Groups (0), Services (See all), App Service Domains, and Service Health. The "App Service Domains" item is highlighted with a red box. At the bottom of the search results panel, there's a message: "Searching 94 of 137 subscriptions. Change Give feedback".

2. Select the domain you want to configure.

3. In the domain's left navigation, select Locks.

A delete lock has been created for your domain. As long as a delete lock exists, you can't delete the App Service domain.

4. Select Delete to remove the lock.

5. In the domain's left navigation, select Overview.

6. If the cancellation period on the purchased domain hasn't elapsed, select Cancel purchase. Otherwise, you see a Delete button instead. To delete the domain without a refund, select Delete.

The screenshot shows the "contoso.com" App Service Domain overview page. The left sidebar has a "Overview" tab selected and is highlighted with a red box. Other tabs include "Access control (IAM)", "Tags", "Properties", and "Locks". The main content area has a "Manage DNS records" button, a "FAQs" link, and a "Cancel purchase" button, which is also highlighted with a red box. Below these, there's an "Essentials" section with details: Resource group (myResourceGroup), Location (Global), Subscription (Visual Studio Ultimate with MSDN), and Domain (contoso.com). It also shows the expiration date (November 27, 2021, 10:15:02 AM GMT+1), Auto renew status (Enabled), and Privacy protection status (Enabled).

7. Confirm the operation by selecting **Yes**.

After the operation is complete, the domain is released from your subscription and available for anyone to purchase again.

Frequently asked questions

- Why do I see "This subscription does not have the billing support to purchase an App Service domain"?
- Why do I get a `SubscriptionExceededMaxDomainLimit` error when creating an App Service domain?
- How do I direct the default URL to a custom directory?

Why do I see "This subscription does not have the billing support to purchase an App Service domain"?

Free subscriptions, which don't require a confirmed credit card, don't have the permissions to buy App Service domains in Azure.

Why do I get a `SubscriptionExceededMaxDomainLimit` error when creating an App Service domain?

The number of App Service domains a subscription can have depends on the subscription type. Subscriptions that have a monthly credit allotment, like the Visual Studio Enterprise subscription, have a limit of one App Service domain. To increase your limit, convert to a pay-per-use subscription.

How do I direct the default URL to a custom directory?

This is not a DNS resolution scenario. By default, App Service directs web requests to the root directory of your app code. To direct them to a subdirectory, such as `public`, see [Redirect to a custom directory](#).

Next step

Learn how to bind a custom TLS/SSL certificate to help secure App Service.

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

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

Configure a custom domain name in Azure App Service with Traffic Manager integration

Article • 02/01/2023

ⓘ Note

For Cloud Services, see [Configuring a custom domain name for an Azure cloud service](#).

When you use [Azure Traffic Manager](#) to load balance traffic to [Azure App Service](#), the App Service app can be accessed using <traffic-manager-endpoint>.trafficmanager.net. You can assign a custom domain name, such as www.contoso.com, with your App Service app in order to provide a more recognizable domain name for your users.

This article shows you how to configure a custom domain name with an App Service app that's integrated with [Traffic Manager](#).

ⓘ Note

Only [CNAME](#) records are supported when you configure a domain name using the Traffic Manager endpoint. Because A records are not supported, a root domain mapping, such as contoso.com is also not supported.

Prepare the app

To map a custom DNS name to an app that's integrated with Azure Traffic Manager, the web app's [App Service plan](#) must be in **Standard** tier or higher. In this step, you make sure that the App Service app is in the supported pricing tier.

Check the pricing tier

In the [Azure portal](#), search for and select **App Services**.

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

The screenshot shows the Azure App Services blade. At the top, there's a navigation bar with 'Home >' and a search bar containing 'myazureaccount (Default Directory)'. Below the search bar are several buttons: '+ Add', 'Manage view', 'Refresh', 'Export to CSV', 'Open query', 'Assign tags', 'Start', 'Restart', 'Stop', and three dots. There are also filters for 'Subscription == all', 'Resource group == all', 'Location == all', and a 'Filter by name...' input field. Below the filters, it says 'Showing 1 to 1 of 1 records.' and there are two dropdown menus: 'No grouping' and 'List view'. The main table has columns: Name, Status, Location, Pricing Tier, and App Service Plan. A single row is listed: 'my-demo-app' (Status: Running, Location: West Europe, Pricing Tier: Free, App Service Plan: myAppServicePlan). The 'my-demo-app' cell is highlighted with a red border.

In the left navigation of the app page, select **Scale up (App Service plan)**.

The screenshot shows the 'cephalin320170403020701' App Service settings page. At the top, there's a search bar labeled 'Search (Ctrl+ /)'. Below it is a 'Deployment Center' section with a blue icon. The main area is titled 'Settings' and contains a list of options: Configuration, Container settings, Authentication / Authorization, Application Insights, Identity, Backups, Custom domains, TLS/SSL settings, Networking, and Scale up (App Service plan). The 'Scale up (App Service plan)' option is highlighted with a red border.

The app's current tier is highlighted by a blue border. Check to make sure that the app is in **Standard** tier or above (any tier in the **Production** or **Isolated** category). If yes, close the **Scale up** page and skip to [Create the CNAME mapping](#).



Dev / Test

For less demanding workloads



Production

For most production workloads



Isolated

Advanced networking and scale

Recommended pricing tiers

F1

Shared infrastructure
1 GB memory
60 minutes/day compute
Free

D1

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

B1

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

▼ See additional options

Included hardware

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

Azure Compute Units (ACU)



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

Memory



Memory available to run applications

Scale up the App Service plan

If you need to scale up your app, select any of the pricing tiers in the **Production** category. For additional options, click [See additional options](#).

Click **Apply**.

Create Traffic Manager endpoint

Following the steps at [Add or Delete Endpoints](#), add your App Service app as an endpoint in your Traffic Manager profile.

Once your App Service app is in a supported pricing tier, it shows up in the list of available App Service targets when you add the endpoint. If your app isn't listed, [verify the pricing tier of your app](#).

Create the CNAME mapping

ⓘ Note

To configure an App Service domain that you purchased, skip this section and go to [Enable custom domain](#).

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

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

2. Find the page for managing DNS records.

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

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

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

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

ADD

3. Select **Add** or the appropriate widget to create a record.

ⓘ Note

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

While the specifics of each domain provider vary, you map *from* a [non-root custom domain name](#) (such as [www.contoso.com](#)) to the Traffic Manager domain name ([contoso.trafficmanager.net](#)) that's integrated with your app.

Note

If a record is already in use and you need to preemptively bind your apps to it, you can create an additional CNAME record. For example, to preemptively bind [www.contoso.com](#) to your app, create a CNAME record from [awverify.www](#) to [contoso.trafficmanager.net](#). You can then add "www.contoso.com" to your app without the need to change the "www" CNAME record. For more information, see [Migrate an active DNS name to Azure App Service](#).

Once you have finished adding or modifying DNS records at your domain provider, save the changes.

What about root domains?

Since Traffic Manager only supports custom domain mapping with CNAME records, and because DNS standards don't support CNAME records for mapping root domains (for example, [contoso.com](#)), Traffic Manager doesn't support mapping to root domains. To work around this issue, use a URL redirect from at the app level. In ASP.NET Core, for example, you can use [URL Rewriting](#). Then, use Traffic Manager to load balance the subdomain ([www.contoso.com](#)). Another approach is you can [create an alias record for your domain name apex to reference an Azure Traffic Manager profile](#). An example is contoso.com. Instead of using a redirecting service, you can configure Azure DNS to reference a Traffic Manager profile directly from your zone.

For high availability scenarios, you can implement a load-balancing DNS setup without Traffic Manager by creating multiple *A records* that point from the root domain to each app copy's IP address. Then, [map the same root domain to all the app copies](#). Since the same domain name cannot be mapped to two different apps in the same region, this setup only works when your app copies are in different regions.

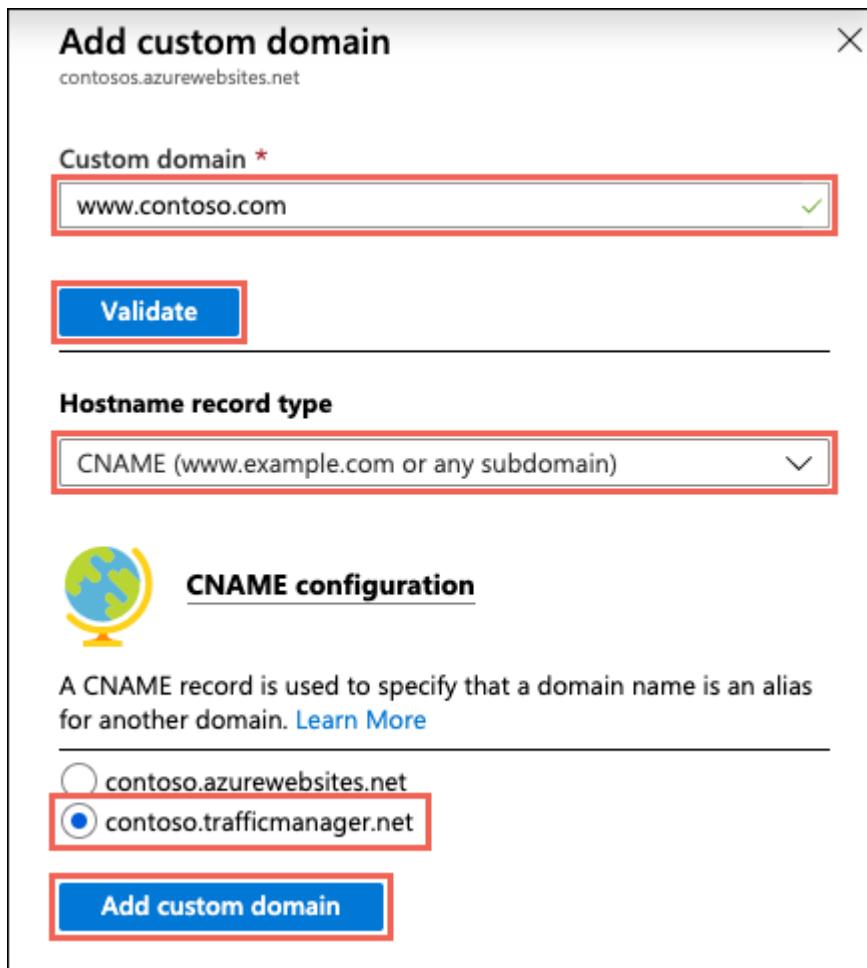
Enable custom domain

After the records for your domain name have propagated, use the browser to verify that your custom domain name resolves to your App Service app.

Note

It can take some time for your CNAME to propagate through the DNS system. You can use a service such as <https://www.digwebinterface.com/> to verify that the CNAME is available.

1. Once domain resolution succeeds, go back to your app page in the [Azure portal](#)
2. From the left navigation, select **Custom domains** > **Add hostname**.
3. Type the custom domain name that you mapped earlier and select **Validate**.
4. Make sure that **Hostname record type** is set to **CNAME** (**www.example.com or any subdomain**).
5. Since the App Service app is now integrated with a Traffic Manager endpoint, you should see the Traffic Manager domain name under **CNAME configuration**. Select it and click **Add custom domain**.



Next steps

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

Migrate an active DNS name to Azure App Service

Article • 02/01/2023

This article shows you how to migrate an active DNS name to [Azure App Service](#) without any downtime.

When you migrate a live site and its DNS domain name to App Service, that DNS name is already serving live traffic. You can avoid downtime in DNS resolution during the migration by binding the active DNS name to your App Service app preemptively.

If you're not worried about downtime in DNS resolution, see [Map an existing custom DNS name to Azure App Service](#).

Prerequisites

To complete the steps, [make sure that your App Service app isn't in FREE tier](#).

1. Get a domain verification ID

When you bind a custom domain preemptively, you accomplish both of the following before making any changes to your existing DNS records:

- Verify domain ownership by adding a domain verification ID with your domain provider.
- Enable the domain name in your App service app.

When you finally migrate your custom DNS name from the old site to the App Service app, there will be no downtime in DNS resolution.

1. In the [Azure portal](#), open the management page of the App Service app.
2. In the left pane of your app page, select **Custom domains**.
3. Copy the ID in the **Custom Domain Verification ID** box in the **Custom Domains** page for the next steps.

The screenshot shows the 'Settings' blade for an Azure App Service. The left sidebar lists several options: Configuration, Authentication, Application Insights, Identity, Backups, and Custom domains, which is highlighted with a red box. The main content area is titled 'Custom domains' and shows a table with one item: 'contoso.azurewebsites.net' under 'Status' with a green checkmark and the word 'Secured'. There are buttons for 'Add custom domain', 'Buy App Service domain', and 'Delete'.

2. Create the DNS records

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

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

2. Find the page for managing DNS records.

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

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

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

The screenshot shows a 'Records' page with a table of DNS records. The columns are Type, Name, Value, and TTL. There are three entries: two NS records pointing to '@' to 'ns31.domaincontrol.com' and 'ns32.domaincontrol.com' with a TTL of '1 Hour', and one SOA record pointing to '@' with the value 'Primary nameserver: ns31.domaincontrol.com.' and a TTL of '600 seconds'. An 'ADD' button is located at the bottom right.

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

3. Select **Add** or the appropriate widget to create a record.

ⓘ Note

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

Add a TXT record for domain verification. The hostname for the TXT record depends on the type of DNS record type you want to map. See the following table (@ typically represents the root domain):

DNS record example	TXT Host	TXT Value
@ (root)	<i>asuid</i>	Domain verification ID shown in the Custom domains management page.
www (sub)	<i>asuid.www</i>	Domain verification ID shown in the Custom domains management page.
* (wildcard)	<i>asuid</i>	Domain verification ID shown in the Custom domains management page.

ⓘ Note

Wildcard * records won't validate subdomains with an existing CNAME's record. You may need to explicitly create a TXT record for each subdomain.

3. Enable the domain for your app

1. Back in the **Custom domains** page, select **Add custom domain**.

The screenshot shows the Azure portal's 'Custom domains' settings page. On the left, a sidebar lists various settings like Configuration, Authentication, Application Insights, Identity, Backups, and Custom domains (which is selected and highlighted with a red box). The main content area shows a table of custom domains. At the top of this area, there are buttons for 'Add custom domain' (highlighted with a red box), 'Buy App Service domain', and 'Delete'. Below these buttons is a table with columns for 'Custom domains', 'Status', and 'Solution'. One row is shown: contoso.azurewebsites.net, Secured, -.

2. For **Domain provider**, select **All other domain services** to configure a third-party domain.
3. For **TLS/SSL certificate**, select **Add certificate later**. You can add an App Service managed certificate after you have completed domain migration.
4. For **TLS/SSL type**, select the binding type you want.

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 2. Remap records for IP based SSL. IP SSL is supported only in Standard tier or above.

5. Type the fully qualified domain name you want to migrate, that corresponds to the TXT record you created, such as `contoso.com`, `www.contoso.com`, or `*.contoso.com`.

Add custom domain

X

Domain provider * ⓘ

App Service Domain

All other domain services

TLS/SSL certificate * ⓘ

App Service Managed Certificate

Add certificate later

Enter your custom domain. We'll give you hostname records to register with your domain provider, and we can validate and add your custom domain here. [Learn more ↗](#)

Domain * ⓘ

www.contoso.com

Hostname record type * ⓘ

CNAME (www.example.com or any subdo... ▾)

Domain validation

To validate your domain ownership, copy the hostname records below and enter them with your domain provider. [Learn more ↗](#)

Type	Host	Value	Status
CNAME	www	contoso.azurewebsites.net	
TXT	asuid.www	XXXXXXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXXXXXX XXXXXXXXXXXX	

 If you use traffic manager, make sure to point it to the CNAME record.

Validate

Add

Cancel

6. Select **Validate**. Although the dialog shows two records you need for the custom domain to be functional for your app, the validation passes with just the domain verification ID (the TXT record).

Domain validation

To validate your domain ownership, copy the hostname records below and enter them with your domain provider. [Learn more](#)

Type	Host	Value	Status
CNAME	www	contoso.azurewebsites.net	
TXT	asuid.www	XXXXXXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXXXXXX XXXXXXXXXXXX	

If you use traffic manager, make sure to point it to the CNAME record.

Validate

Add

Cancel

7. If the **Domain validation** section shows green check marks, then you've configured the domain verification ID correctly. Select **Add**. If it shows any red X, fix any errors in your domain provider's website.

Domain validation

To validate your domain ownership, copy the hostname records below and enter them with your domain provider. [Learn more](#)

Type	Host	Value	Status
CNAME	www	contoso.azurewebsites.net	
TXT	asuid.www	XXXXXXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXXXXXX XXXXXXXXXXXX	

If you use traffic manager, make sure to point it to the CNAME record.

Validation passed. Select **Add** to finish up.

Validate

Add

Cancel

8. You should see the custom domain added to the list. You may also see a red X with **No binding**.

Since you selected **Add certificate later**, you see a red X with **No binding**. It will remain until you [add a private certificate for the domain](#) and [configure the binding](#).

 Refresh  Troubleshoot

 One or more domains are not secured. Check the solution column below for how to secure your domains.

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

IP address: 

Custom Domain Verification ID: 

 Filter by keywords

 Add filter

2 items

 Add custom domain |  Buy App Service domain |  Delete

Custom domains	Status	Solution
<input type="checkbox"/> www.contoso.com	 No binding	Add binding
contoso.azurewebsites.net	 Secured	-

< Previous

Page of 1

Next >

Note

Unless you configure a certificate binding for your custom domain, Any HTTPS request from a browser to the domain will receive an error or warning, depending on the browser.

4. Remap the active DNS name

The only thing left to do is remapping your active DNS record to point to App Service. Right now, it still points to your old site.

1. (A record only) You need the App Service app's external IP address. In the **Custom domains** page, copy the app's IP address.

The screenshot shows the Azure portal's custom domain management interface. At the top, there are 'Refresh' and 'Troubleshoot' buttons. A message box indicates that one or more domains are not secured. Below this, a section titled 'Configure and manage custom domains assigned to your app' includes fields for 'IP address' and 'Custom Domain Verification ID'. There are also 'Filter by keywords' and 'Add filter' buttons. The main table lists 'Custom domains' with columns for 'Status' and 'Solution'. The first domain, 'www.contoso.com', is marked as 'No binding' with a red 'X' icon. The second domain, 'contoso.azurewebsites.net', is marked as 'Secured' with a green checkmark icon. At the bottom, there are navigation buttons for '< Previous', 'Page 1 of 1', and 'Next >'.

2. Back in the DNS records page of your domain provider, select the DNS record to remap.
3. Remap the A or CNAME record like the examples in the following table:

FQDN example	Record type	Host	Value
contoso.com (root)	A	@	IP address from Copy the app's IP address
www.contoso.com (sub)	CNAME	www	<app-name>.azurewebsites.net
*.contoso.com (wildcard)	CNAME	*	<app-name>.azurewebsites.net

4. Save your settings.

DNS queries should start resolving to your App Service app immediately after DNS propagation happens.

Frequently asked questions

- Can I add an App Service managed certificate when migrating a live domain?
- How do I migrate a domain from another app?

Can I add an App Service managed certificate when migrating a live domain?

You can add an App Service managed certificate to a migrated live domain, but only after you [remap the active DNS name](#). To add the App Service managed certificate, see [Create a free managed certificate](#).

How do I migrate a domain from another app?

You can migrate an active custom domain in Azure, between subscriptions or within the same subscription. However, such a migration without downtime requires the source app and the target app are assigned the same custom domain at a certain time. Therefore, you need to make sure that the two apps aren't deployed to the same deployment unit (internally known as a webspace). A domain name can be assigned to only one app in each deployment unit.

You can find the deployment unit for your app by looking at the domain name of the FTP/S URL <deployment-unit>.ftp.azurewebsites.windows.net. Check and make sure the deployment unit is different between the source app and the target app. The deployment unit of an app is determined by the [App Service plan](#) it's in. It's selected randomly by Azure when you create the plan and can't be changed. When you create two apps [in the same resource group and the same region](#), Azure puts them in the same deployment unit. However, there's no way to make sure that the opposite is true. In other words, the only way to create a plan in a different deployment unit is to keep creating a plan in a new resource group or region until you get a different deployment unit.

Next steps

Learn how to bind a custom TLS/SSL certificate to App Service.

[Purchase an App Service domain](#) .

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

Azure App Service TLS overview

Article • 01/31/2025

ⓘ Note

Customers may be aware of [the retirement notification of TLS 1.0 and 1.1 for interactions with Azure services](#). This retirement does not affect applications running on App Service or Azure Functions. Applications on either App Service or Azure Functions configured to accept TLS 1.0 or TLS 1.1 for incoming requests will continue to run unaffected.

What does TLS do in App Service?

Transport Layer Security (TLS) is a widely adopted security protocol designed to secure connections and communications between servers and clients. App Service allows customers to use TLS/SSL certificates to secure incoming requests to their web apps. App Service currently supports different set of TLS features for customers to secure their web apps.

💡 Tip

You can also ask Azure Copilot these questions:

- *What versions of TLS are supported in App Service?*
- *What are the benefits of using TLS 1.3 over previous versions?*
- *How can I change the cipher suite order for my App Service Environment?*

To find Azure Copilot, on the [Azure portal](#) toolbar, select **Copilot**.

Supported TLS Version on App Service?

For incoming requests to your web app, App Service supports TLS versions 1.0, 1.1, 1.2, and 1.3.

Set Minimum TLS Version

Follow these steps to change the Minimum TLS version of your App Service resource:

1. Browse to your app in the [Azure portal](#)
2. In the left menu, select **configuration** and then select the **General settings** tab.
3. On **Minimum Inbound TLS Version**, using the dropdown, select your desired version.
4. Select **Save** to save the changes.

Minimum TLS Version with Azure Policy

You can use Azure Policy to help audit your resources when it comes to minimum TLS version. You can refer to [App Service apps should use the latest TLS version policy definition](#) and change the values to your desired minimum TLS version. For similar policy definitions for other App Service resources, refer to [List of built-in policy definitions - Azure Policy for App Service](#).

Minimum TLS Version and SCM Minimum TLS Version

App Service also allows you to set minimum TLS version for incoming requests to your web app and to SCM site. By default, the minimum TLS version for incoming requests to your web app and to SCM is set to 1.2 on both portal and API.

TLS 1.3

A [Minimum TLS Cipher Suite](#) setting is available with TLS 1.3. This includes two cipher suites at the top of the cipher suite order:

- TLS_AES_256_GCM_SHA384
- TLS_AES_128_GCM_SHA256

TLS 1.0 and 1.1

TLS 1.0 and 1.1 are considered legacy protocols and are no longer considered secure. It's generally recommended for customers to use TLS 1.2 or above as the minimum TLS version. When creating a web app, the default minimum TLS version is TLS 1.2.

To ensure backward compatibility for TLS 1.0 and TLS 1.1, App Service will continue to support TLS 1.0 and 1.1 for incoming requests to your web app. However, since the default minimum TLS version is set to TLS 1.2, you need to update the minimum TLS version configurations on your web app to either TLS 1.0 or 1.1 so the requests won't be rejected.

 **Important**

Incoming requests to web apps and incoming requests to Azure are treated differently. App Service will continue to support TLS 1.0 and 1.1 for incoming requests to the web apps. For incoming requests directly to the Azure control plane, for example through ARM or API calls, it is not recommended to use TLS 1.0 or 1.1.

Minimum TLS cipher suite

 **Note**

Minimum TLS Cipher Suite is supported on Basic SKUs and higher on multi-tenant App Service.

The minimum TLS cipher suite includes a fixed list of cipher suites with an optimal priority order that you cannot change. Reordering or reprioritizing the cipher suites is not recommended as it could expose your web apps to weaker encryption. You also cannot add new or different cipher suites to this list. When you select a minimum cipher suite, the system automatically disables all less secure cipher suites for your web app, without allowing you to selectively disable only some weaker cipher suites.

What are cipher suites and how do they work on App Service?

A cipher suite is a set of instructions that contains algorithms and protocols to help secure network connections between clients and servers. By default, the front-end's OS would pick the most secure cipher suite that is supported by both App Service and the client. However, if the client only supports weak cipher suites, then the front-end's OS would end up picking a weak cipher suite that is supported by them both. If your organization has restrictions on what cipher suites should not be allowed, you may update your web app's minimum TLS cipher suite property to ensure that the weak cipher suites would be disabled for your web app.

App Service Environment (ASE) V3 with cluster setting **FrontEndSSLCipherSuiteOrder**

For App Service Environments with `FrontEndSSLCipherSuiteOrder` cluster setting, you need to update your settings to include two TLS 1.3 cipher suites (TLS_AES_256_GCM_SHA384 and TLS_AES_128_GCM_SHA256). Once updated, restart

your front-end for the change to take effect. You must still include the two required cipher suites as mentioned in the docs.

End-to-end TLS Encryption

End-to-end (E2E) TLS encryption is available in Premium App Service plans (and legacy Standard App Service plans). Front-end intra-cluster traffic between App Service front-ends and the workers running application workloads can now be encrypted.

Next steps

- [Secure a custom DNS name with a TLS/SSL binding](#)
-

Feedback

Was this page helpful?

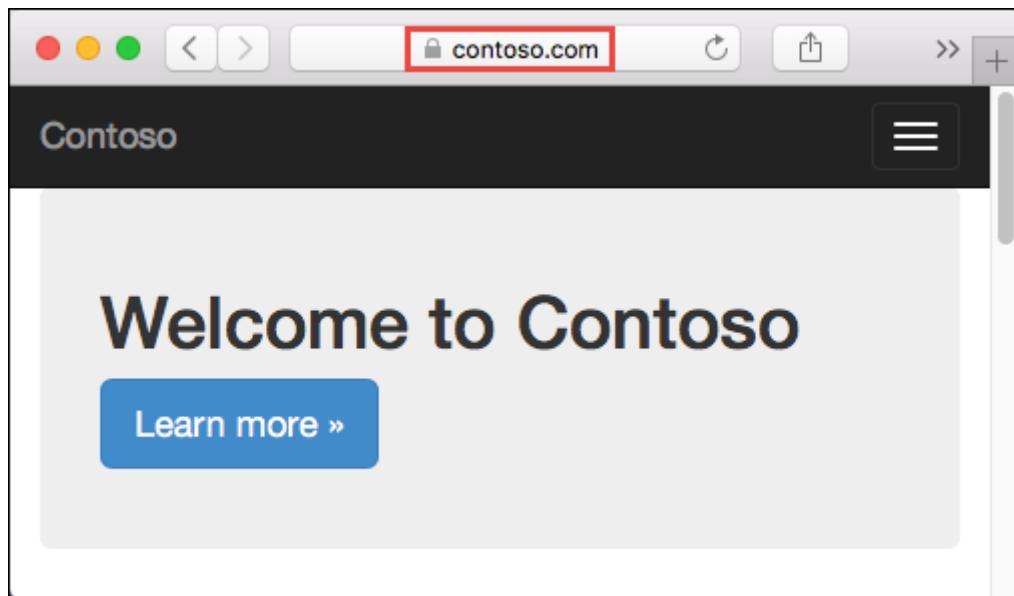


[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

Provide security for a custom DNS name with a TLS/SSL binding in App Service

Article • 09/16/2024

This article shows you how to provide security for 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`).



Prerequisites

- [Scale up your App Service app](#) to one of the supported pricing tiers: Basic, Standard, Premium.
- [Map a domain name to your app](#) or [buy and configure it in Azure](#).

Add the binding

In the [Azure portal](#) :

1. From the left menu, select **App Services** > `<app-name>`.
2. From the left navigation of your app, select **Custom domains**.
3. Next to the custom domain, select **Add binding**.

Custom domains	Status	Solution	Binding type
<input checked="" type="checkbox"/> www.contoso.com	✖ No binding	Add binding	-
my-demo-app.azurewebsites.net	✓ Secured	-	-

4. If your app already has a certificate for the selected custom domain, you can select it in **Certificate**. If not, you must add a certificate using one of the selections in **Source**.

- **Create App Service Managed Certificate** - Let App Service create a managed certificate for your selected domain. This option is the easiest. For more information, see [Create a free managed certificate](#).
- **Import App Service Certificate** - In **App Service Certificate**, select an [App Service certificate](#) you've purchased for your selected domain.
- **Upload certificate (.pfx)** - Follow the workflow at [Upload a private certificate](#) to upload a PFX certificate from your local machine and specify the certificate password.
- **Import from Key Vault** - Select **Select key vault certificate** and select the certificate in the dialog.

5. In **TLS/SSL type**, select either **SNI SSL** or **IP based SSL**.

- **SNI SSL**: Multiple SNI SSL bindings can be added. This option allows multiple TLS/SSL certificates to help secure multiple domains on the same IP address. Most modern browsers (including Microsoft Edge, Chrome, Firefox, and Opera) support SNI. (For more information, see [Server Name Indication](#).)
- **IP based SSL**: Only one IP SSL binding can be added. This option allows only one TLS/SSL certificate to help secure a dedicated public IP address. After you configure the binding, follow the steps in [Remap records for IP-based SSL](#). IP-based SSL is supported only in Standard tier or higher.

6. When adding a new certificate, validate the new certificate by selecting **Validate**.

7. Select Add.

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

Custom domains		Status	Solution	Binding type	
<input type="checkbox"/>	www.contoso.com	✓ Secured	-	SNI SSL	...
	my-demo-app.azurewebsites.net	✓ Secured	-	-	

ⓘ Note

A **Secured** state in **Custom domains** means that a certificate is providing security, 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-based SSL

This step is needed only for IP-based SSL. For an SNI SSL binding, skip to [Test HTTPS](#).

There are potentially two changes you need to make:

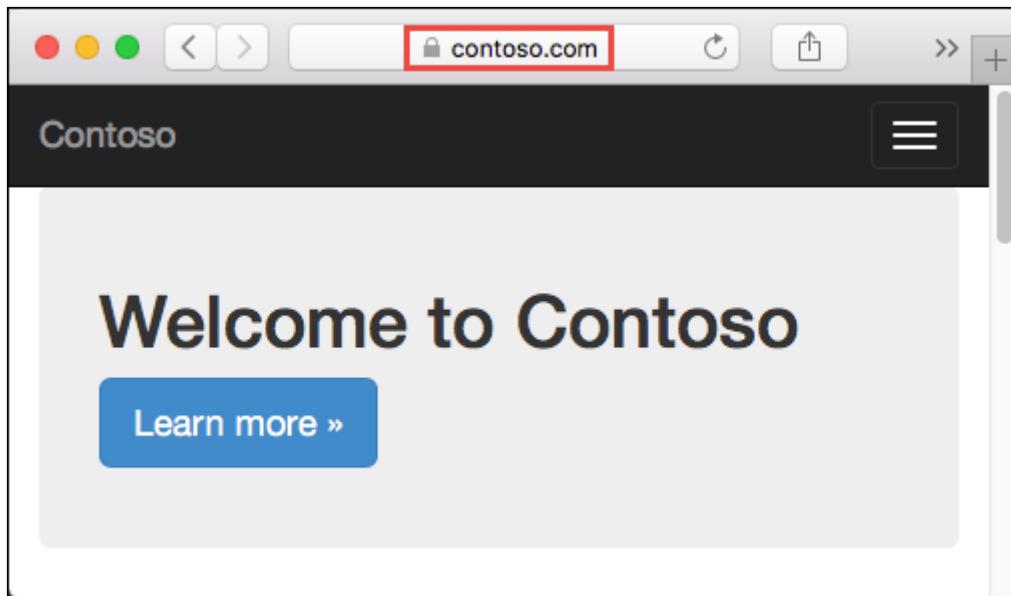
- By default, your app uses a shared public IP address. When you bind a certificate with IP SSL, App Service creates a new, dedicated IP address for your app. If you mapped an A record to your app, update your domain registry with this new, dedicated IP address.

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

Browse to `https://<your.custom.domain>` in various browsers to verify that your app appears.



Your application code can inspect the protocol via the `x-appservice-proto` header. The header has 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 might have left out intermediate certificates when you exported your certificate to the PFX file.

Frequently asked questions

- [How do I make sure that the app's IP address doesn't change when I make changes to the certificate binding?](#)
- [Can I disable the forced redirect from HTTP to HTTPS?](#)
- [How can I change the minimum TLS versions for the app?](#)
- [How do I handle TLS termination in App Service?](#)

How do I make sure that the app's IP address doesn't change when I make changes to the certificate binding?

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.

Can I disable the forced redirect from HTTP to HTTPS?

By default, App Service forces a redirect from HTTP requests to HTTPS. To disable this behavior, see [Configure general settings](#).

How can I change the minimum TLS versions for the app?

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, see [Configure general settings](#).

How do I handle TLS termination in App Service?

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, inspect the `X-Forwarded-Proto` header.

Language-specific configuration guides, such as the [Linux Node.js configuration](#) guide, show how to detect an HTTPS session in your application code.

Automate with scripts

Azure CLI

[Bind a custom TLS/SSL certificate to a web app](#)

PowerShell

```
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.
$webapp = New-AzWebApp -Name $webappname -Location $location -AppServicePlan
$webappname ` 
-ResourceGroupName $webappname

Write-Host "Sign in to your domain provider's website and configure the
following records:"
Write-Host "A CNAME record that maps $fqdn to $webappname.azurewebsites.net"
Write-Host "A TXT record that maps asuid.$fqdn to the domain verification ID
 $($webapp.CustomDomainVerificationId)"
Read-Host "Press [Enter] key when ready ..."

# Before continuing, go to your DNS configuration UI for your custom domain
and follow the
# instructions at https://aka.ms/appservicecustomdns to configure a CNAME
record for the
# hostname "www" and point it to your web app's default domain name.

# Upgrade App Service plan to Basic tier (minimum required by custom SSL
certificates)
Set-AzAppServicePlan -Name $webappname -ResourceGroupName $webappname ` 
-Tier Basic

# Add a custom domain name to the web app.
Set-AzWebApp -Name $webappname -ResourceGroupName $webappname ` 
-HostNames @($fqdn,"$webappname.azurewebsites.net")

# Upload and bind the SSL certificate to the web app.
New-AzWebAppSSLBinding -WebAppName $webappname -ResourceGroupName
$webappname -Name $fqdn ` 
-CertificateFilePath $pfxPath -CertificatePassword $pfxPassword -SslState
SniEnabled

```

Related content

- Use a TLS/SSL certificate in your code in Azure App Service
- Frequently asked questions about creating or deleting resources in Azure App Service

Feedback

Was this page helpful?

 Yes

 No

Add and manage TLS/SSL certificates in Azure App Service

Article • 09/19/2024

ⓘ Note

Starting June 1, 2024, all newly created App Service apps will have the option to generate a unique default hostname using the naming convention <app-name>-<random-hash>. <region>.azurewebsites.net. Existing app names will remain unchanged.

Example: myapp-ds27dh7271ah175.westus-01.azurewebsites.net

For further details, refer to [Unique Default Hostname for App Service Resource](#).

You can add digital security certificates to [use in your application code](#) or to [help secure custom DNS names in Azure App Service](#), which provides a highly scalable, self-patching web hosting service. Currently called Transport Layer Security (TLS) certificates, also previously known as Secure Socket Layer (SSL) certificates, these private or public certificates help you secure internet connections by encrypting data sent between your browser, websites that you visit, and the website server.

The following table lists the options for you to add certificates in App Service:

[+] Expand table

Option	Description
Create a free App Service managed certificate	A private certificate that's free of charge and easy to use if you just need to improve security for your custom domain in App Service.
Import an App Service certificate	A private certificate that's managed by Azure. It combines the simplicity of automated certificate management and the flexibility of renewal and export options.
Import a certificate from Key Vault	Useful if you use Azure Key Vault to manage your PKCS12 certificates . See Private certificate requirements .
Upload a private certificate	If you already have a private certificate from a third-party provider, you can upload it. See Private certificate requirements .

Option	Description
Upload a public certificate	Public certificates aren't used to secure custom domains, but you can load them into your code if you need them to access remote resources.

Prerequisites

- [Create an App Service app](#). The app's **App Service plan** must be in the **Basic**, **Standard**, **Premium**, or **Isolated** tier. See [Scale up an app](#) to update the tier.
- For a private certificate, make sure that it satisfies all [requirements from App Service](#).
- **Free certificate only:**
 - Map the domain where you want the certificate to App Service. For information, see [Tutorial: Map an existing custom DNS name to Azure App Service](#).
 - For a root domain (like contoso.com), make sure your app doesn't have any [IP restrictions](#) configured. Both certificate creation and its periodic renewal for a root domain depend on your app being reachable from the internet.

Private certificate requirements

The [free App Service managed certificate](#) and the [App Service certificate](#) already satisfy the requirements of App Service. If you choose to upload or import a private certificate to App Service, your certificate must meet the following requirements:

- Exported as a [password-protected PFX file](#), encrypted using triple DES
- Contains private key at least 2048 bits long
- Contains all intermediate certificates and the root certificate in the certificate chain

If you want to help secure a custom domain in a TLS binding, the certificate must meet these additional requirements:

- Contains an [Extended Key Usage](#) for server authentication (OID = 1.3.6.1.5.5.7.3.1)
- Signed by a trusted certificate authority

ⓘ Note

[Elliptic Curve Cryptography \(ECC\) certificates](#) work with App Service but aren't covered by this article. For the exact steps to create ECC certificates, work with your

certificate authority.

ⓘ Note

After you add a private certificate to an app, the certificate is stored in a deployment unit that's bound to the App Service plan's resource group, region, and operating system combination, internally called a *webspace*. That way, the certificate is accessible to other apps in the same resource group, region, and OS combination. Private certificates uploaded or imported to App Service are shared with App Services in the same deployment unit.

You can add up to 1000 private certificates per webspace.

Create a free managed certificate

The free App Service managed certificate is a turn-key solution for helping to secure your custom DNS name in App Service. Without any action from you, this TLS/SSL server certificate is fully managed by App Service and is automatically renewed continuously in six-month increments, 45 days before expiration, as long as the prerequisites that you set up stay the same. All the associated bindings are updated with the renewed certificate. You create and bind the certificate to a custom domain, and let App Service do the rest.

ⓘ Important

Before you create a free managed certificate, make sure you have [met the prerequisites](#) for your app.

Free certificates are issued by DigiCert. For some domains, you must explicitly allow DigiCert as a certificate issuer by creating a [CAA domain record](#) ↗ with the value: `issue digicert.com`.

Azure fully manages the certificates on your behalf, so any aspect of the managed certificate, including the root issuer, can change at anytime. These changes are outside your control. Make sure to avoid hard dependencies and "pinning" practice certificates to the managed certificate or any part of the certificate hierarchy. If you need the certificate pinning behavior, add a certificate to your custom domain using any other available method in this article.

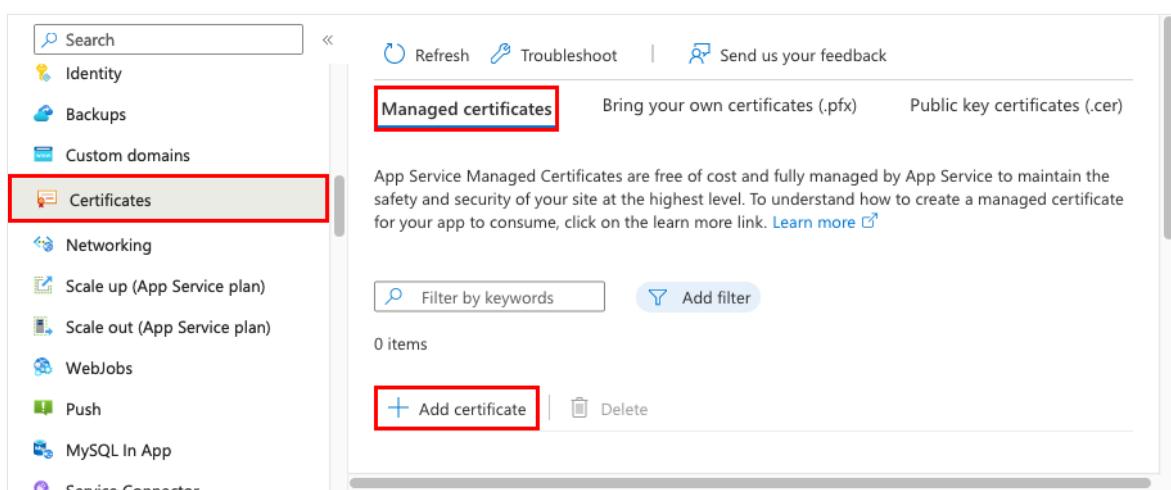
The free certificate comes with the following limitations:

- Doesn't support wildcard certificates.
- Doesn't support usage as a client certificate by using certificate thumbprint, which is planned for deprecation and removal.
- Doesn't support private DNS.
- Isn't exportable.
- Isn't supported in an App Service Environment.
- Only supports alphanumeric characters, dashes (-), and periods (.)
- Only custom domains of length up to 64 characters are supported.

Apex domain

- Must have an A record pointing to your web app's IP address.
- Must be on apps that are publicly accessible.
- Isn't supported with root domains that are integrated with Traffic Manager.
- Must meet all of the above for successful certificate issuances and renewals.

1. In the [Azure portal](#), from the left menu, select **App Services > <app-name>**.
2. On your app's navigation menu, select **Certificates**. In the **Managed certificates** pane, select **Add certificate**.



3. Select the custom domain for the free certificate, and then select **Validate**. When validation completes, select **Add**. You can create only one managed certificate for each supported custom domain.

When the operation completes, the certificate appears in the **Managed certificates** list.

Managed certificates[Bring your own certificates \(.pfx\)](#)[Public key certificates \(.cer\)](#)

App Service Managed Certificates are free of cost and fully managed by App Service to maintain the safety and security of your site at the highest level. To understand how to create a managed certificate for your app to consume, click on the learn more link. [Learn more](#)

[Filter by keywords](#)[Add filter](#)

1 items

[Add certificate](#)[Delete](#)

Certificate Status ↑	Domain	Certificate Name
<input type="checkbox"/> ✓ No action needed	www.contoso.com	Contoso www

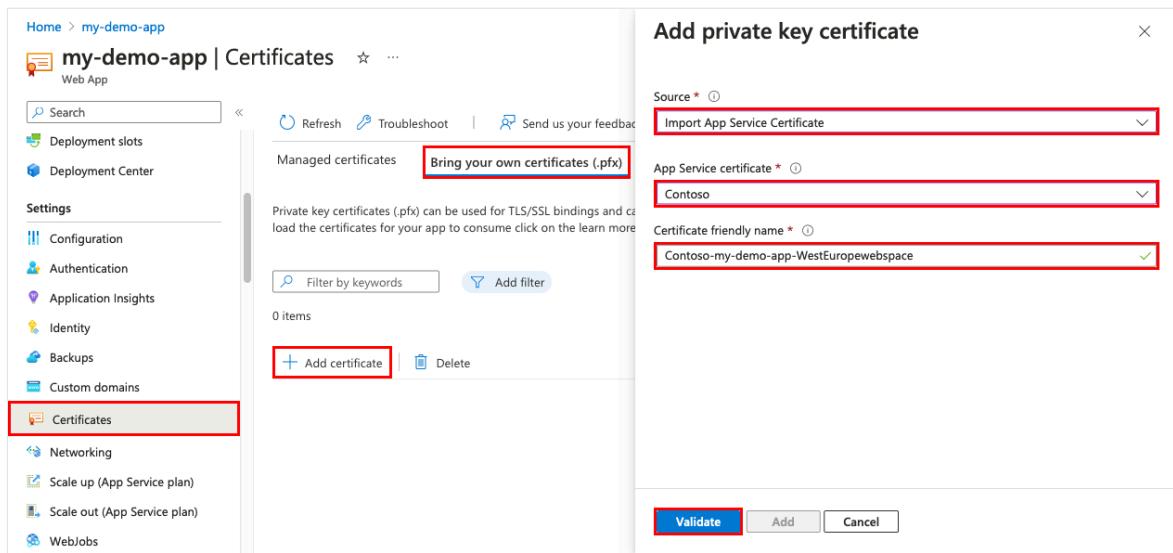
< Previous Items per page: 50 Page 1 of 1 Next >

4. To provide security for a custom domain with this certificate, you still have to create a certificate binding. Follow the steps in [Secure a custom DNS name with a TLS/SSL binding in Azure App Service](#).

Import an App Service certificate

To import an App Service certificate, first [buy and configure an App Service certificate](#), and then follow the steps here.

1. In the [Azure portal](#), from the left menu, select **App Services** > **<app-name>**.
2. From your app's navigation menu, select **Certificates** > **Bring your own certificates (.pfx)** > **Add certificate**.
3. In **Source**, select **Import App Service Certificate**.
4. In **App Service certificate**, select the certificate you just created.
5. In **Certificate friendly name**, give the certificate a name in your app.
6. Select **Validate**. When validation succeeds, select **Add**.



When the operation completes, the certificate appears in the **Bring your own certificates (.pfx)** list.

Certificate Status ↑	Domain	Certificate Name	Expiration D... ↑	Tl
<input type="checkbox"/> ✓ No action needed	contoso.com,www.contoso.com	Contoso-my-demo-app-WestEurope...	29/06/2024	OF

7. To help secure a custom domain with this certificate, you still have to create a certificate binding. Follow the steps in [Secure a custom DNS name with a TLS/SSL binding in Azure App Service](#).

Import a certificate from Key Vault

If you use Azure Key Vault to manage your certificates, you can import a PKCS12 certificate into App Service from Key Vault if you met the [requirements](#).

Authorize App Service to read from the vault

By default, the App Service resource provider doesn't have access to your key vault. To use a key vault for a certificate deployment, you must authorize read access for the resource provider (App Service) to the key vault. You can grant access either with access policy or RBAC.

 **Note**

Currently, the Azure portal does not allow you to configure an App Service certificate in Key Vault to use the RBAC model. You can, however, use Azure CLI, Azure PowerShell, or an ARM template deployment to perform this configuration.

RBAC permissions

 Expand table

Resource provider	Service principal app ID / assignee	Key vault RBAC role
Microsoft Azure App Service or Microsoft.Azure.WebSites	- abfa0a7c-a6b6-4736-8310-5855508787cd for public Azure cloud environment - 6a02c803-daf3-4136-b4c3-5a6f318b4714 for Azure Government cloud environment	Certificate User

The service principal app ID or assignee value is the ID for the App Service resource provider. To learn how to authorize key vault permissions for the App Service resource provider using an access policy, see the [provide access to Key Vault keys, certificates, and secrets with an Azure role-based access control documentation](#).

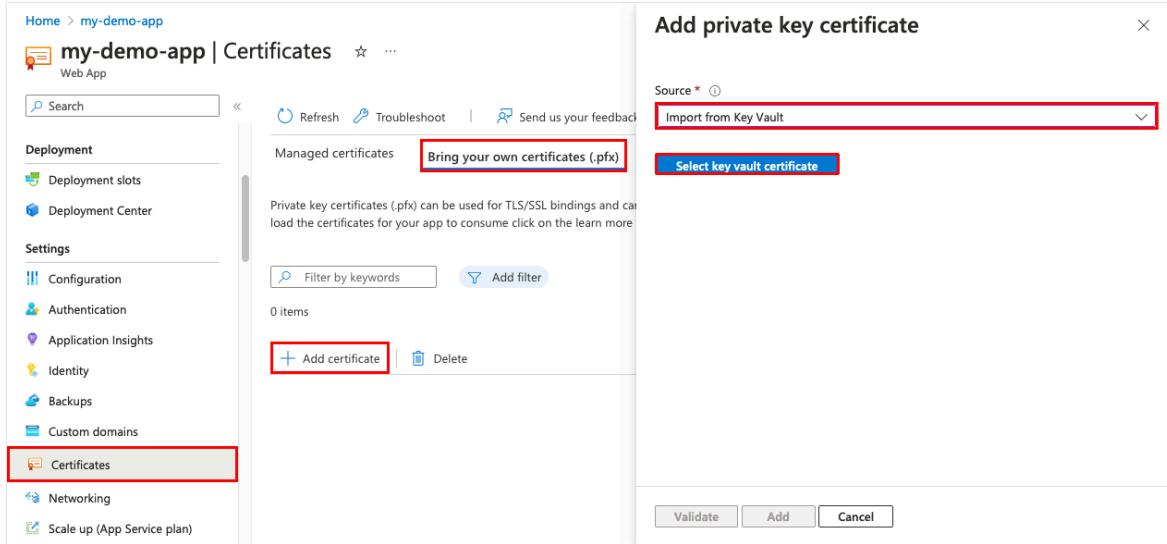
 **Note**

Do not delete these RBAC permissions from key vault. If you do, App Service will not be able to sync your web app with the latest key vault certificate version.

Import a certificate from your vault to your app

1. In the [Azure portal](#), from the left menu, select App Services > <app-name>.

2. From your app's navigation menu, select **Certificates** > **Bring your own certificates (.pfx)** > **Add certificate**.
3. In Source, select **Import from Key Vault**.
4. Select **Select key vault certificate**.



5. To help you select the certificate, use the following table:

[Expand table](#)

Setting	Description
Subscription	The subscription associated with the key vault.
Key vault	The key vault that has the certificate you want to import.
Certificate	From this list, select a PKCS12 certificate that's in the vault. All PKCS12 certificates in the vault are listed with their thumbprints, but not all are supported in App Service.

6. When finished with your selection, select **Select**, **Validate**, and then **Add**.

When the operation completes, the certificate appears in the **Bring your own certificates** list. If the import fails with an error, the certificate doesn't meet the [requirements for App Service](#).

Private key certificates (.pfx) can be used for TLS/SSL bindings and can be loaded to the certificate store for your app to consume. To understand how to load the certificates for your app to consume click on the learn more link. [Learn more](#)

Filter by keywords Add filter

1 items

Add certificate | Delete

Certificate Status ↑	Domain	Certificate Name	Expiration D... ↑	T...
<input type="checkbox"/> ✓ No action needed	contoso.com,www.contoso.com	Contoso-my-demo-app-WestEurope...	29/06/2024	OF

< Previous Items per page: 50 Page 1 of 1 Next >

ⓘ Note

If you update your certificate in Key Vault with a new certificate, App Service automatically syncs your certificate within 24 hours.

7. To help secure custom domain with this certificate, you still have to create a certificate binding. Follow the steps in [Secure a custom DNS name with a TLS/SSL binding in Azure App Service](#).

Upload a private certificate

After you get a certificate from your certificate provider, make the certificate ready for App Service by following the steps in this section.

Merge intermediate certificates

If your certificate authority gives you multiple certificates in the certificate chain, you must merge the certificates following the same order.

1. In a text editor, open each received certificate.
2. To store the merged certificate, create a file named *mergedcertificate.crt*.
3. Copy the content for each certificate into this file. Make sure to follow the certificate sequence specified by the certificate chain, starting with your certificate and ending with the root certificate, for example:

```
-----BEGIN CERTIFICATE-----
<your entire Base64 encoded SSL certificate>
-----END CERTIFICATE-----

-----BEGIN CERTIFICATE-----
<The entire Base64 encoded intermediate certificate 1>
-----END CERTIFICATE-----

-----BEGIN CERTIFICATE-----
<The entire Base64 encoded intermediate certificate 2>
-----END CERTIFICATE-----

-----BEGIN CERTIFICATE-----
<The entire Base64 encoded root certificate>
-----END CERTIFICATE-----
```

Export the merged private certificate to PFX

Now, export your merged TLS/SSL certificate with the private key that was used to generate your certificate request. If you generated your certificate request using OpenSSL, then you created a private key file.

ⓘ Note

OpenSSL v3 changed the default cipher from 3DES to AES256, but this can be overridden on the command line: -keypbe PBE-SHA1-3DES -certpbe PBE-SHA1-3DES -macalg SHA1. OpenSSL v1 uses 3DES as the default, so the PFX files generated are supported without any special modifications.

1. To export your certificate to a PFX file, run the following command, but replace the placeholders <*private-key-file*> and <*merged-certificate-file*> with the paths to your private key and your merged certificate file.

Bash

```
openssl pkcs12 -export -out myserver.pfx -inkey <private-key-file> -in
<merged-certificate-file>
```

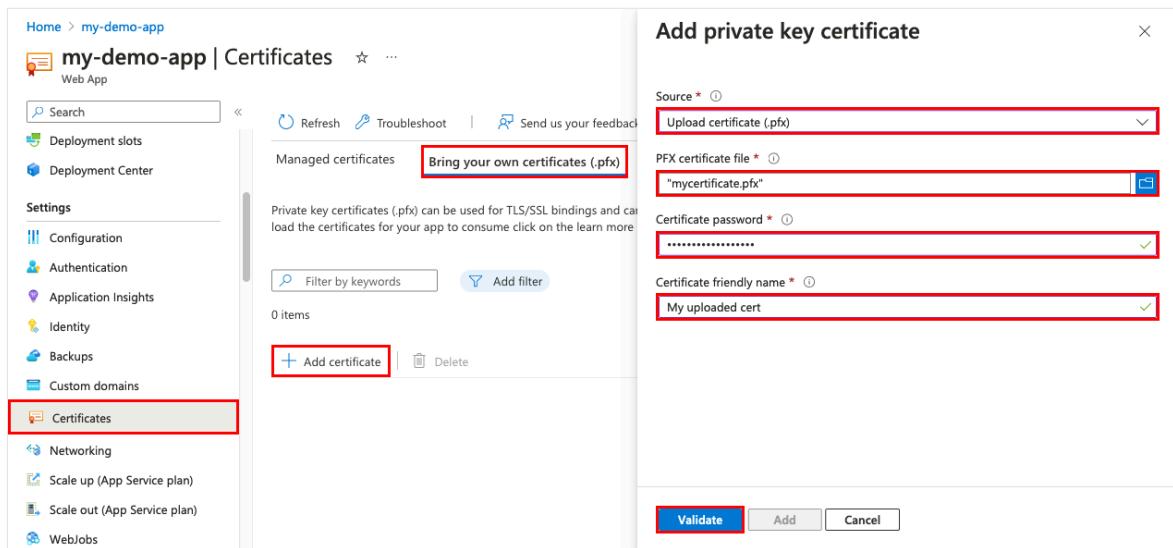
2. When you're prompted, specify a password for the export operation. When you upload your TLS/SSL certificate to App Service later, you must provide this password.

3. If you used IIS or *Certreq.exe* to generate your certificate request, install the certificate to your local computer, and then [export the certificate to a PFX file](#).

Upload the certificate to App Service

You're now ready upload the certificate to App Service.

1. In the [Azure portal](#), from the left menu, select **App Services > <app-name>**.
2. From your app's navigation menu, select **Certificates > Bring your own certificates (.pfx) > Upload Certificate**.



3. To help you upload the .pfx certificate, use the following table:

[+] [Expand table](#)

Setting	Description
PFX certificate file	Select your .pfx file.
Certificate password	Enter the password that you created when you exported the PFX file.
Certificate friendly name	The certificate name that will be shown in your web app.

4. When finished with your selection, select **Select, Validate, and then Add**.

When the operation completes, the certificate appears in the **Bring your own certificates** list.

Private key certificates (.pfx) can be used for TLS/SSL bindings and can be loaded to the certificate store for your app to consume. To understand how to load the certificates for your app to consume click on the learn more link. [Learn more](#)

Filter by keywords Add filter

1 items

Add certificate Delete

Certificate Status ↑	Domain	Certificate Name	Expiration D... ↑	T...
<input type="checkbox"/> ✓ No action needed	contoso.com,www.contoso.com	Contoso-my-demo-app-WestEurope...	29/06/2024	OF

< Previous Items per page: 50 Page 1 of 1 Next >

5. To provide security for a custom domain with this certificate, you still have to create a certificate binding. Follow the steps in [Secure a custom DNS name with a TLS/SSL binding in Azure App Service](#).

Upload a public certificate

Public certificates are supported in the .cer format.

ⓘ Note

After you upload a public certificate to an app, it's only accessible by the app it's uploaded to. Public certificates must be uploaded to each individual web app that needs access. For App Service Environment specific scenarios, refer to [the documentation for certificates and the App Service Environment](#).

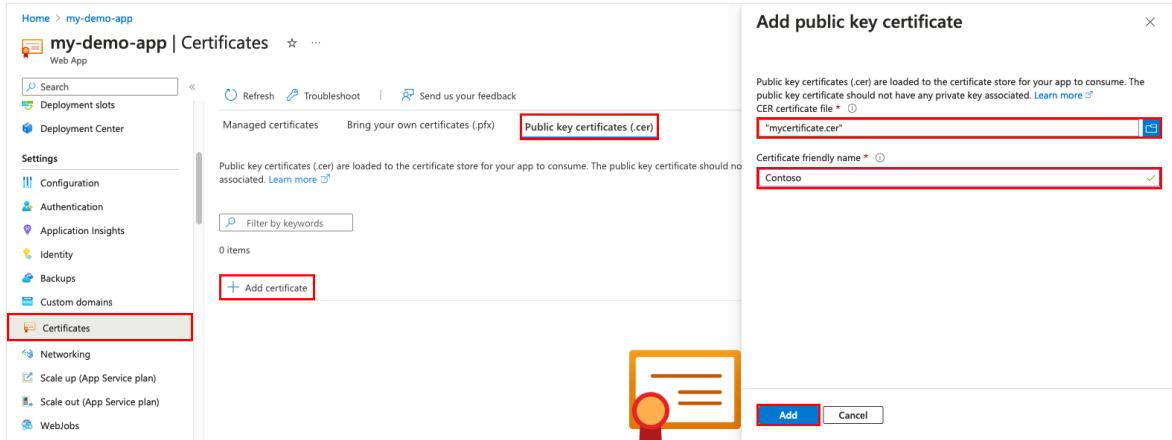
You can upload up to 1000 public certificates per App Service Plan.

1. In the [Azure portal](#), from the left menu, select **App Services** > <app-name>.
2. From your app's navigation menu, select **Certificates** > **Public key certificates (.cer)** > **Add certificate**.
3. To help you upload the .cer certificate, use the following table:

[Expand table](#)

Setting	Description
CER certificate file	Select your .cer file.
Certificate friendly name	The certificate name that will be shown in your web app.

4. When you're done, select **Add**.



5. After the certificate is uploaded, copy the certificate thumbprint, and then review [Make the certificate accessible](#).

Renew an expiring certificate

Before a certificate expires, make sure to add the renewed certificate to App Service, and update any certificate bindings where the process depends on the certificate type. For example, a [certificate imported from Key Vault](#), including an [App Service certificate](#), automatically syncs to App Service every 24 hours and updates the TLS/SSL binding when you renew the certificate. For an [uploaded certificate](#), there's no automatic binding update. Based on your scenario, review the corresponding section:

- [Renew an uploaded certificate](#)
- [Renew an App Service certificate](#)
- [Renew a certificate imported from Key Vault](#)

Renew an uploaded certificate

When you replace an expiring certificate, the way you update the certificate binding with the new certificate might adversely affect the user experience. For example, your inbound IP address might change when you delete a binding, even if that binding is IP-based. This result is especially impactful when you renew a certificate that's already in an IP-based binding. To avoid a change in your app's IP address, and to avoid downtime for your app due to HTTPS errors, follow these steps in the specified sequence:

1. Upload the new certificate.
2. Go to the **Custom domains** page for your app, select the ... button, and then select **Update binding**.
3. Select the new certificate and then select **Update**.
4. Delete the existing certificate.

Renew a certificate imported from Key Vault

ⓘ Note

To renew an App Service certificate, see [Renew an App Service certificate](#).

To renew a certificate that you imported into App Service from Key Vault, review [Renew your Azure Key Vault certificate](#).

After the certificate renews in your key vault, App Service automatically syncs the new certificate and updates any applicable certificate binding within 24 hours. To sync manually, follow these steps:

1. Go to your app's **Certificate** page.
2. Under **Bring your own certificates (.pfx)**, select the ... button for the imported key vault certificate, and then select **Sync**.

Frequently asked questions

How can I automate adding a bring-your-own certificate to an app?

- [Azure CLI: Bind a custom TLS/SSL certificate to a web app](#)
- [Azure PowerShell: Bind a custom TLS/SSL certificate to a web app using PowerShell](#)

Can I use a private CA (certificate authority) certificate for inbound TLS on my app?

You can use a private CA certificate for inbound TLS in [App Service Environment version 3](#). This isn't possible in App Service (multi-tenant). For more information on App Service

multi-tenant vs. single-tenant, see [App Service Environment v3](#) and [App Service public multitenant comparison](#).

Can I make outbound calls using a private CA client certificate from my app?

This is only supported for Windows container apps in multi-tenant App Service. In addition, you can make outbound calls using a private CA client certificate with both code-based and container-based apps in [App Service Environment version 3](#). For more information on App Service multi-tenant vs. single-tenant, see [App Service Environment v3](#) and [App Service public multitenant comparison](#).

Can I load a private CA certificate in my App Service Trusted Root Store?

You can load your own CA certificate into the Trusted Root Store in [App Service Environment version 3](#). You can't modify the list of Trusted Root Certificates in App Service (multi-tenant). For more information on App Service multi-tenant vs. single-tenant, see [App Service Environment v3](#) and [App Service public multitenant comparison](#).

More resources

- [Secure a custom DNS name with a TLS/SSL binding in Azure App Service](#)
- [Enforce HTTPS](#)
- [Enforce TLS 1.1/1.2](#)
- [Use a TLS/SSL certificate in your code in Azure App Service](#)
- [FAQ: App Service Certificates](#)

Feedback

Was this page helpful?



[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

Create and manage an App Service certificate for your web app

Article • 11/01/2024

This article shows how to create an App Service certificate and perform management tasks like renewing, synchronizing, and deleting certificates. Once you have an App Service certificate, you can then import it into an App Service app. An App Service certificate is a private certificate that's managed by Azure. It combines the simplicity of automated certificate management and the flexibility of renewal and export options.

If you purchase an App Service certificate from Azure, Azure manages the following tasks:

- Handles the purchase process from GoDaddy.
- Performs domain verification of the certificate.
- Maintains the certificate in [Azure Key Vault](#).
- Manages [certificate renewal](#).
- Synchronizes the certificate automatically with the imported copies in App Service apps.

ⓘ Note

After you upload a certificate to an app, the certificate is stored in a deployment unit that's bound to the App Service plan's resource group, region, and operating system combination, internally called a *webspace*. That way, the certificate is accessible to other apps in the same resource group and region combination. Certificates uploaded or imported to App Service are shared with App Services in the same deployment unit.

Prerequisites

- [Create an App Service app](#). The app's App Service plan must be in the Basic, Standard, Premium, or Isolated tier. See [Scale up an app](#) to update the tier.

ⓘ Note

Currently, App Service certificates aren't supported in Azure national clouds.

Buy and configure an App Service certificate

Buy the certificate

1. Go to the [Create App Service certificate page](#) to start the purchase.

! Note

App Service Certificates purchased from Azure are issued by GoDaddy. For some domains, you must explicitly allow GoDaddy as a certificate issuer by creating a [CAA domain record](#) with the value `0 issue godaddy.com`.

[Basics](#) [Tags](#) [Review + create](#)

Create a private App Service certificate that's managed by Azure. You can export copies, renew your certificates automatically, and sync them with your App Service apps. [Learn more](#)

Project Details

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

Once created, App Service Certificates can only be used by other App Services within the same subscription.

Subscription * [①](#) Visual Studio Enterprise Subscription [▼](#)

Resource Group * [①](#) (New) ContosoCertDemo [▼](#)
[Create new](#)

Certificate details

SKU

Standard
\$XX.XX USD/year
Can be used with one domain.

Wildcard
\$XX.XX USD/year
Can be used with multiple sub-domains.

Naked domain hostname * contoso.com [✓](#)

Certificate name * Contoso [✓](#)

Auto renewal

Auto renew prevents untimely expiration and unexpected loss of your certificate. The Certificate Authority (CA) will still require you to verify domain ownership when the certificate is in pending issuance state during renew or rekey, if you haven't verified domain ownership in the last 395 days. This process will not be automated and failure to verify domain ownership will result to failed renewals. You'll be billed annually as a new certificate order purchase upon successful certificate renewal. [Learn more](#)

Enable auto renewal Enable Disable

[Review + create](#) [◀ Previous](#) [Next : Tags >](#)

2. To configure the certificate, use the following table. When you're done, select **Review + Create**, and then select **Create**.

[+] Expand table

Setting	Description
Subscription	The Azure subscription to associate with the certificate.
Resource Group	The resource group that will contain the certificate. You can either create a new resource group or select the same resource group as your App Service app.
SKU	Determines the type of certificate to create, either a standard certificate or a wildcard certificate .
Naked domain hostname	Specify the root domain. The issued certificate provides security for <i>both</i> the root domain and the <code>www</code> subdomain. In the issued certificate, the Common Name field specifies the root domain, and the Subject Alternative Name field specifies the <code>www</code> domain. To provide security for only a subdomain, specify the fully qualified domain name for the subdomain, for example, <code>mysubdomain.contoso.com</code> .
Certificate name	The friendly name for your App Service certificate.
Enable auto renewal	Select whether to automatically renew the certificate before expiration. Each renewal extends the certificate expiration by one year. The cost is charged to your subscription.

3. When deployment is complete, select **Go to resource**.

Store the certificate in Azure Key Vault

[Key Vault](#) is an Azure service that helps safeguard cryptographic keys and secrets used by cloud applications and services. For App Service certificates, we recommend that you use Key Vault. After you finish the certificate purchase process, you must complete a few more steps before you start using the certificate.

1. On the [App Service Certificates page](#), select the certificate. On the certificate menu, select **Certificate Configuration > Step 1: Store**.

Search

- Overview
- Activity log
- Access control (IAM)
- Tags
- Diagnose and solve problems

Settings

- Certificate Configuration** (highlighted with a red box)
- Auto Renew Settings
- Timeline
- Rekey and Sync
- Export Certificate
- Properties
- Locks

Automation

Step 1: Store



Import certificate into Key Vault for secure administration.

Step 2: Verify



Verify certificate domain ownership

Step 3: Assign



- On the Key Vault Status page, select **Select from Key Vault**.
- If you create a new vault, set up the vault based on the following table, and make sure to use the same subscription and resource group as your App Service app.

[+] Expand table

Setting	Description
Resource group	Recommended: The same resource group as your App Service certificate.
Key vault name	A unique name that uses only alphanumeric characters and dashes.
Region	The same location as your App Service app.
Pricing tier	For information, see Azure Key Vault pricing details .
Days to retain deleted vaults	The number of days, after deletion, that objects remain recoverable. (See Azure Key Vault soft-delete overview .) Set a value between 7 and 90.
Purge protection	Enabling this option forces all deleted objects to remain in soft-deleted state for the entire duration of the retention period.

- Select **Next** and then select **Vault access policy**. Currently, App Service certificates support only Key Vault access policies, not the RBAC model.
- Select **Review + create**, and then select **Create**.

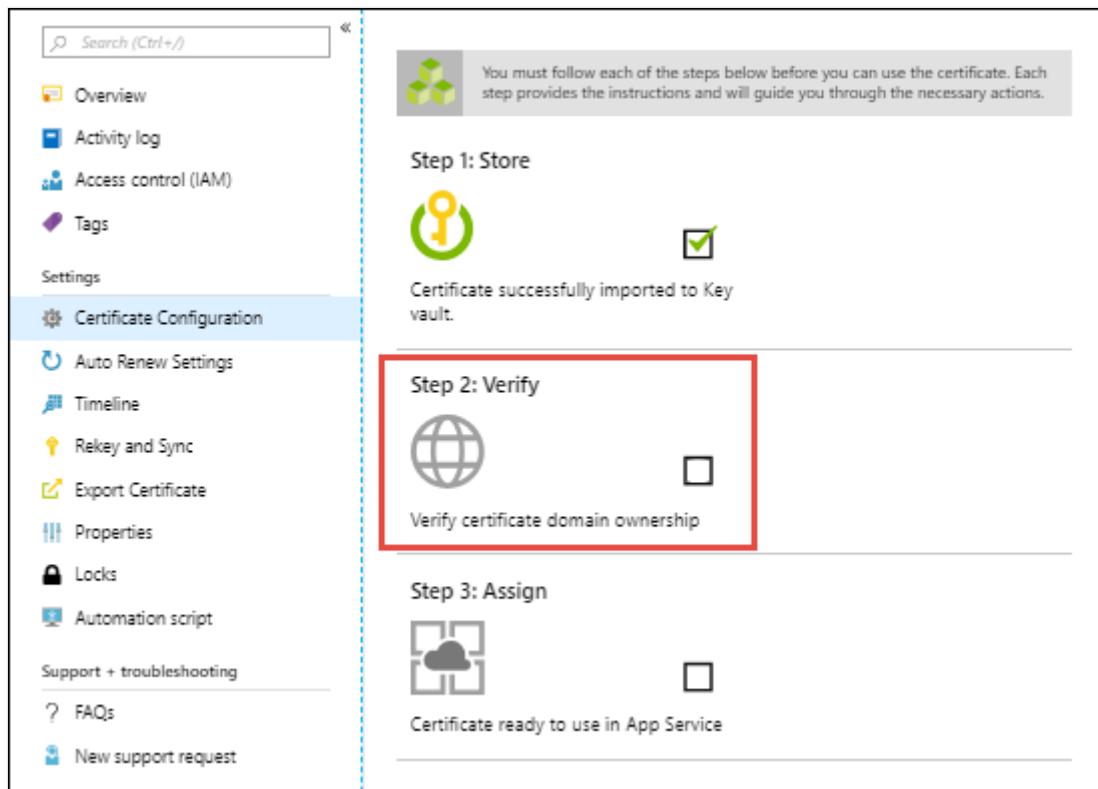
6. After the key vault is created, don't select **Go to resource**. Wait for the **Select key vault from Azure Key Vault** page to reload.

7. Select **Select**.

8. After you select the vault, close the **Key Vault Repository** page. The **Step 1: Store** option should show a green check mark to indicate success. Keep the page open for the next step.

Confirm domain ownership

1. From the same **Certificate Configuration** page as in the previous section, select **Step 2: Verify**.



2. Select **App Service Verification**. Because you mapped the domain to your web app earlier in this section, the domain is already verified. To finish this step, just select **Verify**, and then select **Refresh** until the message **Certificate is Domain Verified** appears.

The following domain verification methods are supported:

 Expand table

Method	Description
App Service Verification	The most convenient option when the domain is already mapped to an App Service app in the same subscription because the App Service app has already verified the domain ownership. Review the last step in Confirm domain ownership .
Domain Verification	Confirm an App Service domain that you purchased from Azure . Azure automatically adds the verification TXT record for you and completes the process.
Mail Verification	Confirm the domain by sending an email to the domain administrator. Instructions are provided when you select the option.
Manual Verification	Confirm the domain by using either a DNS TXT record or an HTML page. (The latter applies only to Standard certificates. See the following note.) The steps are provided after you select the option. The HTML page option doesn't work for web apps with HTTPS Only enabled. For domain verification via DNS TXT record for either the root domain (for example, <code>contoso.com</code>) or the subdomain (for example, <code>www.contoso.com</code> or <code>test.api.contoso.com</code>) and regardless of the certificate SKU, you need to add a TXT record at the root domain level, using <code>@</code> for the name and the domain verification token for the value in your DNS record.

ⓘ Important

With the Standard certificate, you get a certificate for the requested top-level domain *and* the `www` subdomain, for example, `contoso.com` and `www.contoso.com`. However, App Service Verification and Manual Verification both use HTML page verification, which doesn't support the `www` subdomain when you issue, rekey, or renew a certificate. For the Standard certificate, use Domain Verification and Mail Verification to include the `www` subdomain with the requested top-level domain in the certificate.

Once your certificate is domain-verified, [you're ready to import it into an App Service app](#).

Renew an App Service certificate

By default, App Service certificates have a one-year validity period. Before the expiration date, you can automatically or manually renew App Service certificates in one-year increments. The renewal process effectively gives you a new App Service certificate with the expiration date extended to one year from the existing certificate's expiration date.

ⓘ Note

Starting September 23 2021, if you haven't verified the domain in the last 395 days, App Service certificates require domain verification during a renew, auto-renew, or rekey process. The new certificate order remains in "pending issuance" mode during the renew, auto-renew, or rekey process until you complete the domain verification.

Unlike the free App Service managed certificate, purchased App Service certificates don't have automated domain re-verification. Failure to verify domain ownership results in failed renewals. For more information about how to verify your App Service certificate, review [Confirm domain ownership](#).

The renewal process requires that the service principal for App Service has the required permissions on your key vault. These permissions are set up for you when you import an App Service certificate through the Azure portal. Make sure that you don't remove these permissions from your key vault.

1. To change the automatic renewal setting for your App Service certificate at any time, on the [App Service Certificates page](#), select the certificate.
2. On the left menu, select **Auto Renew Settings**.
3. Select **On** or **Off**, and then select **Save**.

If you turn on automatic renewal, certificates can start automatically renewing 32 days before expiration.

Search Refresh Save Discard Manual Renew Sync

Overview Activity log Access control (IAM) Tags Diagnose and solve problems

Settings

Certificate Configuration

Auto Renew Settings

Timeline Rekey and Sync Export Certificate Properties Locks

Automation Tasks (preview) Export template

Manual renewal not allowed at this time

This App Service certificate is not eligible for manual renewal right now. Manual renewal will become available 60 days before expiry to prevent accidental renewal and auto renewal will start 32 days before expiry. The Certificate Authority (CA) will still require you to verify domain ownership when the certificate is in pending issuance state, if you haven't verified domain ownership in the last 395 days. This process will not be automated and failure to verify domain ownership will result in failed renewals. Use Manual Renew ONLY if you really need to get a new certificate issued before you can auto-renew. For rolling keys use the Rekey feature. Turn off the auto renew feature to opt out of auto renewal. [Learn more](#)

Auto Renew App Service Certificate On

PPK Linked Private Certificate

Linked Private Certificates are used in App Service apps. The private certificates can go out of sync after rekey and renew operation. Click sync to update the private certificates and the SSL bindings to the latest certificate.

Current Certificate Thumbprint

Status	Linked Private Certificate	Resource Group	Thumbprint
No linked App Service Private Certificates found, Import your App Service Certificate in App Service apps to create linked private certificates.			

4. To manually renew the certificate instead, select **Manual Renew**. You can request to manually renew your certificate 60 days before expiration, but [certificates can't be issued for longer than 397 days](#).

5. After the renew operation completes, select **Sync**.

The sync operation automatically updates the hostname bindings for the certificate in App Service without causing any downtime to your apps.

Note

If you don't select **Sync**, App Service automatically syncs your certificate within 24 hours.

Rekey an App Service certificate

If you think your certificate's private key is compromised, you can rekey your certificate. This action rotates the certificate with a new certificate issued from the certificate authority.

Note

Starting September 23 2021, if you haven't verified the domain in the last 395 days, App Service certificates require domain verification during a renew, auto-renew, or rekey process. The new certificate order remains in "pending issuance" mode during the renew, auto-renew, or rekey process until you complete the domain verification.

Unlike the free App Service managed certificate, purchased App Service certificates don't have automated domain re-verification. Failure to verify domain ownership results in failed renewals. For more information about how to verify your App Service certificate, review [Confirm domain ownership](#).

The rekey process requires that the service principal for App Service has the required permissions on your key vault. These permissions are set up for you when you import an App Service certificate through the Azure portal. Make sure that you don't remove these permissions from your key vault.

1. On the [App Service Certificates page](#), select the certificate. From the left menu, select **Rekey and Sync**.
2. To start the process, select **Rekey**. This process can take 1-10 minutes to complete.

Rekey Certificate

Rekeying your certificate will roll the certificate with a new certificate issued from the Certificate Authority (CA). While Rekeying, your certificate will go through Pending Issuance state. If you haven't verified your domain in the last 395 days, the Certificate Authority will require you to verify domain ownership again. This process will not be automated and failure to verify domain ownership will result to failed certificate rekey. Once the certificate is issued, make sure you sync your resources using this certificate to prevent disruption to service.

Certificate rekey operations are free and rekey does not incur additional charges.

Linked Private Certificate

Linked Private Certificates are used in App Service apps. The private certificates can go out of sync after rekey and renew operation. Click sync to update the private certificates and the SSL bindings to the latest certificate.

Status	Linked Private Certificate	Resource Group	Thumbprint
No linked App Service Private Certificates found, Import your App Service Certificate in App Service apps to create linked private certific...			

3. You might also be required to [reconfirm domain ownership](#).

4. After the rekey operation completes, select **Sync**.

The sync operation automatically updates the hostname bindings for the certificate in App Service without causing any downtime to your apps.

ⓘ Note

If you don't select **Sync**, App Service automatically syncs your certificate within 24 hours.

Export an App Service certificate

Because an App Service certificate is a [Key Vault secret](#), you can export a copy as a PFX file, which you can use for other Azure services or outside of Azure.

ⓘ Important

The exported certificate is an unmanaged artifact. App Service doesn't sync such artifacts when the App Service Certificate is [renewed](#). You must export and install the renewed certificate where necessary.

Azure portal

1. On the [App Service Certificates page](#), select the certificate.

2. On the left menu, select **Export Certificate**.
3. Select **Open Key Vault Secret**.
4. Select the certificate's current version.
5. Select **Download as a certificate**.

The downloaded PFX file is a raw PKCS12 file that contains both the public and private certificates and has an import password that's an empty string. You can locally install the file by leaving the password field empty. You can't [upload the file as-is into App Service](#) because the file isn't [password protected](#).

Use Azure Advisor for App Service certificate

App Service certificate is integrated with [Azure Advisor](#) to provide reliability recommendations for when your certificate requires domain verification. You must verify domain ownership for your certificate during renew, auto-renew, or rekey process if you haven't verified the domain in the last 395 days. To ensure you do not miss any certificate that requires verification or risk any certificate from expiring, you can utilize Azure Advisor to view and set up alerts for App Service certificate.

View Advisor recommendation

To view Advisor recommendation for App Service certificate:

1. Navigate to the [Azure Advisor page](#).
2. From the left menu, select **Recommendations > Reliability**
3. Select the filter option **Type equals** and search for **App Service Certificates** from the dropdown list. If the value does not exist on the dropdown menu, then that means no recommendation has been generated for your App Service certificate resources because none of them requires domain ownership verification.

Create Advisor Alerts

You [create Azure Advisor alerts on new recommendations] using different configurations. To set up Advisor Alerts specifically for App Service certificate so you can get notifications when your certificate requires domain ownership validation:

1. Navigate to the [Azure Advisor page](#).

- From the left menu, select **Monitoring > Alerts (Preview)**
- Click on **+ New Advisor Alert** on the action bar at the top. This will open a new blade called "Create Advisor Alerts".
- Under **Condition** select the following:

 Expand table

Configured by	Recommendation Type
Recommendation Type	Domain verification required to issue your App Service Certificate

- Fill out the rest of the required fields, then select the **Create alert** button at the bottom.

Delete an App Service certificate

If you delete an App Service certificate, the delete operation is irreversible and final. The result is a revoked certificate, and any binding in App Service that uses the certificate becomes invalid.

- On the [App Service Certificates page](#), select the certificate.
- From the left menu, select **Overview > Delete**.
- When the confirmation box opens, enter the certificate name, and then select **OK**.

Frequently asked questions

My App Service certificate doesn't have any value in Key Vault

Your App Service certificate is probably not yet domain-verified. Until [domain ownership is confirmed](#), your App Service certificate isn't ready for use. As a Key Vault secret, it maintains an `Initialize` tag, and its value and content-type remain empty. When domain ownership is confirmed, the key vault secret shows a value and a content-type, and the tag changes to `Ready`.

I can't export my App Service certificate with PowerShell

Your App Service certificate is probably not yet domain-verified. Until [domain ownership is confirmed](#), your App Service certificate isn't ready for use.

What changes does the App Service certificate creation process make to my existing key vault?

The creation process makes the following changes:

- Adds two access policies in the vault:
 - Microsoft.Azure.WebSites (or Microsoft Azure App Service)
 - Microsoft certificate reseller CSM Resource Provider (or Microsoft.Azure.CertificateRegistration)
- Creates a [delete lock](#) called AppServiceCertificateLock on the vault to prevent accidental deletion of the key vault.

Related content

- [Secure a custom DNS name with a TLS/SSL binding in Azure App Service](#)
- [Enforce HTTPS](#)
- [Enforce TLS 1.1/1.2](#)
- [Use a TLS/SSL certificate in your code in Azure App Service](#)
- [Frequently asked questions about creating or deleting resources in Azure App Service](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

Use a TLS/SSL certificate in your code in Azure App Service

Article • 05/01/2024

In your application code, you can access the [public or private certificates you add to App Service](#). Your app code may act as a client and access an external service that requires certificate authentication, or it may need to perform cryptographic tasks. This how-to guide shows how to use public or private certificates in your application code.

This approach to using certificates in your code makes use of the TLS functionality in App Service, which requires your app to be in **Basic** tier or higher. If your app is in **Free** or **Shared** tier, you can [include the certificate file in your app repository](#).

When you let App Service manage your TLS/SSL certificates, you can maintain the certificates and your application code separately and safeguard your sensitive data.

Prerequisites

To follow this how-to guide:

- [Create an App Service app](#)
- [Add a certificate to your app](#)

Find the thumbprint

In the [Azure portal](#), from the left menu, select **App Services > <app-name>**.

From the left navigation of your app, select **Certificates**, then select **Bring your own certificates (.pfx) or Public key certificates (.cer)**.

Find the certificate you want to use and copy the thumbprint.

Managed certificates

Bring your own certificates (.pfx)

Public key certificates (.cer)

App Service Managed Certificates are free of cost and fully managed by App Service to maintain the safety and security of your site at the highest level. To understand how to create a managed certificate for your app to consume, click on the learn more link. [Learn more](#)

Filter by keywords

Add filter

1 items

Add certificate

Delete

Certificate Status ↑	Domain	Certificate Name
<input type="checkbox"/> No action needed	www.contoso.com	Contoso www

< Previous

Items per page:

50

Page

1

of 1

Next >

Make the certificate accessible

To access a certificate in your app code, add its thumbprint to the `WEBSITE_LOAD_CERTIFICATES` app setting, by running the following command in the [Cloud Shell](#):

Azure CLI

```
az webapp config appsettings set --name <app-name> --resource-group <resource-group-name> --settings WEBSITE_LOAD_CERTIFICATES=<comma-separated-certificate-thumbprints>
```

To make all your certificates accessible, set the value to `*`.

Note

When `WEBSITE_LOAD_CERTIFICATES` is set `*`, all previously added certificates are accessible to application code. If you add a certificate to your app later, restart the app to make the new certificate accessible to your app. For more information, see [When updating \(renewing\) a certificate](#).

Load certificate in Windows apps

The `WEBSITE_LOAD_CERTIFICATES` app setting makes the specified certificates accessible to your Windows hosted app in the Windows certificate store, in `Current User\My`.

In C# code, you access the certificate by the certificate thumbprint. The following code loads a certificate with the thumbprint `E661583E8FABEF4C0BEF694CBC41C28FB81CD870`.

C#

```
using System;
using System.Linq;
using System.Security.Cryptography.X509Certificates;

string certThumbprint = "E661583E8FABEF4C0BEF694CBC41C28FB81CD870";
bool validOnly = false;

using (X509Store certStore = new X509Store(StoreName.My,
StoreLocation.CurrentUser))
{
    certStore.Open(OpenFlags.ReadOnly);

    X509Certificate2Collection certCollection = certStore.Certificates.Find(
        X509FindType.FindByThumbprint,
        // Replace below with your certificate's
thumbprint
        certThumbprint,
        validOnly);
    // Get the first cert with the thumbprint
    X509Certificate2 cert = certCollection.OfType<X509Certificate2>
().FirstOrDefault();

    if (cert is null)
        throw new Exception($"Certificate with thumbprint {certThumbprint} was
not found");

    // Use certificate
    Console.WriteLine(cert.FriendlyName);

    // Consider to call Dispose() on the certificate after it's being used,
available in .NET 4.6 and later
}
```

In Java code, you access the certificate from the "Windows-MY" store using the Subject Common Name field (see [Public key certificate ↴](#)). The following code shows how to load a private key certificate:

Java

```
import org.springframework.web.bind.annotation.RestController;
import org.springframework.web.bind.annotation.RequestMapping;
import java.security.KeyStore;
import java.security.cert.Certificate;
import java.security.PrivateKey;

...
KeyStore ks = KeyStore.getInstance("Windows-MY");
ks.load(null, null);
Certificate cert = ks.getCertificate("<subject-cn>");
PrivateKey privKey = (PrivateKey) ks.getKey("<subject-cn>", ("<password>").toCharArray());

// Use the certificate and key
...
```

For languages that don't support or offer insufficient support for the Windows certificate store, see [Load certificate from file](#).

Load certificate from file

If you need to load a certificate file that you upload manually, it's better to upload the certificate using [FTPS](#) instead of [Git](#), for example. You should keep sensitive data like a private certificate out of source control.

ⓘ Note

ASP.NET and ASP.NET Core on Windows must access the certificate store even if you load a certificate from a file. To load a certificate file in a Windows .NET app, load the current user profile with the following command in the [Cloud Shell](#):

Azure CLI

```
az webapp config appsettings set --name <app-name> --resource-group <resource-group-name> --settings WEBSITE_LOAD_USER_PROFILE=1
```

This approach to using certificates in your code makes use of the TLS functionality in App Service, which requires your app to be in **Basic** tier or higher.

The following C# example loads a public certificate from a relative path in your app:

C#

```

using System;
using System.IO;
using System.Security.Cryptography.X509Certificates;

...
var bytes = File.ReadAllBytes("~/<relative-path-to-cert-file>");
var cert = new X509Certificate2(bytes);

// Use the loaded certificate

```

To see how to load a TLS/SSL certificate from a file in Node.js, PHP, Python, or Java, see the documentation for the respective language or web platform.

Load certificate in Linux/Windows containers

The `WEBSITE_LOAD_CERTIFICATES` app setting makes the specified certificates accessible to your Windows or Linux custom containers (including built-in Linux containers) as files. The files are found under the following directories:

[] [Expand table](#)

Container platform	Public certificates	Private certificates
Windows container	<code>C:\appservice\certificates\public</code>	<code>C:\appservice\certificates\private</code>
Linux container	<code>/var/ssl/certs</code>	<code>/var/ssl/private</code>

The certificate file names are the certificate thumbprints.

! Note

App Service injects the certificate paths into Windows containers as the following environment variables `WEBSITE_PRIVATE_CERTS_PATH`, `WEBSITE_INTERMEDIATE_CERTS_PATH`, `WEBSITE_PUBLIC_CERTS_PATH`, and `WEBSITE_ROOT_CERTS_PATH`. It's better to reference the certificate path with the environment variables instead of hardcoding the certificate path, in case the certificate paths change in the future.

In addition, [Windows Server Core](#) and [Windows Nano Server containers](#) load the certificates into the certificate store automatically, in `LocalMachine\My`. To load the

certificates, follow the same pattern as [Load certificate in Windows apps](#). For Windows Nano based containers, use these file paths [Load the certificate directly from file](#).

Windows

The following C# example shows how to load a public certificate in a .NET Framework app in a Windows Server Core Container.

C#

```
using System;
using System.Linq;
using System.Security.Cryptography.X509Certificates;

string certThumbprint = "E661583E8FABEF4C0BEF694CBC41C28FB81CD870";
bool validOnly = false;

using (X509Store certStore = new X509Store(StoreName.My,
StoreLocation.LocalMachine))
{
    certStore.Open(OpenFlags.ReadOnly);

    X509Certificate2Collection certCollection =
certStore.Certificates.Find(
        X509FindType.FindByThumbprint,
        // Replace below with your certificate's
thumbprint
        certThumbprint,
        validOnly);
    // Get the first cert with the thumbprint
    X509Certificate2 cert = certCollection.OfType<X509Certificate2>
().FirstOrDefault();

    if (cert is null)
        throw new Exception($"Certificate with thumbprint {certThumbprint} was not found");

    // Use certificate
    Console.WriteLine(cert.FriendlyName);

    // Consider to call Dispose() on the certificate after it's being
    used, available in .NET 4.6 and later
}
```

The following C# example shows how to load a public certificate in a .NET Core app in a Windows Server Core or Windows Nano Server Container.

C#

```

using System.Security.Cryptography.X509Certificates;

string Thumbprint = "C0CF730E216F5D690D1834446554DF5DC577A78B";

using X509Store store = new X509Store(StoreName.My,
StoreLocation.LocalMachine);
{
    store.Open(OpenFlags.ReadOnly);

    // Get the first cert with the thumbprint
    var certificate = store.Certificates.OfType<X509Certificate2>()
        .First(c => c.Thumbprint == Thumbprint) ?? throw new
Exception($"Certificate with thumbprint {Thumbprint} was not found");

    // Use certificate
    ViewData["certificateDetails"] =
certificate.IssuerName.Name.ToString();
}

```

To see how to load a TLS/SSL certificate from a file in Node.js, PHP, Python, or Java, see the documentation for the respective language or web platform.

When updating (renewing) a certificate

When you renew a certificate and add it to your app, it gets a new thumbprint, which also needs to be [made accessible](#). How it works depends on your certificate type.

If you manually upload the [public](#) or [private](#) certificate:

- If you list thumbprints explicitly in `WEBSITE_LOAD_CERTIFICATES`, add the new thumbprint to the app setting.
- If `WEBSITE_LOAD_CERTIFICATES` is set to `*`, restart the app to make the new certificate accessible.

If you renew a certificate [in Key Vault](#), such as with an [App Service certificate](#), the daily sync from Key Vault makes the necessary update automatically when synchronizing your app with the renewed certificate.

- If `WEBSITE_LOAD_CERTIFICATES` contains the old thumbprint of your renewed certificate, the daily sync updates the old thumbprint to the new thumbprint automatically.
- If `WEBSITE_LOAD_CERTIFICATES` is set to `*`, the daily sync makes the new certificate accessible automatically.

More resources

- [Secure a custom DNS name with a TLS/SSL binding in Azure App Service](#)
- [Enforce HTTPS](#)
- [Enforce TLS 1.1/1.2](#)
- [FAQ : App Service Certificates](#)
- [Environment variables and app settings reference](#)

Configure TLS mutual authentication for Azure App Service

Article • 01/09/2025

You can restrict access to your Azure App Service app by enabling different types of authentication for it. One way to do it is to request a client certificate when the client request is over TLS/SSL and validate the certificate. This mechanism is called Transport Layer Security (TLS) mutual authentication or client certificate authentication. This article shows how to set up your app to use client certificate authentication.

ⓘ Note

Your app code is responsible for validating the client certificate. App Service doesn't do anything with this client certificate other than forwarding it to your app.

If you access your site over HTTP and not HTTPS, you will not receive any client certificate. So if your application requires client certificates, you should not allow requests to your application over HTTP.

Prepare your web app

To create custom TLS/SSL bindings or enable client certificates for your App Service app, your [App Service plan](#) must be in the **Basic**, **Standard**, **Premium**, or **Isolated** tier. To make sure that your web app is in the supported pricing tier, follow these steps:

Go to your web app

1. In the [Azure portal](#) search box, find and select **App Services**.

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

2. On the App Services page, select your web app's name.

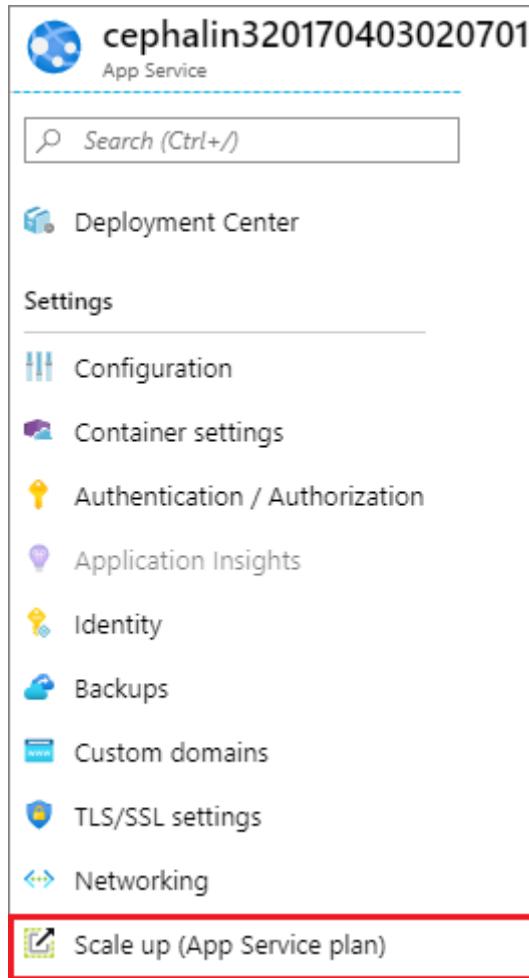
The screenshot shows the "App Services" management page. The top navigation bar includes "Home > App Services", "Documentation", and "More". Below the header, there are buttons for "Add", "Edit columns", "Refresh", "Assign tags", "Start", "Restart", and "More". A note says "Subscriptions: All 2 selected - Don't see a subscription? Open Directory + Subscription settings". There are filter buttons for "Filter by na...", "All subsc...", "All resou...", "All locati...", "All tags", and "No group...". The main table displays "6 items" with columns for "Name", "Status", "App Type", and "App Service". The row for "cephalin320170403020701" is highlighted with a red box. The table also lists "denniseastusbot", "myFirstAzureWebApp20190...", and "WebApplicationASPDotNET...".

Name	Status	App Type	App Service
cephalin320170403020701	Running	Web App	test-sku
denniseastusbot	Running	Web App	z76-z763
myFirstAzureWebApp20190...	Running	Web App	ServicePl
WebApplicationASPDotNET...	Running	Web App	ServicePl

You're now on your web app's management page.

Check the pricing tier

1. In the left menu for your web app, under the **Settings** section, select **Scale up (App Service plan)**.

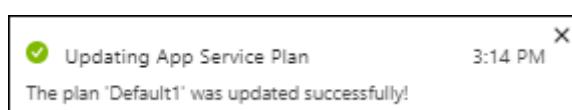


2. Make sure that your web app isn't in the F1 or D1 tier, which doesn't support custom TLS/SSL.
3. If you need to scale up, follow the steps in the next section. Otherwise, close the [Scale up](#) page, and skip the [Scale up your App Service plan](#) section.

Scale up your App Service plan

1. Select any non-free tier, such as **B1**, **B2**, **B3**, or any other tier in the **Production** category.
2. When you're done, select **Select**.

When the following message appears, the scale operation has completed.



Enable client certificates

When you enable client certificate for your app, you should select your choice of client certificate mode. Each mode defines how your app handles incoming client certificates:

[\[+\] Expand table](#)

Client certificate modes	Description
Required	All requests require a client certificate.
Optional	Requests may or may not use a client certificate and clients are prompted for a certificate by default. For example, browser clients will show a prompt to select a certificate for authentication.
Optional Interactive User	Requests may or may not use a client certificate and clients are not prompted for a certificate by default. For example, browser clients won't show a prompt to select a certificate for authentication.

Azure portal

To set up your app to require client certificates in Azure portal:

1. Navigate to your app's management page.
2. From the left navigation of your app's management page, select **Configuration > General Settings**.
3. Select **Client certificate mode** of choice. Select **Save** at the top of the page.

Exclude paths from requiring authentication

When you enable mutual auth for your application, all paths under the root of your app require a client certificate for access. To remove this requirement for certain paths, define exclusion paths as part of your application configuration.

ⓘ Note

Using any client certificate exclusion path triggers TLS renegotiation for incoming requests to the app.

- From the left navigation of your app's management page, select Configuration > General Settings.
- Next to Certificate exclusion paths, select the edit icon.
- Select New path, specify a path, or a list of paths separated by , or ;, and select OK.
- Select Save at the top of the page.

In the following screenshot, any path for your app that starts with `/public` doesn't request a client certificate. Path matching is case-insensitive.

The screenshot shows the Azure App Service configuration interface for an app named "my-demo-app". The left sidebar lists various settings like Overview, Activity log, and Configuration (which is highlighted with a red box). The main content area shows the "General settings" tab selected (also highlighted with a red box). Under "General settings", there are sections for Stack settings, Platform settings, Debugging, and Incoming client certificates. In the "Incoming client certificates" section, the "Client certificate mode" dropdown is set to "Require" (radio button selected), and the "Certificate exclusion paths" input field contains the value `/public`.

Client certificate and TLS renegotiation

For some client certificate settings, App Service requires TLS renegotiation to read a request before knowing whether to prompt for a client certificate. Any of the following settings triggers TLS renegotiation:

- Using "Optional Interactive User" client certificate mode.

2. Using client certificate exclusion path.

Note

TLS 1.3 and HTTP 2.0 don't support TLS renegotiation. These protocols will not work if your app is configured with client certificate settings that use TLS renegotiation.

To disable TLS renegotiation and to have the app negotiate client certificates during TLS handshake, you must configure your app with *all* these settings:

1. Set client certificate mode to "Required" or "Optional"
2. Remove all client certificate exclusion paths

Uploading large files with TLS renegotiation

Client certificate configurations that use TLS renegotiation cannot support incoming requests with large files greater than 100 kb due to buffer size limitations. In this scenario, any POST or PUT requests over 100 kb will fail with a 403 error. This limit isn't configurable and can't be increased.

To address the 100 kb limit, consider these alternative solutions:

1. Disable TLS renegotiation. Update your app's client certificate configurations with *all* these settings:
 - Set client certificate mode to either "Required" or "Optional"
 - Remove all client certificate exclusion paths
2. Send a HEAD request before the PUT/POST request. The HEAD request handles the client certificate.
3. Add the header `Expect: 100-Continue` to your request. This causes the client to wait until the server responds with a `100 Continue` before sending the request body, which bypasses the buffers.

Access client certificate

In App Service, TLS termination of the request happens at the frontend load balancer. When App Service forwards the request to your app code with [client certificates enabled](#), it injects an `X-ARR-ClientCert` request header with the client certificate. App Service doesn't do anything with this client certificate other than forwarding it to your app. Your app code is responsible for validating the client certificate.

For ASP.NET, the client certificate is available through the `HttpRequest.ClientCertificate` property.

For other application stacks (Node.js, PHP, etc.), the client cert is available in your app through a base64 encoded value in the `X-ARR-ClientCert` request header.

ASP.NET Core sample

For ASP.NET Core, middleware is provided to parse forwarded certificates. Separate middleware is provided to use the forwarded protocol headers. Both must be present for forwarded certificates to be accepted. You can place custom certificate validation logic in the [CertificateAuthentication options](#).

C#

```
public class Startup
{
    public Startup(IConfiguration configuration)
    {
        Configuration = configuration;
    }

    public IConfiguration Configuration { get; }

    public void ConfigureServices(IServiceCollection services)
    {
        services.AddControllersWithViews();
        // Configure the application to use the protocol and client ip
address forwarded by the frontend load balancer
        services.Configure<ForwardedHeadersOptions>(options =>
        {
            options.ForwardedHeaders =
                ForwardedHeaders.XForwardedFor |
ForwardedHeaders.XForwardedProto;
            // Only loopback proxies are allowed by default. Clear that
restriction to enable this explicit configuration.
            options.KnownNetworks.Clear();
            options.KnownProxies.Clear();
        });

        // Configure the application to client certificate forwarded the
frontend load balancer
        services.AddCertificateForwarding(options => {
options.CertificateHeader = "X-ARR-ClientCert"; });

        // Add certificate authentication so when authorization is performed
the user will be created from the certificate

        services.AddAuthentication(CertificateAuthenticationDefaults.AuthenticationS
cheme).AddCertificate();
    }
}
```

```
}

public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
{
    if (env.IsDevelopment())
    {
        app.UseDeveloperExceptionPage();
    }
    else
    {
        app.UseExceptionHandler("/Home/Error");
        app.UseHsts();
    }

    app.UseForwardedHeaders();
    app.UseCertificateForwarding();
    app.UseHttpsRedirection();

    app.UseAuthentication()
    app.UseAuthorization();

    app.UseStaticFiles();

    app.UseRouting();

    app.UseEndpoints(endpoints =>
    {
        endpoints.MapControllerRoute(
            name: "default",
            pattern: "{controller=Home}/{action=Index}/{id?}");
    });
}
}
```

ASP.NET WebForms sample

C#

```
using System;
using System.Collections.Specialized;
using System.Security.Cryptography.X509Certificates;
using System.Web;

namespace ClientCertificateUsageSample
{
    public partial class Cert : System.Web.UI.Page
    {
        public string certHeader = "";
        public string errorString = "";
        private X509Certificate2 certificate = null;
        public string certThumbprint = "";
        public string certSubject = "";
    }
}
```

```

public string certIssuer = "";
public string certSignatureAlg = "";
public string certIssueDate = "";
public string certExpiryDate = "";
public bool isValidCert = false;

//
// Read the certificate from the header into an X509Certificate2
object
// Display properties of the certificate on the page
//
protected void Page_Load(object sender, EventArgs e)
{
    NameValueCollection headers = base.Request.Headers;
    certHeader = headers["X-ARR-ClientCert"];
    if (!String.IsNullOrEmpty(certHeader))
    {
        try
        {
            byte[] clientCertBytes =
Convert.FromBase64String(certHeader);
            certificate = new X509Certificate2(clientCertBytes);
            certSubject = certificate.Subject;
            certIssuer = certificate.Issuer;
            certThumbprint = certificate.Thumbprint;
            certSignatureAlg =
certificate.SignatureAlgorithm.FriendlyName;
            certIssueDate =
certificate.NotBefore.ToString(" " +
certificate.NotBefore.ToString(" " +
certExpiryDate =
certificate.NotAfter.ToString(" " +
certificate.NotAfter.ToString(" ");
        }
        catch (Exception ex)
        {
            errorString = ex.ToString();
        }
        finally
        {
            isValidCert = IsValidClientCertificate();
            if (!isValidCert) Response.StatusCode = 403;
            else Response.StatusCode = 200;
        }
    }
    else
    {
        certHeader = "";
    }
}

//
// This is a SAMPLE verification routine. Depending on your
application logic and security requirements,
// you should modify this method

```

```

//  

private bool IsValidClientCertificate()  

{  

    // In this example we will only accept the certificate as a  

    // valid certificate if all the conditions below are met:  

    // 1. The certificate isn't expired and is active for the  

    // current time on server.  

    // 2. The subject name of the certificate has the common  

    // name nildevecc  

    // 3. The issuer name of the certificate has the common name  

    // nildevecc and organization name Microsoft Corp  

    // 4. The thumbprint of the certificate is  

    30757A2E831977D8BD9C8496E4C99AB26CB9622B  

    //  

    // This example doesn't test that this certificate is  

    // chained to a Trusted Root Authority (or revoked) on the server  

    // and it allows for self signed certificates  

    //  

    if (certificate == null ||  

!String.IsNullOrEmpty(errorString)) return false;  

    // 1. Check time validity of certificate  

    if (DateTime.Compare(DateTime.Now, certificate.NotBefore) <  

0 || DateTime.Compare(DateTime.Now, certificate.NotAfter) > 0) return false;  

    // 2. Check subject name of certificate  

    bool foundSubject = false;  

    string[] certSubjectData = certificate.Subject.Split(new  

char[] { ',' }, StringSplitOptions.RemoveEmptyEntries);  

    foreach (string s in certSubjectData)  

    {  

        if (String.Compare(s.Trim(), "CN=nildevecc") == 0)  

        {  

            foundSubject = true;  

            break;
        }
    }
    if (!foundSubject) return false;  

    // 3. Check issuer name of certificate  

    bool foundIssuerCN = false, foundIssuerO = false;  

    string[] certIssuerData = certificate.Issuer.Split(new  

char[] { ',' }, StringSplitOptions.RemoveEmptyEntries);  

    foreach (string s in certIssuerData)  

    {  

        if (String.Compare(s.Trim(), "CN=nildevecc") == 0)
        {
            foundIssuerCN = true;
            if (foundIssuerO) break;
        }
    }
    if (String.Compare(s.Trim(), "O=Microsoft Corp") == 0)
    {
        foundIssuerO = true;
    }
}

```

```

                if (foundIssuerCN) break;
            }

            if (!foundIssuerCN || !foundIssuerO) return false;

            // 4. Check thumbprint of certificate
            if (String.Compare(certificate.Thumbprint.Trim().ToUpper(),
"30757A2E831977D8BD9C8496E4C99AB26CB9622B") != 0) return false;

            return true;
        }
    }
}

```

Node.js sample

The following Node.js sample code gets the `X-ARR-ClientCert` header and uses `node-forge`² to convert the base64-encoded PEM string into a certificate object and validate it:

JavaScript

```

import { NextFunction, Request, Response } from 'express';
import { pki, md, asn1 } from 'node-forge';

export class AuthorizationHandler {
    public static authorizeClientCertificate(req: Request, res: Response,
next: NextFunction): void {
        try {
            // Get header
            const header = req.get('X-ARR-ClientCert');
            if (!header) throw new Error('UNAUTHORIZED');

            // Convert from PEM to pki.CERT
            const pem = `-----BEGIN CERTIFICATE-----${header}-----END
CERTIFICATE-----`;
            const incomingCert: pki.Certificate =
pki.certificateFromPem(pem);

            // Validate certificate thumbprint
            const fingerPrint =
md.sha1.create().update(asn1.toDer(pki.certificateToAsn1(incomingCert)).getBytes()).digest().toHex();
            if (fingerPrint.toLowerCase() !==
'abcdef1234567890abcdef1234567890abcdef12') throw new Error('UNAUTHORIZED');

            // Validate time validity
            const currentDate = new Date();
            if (currentDate < incomingCert.validity.notBefore || currentDate
> incomingCert.validity.notAfter) throw new Error('UNAUTHORIZED');
        }
    }
}

```

```

        // Validate issuer
        if (incomingCert.issuer.hash.toLowerCase() !==
'abcdef1234567890abcdef1234567890abcdef12') throw new Error('UNAUTHORIZED');

        // Validate subject
        if (incomingCert.subject.hash.toLowerCase() !==
'abcdef1234567890abcdef1234567890abcdef12') throw new Error('UNAUTHORIZED');

        next();
    } catch (e) {
        if (e instanceof Error && e.message === 'UNAUTHORIZED') {
            res.status(401).send();
        } else {
            next(e);
        }
    }
}

```

Java sample

The following Java class encodes the certificate from `X-ARR-ClientCert` to an `X509Certificate` instance. `certificateIsValid()` validates that the certificate's thumbprint matches the one given in the constructor and that certificate hasn't expired.

Java

```

import java.io.ByteArrayInputStream;
import java.security.NoSuchAlgorithmException;
import java.security.cert.*;
import java.security.MessageDigest;

import sun.security.provider.X509Factory;

import javax.xml.bind.DatatypeConverter;
import java.util.Base64;
import java.util.Date;

public class ClientCertValidator {

    private String thumbprint;
    private X509Certificate certificate;

    /**
     * Constructor.
     * @param certificate The certificate from the "X-ARR-ClientCert" HTTP
     * header
     * @param thumbprint The thumbprint to check against
     * @throws CertificateException If the certificate factory cannot be
     * created.

```

```

        */
    public ClientCertValidator(String certificate, String thumbprint) throws
CertificateException {
        certificate = certificate
            .replaceAll(X509Factory.BEGIN_CERT, "")
            .replaceAll(X509Factory.END_CERT, "");
        CertificateFactory cf = CertificateFactory.getInstance("X.509");
        byte [] base64Bytes = Base64.getDecoder().decode(certificate);
        X509Certificate X509cert = (X509Certificate)
cf.generateCertificate(new ByteArrayInputStream(base64Bytes));

        this.setCertificate(X509cert);
        this.setThumbprint(thumbprint);
    }

    /**
     * Check that the certificate's thumbprint matches the one given in the
constructor, and that the
     * certificate hasn't expired.
     * @return True if the certificate's thumbprint matches and hasn't
expired. False otherwise.
    */
    public boolean certificateIsValid() throws NoSuchAlgorithmException,
CertificateEncodingException {
        return certificateHasNotExpired() && thumbprintIsValid();
    }

    /**
     * Check certificate's timestamp.
     * @return Returns true if the certificate hasn't expired. Returns false
if it has expired.
    */
    private boolean certificateHasNotExpired() {
        Date currentTime = new java.util.Date();
        try {
            this.getCertificate().checkValidity(currentTime);
        } catch (CertificateExpiredException |
CertificateNotYetValidException e) {
            return false;
        }
        return true;
    }

    /**
     * Check the certificate's thumbprint matches the given one.
     * @return Returns true if the thumbprints match. False otherwise.
    */
    private boolean thumbprintIsValid() throws NoSuchAlgorithmException,
CertificateEncodingException {
        MessageDigest md = MessageDigest.getInstance("SHA-1");
        byte[] der = this.getCertificate().getEncoded();
        md.update(der);
        byte[] digest = md.digest();
        String digestHex = DatatypeConverter.printHexBinary(digest);
        return

```

```

digestHex.toLowerCase().equals(this.getThumbprint().toLowerCase());
}

// Getters and setters

public void setThumbprint(String thumbprint) {
    this.thumbprint = thumbprint;
}

public String getThumbprint() {
    return this.thumbprint;
}

public X509Certificate getCertificate() {
    return certificate;
}

public void setCertificate(X509Certificate certificate) {
    this.certificate = certificate;
}
}

```

Python sample

The following Flask and Django Python code samples implement a decorator named `authorize_certificate` that can be used on a view function to permit access only to callers that present a valid client certificate. It expects a PEM formatted certificate in the `X-ARR-ClientCert` header and uses the Python [cryptography](#) package to validate the certificate based on its fingerprint (thumbprint), subject common name, issuer common name, and beginning and expiration dates. If validation fails, the decorator ensures that an HTTP response with status code 403 (Forbidden) is returned to the client.

Flask

Python

```

from functools import wraps
from datetime import datetime, timezone
from flask import abort, request
from cryptography import x509
from cryptography.x509.oid import NameOID
from cryptography.hazmat.primitives import hashes

def validate_cert(request):

    try:
        cert_value = request.headers.get('X-ARR-ClientCert')

```

```

        if cert_value is None:
            return False

        cert_data = ''.join(['-----BEGIN CERTIFICATE-----\n',
cert_value, '\n-----END CERTIFICATE-----\n',])
        cert = x509.load_pem_x509_certificate(cert_data.encode('utf-8'))

        fingerprint = cert.fingerprint(hashes.SHA1())
        if fingerprint != b'12345678901234567890':
            return False

        subject = cert.subject
        subject_cn = subject.get_attributes_for_oid(NameOID.COMMON_NAME)
[0].value
        if subject_cn != "contoso.com":
            return False

        issuer = cert.issuer
        issuer_cn = issuer.get_attributes_for_oid(NameOID.COMMON_NAME)
[0].value
        if issuer_cn != "contoso.com":
            return False

        current_time = datetime.now(timezone.utc)

        if current_time < cert.not_valid_before_utc:
            return False

        if current_time > cert.not_valid_after_utc:
            return False

        return True

    except Exception as e:
        # Handle any errors encountered during validation
        print(f"Encountered the following error during certificate
validation: {e}")
        return False

def authorize_certificate(f):
    @wraps(f)
    def decorated_function(*args, **kwargs):
        if not validate_cert(request):
            abort(403)
        return f(*args, **kwargs)
    return decorated_function

```

The following code snippet shows how to use the decorator on a Flask view function.

Python

```
@app.route('/hellocert')
@authorize_certificate
def hellocert():
    print('Request for hellocert page received')
    return render_template('index.html')
```

Feedback

Was this page helpful?



[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

Secure connectivity to Azure services and databases from Azure App Service

Article • 07/08/2024

Your app service might need to connect to other Azure services such as a database, storage, or another app. This overview recommends different methods for connecting and when to use them.

Today, the decision for a connectivity approach is closely related to secrets management. The common pattern of using connection secrets in connection strings, such as username and password, secret key, etc. is no longer considered the most secure approach for connectivity. The risk is even higher today because threat actors regularly crawl public GitHub repositories for accidentally committed connection secrets. For cloud applications, the best secrets management is to have no secrets at all. When you migrate to Azure App Service, your app might start with secrets-based connectivity, and App Service lets you keep secrets securely. However, Azure can help secure your app's back-end connectivity through Microsoft Entra authentication, which eliminates secrets altogether in your app.

[] Expand table

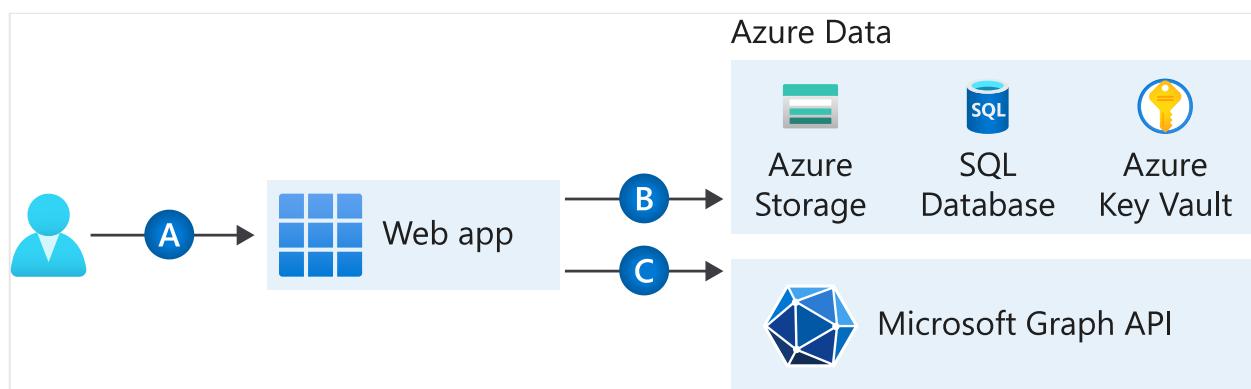
Connection method	When to use
Connect with an app identity	<ul style="list-style-type: none">* You want to remove credentials, keys, or secrets completely from your application.* The downstream Azure service supports Microsoft Entra authentication, such as Microsoft Graph.* The downstream resource doesn't need to know the current signed-in user or doesn't need granular authorization of the current signed-in user.
Connect on behalf of the signed-in user	<ul style="list-style-type: none">* The app must access a downstream resource on behalf of the signed-in user.* The downstream Azure service supports Microsoft Entra authentication, such as Microsoft Graph.* The downstream resource must perform granular authorization of the current signed-in user.
Connect using secrets	<ul style="list-style-type: none">* The downstream resource requires connection secrets.* Your app connects to non-Azure services, such as an on-premises database server.* The downstream Azure service doesn't support Microsoft Entra authentication yet.

Connect with an app identity

If your app already uses a single set of credentials to access a downstream Azure service, you can quickly convert the connection to use an app identity instead. A [managed identity](#) from Microsoft Entra ID lets App Service access resources without secrets, and you can manage its access through role-based access control (RBAC). A managed identity can connect to any Azure resource that supports Microsoft Entra authentication, and the authentication takes place with short-lived tokens.

The following image demonstrates the following an App Service connecting to other Azure services:

- A: User visits Azure app service website.
- B: Securely **connect from App Service to** another Azure service using a managed identity.
- C: Securely **connect from App Service to** Microsoft Graph using a managed identity.



Examples of using application secrets to connect to a database:

- [Tutorial: Connect to Azure databases from App Service without secrets using a managed identity](#)
- [Tutorial: Connect to SQL Database from .NET App Service without secrets using a managed identity](#)
- [Tutorial: Connect to a PostgreSQL Database from Java Tomcat App Service without secrets using a managed identity](#)

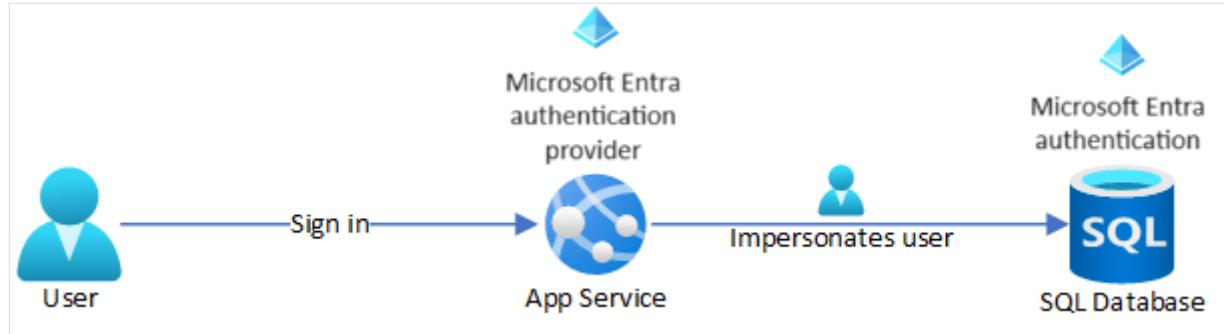
Connect on behalf of the signed-in user

Your app might need to connect to a downstream service on behalf of the signed-in user. App Service lets you easily authenticate users using the most common identity providers (see [Authentication and authorization in Azure App Service and Azure](#)

Functions). If you use the Microsoft provider (Microsoft Entra authentication), you can then flow the signed-in user to any downstream service. For example:

- Run a database query that returns confidential data that the signed-in user is authorized to read.
- Retrieve personal data or take actions as the signed-in user in Microsoft Graph.

The following image demonstrates an application securely accessing an SQL database on behalf of the signed-in user.



Some common scenarios are:

- Connect to Microsoft Graph on behalf of the user
- Connect to an SQL database on behalf the user
- Connect to another App Service app on behalf of the user
- Flow the signed-in user through multiple layers of downstream services

Connect using secrets

There are two recommended ways to use secrets in your app: using secrets stored in Azure Key Vault or secrets in App Service app settings.

Use secrets from Key Vault

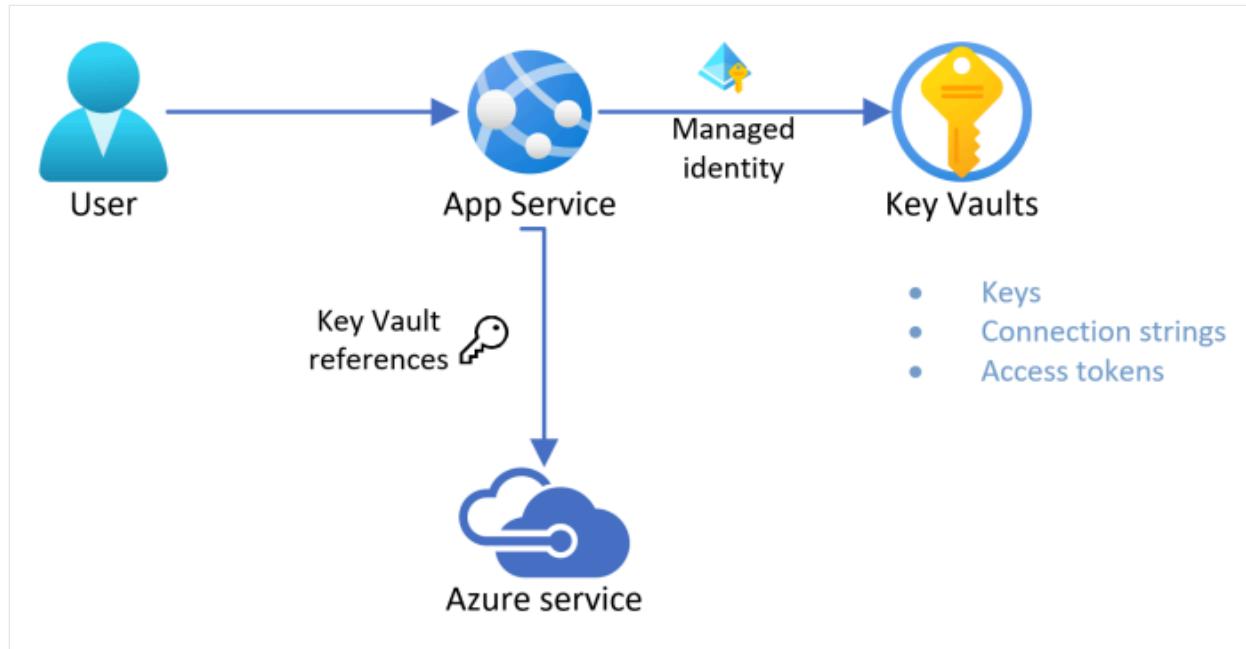
Azure Key Vault can be used to securely store secrets and keys, monitor access and use of secrets, and simplify administration of application secrets. If the downstream service doesn't support Microsoft Entra authentication or requires a connection string or key, use Key Vault to store your secrets and connect your app to Key Vault with a managed identity and retrieve the secrets. Your app can access they key vault secrets as [Key Vault references](#) in the app settings.

Benefits of managed identities integrated with Key Vault include:

- Access to the key vault secret is restricted to the app.

- App contributors, such as administrators, might have complete control of the App Service resources, and at the same time have no access to the key vault secrets.
- No code change is required if your application code already accesses connection secrets with app settings.
- Key Vault provides monitoring and auditing of who accessed secrets.
- Rotation of key vault secrets requires no changes in App Service.

The following image demonstrates App Service connecting to Key Vault using a managed identity and then accessing an Azure service using secrets stored in Key Vault:



Use secrets in app settings

For apps that connect to services using secrets (such as usernames, passwords, and API keys), App Service can store them securely in [app settings](#). These secrets are injected into your application code as environment variables at app startup. App settings are always encrypted when stored (encrypted-at-rest). For more advanced secrets management, such as secrets rotation, access policies, and audit history, try [using Key Vault](#).

Examples of using application secrets to connect to a database:

- [Tutorial: Deploy an ASP.NET Core and Azure SQL Database app to Azure App Service](#)
- [Tutorial: Deploy an ASP.NET app to Azure with Azure SQL Database](#)
- [Tutorial: Deploy a PHP, MySQL, and Redis app to Azure App Service](#)
- [Deploy a Node.js + MongoDB web app to Azure](#)
- [Deploy a Python \(Django or Flask\) web app with PostgreSQL in Azure](#)
- [Tutorial: Build a Tomcat web app with Azure App Service on Linux and MySQL](#)

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

Next steps

Learn how to:

- Securely store secrets in [Azure Key Vault](#).
- Access resources using a [managed identity](#).
- Store secrets using App Service [app settings](#).
- [Connect to Microsoft Graph](#) as the user.
- [Connect to an SQL database](#) as the user.
- [Connect to another App Service app](#) as the user.
- [Connect to another App Service app and then a downstream service](#) as the user.

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#)

Tutorial: Deploy an ASP.NET Core and Azure SQL Database app to Azure App Service

Article • 01/31/2025

In this tutorial, you learn how to deploy a data-driven ASP.NET Core app to Azure App Service and connect to an Azure SQL Database. You'll also deploy an Azure Cache for Redis to enable the caching code in your application. Azure App Service is a highly scalable, self-patching, web-hosting service that can easily deploy apps on Windows or Linux. Although this tutorial uses an ASP.NET Core 8.0 app, the process is the same for other versions of ASP.NET Core.

In this tutorial, you learn how to:

- ✓ Create a secure-by-default App Service, SQL Database, and Redis cache architecture.
- ✓ Secure connection secrets using a managed identity and Key Vault references.
- ✓ Deploy a sample ASP.NET Core app to App Service from a GitHub repository.
- ✓ Access App Service connection strings and app settings in the application code.
- ✓ Make updates and redeploy the application code.
- ✓ Generate database schema by uploading a migrations bundle.
- ✓ Stream diagnostic logs from Azure.
- ✓ Manage the app in the Azure portal.
- ✓ Provision the same architecture and deploy by using Azure Developer CLI.
- ✓ Optimize your development workflow with GitHub Codespaces and GitHub Copilot.

Prerequisites

- An Azure account with an active subscription. If you don't have an Azure account, you [can create one for free ↗](#).
- A GitHub account. you can also [get one for free ↗](#).
- Knowledge of ASP.NET Core development.
- (Optional) To try GitHub Copilot, a [GitHub Copilot account ↗](#). A 30-day free trial is available.

Skip to the end

If you just want to see the sample app in this tutorial running in Azure, just run the following commands in the [Azure Cloud Shell](#), and follow the prompt:

Bash

```
dotnet tool install --global dotnet-ef  
mkdir msdocs-app-service-sqldb-dotnetcore  
cd msdocs-app-service-sqldb-dotnetcore  
azd init --template msdocs-app-service-sqldb-dotnetcore  
azd up
```

1. Run the sample

First, you set up a sample data-driven app as a starting point. For your convenience, the [sample repository](#), includes a [dev container](#) configuration. The dev container has everything you need to develop an application, including the database, cache, and all environment variables needed by the sample application. The dev container can run in a [GitHub codespace](#), which means you can run the sample on any computer with a web browser.

Step 1: In a new browser window:

1. Sign in to your GitHub account.
2. Navigate to <https://github.com/Azure-Samples/msdocs-app-service-sqldb-dotnetcore/fork>.
3. Unselect **Copy the main branch only**. You want all the branches.
4. Select **Create fork**.

Create a new fork

A *fork* is a copy of a repository. Forking a repository allows you to freely experiment with changes without affecting the original project. [View existing forks.](#)

Required fields are marked with an asterisk (*).

Owner *



Repository name *

msdocs-app-service-sqldb-d

msdocs-app-service-sqldb-dotnetcore is available.

By default, forks are named the same as their upstream repository. You can customize the name to distinguish it further.

Description (optional)

Copy the main branch only

Contribute back to Azure-Samples/msdocs-app-service-sqldb-dotnetcore by adding your own branch. [Learn more.](#)

You are creating a fork in your personal account.

[Create fork](#)



Step 2: In the GitHub fork:

1. Select **main > starter-no-infra** for the starter branch. This branch contains just the sample project and no Azure-related files or configuration.
2. Select **Code > Create codespace on starter-no-infra**. The codespace takes a few minutes to set up.

The screenshot shows a GitHub repository page for 'msdocs-app-service-sqldb-dotnetcore'. The 'Code' tab is active. A red box highlights the dropdown menu 'starter-no-infra'. Another red box highlights the 'Code' button in the top right. On the left, there's a sidebar with file navigation. In the center, under 'Codespaces', a red box highlights the green 'Create codespace on starter-no-infra' button.

Step 3: In the codespace terminal:

1. Run database migrations with `dotnet ef database update`.
2. Run the app with `dotnet run`.
3. When you see the notification `Your application running on port 5093 is available.`, select **Open in Browser**. You should see the sample application in a new browser tab. To stop the application, type `Ctrl + C`.

The screenshot shows the Microsoft Visual Studio Code interface. In the left sidebar (Explorer), there's a tree view of a project named 'MSDOCS-APP-SERVICE-SQLDB-DOTNETCORE'. The project structure includes files like '.devcontainer', '.github', 'bin', 'Controllers', 'Data', 'Migrations', 'Models', 'obj', 'Properties', 'Views', and 'wwwroot'. There are also configuration files like '.gitignore', 'ActionTimerFilter.cs', 'appsettings.Development.json', 'appsettings.json', 'CONTRIBUTING.md', and 'DotNetCoreSqlDb.csproj'. The right side of the interface has a preview of a README.md file titled 'Deploy an ASP.NET Core web app with SQL Database in Azure'. Below the preview is a terminal window showing command-line output for a 'dotnet ef database update' command. A tooltip in the terminal area says 'Your application running on port 5093 is available. See all forwarded ports'. At the bottom right of the terminal, there are buttons for 'Open in Browser' and 'Make Public'.

💡 Tip

You can ask [GitHub Copilot](#) about this repository. For example:

- @workspace *What does this project do?*
- @workspace *What does the .devcontainer folder do?*

Having issues? Check the [Troubleshooting section](#).

2. Create App Service, database, and cache

In this step, you create the Azure resources. The steps used in this tutorial create a set of secure-by-default resources that include App Service, Azure SQL Database, and Azure Cache. For the creation process, you'll specify:

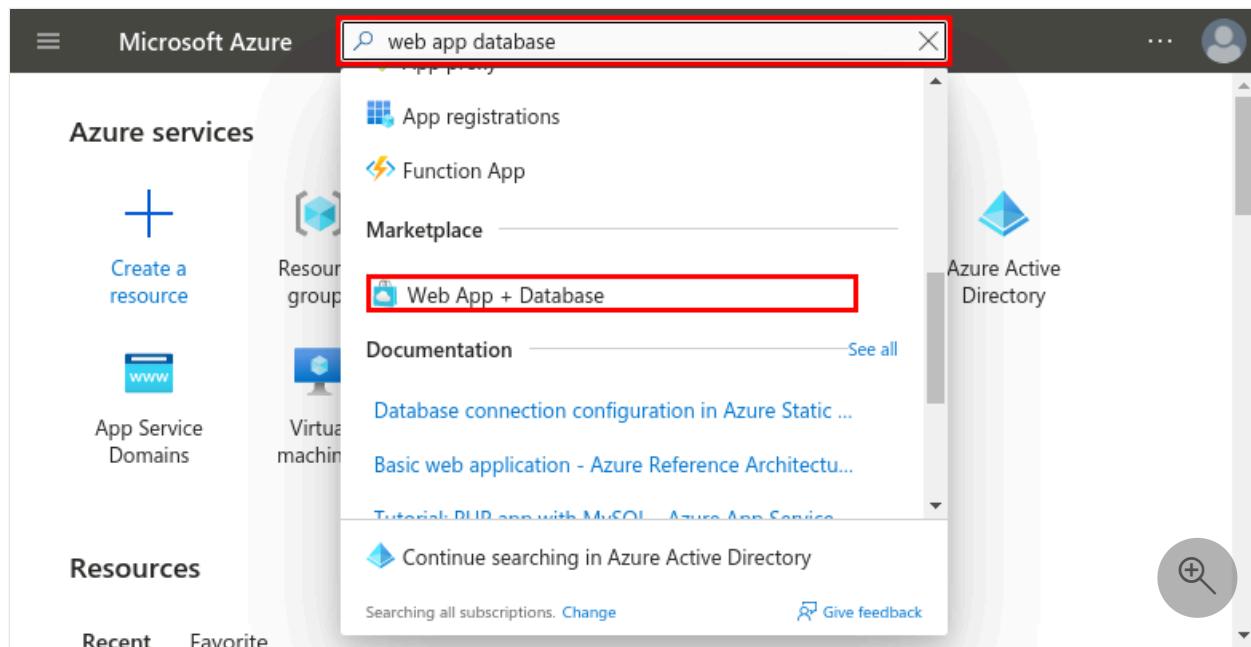
- The **Name** for the web app. It's used as part of the DNS name for your app in the form of `https://<app-name>-<hash>.azurewebsites.net`.
- The **Region** to run the app physically in the world. It's also used as part of the DNS name for your app.
- The **Runtime stack** for the app. It's where you select the .NET version to use for your app.

- The **Hosting plan** for the app. It's the pricing tier that includes the set of features and scaling capacity for your app.
- The **Resource Group** for the app. A resource group lets you group (in a logical container) all the Azure resources needed for the application.

Sign in to the [Azure portal](#) and follow these steps to create your Azure App Service resources.

Step 1: In the Azure portal:

1. Enter "web app database" in the search bar at the top of the Azure portal.
2. Select the item labeled **Web App + Database** under the **Marketplace** heading. You can also navigate to the [creation wizard](#) directly.



Step 2: In the **Create Web App + Database** page, fill out the form as follows.

1. **Resource Group:** Select **Create new** and use a name of **msdocs-core-sql-tutorial**.
2. **Region:** Any Azure region near you.
3. **Name:** **msdocs-core-sql-XYZ** where **XYZ** is any three random characters. This name must be unique across Azure.
4. **Runtime stack:** **.NET 8 (LTS)**.
5. **Engine:** **SQLAzure**. Azure SQL Database is a fully managed platform as a service (PaaS) database engine that's always running on the latest stable version of the SQL Server.
6. **Add Azure Cache for Redis?: Yes**.
7. **Hosting plan:** **Basic**. When you're ready, you can [scale up](#) to a production pricing tier.
8. Select **Review + create**.
9. After validation completes, select **Create**.

Microsoft Azure (Preview)  

Home > Create Web App + Database X

Basics Tags Review + create

This template will create a secure by default configuration where the only publicly accessible endpoint will be your app following the recommended security best practices. [Learn more](#)

Project Details

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

Subscription * 

Resource Group * 
 (New) msdocs-core-sql-tutorial 
[Create new](#)

Region * 
 West Europe

Web App Details

Name * 
 msdocs-core-sql-251 
 .azurewebsites.net

Try a unique default hostname (preview). [More about this update](#)

Runtime stack * 
 .NET 8 (LTS)

Database

Info Database access will be locked down and not exposed to the public internet. This is in compliance with recommended best practices for security.

Engine * 
 SQLAzure (recommended)

Server name * 
 msdocs-core-sql-251-server 

Database name * 
 msdocs-core-sql-251-database 

Azure Cache for Redis

Add Azure Cache for Redis? Yes No

Cache name * 
 msdocs-core-sql-251-cache 

Hosting

Hosting plan * Basic - For hobby or research purposes Standard - General purpose production apps

Review + create **< Previous** **Next : Tags >** 

Step 3: The deployment takes a few minutes to complete. Once deployment completes, select the **Go to resource** button. You're taken directly to the App Service app, but the following resources are created:

- **Resource group:** The container for all the created resources.
- **App Service plan:** Defines the compute resources for App Service. A Linux plan in the *Basic* tier is created.
- **App Service:** Represents your app and runs in the App Service plan.
- **Virtual network:** Integrated with the App Service app and isolates back-end network traffic.
- **Private endpoints:** Access endpoints for the key vault, the database server, and the Redis cache in the virtual network.
- **Network interfaces:** Represents private IP addresses, one for each of the private endpoints.
- **Azure SQL Database server:** Accessible only from behind its private endpoint.
- **Azure SQL Database:** A database and a user are created for you on the server.
- **Azure Cache for Redis:** Accessible only from behind its private endpoint.
- **Key vault:** Accessible only from behind its private endpoint. Used to manage secrets for the App Service app.
- **Private DNS zones:** Enable DNS resolution of the key vault, the database server, and the Redis cache in the virtual network.

Your deployment is complete



Deployment name : Microsoft.Web-WebAppDatabase-Portal-13aec024-a580

Subscription :

Resource group : [msdocs-core-sql-tutorial](#)

Start time : 6/28/2024, 4:33:41 PM

Correlation ID :

 Deployment details

 Next steps

[Go to resource](#)

[Give feedback](#)

 [Tell us about your experience with deployment](#)



3. Secure connection secrets

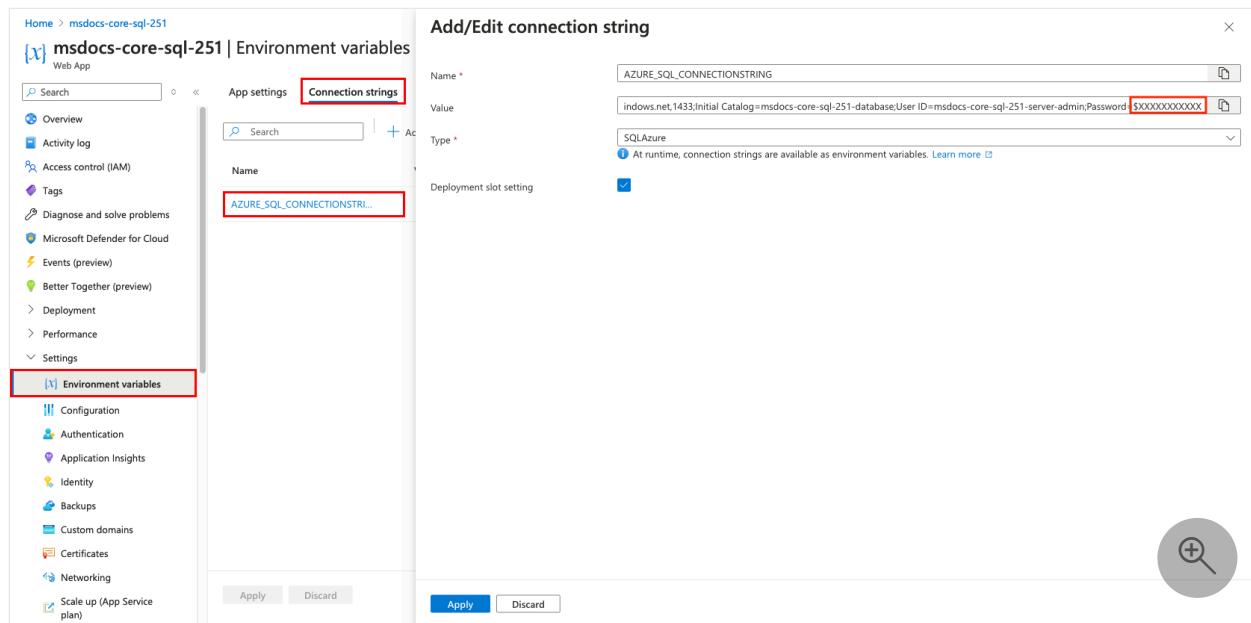
The creation wizard generated the connectivity variable for you already as [.NET connection strings](#) and [app settings](#). However, the security best practice is to keep secrets out of App Service completely. You'll move your secrets to a key vault and change your app setting to [Key Vault references](#) with the help of Service Connectors.

💡 Tip

To use passwordless authentication, see [How do I change the SQL Database connection to use a managed identity instead?](#)

Step 1: Retrieve the existing connection string

1. In the left menu of the App Service page, select **Settings > Environment variables > Connection strings**.
2. Select **AZURE_SQL_CONNECTIONSTRING**.
3. In **Add/Edit connection string**, in the **Value** field, find the *Password=* part at the end of the string.
4. Copy the password string after *Password=* for use later. This connection string lets you connect to the SQL database secured behind a private endpoint. However, the secrets are saved directly in the App Service app, which isn't the best. Likewise, the Redis cache connection string in the **App settings** tab contains a secret. You'll change this.



Step 2: Create a key vault for secure management of secrets

1. In the top search bar, type "key vault", then select **Marketplace > Key Vault**.
2. In **Resource Group**, select **msdocs-core-sql-tutorial**.
3. In **Key vault name**, type a name that consists of only letters and numbers.
4. In **Region**, set it to the same location as the resource group.

Basics [Access configuration](#) [Networking](#) [Tags](#) [Review + create](#)

Azure Key Vault is a cloud service used to manage keys, secrets, and certificates. Key Vault eliminates the need for developers to store security information in their code. It allows you to centralize the storage of your application secrets which greatly reduces the chances that secrets may be leaked. Key Vault also allows you to securely store secrets and keys backed by Hardware Security Modules or HSMs. The HSMs used are Federal Information Processing Standards (FIPS) 140-2 Level 2 validated. In addition, key vault provides logs of all access and usage attempts of your secrets so you have a complete audit trail for compliance.

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

Key vault name * 

Region * 

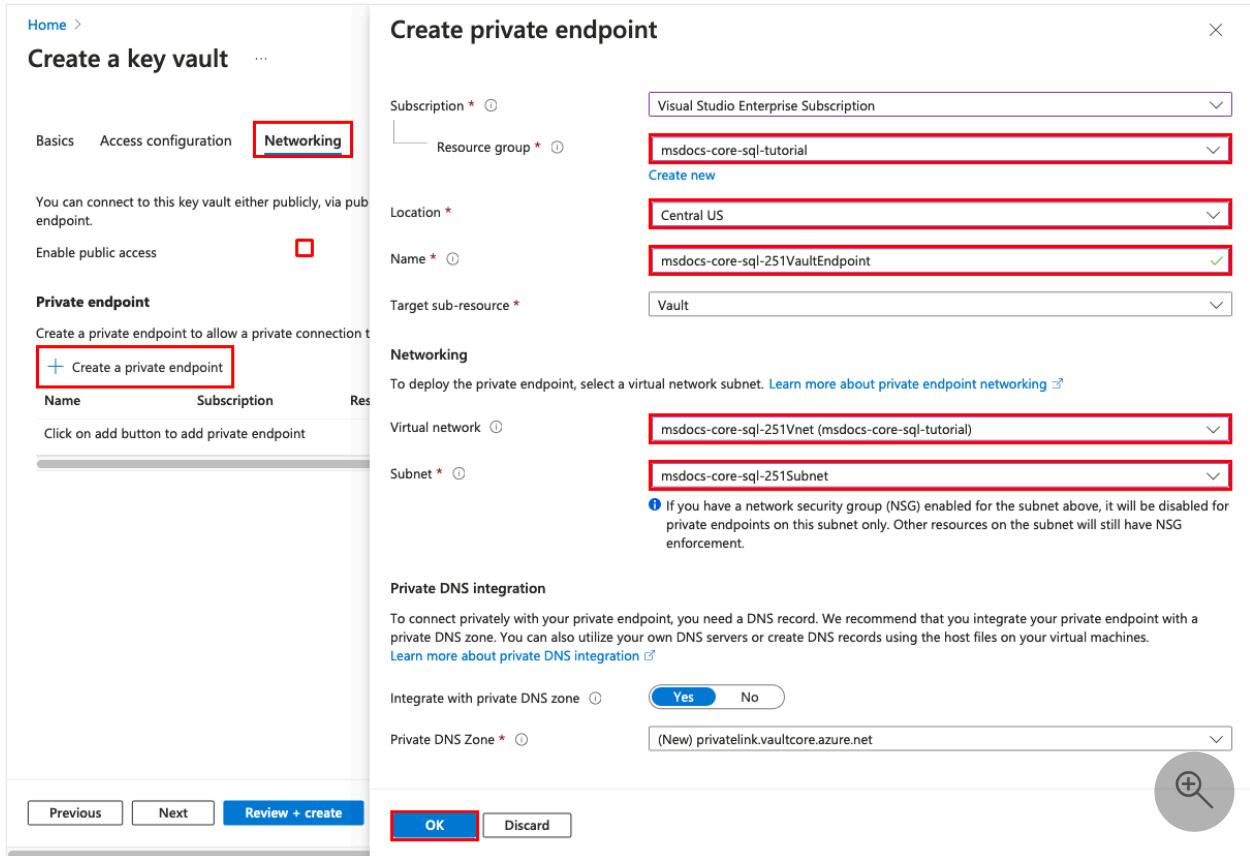
Pricing tier * 

Recovery options

[Previous](#) [Next](#) [Review + create](#) 

Step 3: Secure the key vault with a Private Endpoint

1. Select the **Networking** tab.
2. Unselect **Enable public access**.
3. Select **Create a private endpoint**.
4. In **Resource Group**, select **msdocs-core-sql-tutorial**.
5. In the dialog, in **Location**, select the same location as your App Service app.
6. In **Name**, type **msdocs-core-sql-XYZVaultEndpoint**.
7. In **Virtual network**, select **msdocs-core-sql-XYZVnet**.
8. In **Subnet**, **msdocs-core-sql-XYZSubnet**.
9. Select **OK**.
10. Select **Review + create**, then select **Create**. Wait for the key vault deployment to finish. You should see "Your deployment is complete."



Step 4:

1. In the top search bar, type *msdocs-core-sql*, then the App Service resource called **msdocs-core-sql-XYZ**.
2. In the App Service page, in the left menu, select **Settings > Service Connector**.
There are already two connectors, which the app creation wizard created for you.
3. Select checkbox next to the SQL Database connector, then select **Edit**.
4. Select the **Authentication** tab.
5. In **Password**, paste the password you copied earlier.
6. Select **Store Secret in Key Vault**.
7. Under **Key Vault Connection**, select **Create new**. A **Create connection** dialog is opened on top of the edit dialog.

The screenshot shows the Azure portal interface for managing a Service Connector named 'msdocs-core-sql-251'. On the left, the navigation menu is open, showing various settings like Deployment, Performance, and Settings. The 'Service Connector' option is highlighted with a red box. On the right, the 'defaultConnector' configuration page is displayed. The 'Authentication' tab is selected and highlighted with a red box. The 'Edit' button is also highlighted with a red box. The 'Connection string' authentication type is selected. Below it, there are links to 'Database credentials' and 'Key Vault'. Under 'Continue with...', 'Database credentials' is selected. The 'Username' field contains 'msdocs-core-sql-251-server-admin'. The 'Password' field is masked. A 'Forgot password?' link is present. There is a checked checkbox for 'Store Secret In Key Vault'. The 'Key Vault Connection' dropdown shows 'No item available' and has a 'Create new' button highlighted with a red box. A 'Store Configuration in App Configuration' checkbox is also present. At the bottom, there are 'Next : Networking', 'Previous', and 'Cancel' buttons.

Step 5: Establish the Key Vault connection

1. In the **Create connection** dialog for the Key Vault connection, in **Key Vault**, select the key vault you created earlier.
2. Select **Review + Create**.
3. When validation completes, select **Create**.

Create connection

X

Basics

Networking

Review + Create

Select the service instance and client type.

Service type * ⓘ

Key Vault

Connection name * ⓘ

keyvault_ca5fa

Subscription * ⓘ

Visual Studio Enterprise Subscription

Key vault * ⓘ

vault03940324

[Create new](#)

Client type * ⓘ

.NET

[Next : Networking](#)

[Cancel](#)

[Report an issue](#)

Step 6: Finalize the SQL Database connector settings

1. You're back in the edit dialog for **defaultConnector**. In the **Authentication** tab, wait for the key vault connector to be created. When it's finished, the **Key Vault Connection** dropdown automatically selects it.
2. Select **Next: Networking**.
3. Select **Save**. Wait until the **Update succeeded** notification appears.

defaultConnector

X

Basics

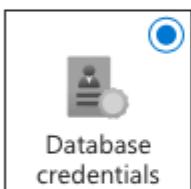
Authentication

Networking

Select the authentication type you'd like to use between your compute service and target service. [Learn more](#)

- System assigned managed identity(Supported via Azure CLI. [Learn more](#)) ⓘ
- User assigned managed identity(Supported via Azure CLI. [Learn more](#)) ⓘ
- Connection string ⓘ
- Service principal(Supported via Azure CLI. [Learn more](#)) ⓘ

Continue with...



Username *

msdocs-core-sql-251-server-admin



Password *



[Forgot password?](#)

Store Secret In Key Vault ⓘ

Key Vault Connection * ⓘ

vault03940324 (keyvault_ca5fa)



[Create new](#)

Store Configuration in App Configuration ⓘ

[Next : Networking](#)

[Previous](#)

[Cancel](#)



Step 7: Configure the Redis connector to use Key Vault secrets

1. In the Service Connectors page, select the checkbox next to the Cache for Redis connector, then select **Edit**.
2. Select the **Authentication** tab.

3. Select **Store Secret** in Key Vault.
4. Under **Key Vault Connection**, select the key vault you created.
5. Select **Next: Networking**.
6. Select **Configure firewall rules** to enable access to target service. The app creation wizard already secured the SQL database with a private endpoint.
7. Select **Save**. Wait until the **Update succeeded** notification appears.

The screenshot shows the Azure App Service configuration interface for the application 'msdocs-core-sql-251'. On the left, the 'Service Connector' section is selected, highlighted with a red box. Within this section, the 'Edit' button is also highlighted with a red box. The right pane displays the 'RedisConnector' configuration page. The 'Authentication' tab is selected, also highlighted with a red box. Under 'Service type', the 'Cache for Redis' option is selected and highlighted with a red box. The 'Store Secret In Key Vault' checkbox is checked and highlighted with a red box. The 'Key Vault Connection' dropdown is set to 'vault03940324 (keyvault_ca5fa)', which is also highlighted with a red box. At the bottom of the page, the 'Next : Networking' button is highlighted with a red box.

Step 8: Verify the Key Vault integration

1. From the left menu, select **Settings > Environment variables > Connection strings** again.
2. Next to **AZURE_SQL_CONNECTIONSTRING**, select **Show value**. The value should be `@Microsoft.KeyVault(...)`, which means that it's a **key vault reference** because the secret is now managed in the key vault.
3. To verify the Redis connection string, select the **App setting** tab. Next to **AZURE_REDIS_CONNECTIONSTRING**, select **Show value**. The value should be `@Microsoft.KeyVault(...)` too.

The screenshot shows the 'Connection strings' section of the Azure App Service configuration. On the left, there's a sidebar with various settings like Events (preview), Better Together (preview), Deployment, Performance, and Settings. Under Settings, 'Environment variables' is selected and highlighted with a red box. The main area shows a table with columns 'Name', 'Value', and 'Deploy'. There's one row for 'AZURE_SQL_CONNECTI...' with a 'Show value' link next to it, which is also highlighted with a red box. At the bottom, there are 'Apply' and 'Discard' buttons, and a 'Send us your feedback' button.

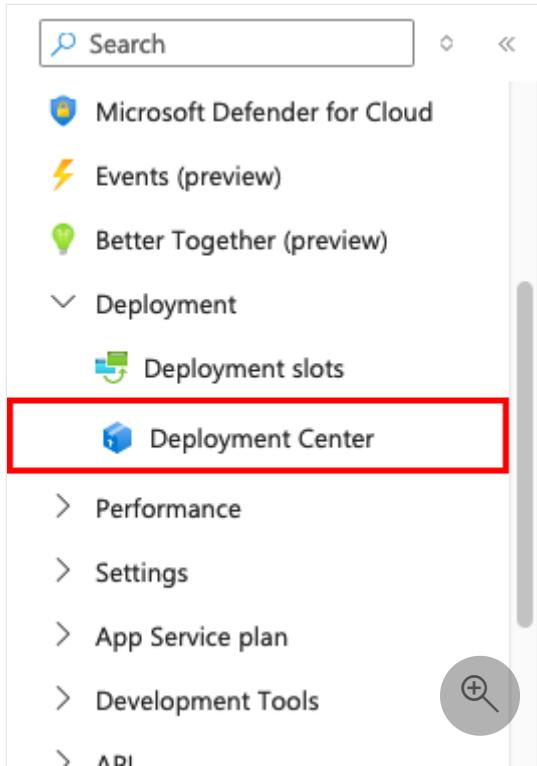
To summarize, the process for securing your connection secrets involved:

- Retrieving the connection secrets from the App Service app's environment variables.
- Creating a key vault.
- Creating a Key Vault connection with the system-assigned managed identity.
- Updating the service connectors to store the secrets in the key vault.

4. Deploy sample code

In this step, you configure GitHub deployment using GitHub Actions. It's just one of many ways to deploy to App Service, but also a great way to have continuous integration in your deployment process. By default, every `git push` to your GitHub repository kicks off the build and deploy action.

Step 1: In the left menu, select **Deployment > Deployment Center**.



Step 2: In the Deployment Center page:

1. In **Source**, select **GitHub**. By default, **GitHub Actions** is selected as the build provider.
2. Sign in to your GitHub account and follow the prompt to authorize Azure.
3. In **Organization**, select your account.
4. In **Repository**, select **msdocs-app-service-sqldb-dotnetcore**.
5. In **Branch**, select **starter-no-infra**. This is the same branch that you worked in with your sample app, without any Azure-related files or configuration.
6. For **Authentication type**, select **User-assigned identity**.
7. In the top menu, select **Save**. App Service commits a workflow file into the chosen GitHub repository, in the `.github/workflows` directory. By default, the deployment center [creates a user-assigned identity](#) for the workflow to authenticate using Microsoft Entra (OIDC authentication). For alternative authentication options, see [Deploy to App Service using GitHub Actions](#).

[Save](#) [Discard](#) [Browse](#) [Manage publish profile](#) [Sync](#) [Leave Feedback](#)

Settings * [Logs](#) [FTPS credentials](#)

Info You're now in the production slot, which is not recommended for setting up CI/CD. [Learn more](#) [X](#)

Deploy and build code from your preferred source and build provider. [Learn more](#)

Source *

[Change provider](#)

Building with GitHub Actions. [Change provider](#).

GitHub

App Service will place a GitHub Actions workflow in your chosen repository to build and deploy your app whenever there is a commit on the chosen branch. If you can't find an organization or repository, you may need to enable additional permissions on GitHub. You must have write access to your chosen GitHub repository to deploy with GitHub Actions. [Learn more](#)

Signed in as

<github-alias> [Change Account](#) ⓘ

Organization *

Repository *

[...](#)

Branch *

[...](#)

Build

Runtime stack

.NET

Version

.NET Core 8.0

Authentication settings

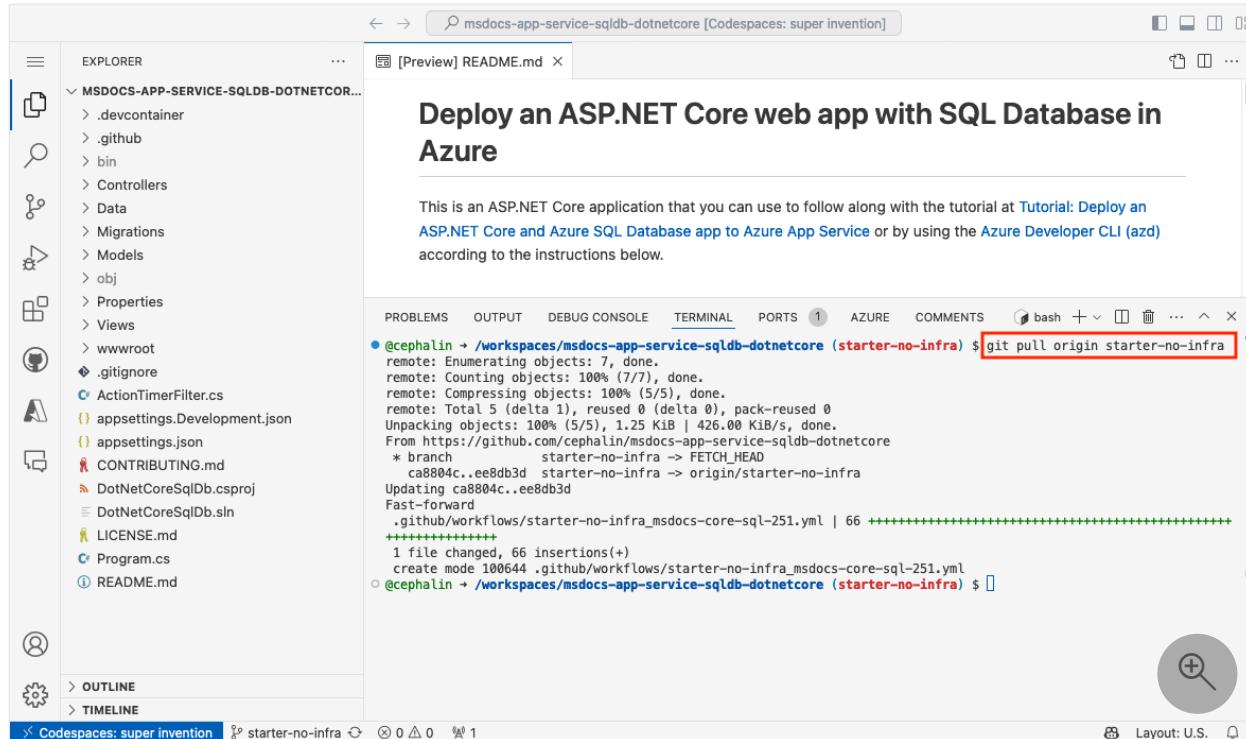
Select how you want your GitHub Action workflow to authenticate to Azure. If you choose user-assigned identity, the identity selected will be federated with GitHub as an authorized client and given write permissions on the app. [Learn more](#)

Authentication type *

User-assigned identity [+ New](#)

Basic authentication

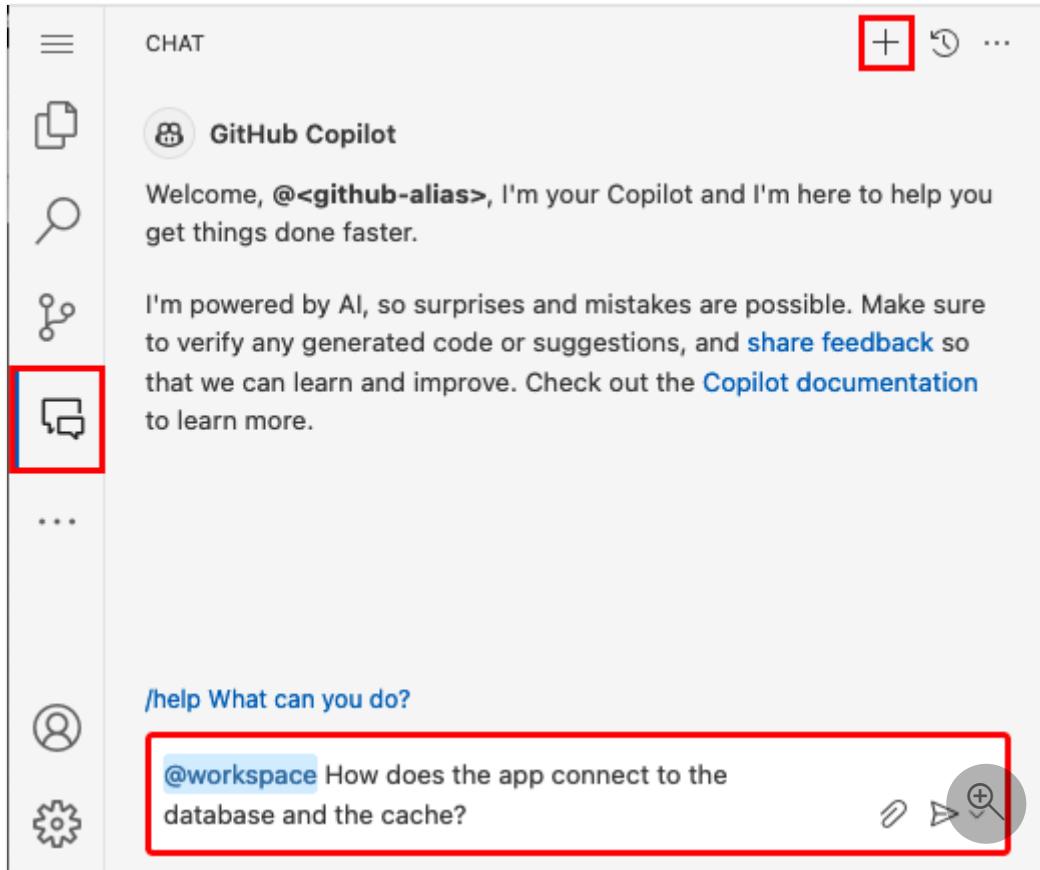
Step 3: Back in the GitHub codespace of your sample fork, run `git pull origin starter-no-infra`. This pulls the newly committed workflow file into your codespace.



The screenshot shows the GitHub Codespace interface for the repository "msdocs-app-service-sqldb-dotnetcore". The left sidebar displays the project structure under "EXPLORER", including files like ".devcontainer", ".github", "bin", "Controllers", "Data", "Migrations", "Models", "obj", "Properties", "Views", "wwwroot", ".gitignore", "ActionTimerFilter.cs", "appsettings.Development.json", "appsettings.json", "CONTRIBUTING.md", "DotNetCoreSqlDb.csproj", "DotNetCoreSqlDb.sln", "LICENSE.md", "Program.cs", and "README.md". The right side shows the "TERMINAL" tab open, displaying the command `git pull origin starter-no-infra` and its execution output. The output shows the cloning of the "starter-no-infra" branch from the "origin" repository, compressing objects, unpacking objects, and updating the local repository. The terminal also shows a fast-forward merge of the "251" branch into the "starter-no-infra" branch. The status bar at the bottom indicates "Layout: U.S.".

Step 4 (Option 1: with GitHub Copilot):

1. Start a new chat session by selecting the **Chat** view, then selecting **+**.
2. Ask, "*@workspace How does the app connect to the database and the cache?*"
Copilot might give you some explanation about the `MyDbContext` class and how it's configured in `Program.cs`.
3. Ask, "In production mode, I want the app to use the connection string called `AZURE_SQL_CONNECTIONSTRING` for the database and the app setting called `AZURE_REDIS_CONNECTIONSTRING`." Copilot might give you a code suggestion similar to the one in the **Option 2: without GitHub Copilot** steps below and even tell you to make the change in the `Program.cs` file.
4. Open `Program.cs` in the explorer and add the code suggestion. GitHub Copilot doesn't give you the same response every time, and it's not always correct. You might need to ask more questions to fine-tune its response. For tips, see [What can I do with GitHub Copilot in my codespace?](#)



Step 4 (Option 2: without GitHub Copilot):

1. Open `Program.cs` in the explorer.
2. Find the commented code (lines 12-21) and uncomment it. This code connects to the database by using `AZURE_SQL_CONNECTIONSTRING` and connects to the Redis cache by using the app setting `AZURE_REDIS_CONNECTIONSTRING`.

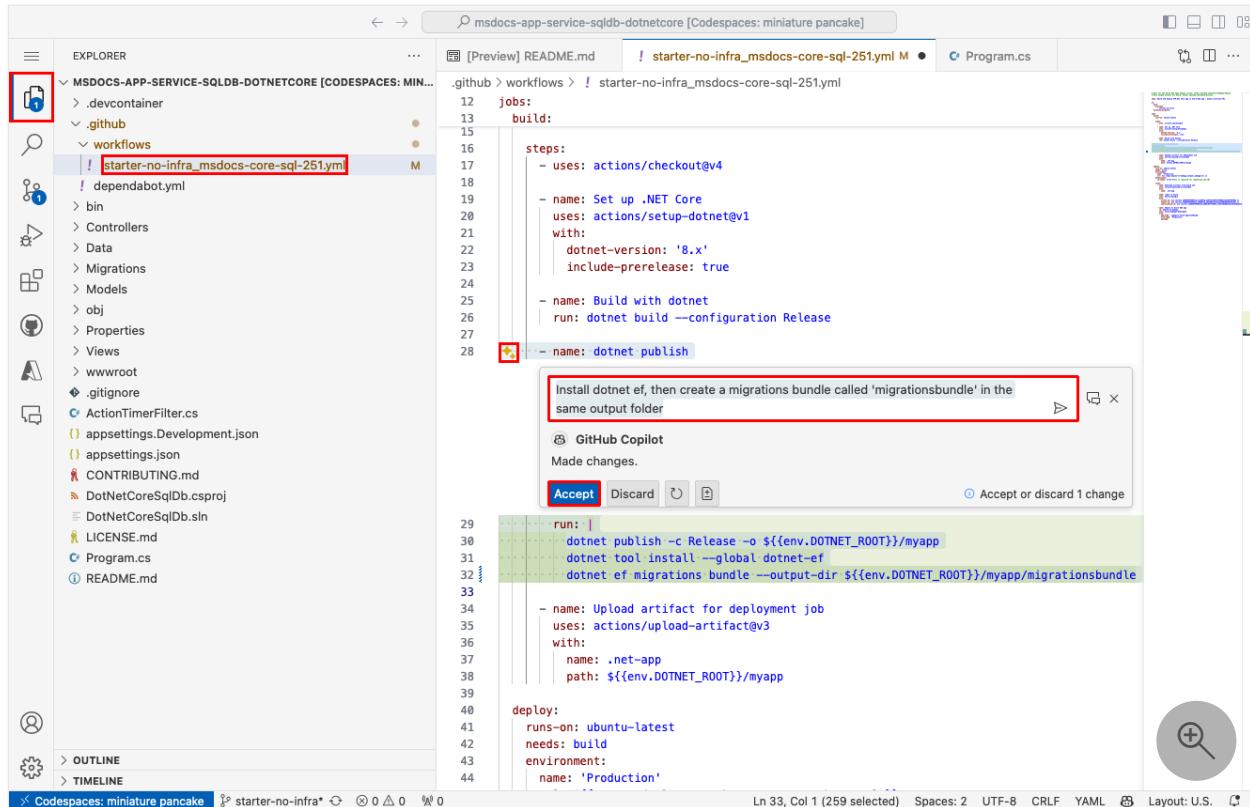
```

1 //<!-- using Microsoft.EntityFrameworkCore;
2 //<!-- using DotNetCoreSqlDb.Data;
3 //<!-- var builder = WebApplication.CreateBuilder(args);
4 //<!--
5 //<!-- Add database context and cache
6 //<!-- if(builder.Environment.IsDevelopment())
7 //<!-- {
8 //<!--     builder.Services.AddDbContext<MyDbContext>(options =>
9 //<!--     options.UseSqlServer(builder.Configuration.GetConnectionString("MyDbConnection")));
10 //<!--     builder.Services.AddDistributedMemoryCache();
11 //<!-- }
12 //<!-- else
13 //<!-- {
14 //<!--     builder.Services.AddDbContext<MyDbContext>(options =>
15 //<!--     options.UseSqlServer(builder.Configuration.GetConnectionString("AZURE_SQL_CONNECTIONSTRING")));
16 //<!--     builder.Services.AddStackExchangeRedisCache(options =>
17 //<!-- {
18 //<!--     options.Configuration = builder.Configuration["AZURE_REDIS_CONNECTIONSTRING"];
19 //<!--     options.InstanceName = "SampleInstance";
20 //<!-- });
21 //<!-- }
22 //<!--
23 //<!-- Add services to the container.
24 //<!-- builder.Services.AddControllersWithViews();
25 //<!--
26 //<!-- Add App Service logging
27 //<!-- builder.Logging.AddAzureWebAppDiagnostics();
28 //<!--
29 //<!-- var app = builder.Build();
30 //<!--
31 //<!-- Configure the HTTP request pipeline.
32 //<!-- if (!app.Environment.IsDevelopment())

```

Step 5 (Option 1: with GitHub Copilot):

1. Open `.github/workflows/starter-no-infra_msdocs-core-sql-XYZ` in the explorer. This file was created by the App Service create wizard.
2. Highlight the `dotnet publish` step and select .
3. Ask Copilot, "Install dotnet ef, then create a migrations bundle in the same output folder."
4. If the suggestion is acceptable, select **Accept**. GitHub Copilot doesn't give you the same response every time, and it's not always correct. You might need to ask more questions to fine-tune its response. For tips, see [What can I do with GitHub Copilot in my codespace?](#).



The screenshot shows the GitHub Codespace interface with the workflow editor open. The workflow file is named `starter-no-infra_msdocs-core-sql-251.yml`. A tooltip from GitHub Copilot appears over the `dotnet publish` step, suggesting to "Install dotnet ef, then create a migrations bundle called 'migrationsbundle' in the same output folder". The tooltip has a red border around the text. The GitHub Copilot interface shows "Made changes." and buttons for "Accept" (highlighted in blue), "Discard", and "Accept or discard 1 change".

```

.starter-no-infra_msdocs-core-sql-251.yml
  jobs:
    build:
      steps:
        - uses: actions/checkout@v4
        - name: Set up .NET Core
          uses: actions/setup-dotnet@v1
          with:
            dotnet-version: '8.x'
            include-prerelease: true
        - name: Build with dotnet
          run: dotnet build --configuration Release
        - name: dotnet publish
          run: |
            dotnet publish -c Release -o ${env.DOTNET_ROOT}/myapp
            dotnet tool install --global dotnet-ef
            dotnet ef migrations bundle --output-dir ${env.DOTNET_ROOT}/myapp/migrationsbundle
      - name: Upload artifact for deployment job
        uses: actions/upload-artifact@v3
        with:
          name: .net-app
          path: ${env.DOTNET_ROOT}/myapp
    deploy:
      runs-on: ubuntu-latest
      needs: build
      environment:
        name: 'Production'

```

Step 5 (Option 2: without GitHub Copilot):

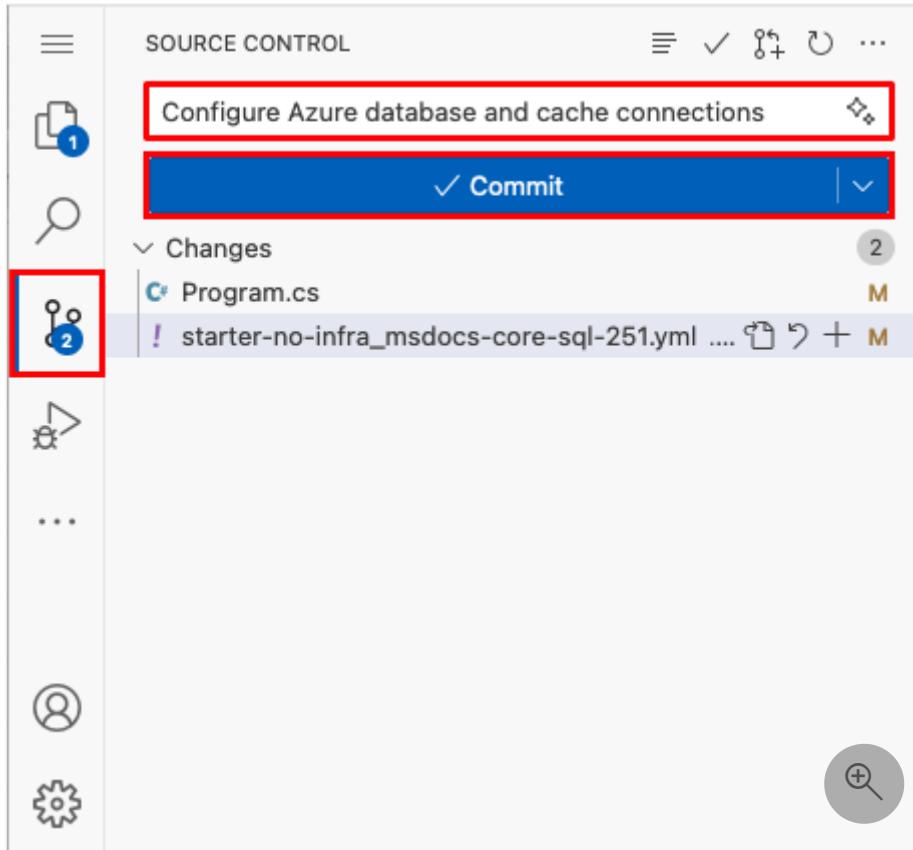
1. Open `.github/workflows/starter-no-infra_msdocs-core-sql-XYZ` in the explorer. This file was created by the App Service create wizard.
2. Under the `dotnet publish` step, add a step to install the [Entity Framework Core tool](#) with the command `dotnet tool install -g dotnet-ef --version 8.*`.
3. Under the new step, add another step to generate a database [migration bundle](#) in the deployment package: `dotnet ef migrations bundle --runtime linux-x64 -o ${env.DOTNET_ROOT}/myapp/migrationsbundle`. The migration bundle is a self-contained executable that you can run in the production environment without needing the .NET SDK. The App Service linux container only has the .NET runtime and not the .NET SDK.

```

    .github > workflows > ! starter-no-infra_msdocs-core-sql-251.yml
    1 # Docs for the Azure Web Apps Deploy action: https://github.com/Azure/webapps-deploy
    2 # More GitHub Actions for Azure: https://github.com/Azure/actions
    3
    4 name: Build and deploy ASP.NET Core app to Azure Web App - msdocs-core-sql-251
    5
    6 on:
    7   push:
    8     branches:
    9       - starter-no-infra
    10  workflow_dispatch:
    11
    12 jobs:
    13   build:
    14     runs-on: ubuntu-latest
    15
    16   steps:
    17     - uses: actions/checkout@v4
    18
    19     - name: Set up .NET Core
    20       uses: actions/setup-dotnet@v1
    21       with:
    22         dotnet-version: '8.x'
    23         include-prerelease: true
    24
    25     - name: Build with dotnet
    26       run: dotnet build --configuration Release
    27
    28     - name: dotnet publish
    29       run: dotnet publish -c Release -o ${env.DOTNET_ROOT}/myapp
    30
    31     - name: Install dotnet ef
    32       run: dotnet tool install --global dotnet-ef --version 8.+
    33
    34     - name: Create migrations bundle
    35       run: dotnet ef migrations bundle --runtime linux-x64 -o ${env.DOTNET_ROOT}/myapp/migrationsbundle
    36
    37     - name: Upload artifact for deployment job
    38       uses: actions/upload-artifact@v3
    39       with:
  
```

Step 6:

1. Select the **Source Control** extension.
2. In the textbox, type a commit message like `Configure Azure database and cache connections`. Or, select and let GitHub Copilot generate a commit message for you.
3. Select **Commit**, then confirm with **Yes**.
4. Select **Sync changes 1**, then confirm with **OK**.



Step 7: Back in the Deployment Center page in the Azure portal:

1. Select the **Logs** tab, then select **Refresh** to see the new deployment run.
2. In the log item for the deployment run, select the **Build/Deploy Logs** entry with the latest timestamp.

The screenshot shows the Azure Deployment Center page with the 'Logs' tab selected. The top navigation bar includes Save, Discard, Browse, Manage publish profile, Sync, and Leave Feedback. Below the tabs, there are Refresh and Delete buttons. The main area displays deployment logs for Friday, June 28, 2024. The logs table has columns: Time, Commit ID, Log Type, Commit Author, Status, and Message. One log entry is highlighted with a red box around its 'Log Type' column, which shows 'Build/Deploy Lo...'. The log details are as follows:

Time	Commit ID	Log Type	Commit Author	Status	Message
06/28/2024, 5:44...	016986b	Build/Deploy Lo...	Cephas Lin	In Progress...	Configure Azure database and cache connections
06/28/2024, 5:25...	a5858da	App Logs	N/A	Success (Active)	Add or update the Azure App Service build and deployment workflow config
06/28/2024, 5:23...	ee8db3d	Build/Deploy Lo...	Cephas Lin	Success	Add or update the Azure App Service build and deployment workflow config

Step 8: You're taken to your GitHub repository and see that the GitHub action is running. The workflow file defines two separate stages, build and deploy. Wait for the

GitHub run to show a status of Success. It takes about 5 minutes.

The screenshot shows a GitHub Actions run summary for a repository named 'msdocs-app-service-sqldb-dotnetcore'. The run was triggered via push 6 minutes ago by '`<github alias>`' pushing commit `9ab82de`. The status is **Success**. The total duration was **3m 17s**, and there was 1 artifact. The workflow file is `starter-no-infra_msdocs-core-sql-251.yml`, which triggers on push. It consists of two steps: **build** (1m 55s) and **deploy** (1m 0s) to `msdocs-core-sql-251.azurewebsites.net`. The build step is green, indicating success. The deploy step is also green, indicating success. There are buttons for 'Re-run all jobs' and '...' at the top right of the summary card.

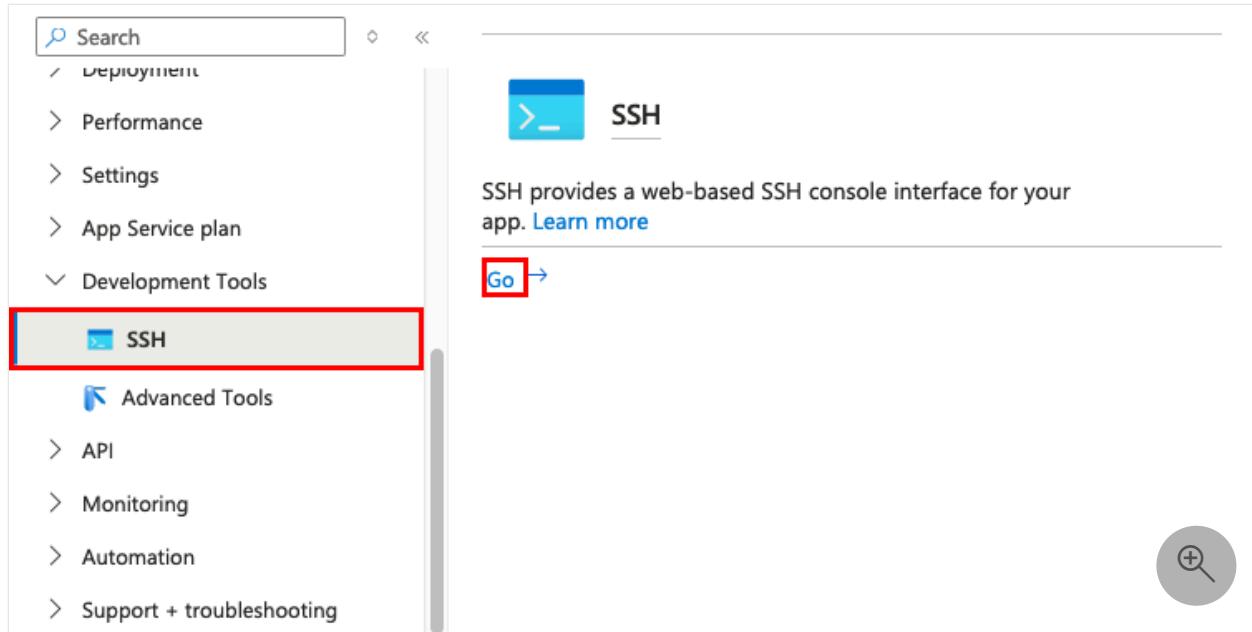
Having issues? Check the [Troubleshooting section](#).

5. Generate database schema

With the SQL Database protected by the virtual network, the easiest way to run [dotnet database migrations](#) is in an SSH session with the Linux container in App Service.

Step 1: Back in the App Service page, in the left menu,

1. Select **Development Tools > SSH**.
2. Select **Go**.



Step 2: In the SSH session:

1. Run `cd /home/site/wwwroot`. Here are all your deployed files.
2. Run the migration bundle that the GitHub workflow generated, with the command `./migrationsbundle -- --environment Production`. If it succeeds, App Service is connecting successfully to the SQL Database. Remember that `--environment Production` corresponds to the code changes you made in *Program.cs*.

```

APP SERVICE ON LINUX

Documentation: http://aka.ms/webapp-linux
Dotnet quickstart: https://aka.ms/dotnet-qs
ASP .NETCore Version: 8.0.3
Note: Any data outside '/home' is not persisted
root@msdocs-cor_263242abe8:~/site/wwwroot# cd /home/site/wwwroot
root@msdocs-cor_263242abe8:~/site/wwwroot# ./migrationsbundle -- --environment Production
info: Microsoft.EntityFrameworkCore.Database.Command[20101]
    Executed DbCommand (68ms) [Parameters=[], CommandType='Text', CommandTimeout='30']
        SELECT 1
info: Microsoft.EntityFrameworkCore.Database.Command[20101]
    Executed DbCommand (62ms) [Parameters=[], CommandType='Text', CommandTimeout='30']
        SELECT OBJECT_ID(N'[__EFMigrationsHistory']");
info: Microsoft.EntityFrameworkCore.Database.Command[20101]
    Executed DbCommand (8ms) [Parameters=[], CommandType='Text', CommandTimeout='30']
        SELECT 1
info: Microsoft.EntityFrameworkCore.Database.Command[20101]
    Executed DbCommand (17ms) [Parameters=[], CommandType='Text', CommandTimeout='30']
        CREATE TABLE [__EFMigrationsHistory] (
            [MigrationId] nvarchar(150) NOT NULL,
            [ProductVersion] nvarchar(32) NOT NULL,
            CONSTRAINT [PK__EFMigrationsHistory] PRIMARY KEY ([MigrationId])
        );
info: Microsoft.EntityFrameworkCore.Database.Command[20101]
    Executed DbCommand (8ms) [Parameters=[], CommandType='Text', CommandTimeout='30']
        SELECT 1
info: Microsoft.EntityFrameworkCore.Database.Command[20101]
    Executed DbCommand (10ms) [Parameters=[], CommandType='Text', CommandTimeout='30']
        SELECT OBJECT_ID(N'[__EFMigrationsHistory"]');
info: Microsoft.EntityFrameworkCore.Database.Command[20101]
    Executed DbCommand (21ms) [Parameters=[], CommandType='Text', CommandTimeout='30']
        SELECT [MigrationId], [ProductVersion]
        FROM [__EFMigrationsHistory]
        ORDER BY [MigrationId];
info: Microsoft.EntityFrameworkCore.Migrations[20402]
    Applying migration '20240621154946_InitialCreate'.
Applying migration '20240621154946_InitialCreate'.
info: Microsoft.EntityFrameworkCore.Database.Command[20101]
    Executed DbCommand (7ms) [Parameters=[], CommandType='Text', CommandTimeout='30']
        CREATE TABLE [Todo] (
            [ID] int NOT NULL IDENTITY,
            [Description] nvarchar(max) NULL,
            [CreatedDate] datetime2 NOT NULL,
            CONSTRAINT [PK_Todo] PRIMARY KEY ([ID])
        );
info: Microsoft.EntityFrameworkCore.Database.Command[20101]
    Executed DbCommand (9ms) [Parameters=[], CommandType='Text', CommandTimeout='30']
        INSERT INTO [__EFMigrationsHistory] ([MigrationId], [ProductVersion])
        VALUES (N'20240621154946_InitialCreate', N'8.0.6');
Done.
root@msdocs-cor_263242abe8:~/site/wwwroot# []

```



☰ Menu | ssh://root@ | SSH CONNECTION ESTABLISHED |

In the SSH session, only changes to files in `/home` can persist beyond app restarts.
Changes outside of `/home` aren't persisted.

Having issues? Check the [Troubleshooting section](#).

6. Browse to the app

Step 1: In the App Service page:

1. From the left menu, select **Overview**.
2. Select the URL of your app.

The screenshot shows the Azure portal's Overview page for an App Service. The left sidebar lists various monitoring and management links. The main pane displays the app's configuration, including its resource group (msdocs-core-sql-tutorial), status (Running), location (East US), subscription (Visual Studio Enterprise Subscription), and other details like the App Service Plan (msdocs-core-sql-251-plan) and Operating System (Linux). The Default domain is listed as msdocs-core-sql-251.azurewebsites.net, which is highlighted with a red box. A JSON View link is also present in the top right.

Step 2: Add a few tasks to the list. Congratulations, you're running a web app in Azure App Service, with secure connectivity to Azure SQL Database.

The screenshot shows the application's index page. At the top, there is a navigation bar with the application name "DotNetCoreSqlDb" and links for "Home" and "Privacy". Below the navigation bar is the main content area with a title "Index" and a "Create New" button. The main content area contains a table with three rows, each representing a task. The columns are "Description" and "Created Date". The tasks listed are "Deploy app to App Service" (Created on 2023-05-17), "Walk dog" (Created on 2023-05-17), and "Feed cats" (Created on 2023-05-17). Each row has "Edit | Details | Delete" links. Below the table, the text "Processing Time: 00:00:00.0498600" is displayed. At the bottom of the page, there is a copyright notice "© 2023 - DotNetCoreSqlDb - [Privacy](#)". A search icon is located in the bottom right corner of the content area.

💡 Tip

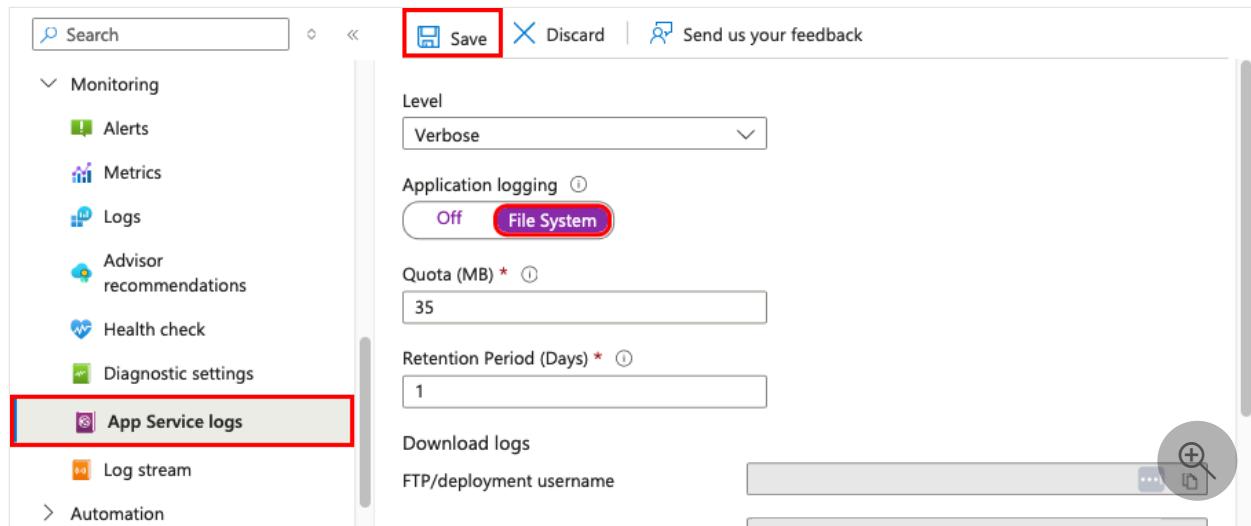
The sample application implements the [cache-aside](#) pattern. When you visit a data view for the second time, or reload the same page after making data changes, **Processing time** in the webpage shows a much faster time because it's loading the data from the cache instead of the database.

7. Stream diagnostic logs

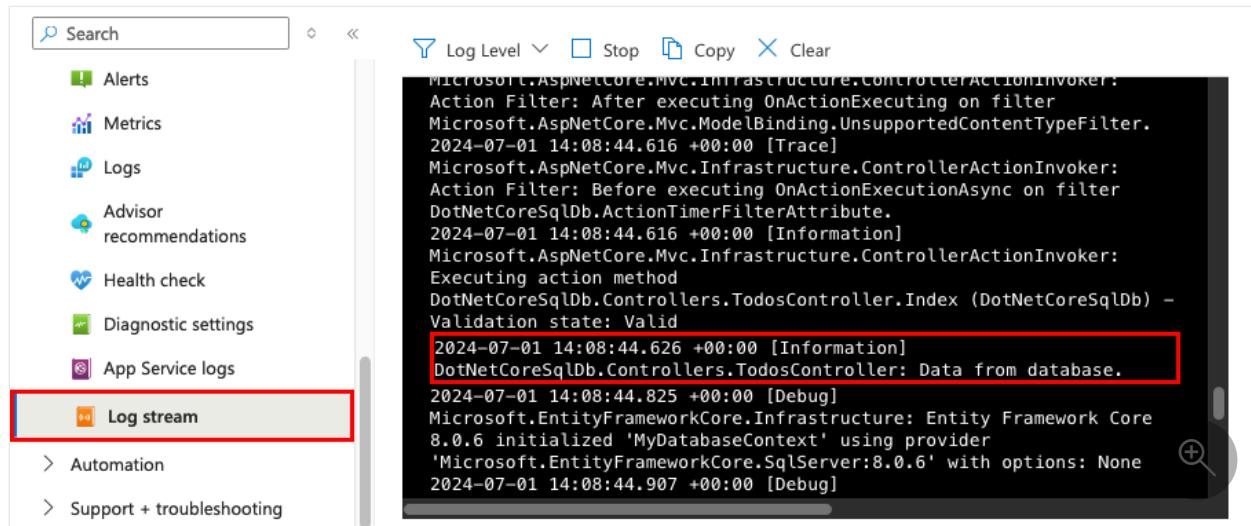
Azure App Service captures all console logs to help you diagnose issues with your application. The sample app includes logging code in each of its endpoints to demonstrate this capability.

Step 1: In the App Service page:

1. From the left menu, select **Monitoring > App Service logs**.
2. Under **Application logging**, select **File System**.
3. In the top menu, select **Save**.



Step 2: From the left menu, select **Log stream**. You see the logs for your app, including platform logs and logs from inside the container.



8. Clean up resources

When you're finished, you can delete all of the resources from your Azure subscription by deleting the resource group.

Step 1: In the search bar at the top of the Azure portal:

1. Enter the resource group name.

2. Select the resource group.

The screenshot shows the Microsoft Azure portal interface. At the top, there is a search bar with the text "msdocs-core-sql-tutorial" and a red box highlighting it. Below the search bar, there are several navigation links: "All" (highlighted), "Resource Groups (1)", "Documentation (99+)", "Services (0)", "Resources (0)", "Marketplace (0)", and "Azure Active Directory (0)". On the left sidebar, under "Azure services", there are icons for "Create a resource", "App Service Domains", and "Virtual machines". Under "Resources", there are "Recent" and "Favorite" sections. In the center, a "Resource Groups" section lists one item: "msdocs-core-sql-tutorial" (highlighted with a red box). Below this, there is a "Documentation" section with a link to "See all". At the bottom of the search results, there is a note: "Continue searching in Azure Active Directory" and a "Give feedback" link. A "Give feedback" button is also located in the bottom right corner of the search results area.

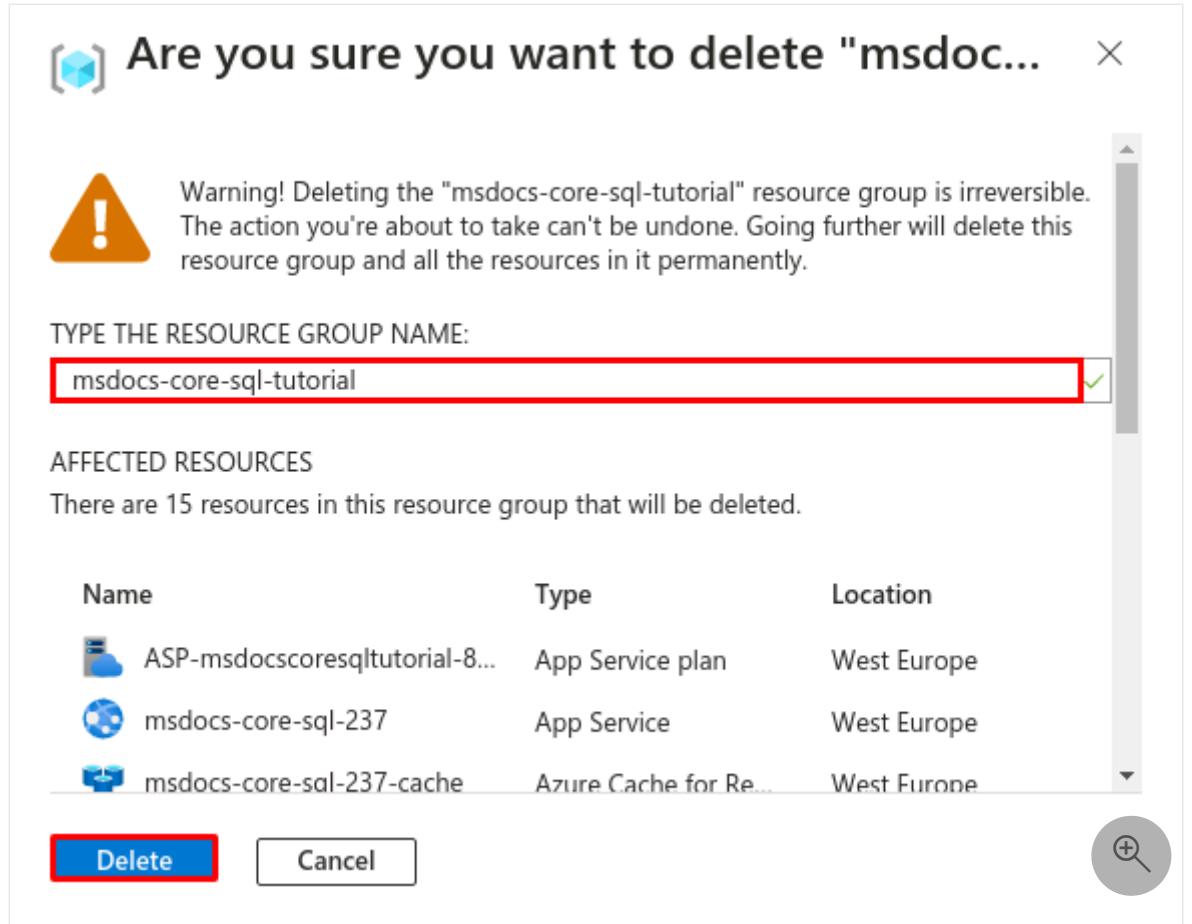
Step 2: In the resource group page, select **Delete resource group**.

The screenshot shows the "msdocs-core-sql-tutorial" resource group page. At the top, there is a toolbar with "Create", "Manage view" (with a dropdown menu), "Delete resource group" (highlighted with a red box), and "..." buttons. Below the toolbar, there is a section titled "Essentials" with a "JSON View" button. Under "Resources", there is a "Recommendations (1)" section. Below that, there is a search bar with "Filter for any field...", an "Add filter" button, and a "More (2)" button. There are also grouping and sorting options: "No grouping" and "List view". The main table lists three resources: "ASP-msdocscoresqltutori..." (App Service plan, West Europe), "msdocs-core-sql-237" (App Service, West Europe), and "msdocs-core-sql-237-cac..." (Azure Cache for Redis, West Europe). Each resource has a checkbox next to its name. A magnifying glass icon is located in the bottom right corner of the table area.

Step 3:

1. Enter the resource group name to confirm your deletion.

2. Select **Delete**.



Troubleshooting

- The portal deployment view for Azure SQL Database shows a Conflict status
- In the Azure portal, the log stream UI for the web app shows network errors
- The SSH session in the browser shows SSH CONN CLOSED
- The portal log stream page shows Connected! but no logs

The portal deployment view for Azure SQL Database shows a Conflict status

Depending on your subscription and the region you select, you might see the deployment status for Azure SQL Database to be `conflict`, with the following message in Operation details:

```
Location '<region>' is not accepting creation of new Windows Azure SQL Database servers at this time.
```

This error is most likely caused by a limit on your subscription for the region you select. Try choosing a different region for your deployment.

In the Azure portal, the log stream UI for the web app shows network errors

You might see this error:

Unable to open a connection to your app. This may be due to any network security groups or IP restriction rules that you have placed on your app. To use log streaming, please make sure you are able to access your app directly from your current network.

This is usually a transient error when the app is first started. Wait a few minutes and check again.

The SSH session in the browser shows **SSH CONN CLOSED**

It takes a few minutes for the Linux container to start up. Wait a few minutes and check again.

The portal log stream page shows **Connected!** but no logs

After you configure diagnostic logs, the app is restarted. You might need to refresh the page for the changes to take effect in the browser.

Frequently asked questions

- How much does this setup cost?
- How do I connect to the Azure SQL Database server that's secured behind the virtual network with other tools?
- How does local app development work with GitHub Actions?
- How do I debug errors during the GitHub Actions deployment?
- How do I change the SQL Database connection to use a managed identity instead?
- I don't have permissions to create a user-assigned identity
- What can I do with GitHub Copilot in my codespace?

How much does this setup cost?

Pricing for the created resources is as follows:

- The App Service plan is created in **Basic** tier and can be scaled up or down. See [App Service pricing ↗](#).

- The Azure SQL Database is created in general-purpose, serverless tier on Standard-series hardware with the minimum cores. There's a small cost and can be distributed to other regions. You can minimize cost even more by reducing its maximum size, or you can scale it up by adjusting the serving tier, compute tier, hardware configuration, number of cores, database size, and zone redundancy. See [Azure SQL Database pricing](#).
- The Azure Cache for Redis is created in **Basic** tier with the minimum cache size. There's a small cost associated with this tier. You can scale it up to higher performance tiers for higher availability, clustering, and other features. See [Azure Cache for Redis pricing](#).
- The virtual network doesn't incur a charge unless you configure extra functionality, such as peering. See [Azure Virtual Network pricing](#).
- The private DNS zone incurs a small charge. See [Azure DNS pricing](#).

How do I connect to the Azure SQL Database server that's secured behind the virtual network with other tools?

- For basic access from a command-line tool, you can run `sqlcmd` from the app's SSH terminal. The app's container doesn't come with `sqlcmd`, so you must [install it manually](#). Remember that the installed client doesn't persist across app restarts.
- To connect from a SQL Server Management Studio client or from Visual Studio, your machine must be within the virtual network. For example, it could be an Azure VM that's connected to one of the subnets, or a machine in an on-premises network that has a [site-to-site VPN](#) connection with the Azure virtual network.

How does local app development work with GitHub Actions?

Take the autogenerated workflow file from App Service as an example, each `git push` kicks off a new build and deployment run. From a local clone of the GitHub repository, you make the desired updates push it to GitHub. For example:

terminal

```
git add .
git commit -m "<some-message>"
git push origin main
```

How do I debug errors during the GitHub Actions deployment?

If a step fails in the autogenerated GitHub workflow file, try modifying the failed command to generate more verbose output. For example, you can get more output from any of the `dotnet` commands by adding the `-v` option. Commit and push your changes to trigger another deployment to App Service.

I don't have permissions to create a user-assigned identity

See [Set up GitHub Actions deployment from the Deployment Center](#).

How do I change the SQL Database connection to use a managed identity instead?

The default connection string to the SQL database is managed by Service Connector, with the name `defaultConnector`, and it uses SQL authentication. To replace it with a connection that uses a managed identity, run the following commands in the [cloud shell](#) after replacing the placeholders:

Azure CLI

```
az extension add --name serviceconnector-passwordless --upgrade
az sql server update --enable-public-network true
az webapp connection delete sql --connection defaultConnector --resource-group <group-name> --name <app-name>
az webapp connection create sql --connection defaultConnector --resource-group <group-name> --name <app-name> --target-resource-group <group-name> --server <database-server-name> --database <database-name> --client-type dotnet --system-identity --config-connstr true
az sql server update --enable-public-network false
```

By default, the command `az webapp connection create sql --client-type dotnet --system-identity --config-connstr` does the following:

- Sets your user as the Microsoft Entra ID administrator of the SQL database server.
- Create a system-assigned managed identity and grants it access to the database.
- Generates a passwordless connection string called `AZURE_SQL_CONNECTIONSTRING`, which your app is already using at the end of the tutorial.

Your app should now have connectivity to the SQL database. For more information, see [Tutorial: Connect to Azure databases from App Service without secrets using a managed identity](#).

Tip

Don't want to enable public network connection? You can skip `az sql server update --enable-public-network true` by running the commands from an [Azure cloud shell that's integrated with your virtual network](#) if you have the Owner role assignment on your subscription.

To grant the identity the required access to the database that's secured by the virtual network, `az webapp connection create sql` needs direct connectivity with Entra ID authentication to the database server. By default, the Azure cloud shell doesn't have this access to the network-secured database.

What can I do with GitHub Copilot in my codespace?

You might have noticed that the GitHub Copilot chat view was already there for you when you created the codespace. For your convenience, we include the GitHub Copilot chat extension in the container definition (see `.devcontainer/devcontainer.json`). However, you need a [GitHub Copilot account ↗](#) (30-day free trial available).

A few tips for you when you talk to GitHub Copilot:

- In a single chat session, the questions and answers build on each other and you can adjust your questions to fine-tune the answer you get.
- By default, GitHub Copilot doesn't have access to any file in your repository. To ask questions about a file, open the file in the editor first.
- To let GitHub Copilot have access to all of the files in the repository when preparing its answers, begin your question with `@workspace`. For more information, see [Use the @workspace agent ↗](#).
- In the chat session, GitHub Copilot can suggest changes and (with `@workspace`) even where to make the changes, but it's not allowed to make the changes for you. It's up to you to add the suggested changes and test it.

Here are some other things you can say to fine-tune the answer you get:

- I want this code to run only in production mode.
- I want this code to run only in Azure App Service and not locally.
- The `--output-path` parameter seems to be unsupported.

Related content

Advance to the next tutorial to learn how to secure your app with a custom domain and certificate.

[Secure with custom domain and certificate](#)

Or, check out other resources:

[Tutorial: Connect to SQL Database from App Service without secrets using a managed identity](#)

[Configure ASP.NET Core app](#)

Feedback

Was this page helpful?

 Yes

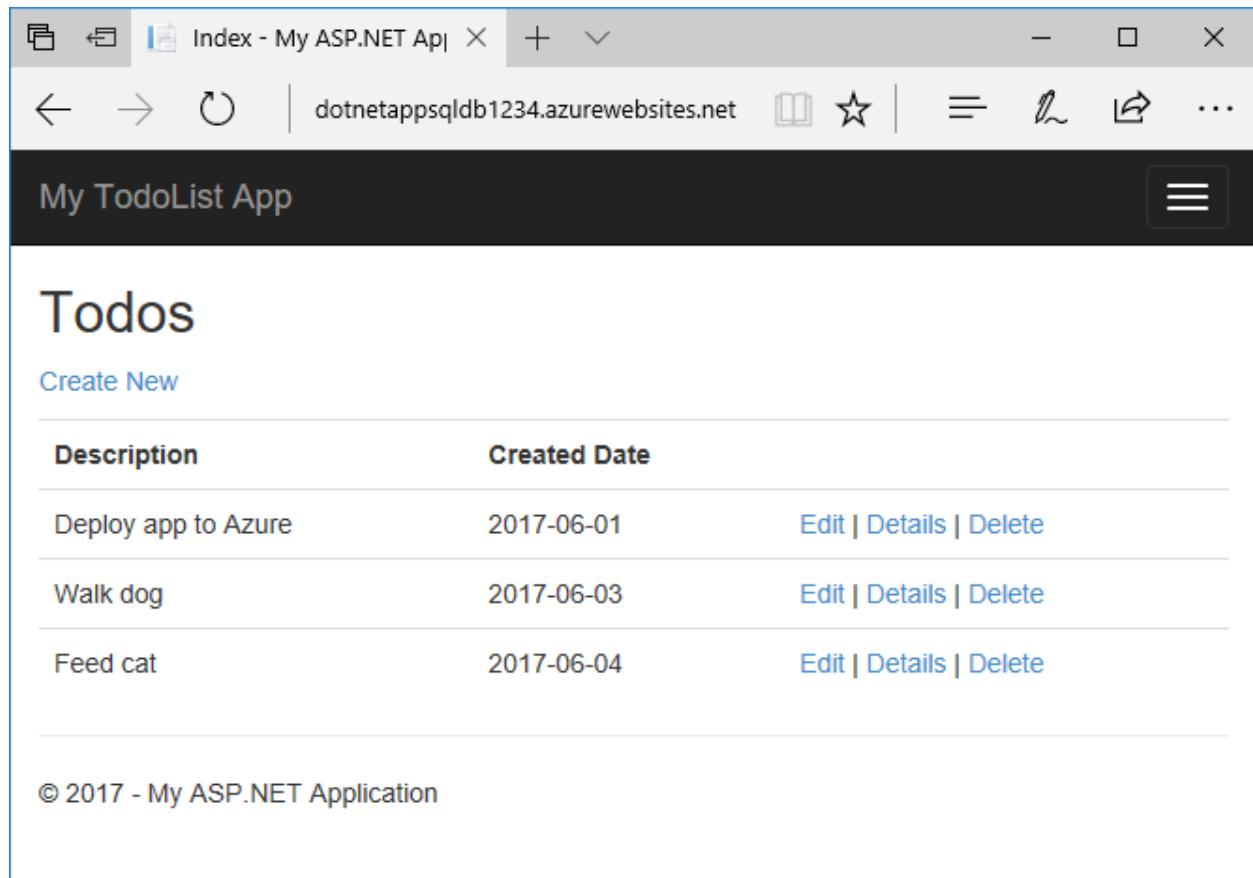
 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

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

Article • 09/21/2022

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

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

Prerequisites

To complete this tutorial:

Install [Visual Studio 2022](#) with the **ASP.NET and web development** and **Azure development** workloads.

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](#).

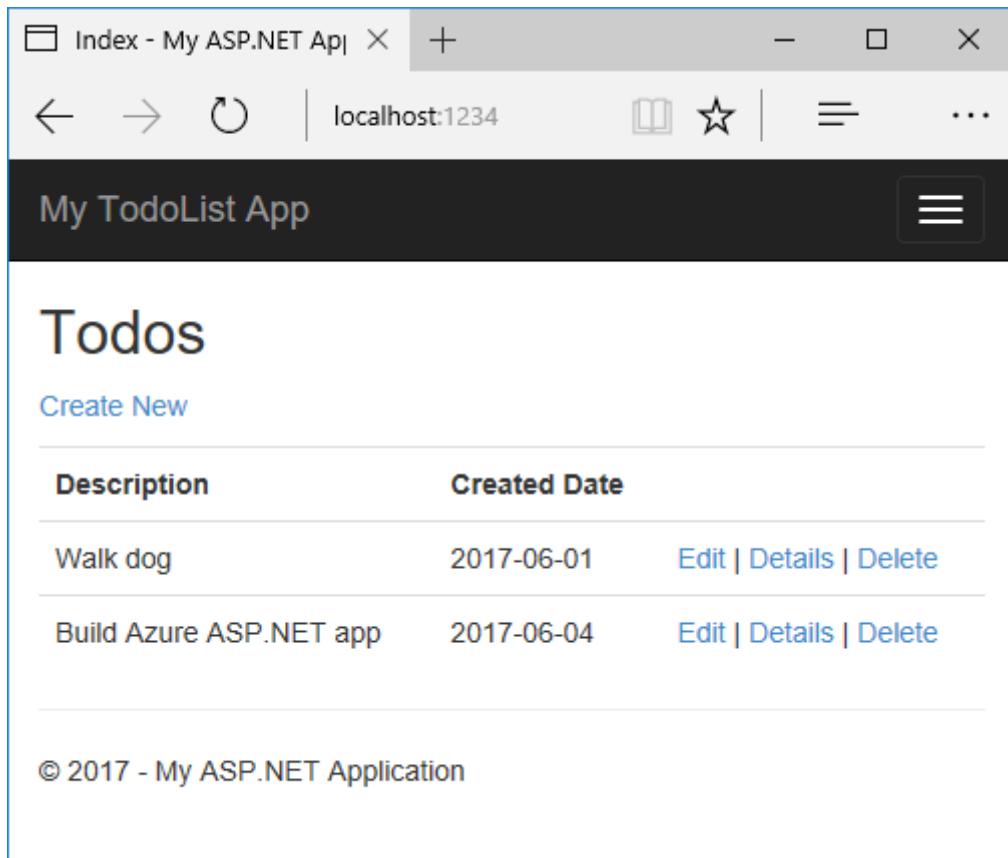
Run the app

1. Open the *dotnet-sqldb-tutorial-master/DotNetAppSqlDb.sln* file in Visual Studio.
2. Type `F5` to run the app. The app is displayed in your default browser.

ⓘ Note

If you only installed Visual Studio and the prerequisites, you may have to [install missing packages via NuGet](#).

3. Select the **Create New** link and create a couple *to-do* items.

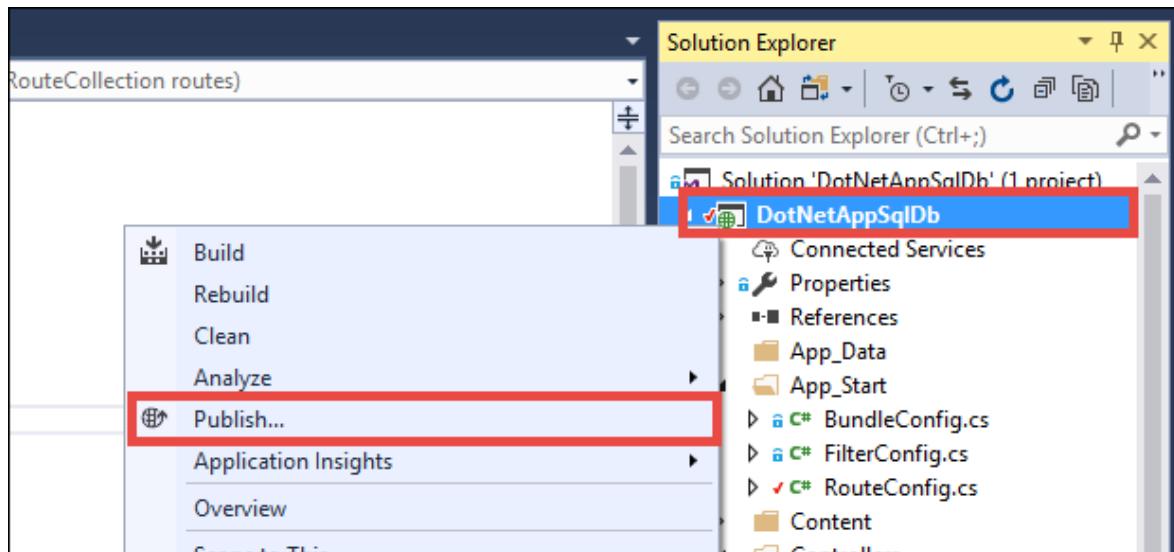


4. Test the **Edit**, **Details**, and **Delete** links.

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

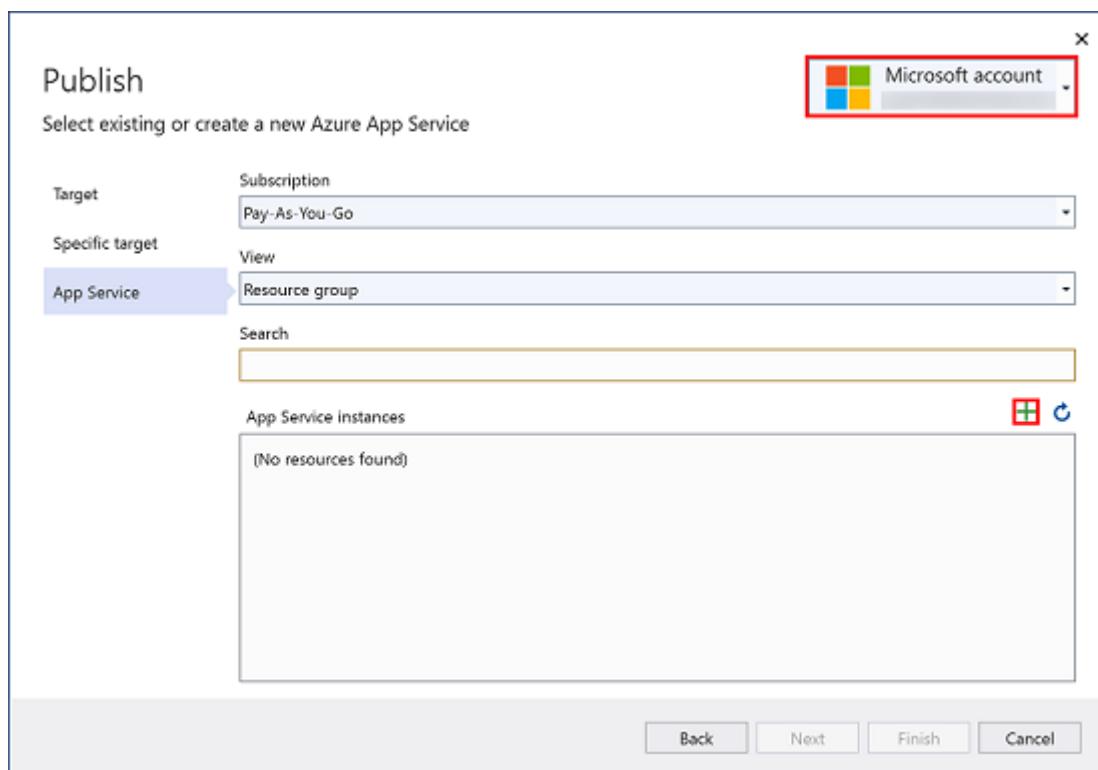
1. In the **Solution Explorer**, right-click your **DotNetAppSqlDb** project and select **Publish**.



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

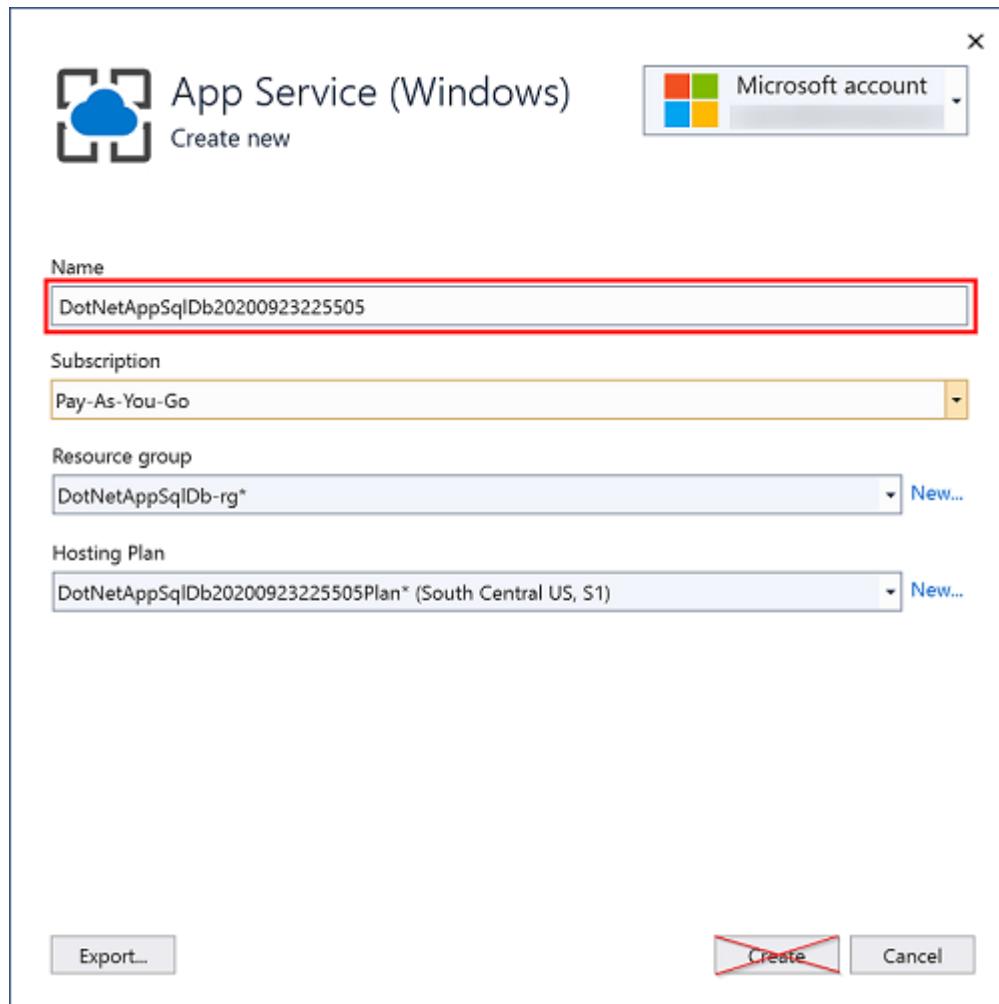


Configure the web app name

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

ⓘ Note

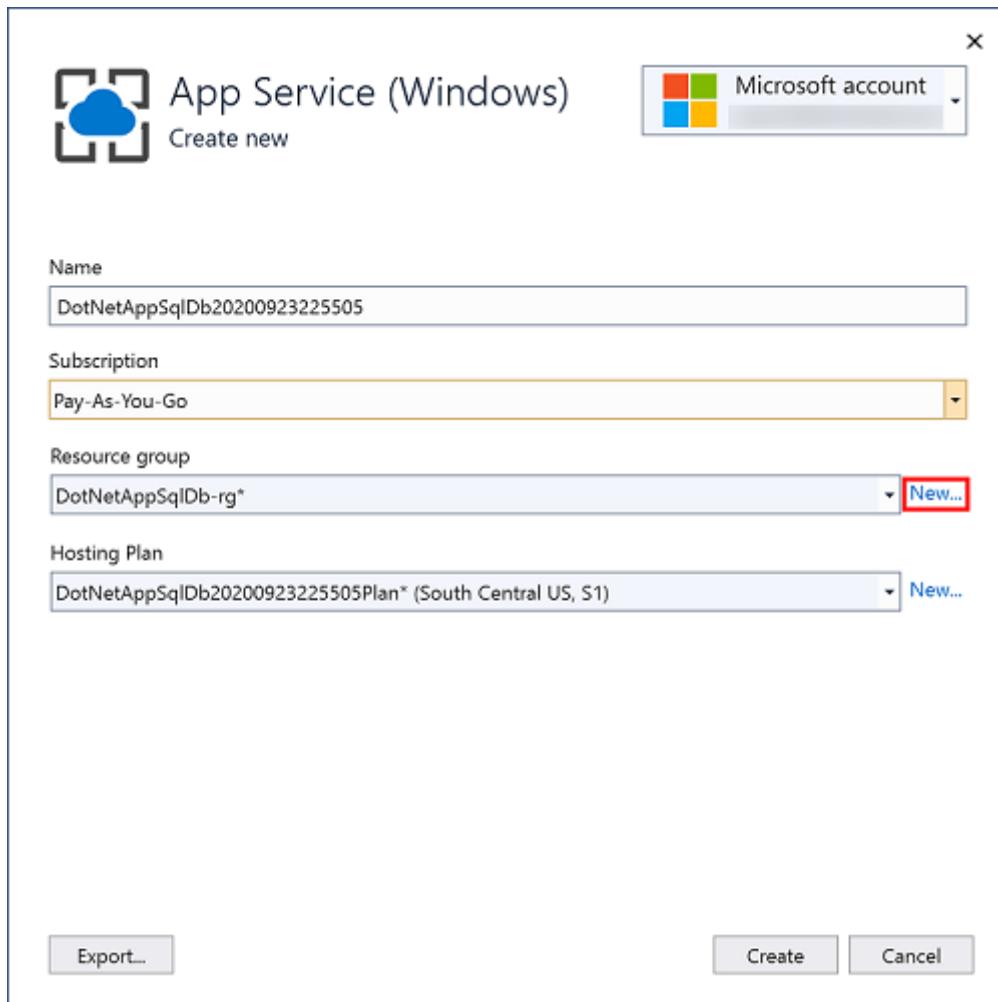
Don't select **Create** yet.



Create a resource group

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

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



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

Create an App Service plan

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

App Service plans define:

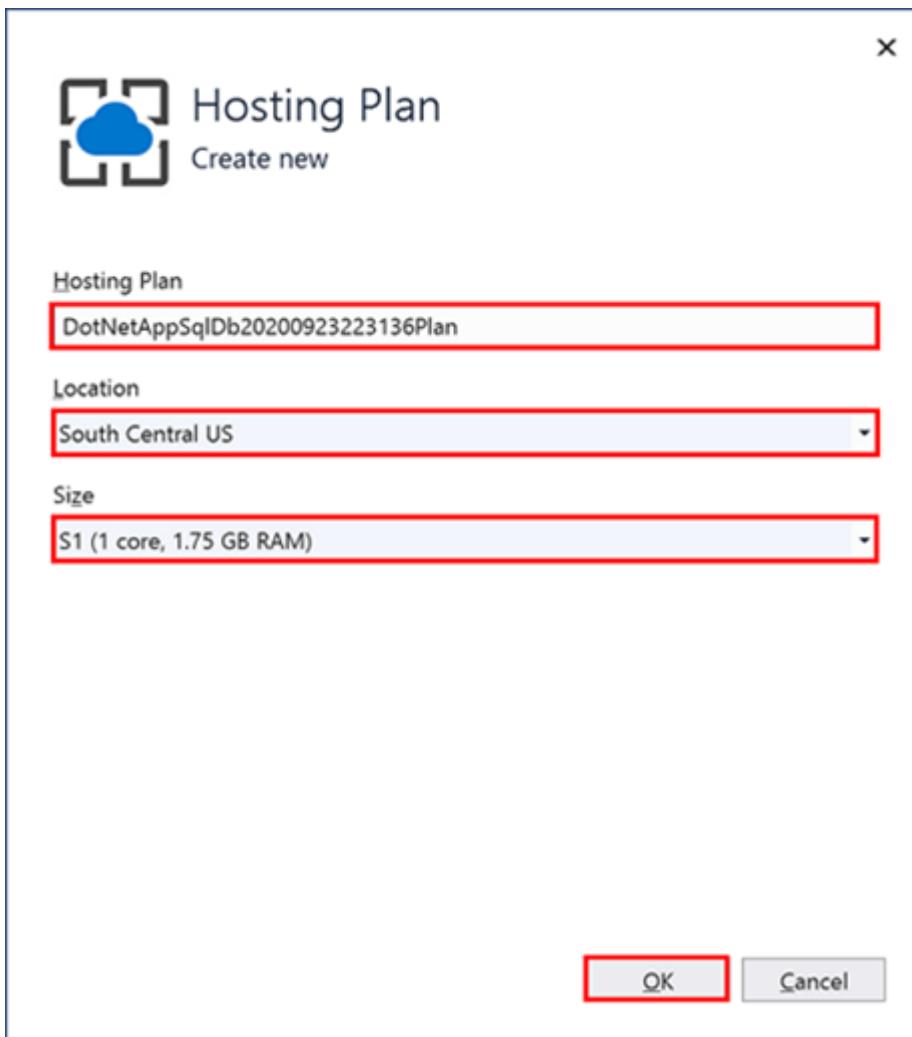
- Region (for example: North Europe, East US, or Southeast Asia)
- Instance size (small, medium, or large)
- Scale count (1 to 20 instances)
- SKU (Free, Shared, Basic, Standard, or Premium)

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

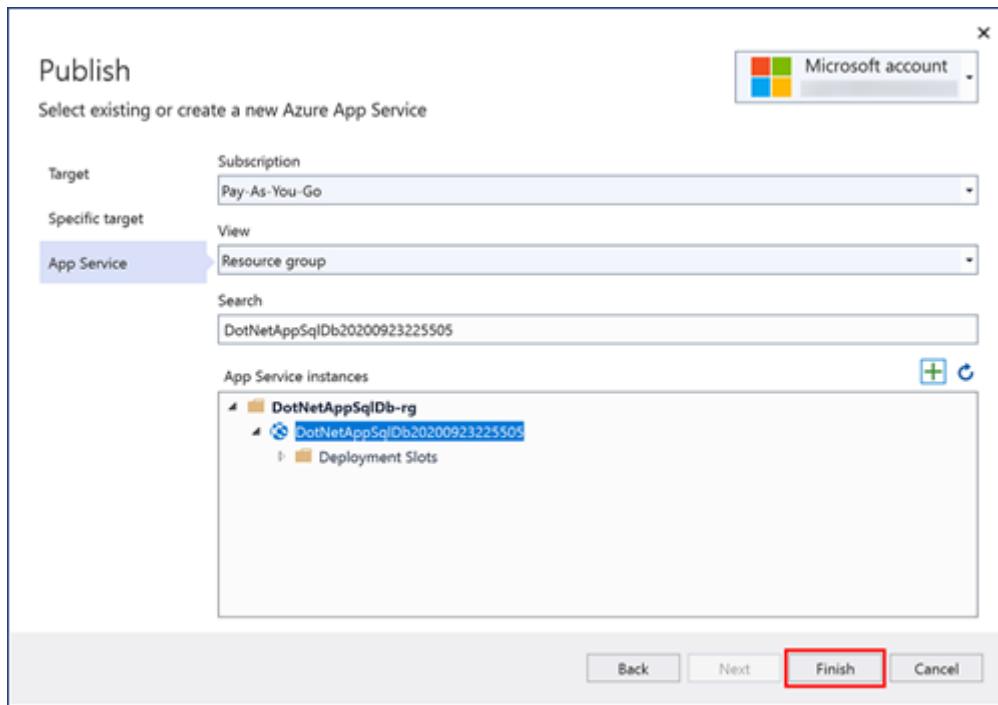
2. In the **Configure App Service Plan** dialog, configure the new App Service plan with the following settings and click **OK**:

Setting	Suggested value	For more information

Setting	Suggested value	For more information
App Service Plan	myAppServicePlan	App Service plans
Location	West Europe	Azure regions
Size	Free	Pricing tiers



3. Click **Create** and wait for the Azure resources to be created.
4. The **Publish** dialog shows the resources you've configured. Click **Finish**.



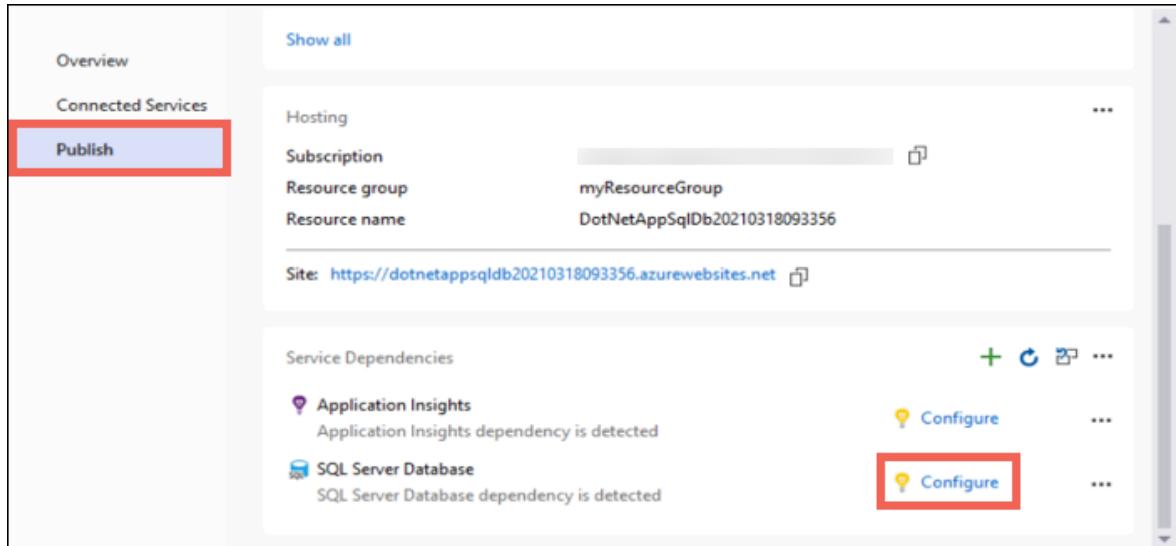
Create a server and database

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

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

ⓘ Note

Be sure to configure the SQL Database from the **Publish** page instead of the **Connected Services** page.



2. Select **Azure SQL Database** and click **Next**.

3. In the **Configure Azure SQL Database** dialog, click **+**.

4. Next to **Database server**, click **New**.

The server name is used as part of the default URL for your server, `<server_name>.database.windows.net`. It must be unique across all servers in Azure SQL. Change the server name to a value you want.

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

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

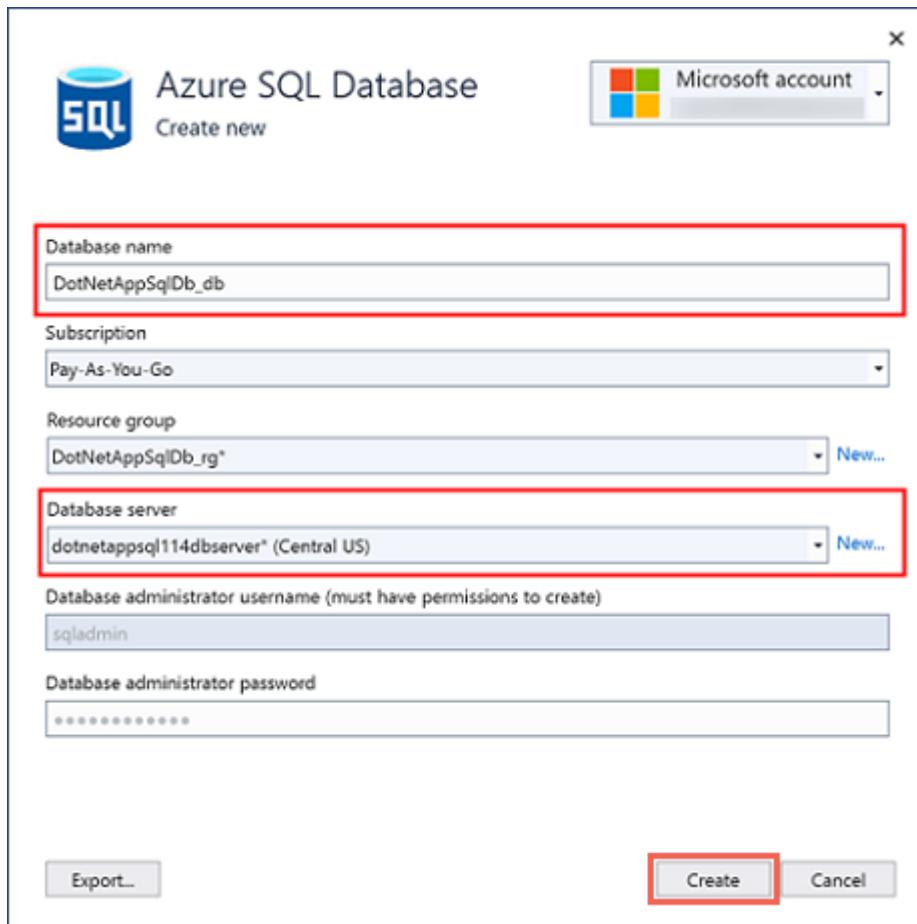


i Important

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

6. Click OK.

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

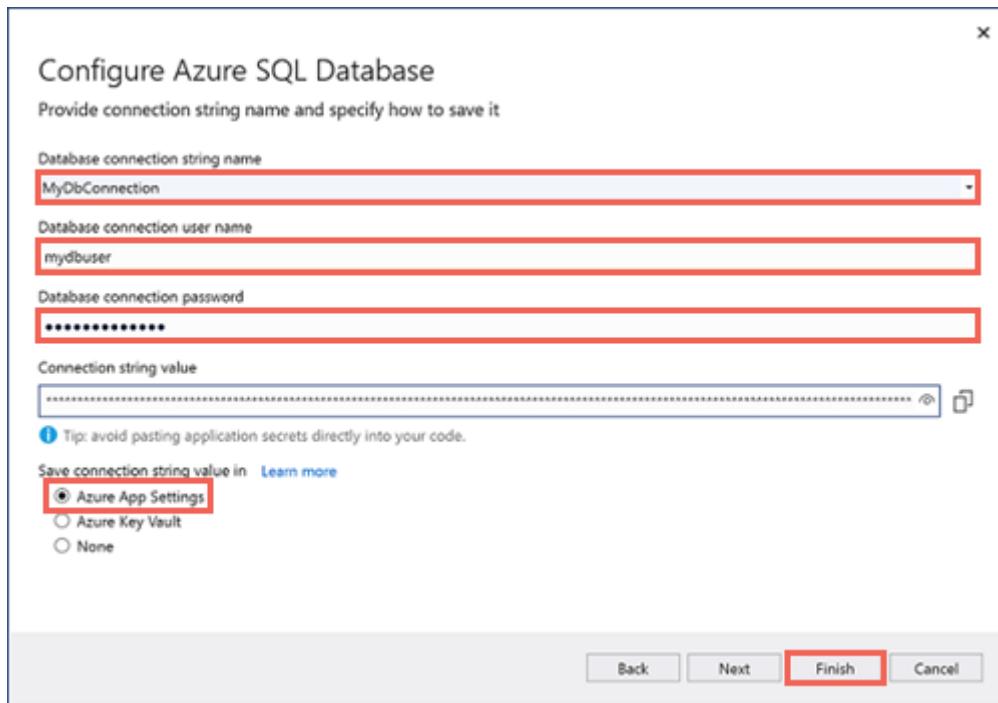


Configure database connection

1. When the wizard finishes creating the database resources, click **Next**.
2. In the **Database connection string Name**, type *MyDbConnection*. This name must match the connection string that is referenced in *Models/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**.

(!) Note

If you see **Local user secrets files** instead, you must have configured SQL Database from the **Connected Services** page instead of the **Publish** page.



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

Deploy your ASP.NET app

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

The screenshot shows a web browser window titled 'Index - My ASP.NET App'. The address bar displays 'dotnetappsql1234.azurewebsites.net'. The page content is titled 'My TodoList App' and shows a table of 'Todos'. The table has columns for 'Description' and 'Created Date'. Three items are listed: 'Deploy app to Azure' (Created 2017-06-01), 'Walk dog' (Created 2017-06-03), and 'Feed cat' (Created 2017-06-04). Each item has 'Edit | Details | Delete' links. At the bottom of the page is a footer with the text '© 2017 - My ASP.NET Application'.

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

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

Access the database locally

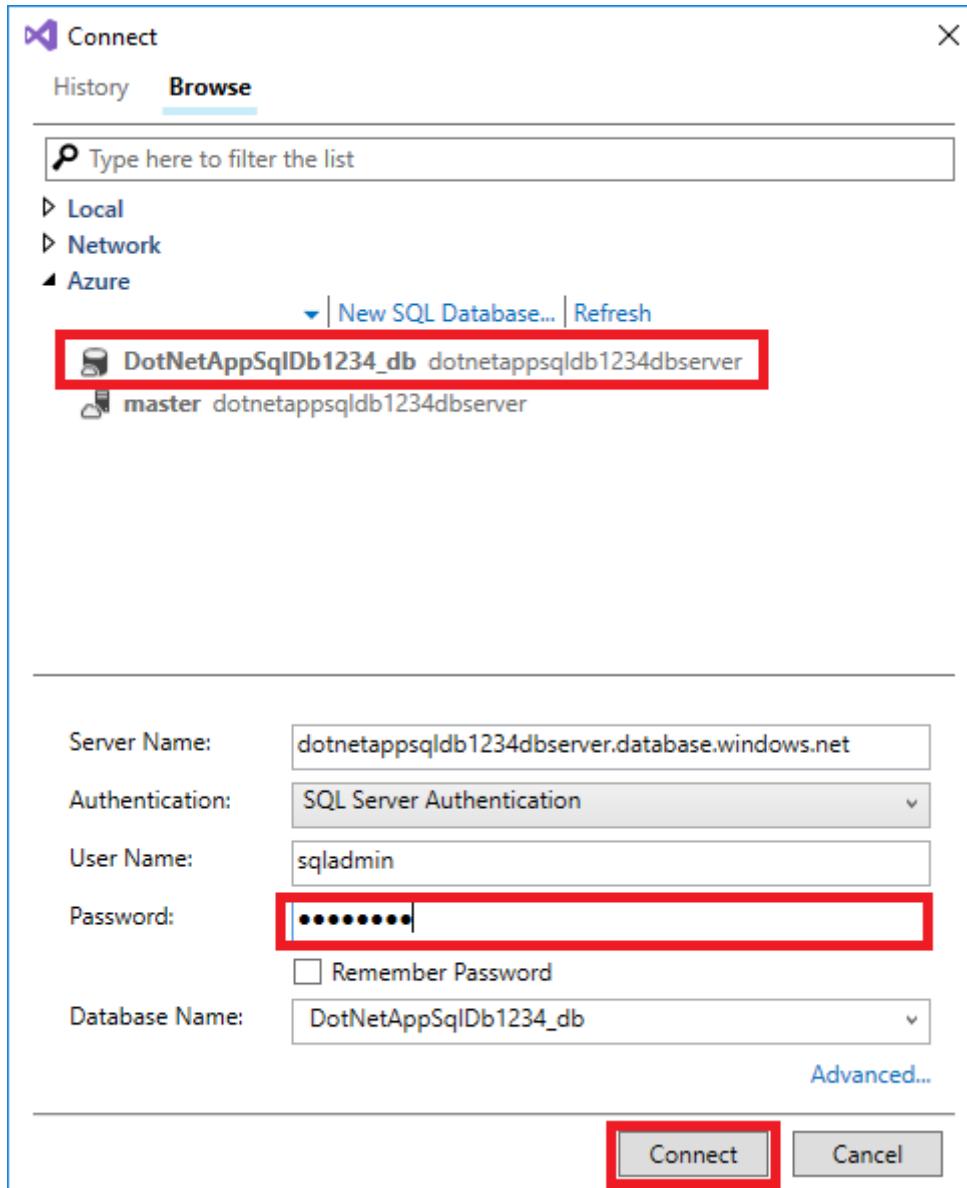
Visual Studio lets you explore and manage your new database in Azure easily in the **SQL Server Object Explorer**. The new database already opened its firewall to the App Service app that you created. But to access it from your local computer (such as from Visual Studio), you must open a firewall for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

Create a database connection

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

Configure the database connection

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

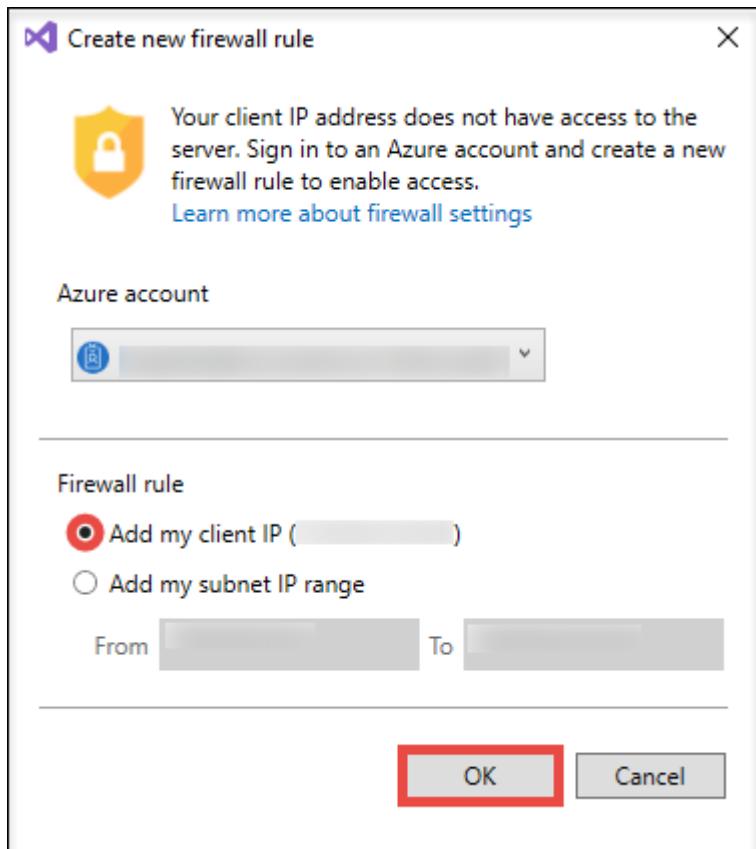


Allow client connection from your computer

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

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

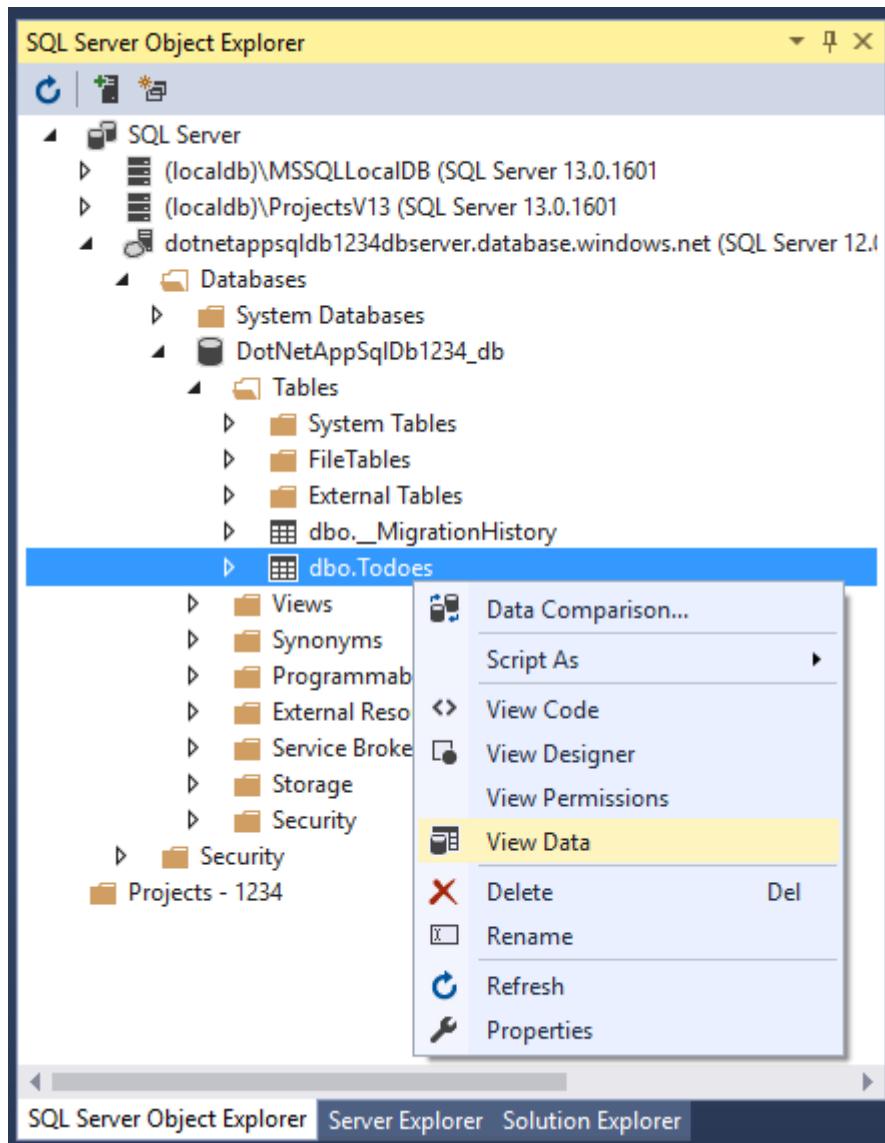
1. Make sure that **Add my client IP** is selected and click **OK**.



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

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

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



Update app with Code First Migrations

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

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

Update your data model

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

C#

```
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:

```
PowerShell
```

```
Enable-Migrations
```

3. Add a migration:

```
PowerShell
```

```
Add-Migration AddProperty
```

4. Update the local database:

```
PowerShell
```

```
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:

```
C#
```

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

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

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

C#

```
<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:

C#

```
<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:

C#

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

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

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

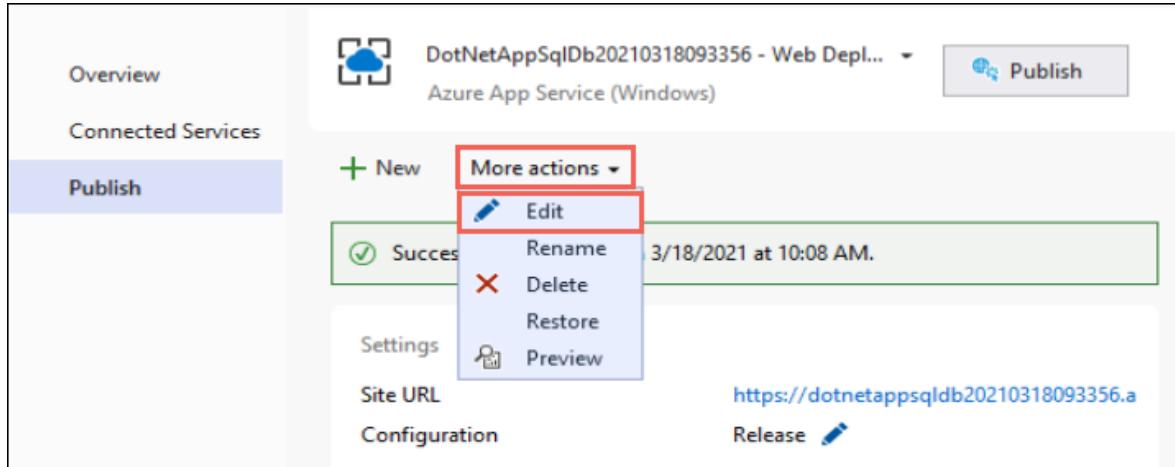
You can now add a to-do item and check **Done**. Then it should show up in your homepage as a completed item. Remember that the **Edit** view doesn't show the **Done** field, because you didn't change the **Edit** view.

Enable Code First Migrations in Azure

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

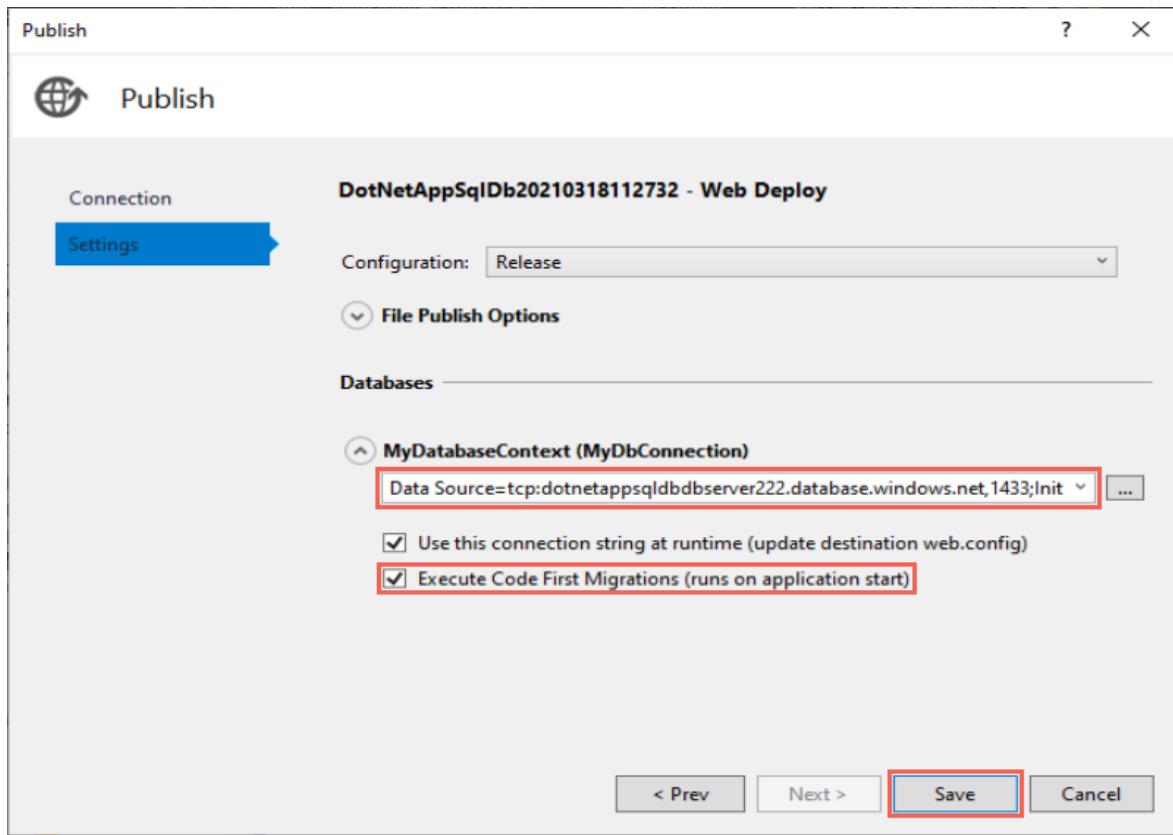
1. Just like before, right-click your project and select **Publish**.

2. Click **More actions > Edit** to open the publish settings.



3. In the **MyDbContext** dropdown, select the database connection for your Azure SQL Database.

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



Publish your changes

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

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

The screenshot shows a web browser window with the title "Index - My ASP.NET App". The address bar shows the URL "dotnetappsqldb1234.azurewebsites.net". The main content area is titled "My TodoList App" and contains a heading "Todos". Below the heading is a table with three rows of todo items. The table has columns for "Description", "Created Date", and "Done". The first item is "Deploy app to Azure" (Created Date: 2017-06-01, Done: checked), the second is "Walk dog" (Created Date: 2017-06-03, Done: unchecked), and the third is "Feed cat" (Created Date: 2017-06-04, Done: unchecked). Each row has "Edit | Details | Delete" links. At the top left of the content area is a "Create New" link. At the bottom is a copyright notice: "© 2017 - My ASP.NET Application".

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

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

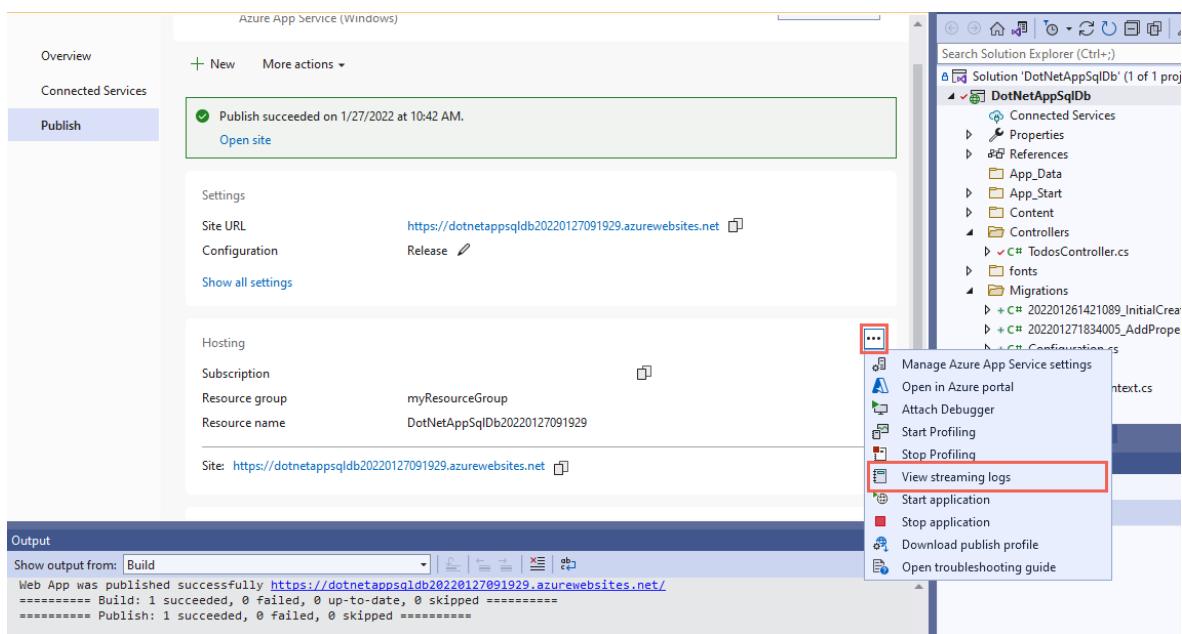
You can stream tracing messages directly from your Azure app to Visual Studio.

Open *Controllers\TodosController.cs*.

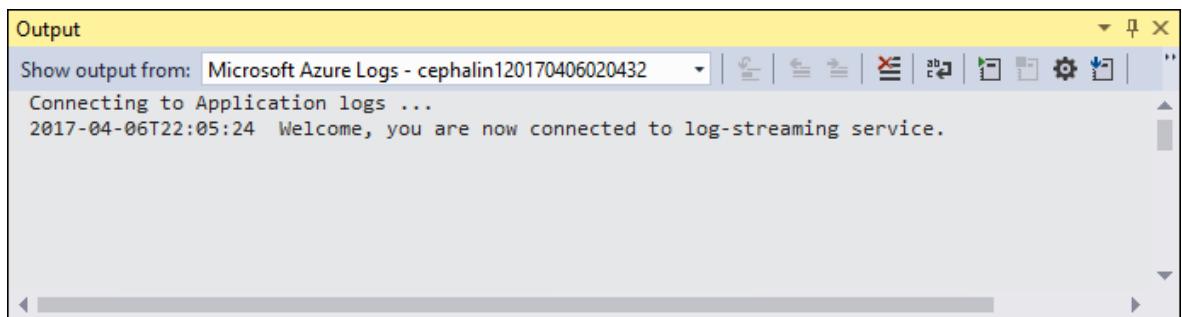
Each action starts with a `Trace.WriteLine()` method. This code is added to show you how to add trace messages to your Azure app.

Enable log streaming

1. In the publish page, scroll down to the **Hosting** section.
2. At the right-hand corner, click ... > **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

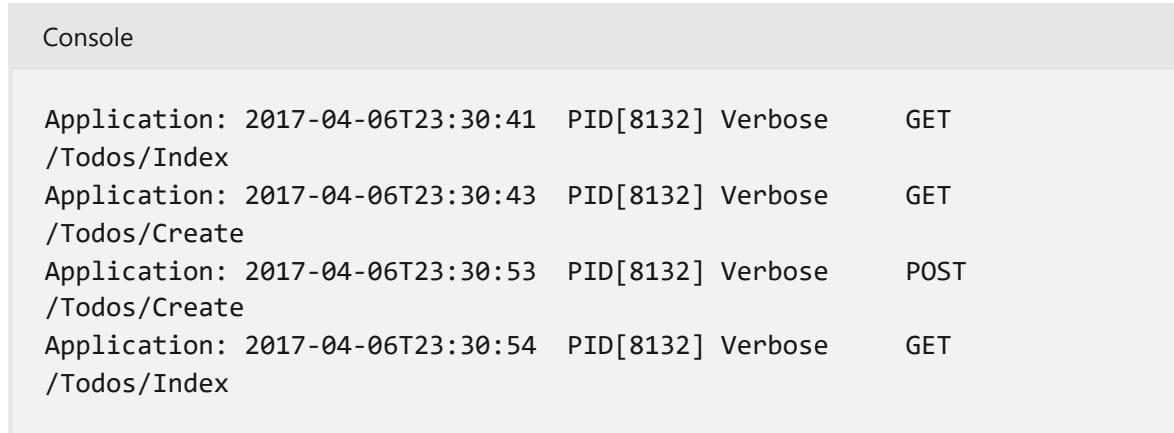
1. To change the trace levels to output other trace messages, go back to the publish page.
2. In the Hosting section, click ... > **Open in Azure portal**.
3. In the portal management page for your app, from the left menu, select **App Service logs**.
4. Under **Application Logging (File System)**, select **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()`.

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

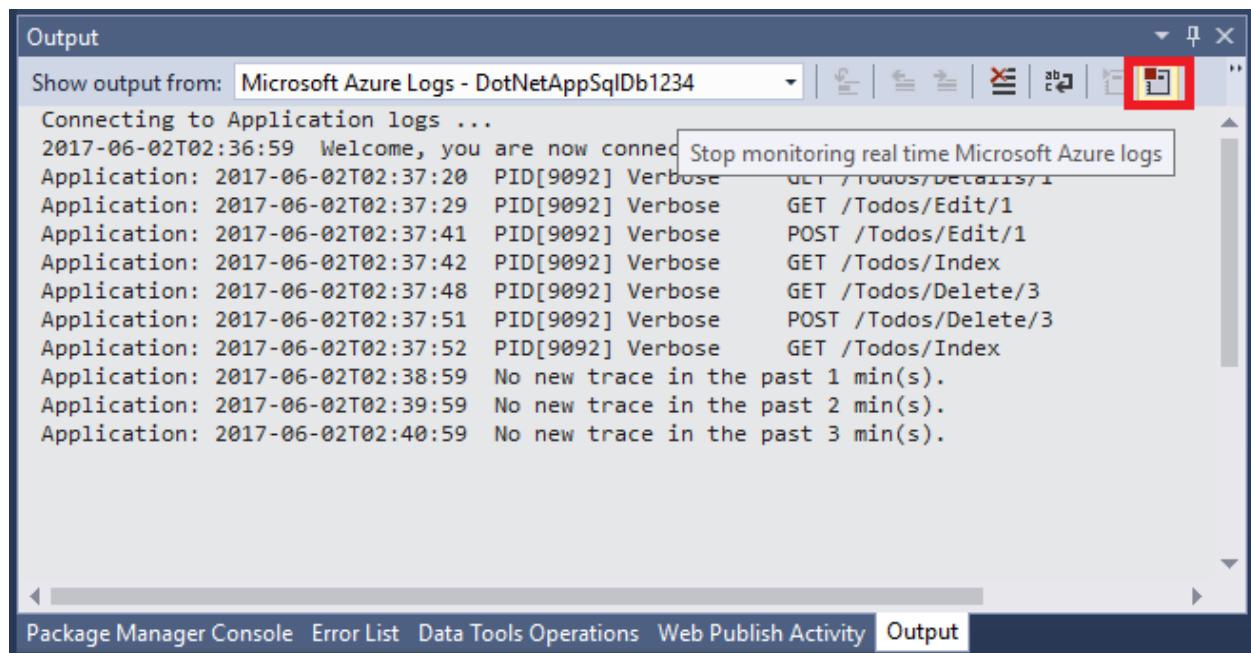


The screenshot shows the Visual Studio Output window titled "Console". It displays a list of log entries from an application named "Todos". The entries show various API requests (GET, POST) made to the "/Todos" endpoint at different times (e.g., 2017-04-06T23:30:41, 2017-04-06T23:30:43, 2017-04-06T23:30:53, 2017-04-06T23:30:54). Each entry includes the timestamp, PID, log level (Verbose), and method (GET or POST).

Timestamp	PID	Log Level	Method	Endpoint
2017-04-06T23:30:41	8132	Verbose	GET	/Todos/Index
2017-04-06T23:30:43	8132	Verbose	GET	/Todos/Create
2017-04-06T23:30:53	8132	Verbose	POST	/Todos/Create
2017-04-06T23:30:54	8132	Verbose	GET	/Todos/Index

Stop log streaming

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



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

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

[Tutorial: Connect to SQL Database from App Service without secrets using a managed identity](#)

More resources:

[Configure ASP.NET app](#)

Want to optimize and save on your cloud spending?

[Start analyzing costs with Cost Management](#)

Tutorial: Deploy a PHP, MySQL, and Redis app to Azure App Service

Article • 06/30/2023

This tutorial shows how to create a secure PHP app in Azure App Service that's connected to a MySQL database (using Azure Database for MySQL flexible server). You'll also deploy an Azure Cache for Redis to enable the caching code in your application. Azure App Service is a highly scalable, self-patching, web-hosting service that can easily deploy apps on Windows or Linux. When you're finished, you'll have a Laravel app running on Azure App Service on Linux.

The screenshot displays a user interface for managing tasks. At the top, there is a "Task List" header. Below it, a "New Task" section contains a "Task" input field and a "+ Add Task" button. The main area is titled "Current Tasks" and lists three items:

Task	Action
Create app and database in Azure	Delete
Deploy data-driven app	Delete
That's it!	Delete

At the bottom of the screen, a message indicates a response time of "Response time: 78.22585105896 milliseconds."

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

Sample application

To follow along with this tutorial, clone or download the sample [Laravel](#) application from the repository:

```
terminal
```

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

If you want to run the application locally, do the following:

- In `.env`, configure the database settings (like `DB_DATABASE`, `DB_USERNAME`, and `DB_PASSWORD`) using settings in your local MySQL database. You need a local MySQL server to run this sample.
- From the root of the repository, start Laravel with the following commands:

```
terminal
```

```
composer install
php artisan migrate
php artisan key:generate
php artisan serve
```

1 - Create App Service and MySQL resources

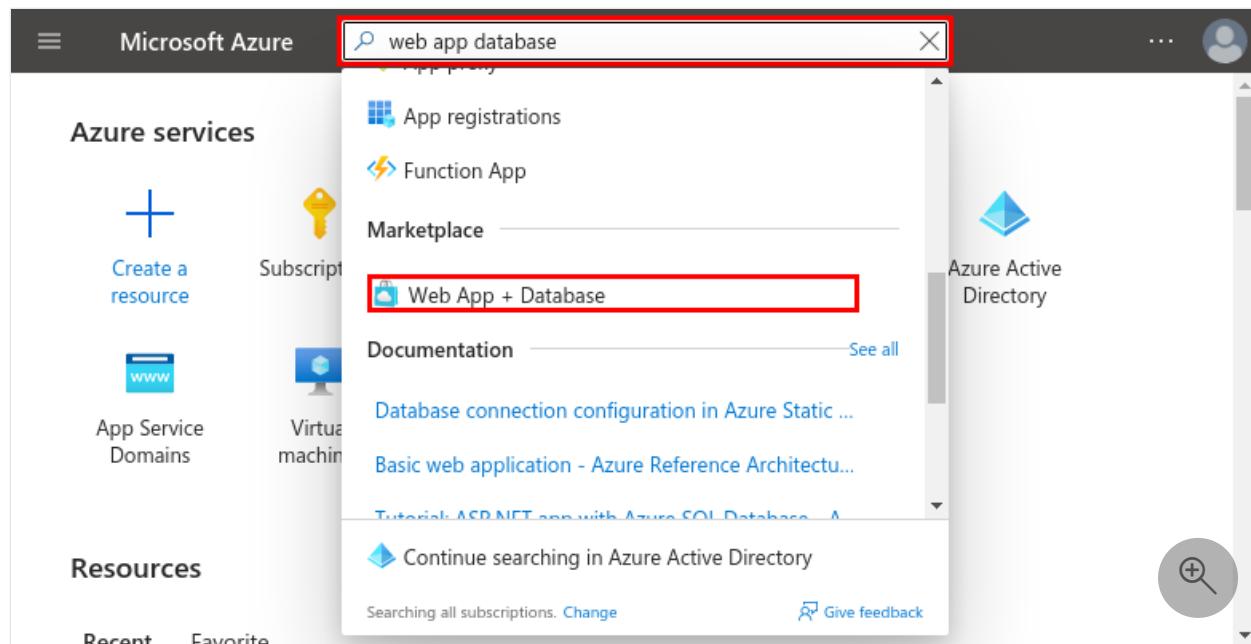
In this step, you create the Azure resources. The steps used in this tutorial create an App Service and Azure Database for MySQL configuration that's secure by default. For the creation process, you'll specify:

- The **Name** for the web app. It's the name used as part of the DNS name for your webapp in the form of `https://<app-name>.azurewebsites.net`.
- The **Runtime** for the app. It's where you select the version of PHP to use for your app.
- The **Resource Group** for the app. A resource group lets you group (in a logical container) all the Azure resources needed for the application.

Sign in to the [Azure portal](#) and follow these steps to create your Azure App Service resources.

Step 1. In the Azure portal:

1. Enter "web app database" in the search bar at the top of the Azure portal.
2. Select the item labeled **Web App + Database** under the **Marketplace** heading. You can also navigate to the [creation wizard](#) directly.



Step 2. In the **Create Web App + Database** page, fill out the form as follows.

1. *Resource Group* → Select **Create new** and use a name of **msdocs-laravel-mysql-tutorial**.
2. *Region* → Any Azure region near you.
3. *Name* → **msdocs-laravel-mysql-XYZ** where **XYZ** is any three random characters.
This name must be unique across Azure.
4. *Runtime stack* → **PHP 8.2**.
5. *Add Azure Cache for Redis?* → **Yes**.
6. *Hosting plan* → **Basic**. When you're ready, you can **scale up** to a production pricing tier later.
7. **MySQL - Flexible Server** is selected for you by default as the database engine.
Azure Database for MySQL is a fully managed MySQL database as a service on Azure, compatible with the latest community editions.
8. Select **Review + create**.
9. After validation completes, select **Create**.

Create Web App + Database

X

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

Subscription * ⓘ

Visual Studio Enterprise Subscription

Resource Group * ⓘ

(New) msdocs-laravel-mysql-tutorial

Create new

Region *

West Europe

Web App Details

Name *

msdocs-laravel-mysql-234

.azurewebsites.net

Runtime stack *

PHP 8.2

Database

i Database access will be locked down and not exposed to the public internet. This is in compliance with recommended best practices for security.

Engine * ⓘ

MySQL - Flexible Server (recommended)

Server name *

msdocs-laravel-mysql-234-server

Database name *

msdocs-laravel-mysql-234-database

Azure Cache for Redis

Add Azure Cache for Redis?

Yes

No

Cache name *

msdocs-laravel-mysql-234-cache

Hosting

Hosting plan *

Basic - For hobby or research purposes

Standard - General purpose production apps

Review + create

< Previous

Next : Tags >



Step 3. The deployment takes a few minutes to complete. Once deployment completes, select the **Go to resource** button. You're taken directly to the App Service app, but the following resources are created:

- **Resource group** → The container for all the created resources.
- **App Service plan** → Defines the compute resources for App Service. A Linux plan in the *Basic* tier is created.
- **App Service** → Represents your app and runs in the App Service plan.

- **Virtual network** → Integrated with the App Service app and isolates back-end network traffic.
- **Private endpoints** → Access endpoints for the database server and the Redis cache in the virtual network.
- **Network interfaces** → Represents private IP addresses, one for each of the private endpoints.
- **Azure Database for MySQL flexible server** → Accessible only from behind its private endpoint. A database and a user are created for you on the server.
- **Azure Cache for Redis** → Accessible only from behind its private endpoint.
- **Private DNS zones** → Enable DNS resolution of the database server and the Redis cache in the virtual network.

✓ Your deployment is complete

 Deployment name : Microsoft.Web-WebAppDatabase-Portal-5b97fff7-910d
Subscription : [Visual Studio Enterprise Subscription](#)
Resource group : [msdocs-laravel-mysql-tutorial](#)
Start time : 6/15/2023, 9:15:18 AM
Correlation ID : f833e5d2-2dfa-4363-bc63-6db91eddc95d

› Deployment details

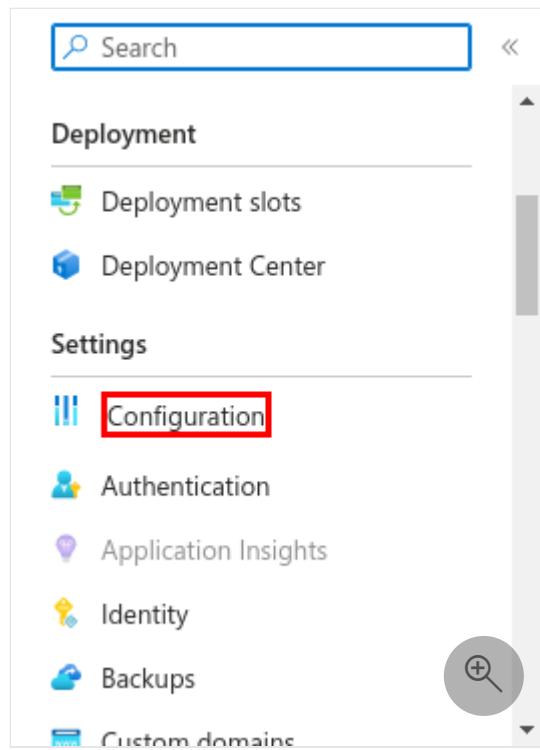
∨ Next steps

[Go to resource](#)



2 - Set up database connectivity

Step 1. In the App Service page, in the left menu, select **Configuration**.



Step 2.

1. Find app settings that begin with **AZURE_MYSQL_**. They were generated from the new MySQL database by the creation wizard.
2. Also, find app settings that begin with **AZURE_REDIS_**. They were generated from the new Redis cache by the creation wizard. To set up your application, this name is all you need.
3. If you want, you can select the **Edit** button to the right of each setting and see or copy its value. Later, you'll change your application code to use these settings.

Custom Error pages requires a premium App Service Plan.

AZURE_MYSQL_DBNAME

Hidden value. Click to show value

AZURE_MYSQL_FLAG

Hidden value. Click to show value

AZURE_MYSQL_HOST

Hidden value. Click to show value

AZURE_MYSQL_PASSWORD

Hidden value. Click to show value

AZURE_MYSQL_PORT

Hidden value. Click to show value

AZURE_MYSQL_USERNAME

Hidden value. Click to show value

AZURE_REDIS_DATABASE

Hidden value. Click to show value

AZURE_REDIS_HOST

Hidden value. Click to show value

AZURE_REDIS_PASSWORD

Hidden value. Click to show value

AZURE_REDIS_PORT

Hidden value. Click to show value

AZURE_REDIS_SSL

Hidden value. Click to show value

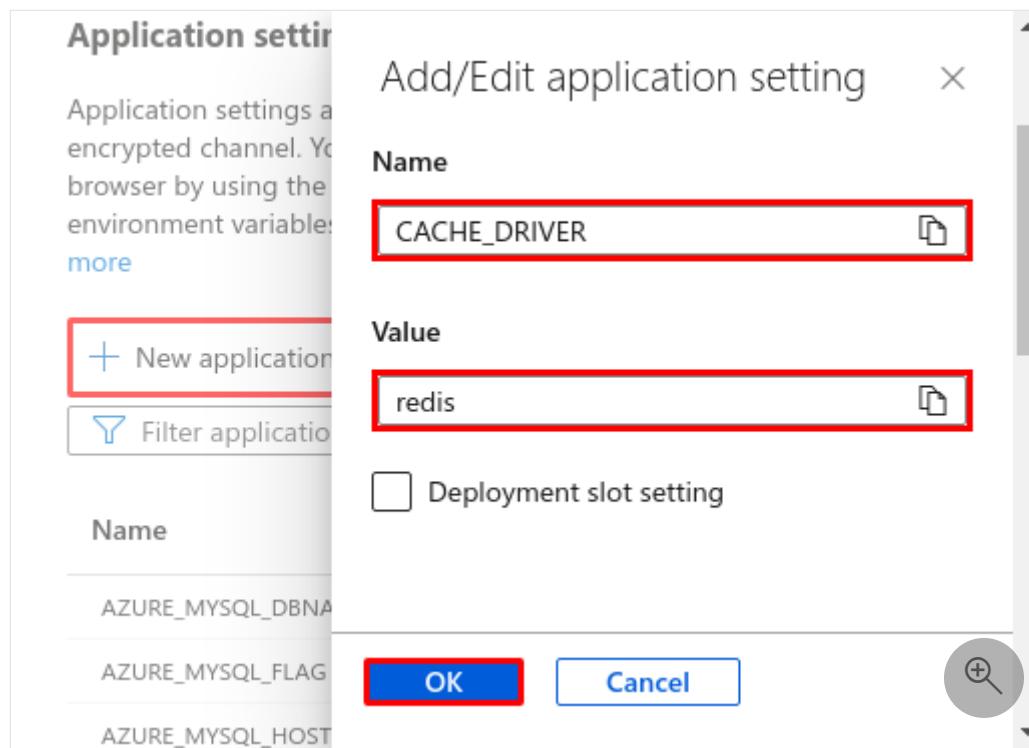
Connection strings



Connection strings are encrypted at rest and transmitted over an

Step 3. In the Application settings tab of the Configuration page, create a `CACHE_DRIVER` setting:

1. Select New application setting.
2. In the Name field, enter `CACHE_DRIVER`.
3. In the Value field, enter `redis`.
4. Select OK. `CACHE_DRIVER` is already used in the Laravel application code. This setting tells Laravel to use Redis as its cache.



Step 4. Using the same steps in **Step 3**, create the following app settings:

- **MYSQL_ATTR_SSL_CA**: Use `/home/site/wwwroot/ssl/DigiCertGlobalRootCA.crt.pem` as the value. This app setting points to the path of the [TLS/SSL certificate you need to access the MySQL server](#). It's included in the sample repository for convenience.
- **LOG_CHANNEL**: Use `stderr` as the value. This setting tells Laravel to pipe logs to `stderr`, which makes it available to the App Service logs.
- **APP_DEBUG**: Use `true` as the value. It's a [Laravel debugging variable](#) that enables debug mode pages.
- **APP_KEY**: Use `base64:Dsz40HWwbCqnq0oxMsjq7fltmKleBfCBGORfspal1Kw=` as the value. It's a [Laravel encryption variable](#).

1. In the menu bar at the top, select **Save**.
2. When prompted, select **Continue**.

The screenshot shows the Azure portal interface for managing environment variables and connection strings. At the top, there are buttons for Refresh, Save (which is highlighted with a red box), Discard, and Leave Feedback. A purple banner at the top states: "Custom Error pages requires a premium App Service Plan." Below this, a list of environment variables is shown:

Name	Description
AZURE_REDIS_PORT	Hidden value. Click to show value
AZURE_REDIS_SSL	Hidden value. Click to show value
CACHE_DRIVER	Hidden value. Click to show value
LOG_CHANNEL	Hidden value. Click to show value
MYSQL_ATTR_SSL_CA	Hidden value. Click to show value

Below the environment variables, there is a section titled "Connection strings". It includes a "New connection string" button, a "Show values" button, and an "Advanced edit" button. A "Filter connection strings" input field is also present. A message indicates "(no connection strings to display)".

ⓘ Important

The `APP_KEY` value is used here for convenience. For production scenarios, it should be generated specifically for your deployment using `php artisan key:generate --show` in the command line.

3 - Deploy sample code

In this step, you'll configure GitHub deployment using GitHub Actions. It's just one of many ways to deploy to App Service, but also a great way to have continuous integration in your deployment process. By default, every `git push` to your GitHub repository kicks off the build and deploy action. You'll make some changes to your codebase with Visual Studio Code directly in the browser, then let GitHub Actions deploy automatically for you.

Step 1. In a new browser window:

1. Sign in to your GitHub account.
2. Navigate to <https://github.com/Azure-Samples/laravel-tasks>.
3. Select **Fork**.
4. Select **Create fork**.

A screenshot of a GitHub repository page. At the top, there's a navigation bar with links for Pulls, Issues, Codespaces, Marketplace, Explore, and a user icon. Below the navigation bar, the repository name 'Azure-Samples / laravel-tasks' is shown, followed by a 'Public' badge. In the top right corner of the main content area, there are buttons for Watch (15), Fork (592), and Star (38). The 'Fork' button is specifically highlighted with a red box. Below these buttons, there are tabs for Code, Issues (6), Pull requests (17), Actions, Projects, Security, and more. The 'Code' tab is selected. On the left side, there's a sidebar with a 'main' dropdown, 'Go to file', 'Add file', and a 'Code' dropdown. The main content area shows a list of commits from a user named '<github-alias>'. The commits are: 'upgrade to Laravel 8.x' (app folder, last year), 'upgrade to Laravel 8.x' (bootstrap folder, last year), 'revert mistaken changes' (config folder, last year), and 'upgrade to Laravel 8.x' (database folder, last year). To the right of the commits, there's an 'About' section with a brief description of the repository: 'A simple Laravel application that demonstrates how to build data-driven PHP apps in Azure App Service.' Below the description are links for Readme, MIT license, Code of conduct, and Activity. A magnifying glass icon is also present.

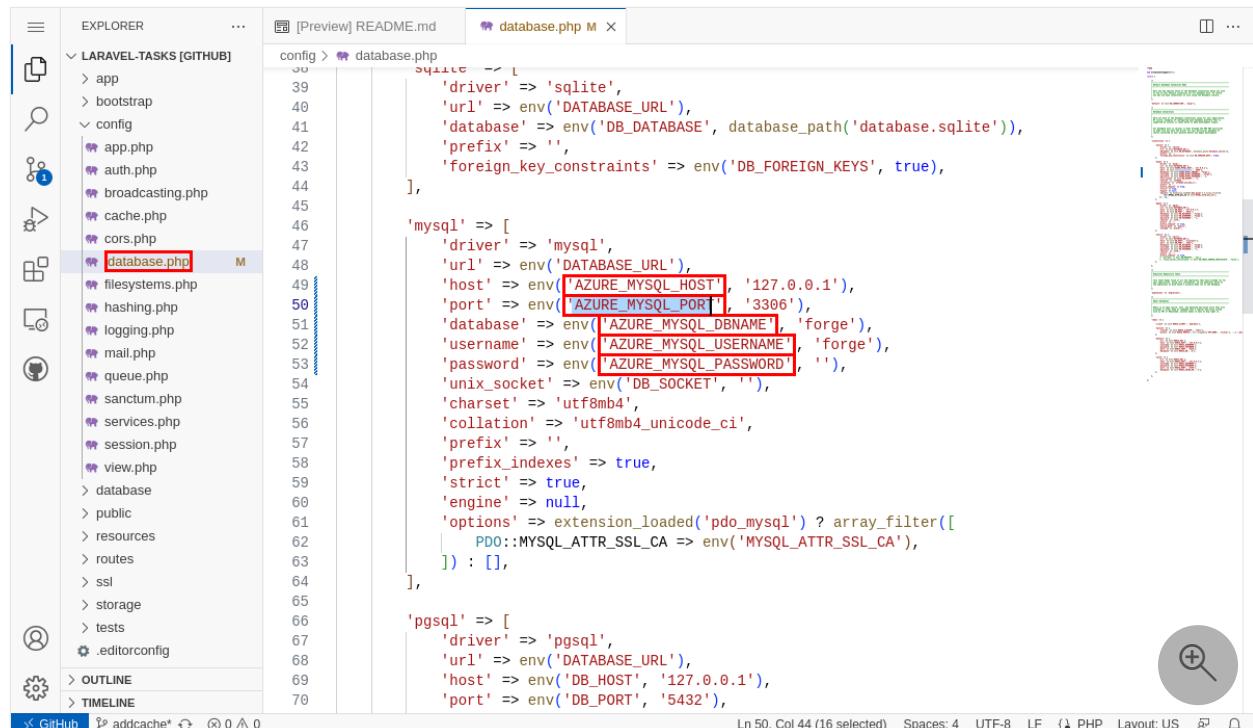
Step 2. In the GitHub page, open Visual Studio Code in the browser by pressing the `.` key.

A screenshot of a GitHub repository page for a forked repository, '<github-alias> / laravel-tasks'. The top navigation bar and repository details are identical to the original page. However, the main content area has a large red box highlighting the text 'Press the '.' key' in the center. Below this text, there's a message about the main branch: 'Your main branch isn't protected' with a note to protect it or dismiss the message. The right side of the page contains the 'About' section and other repository details. A magnifying glass icon is also present.

Step 3. In Visual Studio Code in the browser, open `config/database.php` in the explorer.

Find the `mysql` section and make the following changes:

1. Replace `DB_HOST` with `AZURE_MYSQL_HOST`.
2. Replace `DB_DATABASE` with `AZURE_MYSQL_DBNAME`.
3. Replace `DB_USERNAME` with `AZURE_MYSQL_USERNAME`.
4. Replace `DB_PASSWORD` with `AZURE_MYSQL_PASSWORD`.
5. Replace `DB_PORT` with `AZURE_MYSQL_PORT`. Remember that these `AZURE_MYSQL_` settings were created for you by the create wizard.



```
39     'driver' => 'sqlite',
40     'url' => env('DATABASE_URL'),
41     'database' => env('DB_DATABASE', database_path('database.sqlite')),
42     'prefix' => '',
43     'foreign_key_constraints' => env('DB_FOREIGN_KEYS', true),
44 ],
45
46 'mysql' => [
47     'driver' => 'mysql',
48     'url' => env('DATABASE_URL'),
49     'host' => env('AZURE_MYSQL_HOST', '127.0.0.1'),
50     'port' => env('AZURE_MYSQL_PORT', '3306'),
51     'database' => env('AZURE_MYSQL_DBNAME', 'forge'),
52     'username' => env('AZURE_MYSQL_USERNAME', 'forge'),
53     'password' => env('AZURE_MYSQL_PASSWORD', ''),
54     'unix_socket' => env('DB_SOCKET', ''),
55     'charset' => 'utf8mb4',
56     'collation' => 'utf8mb4_unicode_ci',
57     'prefix' => '',
58     'prefix_indexes' => true,
59     'strict' => true,
60     'engine' => null,
61     'options' => extension_loaded('pdo_mysql') ? array_filter([
62         PDO::MYSQL_ATTR_SSL_CA => env('MYSQL_ATTR_SSL_CA'),
63     ]) : [],
64 ],
65
66 'pgsql' => [
67     'driver' => 'pgsql',
68     'url' => env('DATABASE_URL'),
69     'host' => env('DB_HOST', '127.0.0.1'),
70     'port' => env('DB_PORT', '5432'),
```

Step 4. In `config/database.php` scroll to the Redis `cache` section and make the following changes:

1. Replace `REDIS_HOST` with `AZURE_REDIS_HOST`.
2. Replace `REDIS_PASSWORD` with `AZURE_REDIS_PASSWORD`.
3. Replace `REDIS_PORT` with `AZURE_REDIS_PORT`.
4. Replace `REDIS_CACHE_DB` with `AZURE_REDIS_DATABASE`.
5. In the same section, add a line with `'scheme' => 'tls'`. This configuration tells Laravel to use encryption to connect to Redis. Remember that these `AZURE_REDIS_` settings were created for you by the create wizard.

The screenshot shows the Visual Studio Code interface with the file `database.php` open in the editor. The code contains several environment variable assignments for Redis connections. A red box highlights the section where the `AZURE` variables are being set:

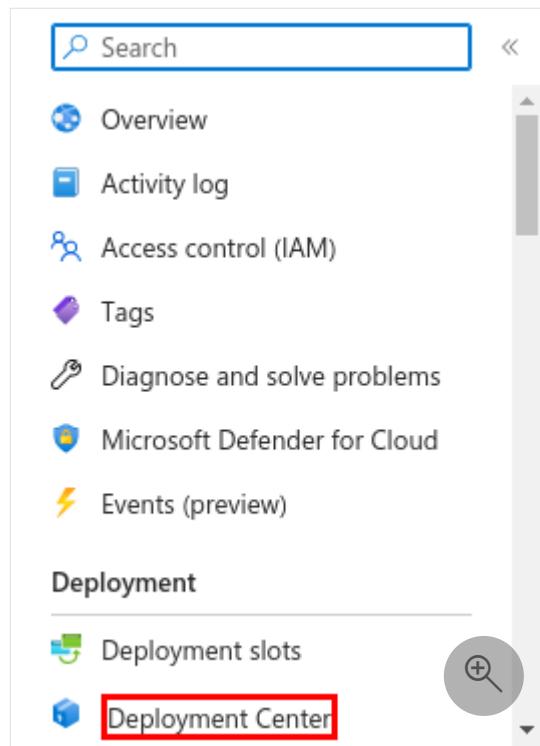
```
        'options' => [
            'cluster' => env('REDIS_CLUSTER', 'redis'),
            'prefix' => env('REDIS_PREFIX', Str::slug(env('APP_NAME', 'laravel'), '_').'_data'),
        ],
        'default' => [
            'url' => env('REDIS_URL'),
            'host' => env('REDIS_HOST', '127.0.0.1'),
            'username' => env('REDIS_USERNAME'),
            'password' => env('REDIS_PASSWORD'),
            'port' => env('REDIS_PORT', '6379'),
            'database' => env('REDIS_DB', '0'),
        ],
        'cache' => [
            'scheme' => 'tls',
            'host' => env('AZURE_REDIS_HOST', '127.0.0.1'),
            'username' => env('REDIS_USERNAME'),
            'password' => env('AZURE_REDIS_PASSWORD'),
            'port' => env('AZURE_REDIS_PORT', '6379'),
            'database' => env('AZURE_REDIS_DATABASE', '1'),
        ],
    ],
};
```

Step 5.

1. Select the **Source Control** extension.
2. In the textbox, type a commit message like `Configure DB & Redis variables`.
3. Select **Commit and Push**.

The screenshot shows the Visual Studio Code interface with the Source Control extension active. A red box highlights the commit message input field containing `Configure DB & Redis variables`. The `Commit & Push` button is also highlighted with a red box.

Step 6. Back in the App Service page, in the left menu, select **Deployment Center**.



Step 7. In the Deployment Center page:

1. In **Source**, select **GitHub**. By default, **GitHub Actions** is selected as the build provider.
2. Sign in to your GitHub account and follow the prompt to authorize Azure.
3. In **Organization**, select your account.
4. In **Repository**, select **laravel-task**.
5. In **Branch**, select **main**.
6. In the top menu, select **Save**. App Service commits a workflow file into the chosen GitHub repository, in the `.github/workflows` directory.

Microsoft Azure Search resources, services, and docs (G+/-) ...

Home > msdocs-laravel-mysql-234

msdocs-laravel-mysql-234 | Deployment Center

Web App

Save Discard Browse Manage publish profile ...

Search

Overview Activity log Access control (IAM) Tags Diagnose and solve problems Microsoft Defender for Cloud Events (preview)

Deployment

Deployment slots Deployment Center

Settings

Configuration Authentication Application Insights Identity Backups Custom domains Certificates Networking Scale up (App Service plan) Scale out (App Service plan) Service Connector Locks

App Service plan

App Service plan

Settings * Logs FTPS credentials

Info You're now in the production slot, which is not recommended for setting up CI/CD. [Learn more](#)

Deploy and build code from your preferred source and build provider. [Learn more](#)

Source*

GitHub

Building with GitHub Actions. [Change provider](#).

GitHub

App Service will place a GitHub Actions workflow in your chosen repository to build and deploy your app whenever there is a commit on the chosen branch. If you can't find an organization or repository, you may need to enable additional permissions on GitHub. You must have write access to your chosen GitHub repository to deploy with GitHub Actions. [Learn more](#)

Signed in as

<github-alias> [Change Account](#)

Organization*

<github-alias>

Repository*

laravel-tasks

Branch*

addcache

Step 8. In the Deployment Center page:

1. Select **Logs**. A deployment run is already started.
2. In the log item for the deployment run, select **Build/Deploy Logs**.

The screenshot shows the Microsoft Azure Deployment Center for a web app named 'msdocs-laravel-mysql-234'. The 'Logs' tab is selected. In the commit list, the first entry for 'Friday, June 16, 2023 (1)' has a link labeled 'Build/Deploy Log...' which is also highlighted with a red box.

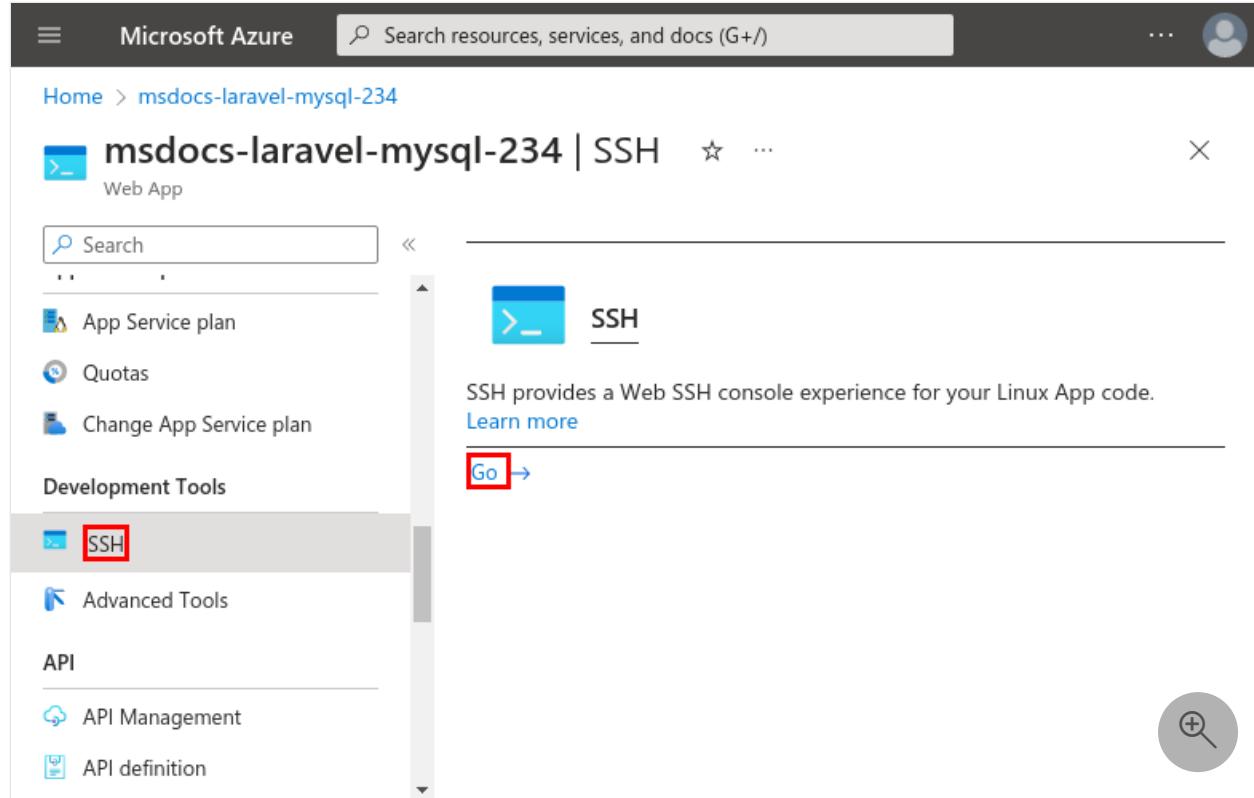
Step 9. You're taken to your GitHub repository and see that the GitHub action is running. The workflow file defines two separate stages, build and deploy. Wait for the GitHub run to show a status of **Complete**. It takes about 15 minutes.

The screenshot shows a GitHub repository page for '`<github-alias>/laravel-tasks`'. The 'Actions' tab is selected. A workflow named 'Add or update the Azure App Service build and deployment workflow config #1' is shown as 'In progress'. The status bar indicates 'Triggered via push now' and shows the commit hash '68a20c8' and the action 'addcache'.

4 - Generate database schema

The creation wizard puts the MySQL database server behind a private endpoint, so it's accessible only from the virtual network. Because the App Service app is already integrated with the virtual network, the easiest way to run database migrations with your database is directly from within the App Service container.

Step 1. Back in the App Service page, in the left menu, select SSH.



The screenshot shows the Microsoft Azure portal interface. At the top, there is a navigation bar with the Microsoft Azure logo, a search bar, and a user profile icon. Below the navigation bar, the URL 'msdocs-laravel-mysql-234' is visible. The main content area displays the details for the 'msdocs-laravel-mysql-234 | SSH' application, which is categorized as a 'Web App'. On the left side, there is a vertical sidebar with the following sections and their corresponding icons:

- Search
- App Service plan
- Quotas
- Change App Service plan
- Development Tools
 - SSH (selected and highlighted with a red box)
 - Advanced Tools
- API
 - API Management
 - API definition

The 'SSH' section on the right contains the following text:
SSH provides a Web SSH console experience for your Linux App code.
[Learn more](#)
[Go →](#)

Step 2. In the SSH terminal:

1. Run `cd /home/site/wwwroot`. Here are all your deployed files.
2. Run `php artisan migrate --force`. If it succeeds, App Service is connecting successfully to the MySQL database. Only changes to files in `/home` can persist beyond app restarts. Changes outside of `/home` aren't persisted.

```
APP SERVICE ON LINUX

Documentation: http://aka.ms/webapp-linux
PHP quickstart: https://aka.ms/php-qs
PHP version : 8.2.5
Note: Any data outside '/home' is not persisted
root@f4a9ddd627e8:/home# cd /home/site/wwwroot
root@f4a9ddd627e8:/home/site/wwwroot# php artisan migrate --force

INFO Preparing database.

Creating migration table ..... 241ms DONE
INFO Running migrations.

2014_10_12_000000_create_users_table ..... 399ms DONE
2014_10_12_100000_create_password_reset_tokens_table ..... 502ms DONE
2019_08_19_000000_create_failed_jobs_table ..... 413ms DONE
2019_12_14_000001_create_personal_access_tokens_table ..... 568ms DONE
2022_06_17_104844_create_tasks_table ..... 159ms DONE

root@f4a9ddd627e8:/home/site/wwwroot# 
☰ Menu | ssh://root@169.254.130.3:2222 | SSH CONNECTION ESTABLISHED
```

5 - Change site root

Laravel application lifecycle [begins](#) in the `/public` directory instead. The default PHP container for App Service uses Nginx, which starts in the application's root directory. To change the site root, you need to change the Nginx configuration file in the PHP container (`/etc/nginx/sites-available/default`). For your convenience, the sample repository contains a custom configuration file called `default`. As noted previously, you don't want to replace this file using the SSH shell, because the change is outside of `/home` and will be lost after an app restart.

Step 1.

1. From the left menu, select **Configuration**.
2. Select the **General settings** tab.

Microsoft Azure Search resources, services, and docs (G+/)

Home > msdocs-laravel-mysql-234

msdocs-laravel-mysql-234 | Configuration

Web App

Search Refresh Save Discard Leave Feedback

Custom Error pages requires a premium App Service Plan.

Deployment

- Deployment slots
- Deployment Center

Settings

- Configuration
- Authentication
- Application Insights
- Identity
- Backups
- Custom domains

Application settings General settings (highlighted with red box) Path mappings ...

Application settings

Application settings are encrypted at rest and transmitted over an encrypted channel. You can choose to display them in plain text in your browser by using the controls below. Application Settings are exposed as environment variables for access by your application at runtime. [Learn more](#)

+ New application setting Show values Advanced edit Filter application settings

Step 2. In the General settings tab:

1. In the **Startup Command** box, enter the following command:
`cp /home/site/wwwroot/default /etc/nginx/sites-available/default && service nginx reload.`
2. Select **Save**. The command replaces the Nginx configuration file in the PHP container and restarts Nginx. This configuration ensures that the same change is made to the container each time it starts.

Refresh Save (highlighted with red box) Discard Leave Feedback

Custom Error pages requires a premium App Service Plan.

minor version

PHP 8.2

Startup Command

```
cp /home/site/wwwroot/default /etc/nginx/sites-available/default && service nginx reload
```

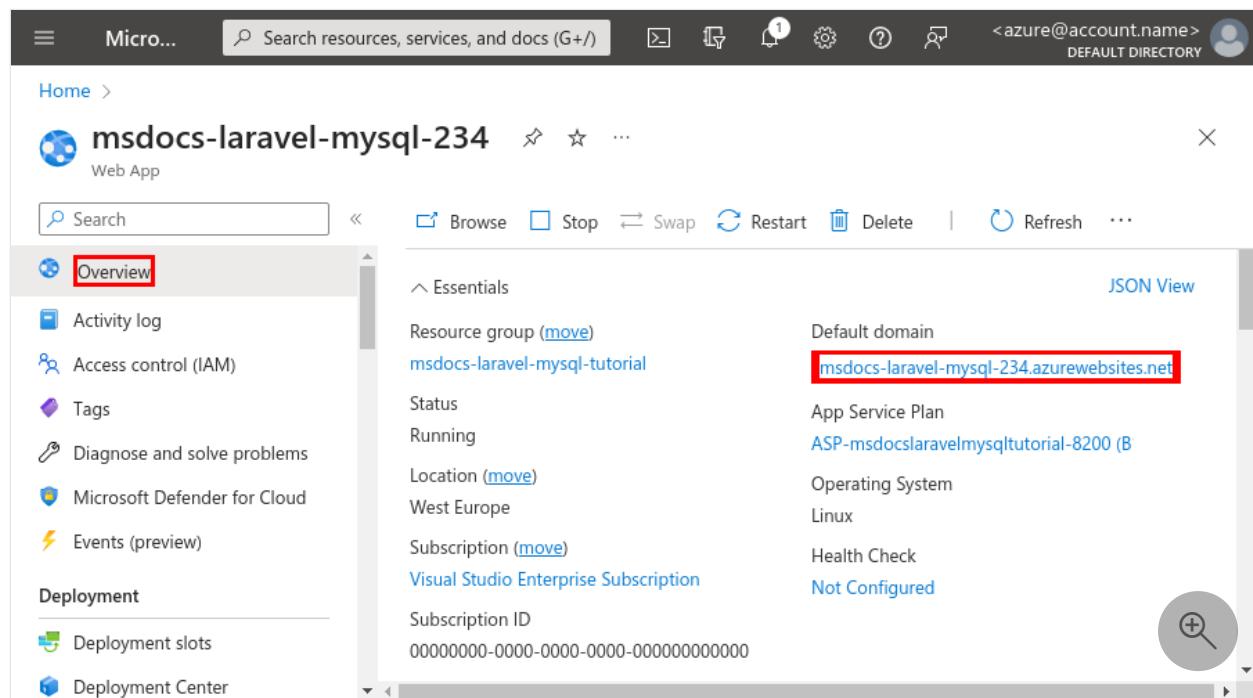
Provide an optional startup command that will be run as part of container startup. [Learn more](#)

Platform settings

6 - Browse to the app

Step 1. In the App Service page:

1. From the left menu, select **Overview**.
2. Select the URL of your app. You can also navigate directly to `https://<app-name>.azurewebsites.net`.



The screenshot shows the Azure App Service Overview page for the application 'msdocs-laravel-mysql-234'. The left sidebar has a 'Overview' tab selected, which is highlighted with a red box. The main content area displays various configuration details under the 'Essentials' section. The 'Default domain' field is explicitly highlighted with a red box, showing the value 'msdocs-laravel-mysql-234.azurewebsites.net'. Other visible details include the Resource group ('msdocs-laravel-mysql-tutorial'), Status ('Running'), Location ('West Europe'), App Service Plan ('ASP-msdocslaravelmysqltutorial-8200 (B)'), Operating System ('Linux'), Subscription ('Visual Studio Enterprise Subscription'), and Health Check status ('Not Configured'). A 'Deployment' section at the bottom includes 'Deployment slots' and 'Deployment Center' options.

Step 2. Add a few tasks to the list. Congratulations, you're running a secure data-driven PHP app in Azure App Service.

Task List

New Task

Task

+ Add Task

Current Tasks

Task	
Create app and database in Azure	Delete
Deploy data-driven app	Delete
That's it!	Delete

Response time: 78.22585105896 milliseconds.

💡 Tip

The sample application implements the **cache-aside** pattern. When you reload the page after making data changes, **Response time** in the webpage shows a much faster time because it's loading the data from the cache instead of the database.

7 - Stream diagnostic logs

Azure App Service captures all messages logged to the console to assist you in diagnosing issues with your application. The sample app outputs console log messages in each of its endpoints to demonstrate this capability. By default, Laravel's logging functionality (for example, `Log::info()`) outputs to a local file. Your `LOG_CHANNEL` app setting from earlier makes log entries accessible from the App Service log stream.

Step 1. In the App Service page:

1. From the left menu, select **App Service logs**.
2. Under **Application logging**, select **File System**.

msdocs-laravel-mysql-234 | App Service logs

Web App

Save

Application logging

Off File System

Quota (MB) * 35

Retention Period (Days)

Download logs

FTP/deployment username msdocs-laravel-mysql-234\user2234

FTP

ftp://waws-prod-am2-729.ftp.azurewebsites.windows.net/site/wwwroot

Step 2. From the left menu, select **Log stream**. You see the logs for your app, including platform logs and logs from inside the container.

msdocs-laravel-mysql-234 | Log stream

Web App

Reconnect Copy Pause Clear

2023-06-16T10:37:35.098176479Z 127.0.0.1 - 16/Jun/2023:10:37:34 +0000 "POST /index.php" 302 [2023-06-16 10:37:35] local.INFO: Get /

2023-06-16T10:37:35.125431007Z 127.0.0.1 - 16/Jun/2023:10:37:35 +0000 "GET /index.php" 200 [2023-06-16 10:37:35] local.INFO: Post /task

2023-06-16T10:37:35.547549950Z 127.0.0.1 - 16/Jun/2023:10:37:35 +0000 "POST /index.php" 302 [2023-06-16 10:37:35] local.INFO: Get /

2023-06-16T10:37:35.727907014Z 127.0.0.1 - 16/Jun/2023:10:37:35 +0000 "GET /index.php" 200 [2023-06-16 10:37:35]

Clean up resources

When you're finished, you can delete all of the resources from your Azure subscription by deleting the resource group.

Step 1. In the search bar at the top of the Azure portal:

1. Enter the resource group name.
2. Select the resource group.

The screenshot shows the Microsoft Azure portal interface. At the top, there is a search bar with the text "msdocs-laravel-mysql-tutorial" and a red box highlighting it. Below the search bar, there are several navigation links: "All" (highlighted), "Resource Groups (1)", "Documentation (99+)", "Services (0)", "Resources (0)", "Marketplace (0)", and "Azure Active Directory (0)". On the left sidebar, under "Azure services", there are icons for "Create a resource", "Resource group", "App Service Domains", and "Virtual machine". Under "Resources", there are "Recent" and "Favorite" sections. The main content area displays a list of "Resource Groups" with one item highlighted: "msdocs-laravel-mysql-tutorial". Below this, there is a "Documentation" section with a link to "Continue searching in Azure Active Directory". At the bottom of the page, there is a "Give feedback" button and a note about searching all subscriptions.

Step 2. In the resource group page, select **Delete resource group**.

The screenshot shows the "msdocs-laravel-mysql-tutorial" Resource Group page. At the top, there are buttons for "Create", "Manage view" (with a dropdown arrow), "Delete resource group" (highlighted with a red box), and "...". Below this, there is a warning message: "⚠ Deleting". The "Essentials" section is expanded, showing "Resources" (selected) and "Recommendations (1)". There is a "Filter for any field..." input field, an "Add filter" button, and a "More (2)" button. At the bottom, there are filters for "Name" (sorted by ascending), "Type" (sorted by ascending), and "Location" (sorted by ascending). A search icon is located next to the location filter. The table lists one record: "msdocs-laravel-mysql-234..." (Type: Azure Cache for Redis, Location: West Europe).

Step 3.

1. Enter the resource group name to confirm your deletion.
2. Select **Delete**.

Delete a resource group

X

The following resource group and all its dependent resources will be permanently deleted.

Resource group to be deleted



Dependent resources to be deleted (11)

All dependent resources, including hidden types, are shown

Name	Resource type
ASP-msdocslaravelmysqltutorial-9b42	App Service plan
msdocs-laravel-mysql-234	App Service
msdocs-laravel-mysql-234-cache	Azure Cache for Redis
msdocs-laravel-mysql-234-cache-privateEndp	Network interface
msdocs-laravel-mysql-234-cache-privateEndp	Private endpoint
msdocs-laravel-mysql-234-server	Azure Database for MySQL flexible ...

Enter resource group name to confirm deletion *

Delete

Cancel



Frequently asked questions

- How much does this setup cost?
- How do I connect to the MySQL database that's secured behind the virtual network with other tools?
- How does local app development work with GitHub Actions?
- Why is the GitHub Actions deployment so slow?

How much does this setup cost?

Pricing for the create resources is as follows:

- The App Service plan is created in **Basic** tier and can be scaled up or down. See [App Service pricing ↗](#).
- The MySQL flexible server is created in **B1ms** tier and can be scaled up or down. With an Azure free account, **B1ms** tier is free for 12 months, up to the monthly limits. See [Azure Database for MySQL pricing ↗](#).
- The Azure Cache for Redis is created in **Basic** tier with the minimum cache size. There's a small cost associated with this tier. You can scale it up to higher performance tiers for higher availability, clustering, and other features. See [Azure Cache for Redis pricing ↗](#).
- The virtual network doesn't incur a charge unless you configure extra functionality, such as peering. See [Azure Virtual Network pricing ↗](#).
- The private DNS zone incurs a small charge. See [Azure DNS pricing ↗](#).

How do I connect to the MySQL database that's secured behind the virtual network with other tools?

- For basic access from a command-line tool, you can run `mysql` from the app's SSH terminal.
- To connect from a desktop tool like MySQL Workbench, your machine must be within the virtual network. For example, it could be an Azure VM that's connected to one of the subnets, or a machine in an on-premises network that has a [site-to-site VPN](#) connection with the Azure virtual network.
- You can also [integrate Azure Cloud Shell](#) with the virtual network.

How does local app development work with GitHub Actions?

Take the autogenerated workflow file from App Service as an example, each `git push` kicks off a new build and deployment run. From a local clone of the GitHub repository, you make the desired updates push it to GitHub. For example:

```
terminal
git add .
git commit -m "<some-message>"
git push origin main
```

Why is the GitHub Actions deployment so slow?

The autogenerated workflow file from App Service defines build-then-deploy, two-job run. Because each job runs in its own clean environment, the workflow file ensures that the `deploy` job has access to the files from the `build` job:

- At the end of the `build` job, [upload files as artifacts ↗](#).
- At the beginning of the `deploy` job, download the artifacts.

Most of the time taken by the two-job process is spent uploading and download artifacts. If you want, you can simplify the workflow file by combining the two jobs into one, which eliminates the need for the upload and download steps.

Next steps

Advance to the next tutorial to learn how to secure your app with a custom domain and certificate.

[Secure with custom domain and certificate](#)

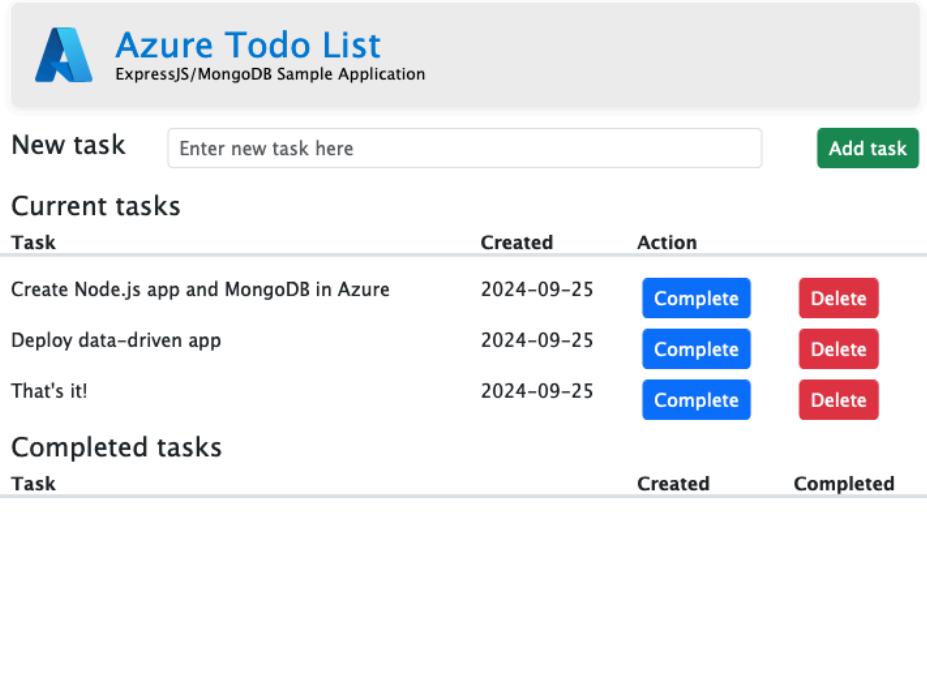
Or, check out other resources:

[Configure PHP app](#)

Tutorial: Deploy a Node.js + MongoDB web app to Azure

Article • 09/29/2024

Azure App Service provides a highly scalable, self-patching web hosting service using the Linux operating system. This tutorial shows how to create a secure Node.js app in Azure App Service that's connected to a Azure Cosmos DB for MongoDB database. When you're finished, you'll have an Express.js app running on Azure App Service on Linux.



The screenshot shows the Azure Todo List application. At the top, there's a logo and the title "Azure Todo List" followed by the subtitle "ExpressJS/MongoDB Sample Application". Below this is a search bar labeled "New task" with the placeholder "Enter new task here" and a green "Add task" button. The main area is divided into two sections: "Current tasks" and "Completed tasks".

Task	Created	Action
Create Node.js app and MongoDB in Azure	2024-09-25	<button>Complete</button> <button>Delete</button>
Deploy data-driven app	2024-09-25	<button>Complete</button> <button>Delete</button>
That's it!	2024-09-25	<button>Complete</button> <button>Delete</button>

Task	Created	Completed

In this tutorial, you learn how to:

- ✓ Create a secure-by-default architecture for Azure App Service and Azure Cosmos DB with MongoDB API.
- ✓ Secure connection secrets using a managed identity and Key Vault references.
- ✓ Deploy a Node.js sample app to App Service from a GitHub repository.
- ✓ Access App Service app settings in the application code.
- ✓ Make updates and redeploy the application code.
- ✓ Stream diagnostic logs from App Service.
- ✓ Manage the app in the Azure portal.
- ✓ Provision the same architecture and deploy by using Azure Developer CLI.
- ✓ Optimize your development workflow with GitHub Codespaces and GitHub Copilot.

Prerequisites

- An Azure account with an active subscription. If you don't have an Azure account, you [can create one for free](#).
- A GitHub account. you can also [get one for free](#).
- Knowledge of Express.js development.
- **(Optional)** To try GitHub Copilot, a [GitHub Copilot account](#). A 30-day free trial is available.

Skip to the end

You can quickly deploy the sample app in this tutorial and see it running in Azure. Just run the following commands in the [Azure Cloud Shell](#), and follow the prompt:

Bash

```
mkdir msdocs-nodejs-mongodb-azure-sample-app
cd msdocs-nodejs-mongodb-azure-sample-app
azd init --template msdocs-nodejs-mongodb-azure-sample-app
azd up
```

1. Run the sample

First, you set up a sample data-driven app as a starting point. For your convenience, the [sample repository](#), includes a [dev container](#) configuration. The dev container has everything you need to develop an application, including the database, cache, and all environment variables needed by the sample application. The dev container can run in a [GitHub codespace](#), which means you can run the sample on any computer with a web browser.

Step 1: In a new browser window:

1. Sign in to your GitHub account.
2. Navigate to <https://github.com/Azure-Samples/msdocs-nodejs-mongodb-azure-sample-app/fork>.
3. Unselect **Copy the main branch only**. You want all the branches.
4. Select **Create fork**.

The screenshot shows the GitHub fork creation interface for the repository `msdocs-nodejs-mongodb-azure-sample-app`. The top navigation bar includes links for Code, Pull requests, Actions, Projects, Security, and Insights. The main section is titled "Create a new fork". It explains that a fork is a copy of a repository, allowing experimentation without affecting the original project. It notes that required fields are marked with an asterisk (*). The "Owner" dropdown is set to "cephalin" and the "Repository name" field contains "msdocs-nodejs-mongodb-az". A note indicates that the repository is available. The "Description (optional)" field contains "A sample Express.js app using a MongoDB database to show how to host Node.js app in Azure App service". There is a checkbox for "Copy the main branch only", which is unchecked. A note below it says "Contribute back to cephalin/msdocs-nodejs-mongodb-azure-sample-app by adding your own branch." A link to "Learn more" is provided. A note at the bottom left says "You are creating a fork in your personal account." On the right side, there is a red-bordered "Create fork" button and a circular icon with a plus sign and a magnifying glass.

Create a new fork

A **fork** is a copy of a repository. Forking a repository allows you to freely experiment with changes without affecting the original project. [View existing forks](#).

Required fields are marked with an asterisk (*).

Owner * Repository name *

 / msdocs-nodejs-mongodb-az

msdocs-nodejs-mongodb-azure-sample-app is available.

By default, forks are named the same as their upstream repository. You can customize the name to distinguish it further.

Description (optional)

A sample Express.js app using a MongoDB database to show how to host Node.js app in Azure App service

Copy the `main` branch only

Contribute back to [cephalin/msdocs-nodejs-mongodb-azure-sample-app](#) by adding your own branch. [Learn more](#).

ⓘ You are creating a fork in your personal account.

Create fork

Step 2: In the GitHub fork:

1. Select **main > starter-no-infra** for the starter branch. This branch contains just the sample project and no Azure-related files or configuration.
2. Select **Code > Create codespace on starter-no-infra**. The codespace takes a few minutes to set up.

A sample Express.js app using a MongoDB database to show how to host Node.js app in Azure App service using Azure Cosmos DB

No codespaces

You don't have any codespaces with this repository checked out

Create codespace on starter-no-infra

Learn more about codespaces...

Codespace usage for this repository is paid for by .

About

- Readme
- MIT license
- Activity
- 0 stars
- 0 watching
- 0 forks

Releases

No releases published

Create a new release

Step 3: In the codespace terminal:

1. Run `npm install && npm start`.
2. When you see the notification `Your application running on port 3000 is available.`, select **Open in Browser**. You should see the sample application in a new browser tab. To stop the Express.js application, type `Ctrl + C`.

msdocs-nodejs-mongodb-azure-sample-app [Codespaces: urban fiesta]

Deploy a Express.js web app with MongoDB in Azure

This is a CRUD (create-read-update-delete) web app that uses Express.js and Azure Cosmos DB. The Node.js app is hosted in a fully managed Azure App Service. This app is designed to be run locally Linux Node.js container in Azure App Service. You can either deploy this project by following the tutorial [Tutorial: Deploy a Node.js + MongoDB web app to Azure](#) or by using the [Azure Developer CLI \(azd\)](#) according to the instructions below.

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS AZURE COMMENTS

```
@icephas + /workspaces/msdocs-nodejs-mongodb-azure-sample-app (starter-no-infra) $ npm install && npm start
up to date, audited 141 packages in 682ms
22 packages are looking for funding
  run 'npm fund' for details
found 0 vulnerabilities
> todolist@0.0.0 start
> node ./bin/www
Connected to database
```

Your application running on port 3000 is available. See all forwarded ports

Open in Browser

💡 Tip

You can ask [GitHub Copilot](#) about this repository. For example:

- *@workspace What does this project do?*
- *@workspace What does the .devcontainer folder do?*

Having issues? Check the [Troubleshooting section](#).

2. Create App Service and Azure Cosmos DB

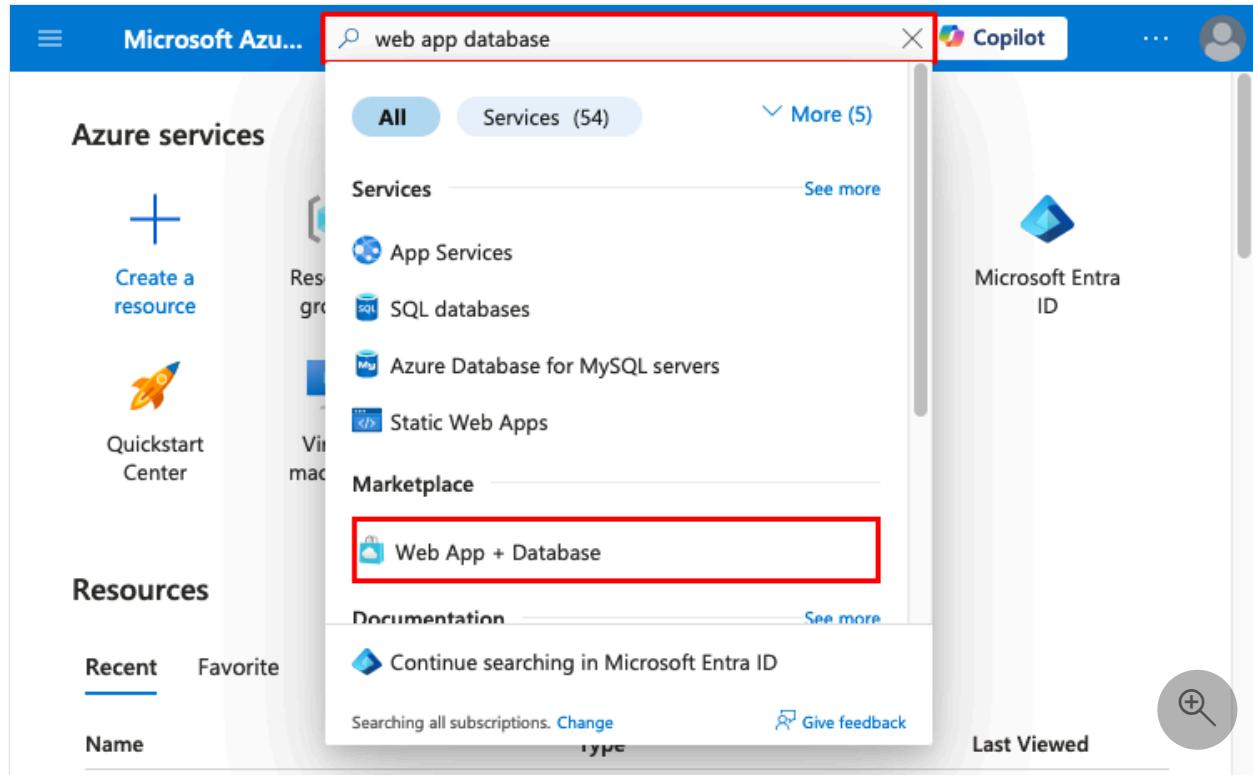
In this step, you create the Azure resources. The steps used in this tutorial create a set of secure-by-default resources that include App Service and Azure Cosmos DB for MongoDB. For the creation process, you'll specify:

- The **Name** for the web app. It's used as part of the DNS name for your app in the form of `https://<app-name>-<hash>.azurewebsites.net`.
- The **Region** to run the app physically in the world. It's also used as part of the DNS name for your app.
- The **Runtime stack** for the app. It's where you select the version of Node to use for your app.
- The **Hosting plan** for the app. It's the pricing tier that includes the set of features and scaling capacity for your app.
- The **Resource Group** for the app. A resource group lets you group (in a logical container) all the Azure resources needed for the application.

Sign in to the [Azure portal](#) and follow these steps to create your Azure App Service resources.

Step 1: In the Azure portal:

1. Enter "web app database" in the search bar at the top of the Azure portal.
2. Select the item labeled **Web App + Database** under the **Marketplace** heading. You can also navigate to the [creation wizard](#) directly.



Step 2: In the **Create Web App + Database** page, fill out the form as follows.

1. **Resource Group:** Select **Create new** and use a name of **msdocs-expressjs-mongodb-tutorial**.
2. **Region:** Any Azure region near you.
3. **Name:** **msdocs-expressjs-mongodb-XYZ**, where **XYZ** is any three random characters.
4. **Runtime stack:** **Node 20 LTS**.
5. **Engine:** **Cosmos DB API for MongoDB**. Azure Cosmos DB is a cloud native database offering a 100% MongoDB compatible API. Note the database name that's generated for you (`<app-name>-database`). You'll need it later.
6. **Hosting plan:** **Basic**. When you're ready, you can **scale up** to a production pricing tier.
7. Select **Review + create**.
8. After validation completes, select **Create**.

Basics Tags Review + create

This template will create a secure by default configuration where the only publicly accessible endpoint will be your app following the recommended security best practices. [Learn more](#)

Project Details

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

Subscription * ⓘ

Resource Group * ⓘ

(New) msdocs-expressjs-mongodb-tutorial

[Create new](#)

Region *

Central US

Web App Details

Name

msdocs-expressjs-mongodb-125

-fuafe8euaufeedfx.centralus-01.azurewebsites.net

Unique default hostname (preview) on. [More about this update](#)

Runtime stack *

Node 20 LTS

Database



Database access will be locked down and not exposed to the public internet. This is in compliance with recommended best practices for security.

Engine * ⓘ

Cosmos DB API for MongoDB (recommended)

Account name *

msdocs-expressjs-mongodb-125-server

Database name *

msdocs-expressjs-mongodb-125-database

Azure Cache for Redis

Add Azure Cache for Redis?

Yes

No

Hosting

Hosting plan *

Basic - For hobby or research purposes

Standard - General purpose production apps

[Review + create](#)

< Previous

Next : Tags >



Step 3: The deployment takes a few minutes to complete. Once deployment completes, select the **Go to resource** button. You're taken directly to the App Service app, but the following resources are created:

- **Resource group** → The container for all the created resources.
- **App Service plan** → Defines the compute resources for App Service. A Linux plan in the *Basic* tier is created.
- **App Service** → Represents your app and runs in the App Service plan.
- **Virtual network** → Integrated with the App Service app and isolates back-end network traffic.
- **Private endpoint** → Access endpoint for the database resource in the virtual network.
- **Network interface** → Represents a private IP address for the private endpoint.
- **Azure Cosmos DB for MongoDB** → Accessible only from behind the private endpoint. A database and a user are created for you on the server.
- **Private DNS zone** → Enables DNS resolution of the Azure Cosmos DB server in the virtual network.

The screenshot shows a deployment summary page in the Azure portal. At the top, there are buttons for Delete, Cancel, Redeploy, Download, and Refresh. Below this, a green checkmark icon indicates "Your deployment is complete". Deployment details are listed:

- Deployment name : Microsoft.Web-WebAppDatabase-Portal-037a6234-825c
- Subscription :
- Resource group : [msdocs-expressjs-mongodb-tutorial](#)
- Start time : 25/09/2024, 15:11:13
- Correlation ID : c75d0531-840a-45f1-9a96-566fb6d585a3

Below the details are two collapsed sections: "Deployment details" and "Next steps". A red box highlights the "Go to resource" button, which is located under the "Next steps" section. At the bottom left, there are links for "Give feedback" and "Tell us about your experience with deployment". On the right side, there is a circular icon with a plus sign and a magnifying glass.

Having issues? Check the [Troubleshooting section](#).

3. Secure connection secrets

The creation wizard generated the connectivity string for you already as an [app setting](#). However, the security best practice is to keep secrets out of App Service completely. You'll move your secrets to a key vault and change your app setting to a [Key Vault reference](#) with the help of Service Connectors.

Step 1: In the App Service page:

1. In the left menu, select **Settings > Environment variables**.

2. Next to AZURE_COSMOS_CONNECTIONSTRING, select **Show value**. This connection string lets you connect to the Cosmos DB database secured behind a private endpoint. However, the secret is saved directly in the App Service app, which isn't the best. You'll change this.

The screenshot shows the 'App settings' section of the Azure portal. On the left, there's a sidebar with various links like Access control (IAM), Tags, Diagnose and solve problems, Microsoft Defender for Cloud, Events (preview), Better Together (preview), Deployment slots, Deployment Center, and Settings. Under Settings, 'Environment variables' is selected and highlighted with a red box. The main area shows a table with one row for 'AZURE_COSMOS_CONNECTIONSTRING'. The 'Value' column contains a placeholder '(redacted)' with a red box around it. To the right of the value is a blue eye icon labeled 'Show value', also highlighted with a red box. At the bottom of the table are 'Apply' and 'Discard' buttons, and a 'Send us your feedback' button with a magnifying glass icon.

Name	Value
AZURE_COSMOS_CONNECTIONSTRING	(redacted) Show value

Step 2: Create a key vault for secure management of secrets.

1. In the top search bar, type "key vault", then select Marketplace > Key Vault.
2. In Resource Group, select msdocs-expressjs-mongodb-tutorial.
3. In Key vault name, type a name that consists of only letters and numbers.
4. In Region, set it to the sample location as the resource group.

[Basics](#) [Access configuration](#) [Networking](#) [Tags](#) [Review + create](#)

Azure Key Vault is a cloud service used to manage keys, secrets, and certificates. Key Vault eliminates the need for developers to store security information in their code. It allows you to centralize the storage of your application secrets which greatly reduces the chances that secrets may be leaked. Key Vault also allows you to securely store secrets and keys backed by Hardware Security Modules or HSMs. The HSMs used are Federal Information Processing Standards (FIPS) 140-2 Level 2 validated. In addition, key vault provides logs of all access and usage attempts of your secrets so you have a complete audit trail for compliance.

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](#)

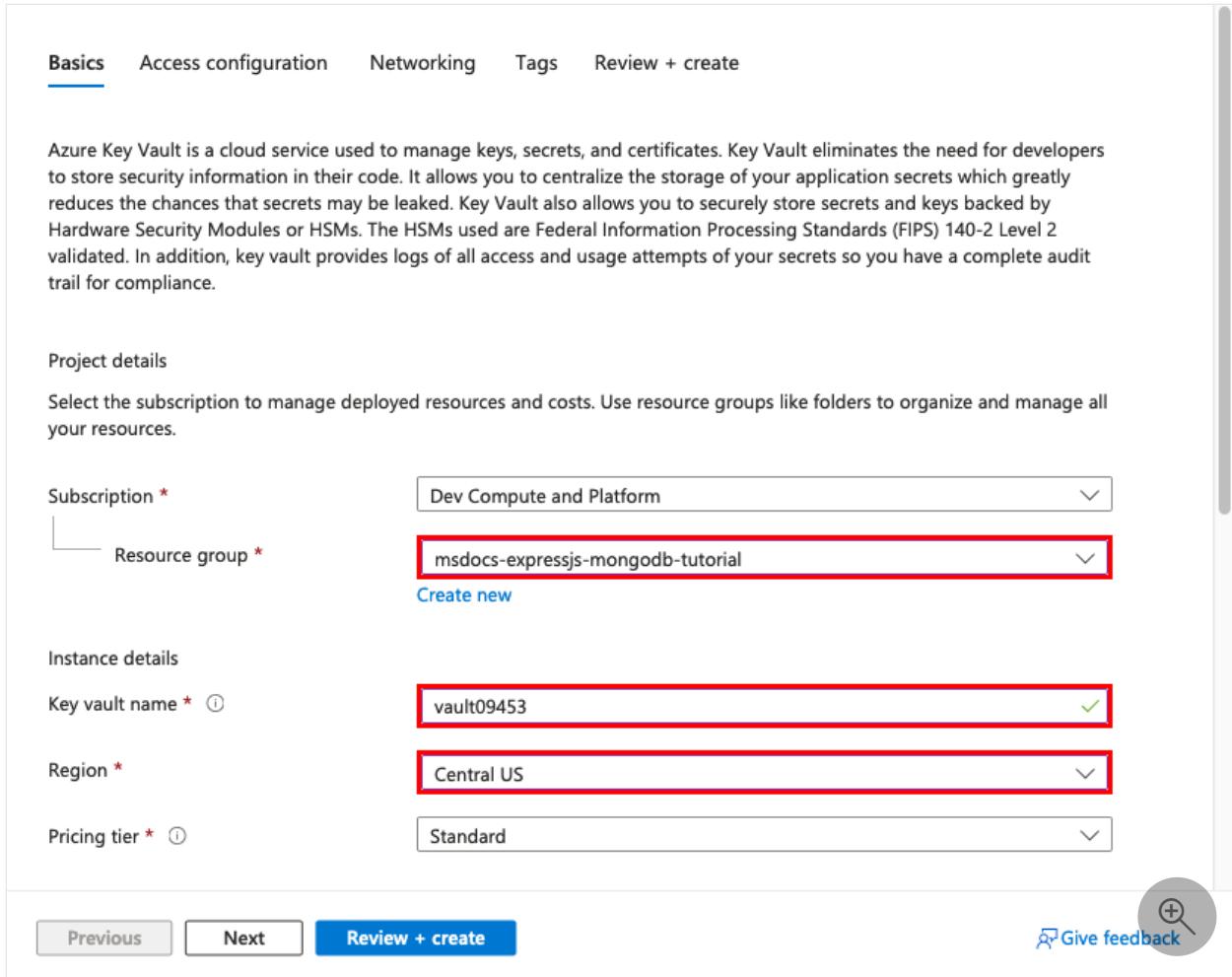
Instance details

Key vault name * 

Region *

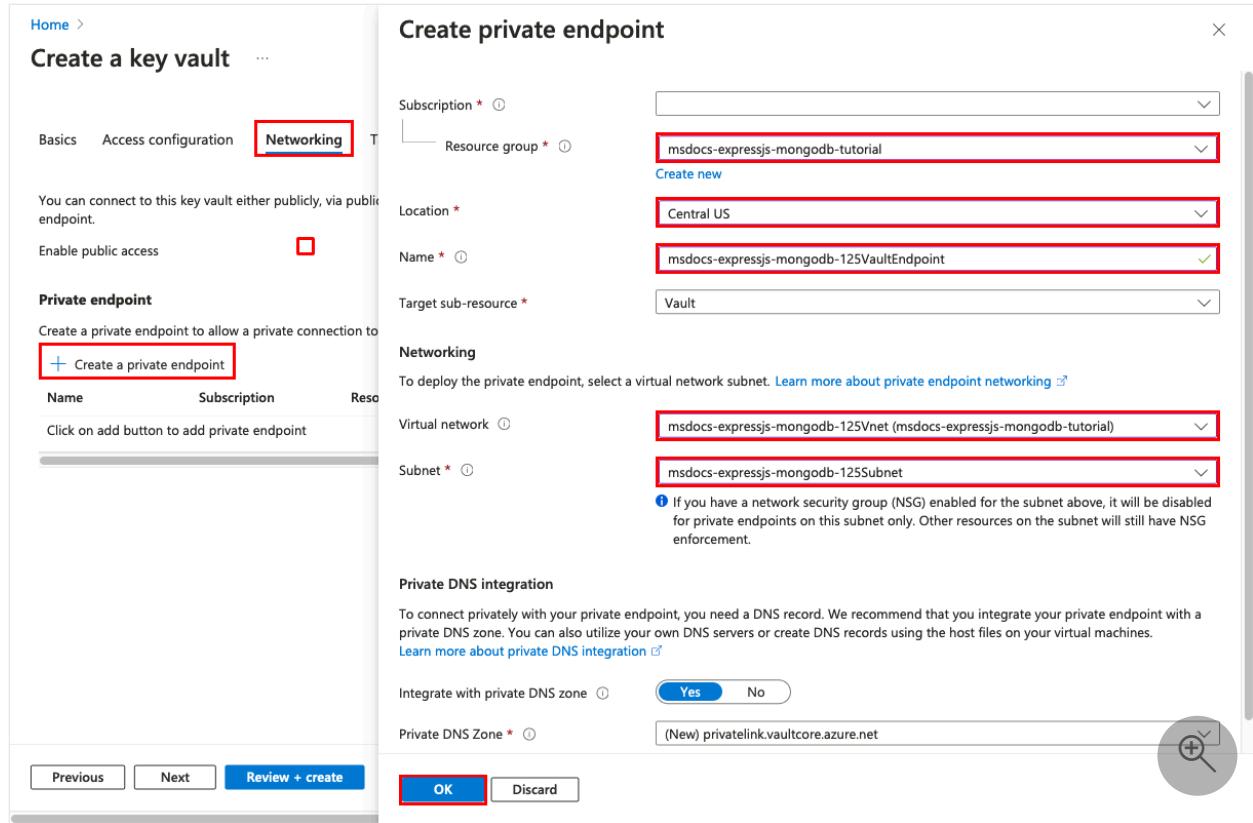
Pricing tier *

[Previous](#) [Next](#) [Review + create](#) [Give feedback](#)



Step 3:

1. Select the **Networking** tab.
2. Unselect **Enable public access**.
3. Select **Create a private endpoint**.
4. In **Resource Group**, select **msdocs-expressjs-mongodb-tutorial**.
5. In **Key vault name**, type a name that consists of only letters and numbers.
6. In **Region**, set it to the sample location as the resource group.
7. In the dialog, in **Location**, select the same location as your App Service app.
8. In **Resource Group**, select **msdocs-expressjs-mongodb-tutorial**.
9. In **Name**, type **msdocs-expressjs-mongodb-XYZVaultEndpoint**.
10. In **Virtual network**, select **msdocs-expressjs-mongodb-XYZVnet**.
11. In **Subnet**, **msdocs-expressjs-mongodb-XYZSubnet**.
12. Select **OK**.
13. Select **Review + create**, then select **Create**. Wait for the key vault deployment to finish. You should see "Your deployment is complete."



Step 4:

1. In the top search bar, type `msdocs-expressjs-mongodb`, then the App Service resource called **msdocs-expressjs-mongodb-XYZ**.
2. In the App Service page, in the left menu, select **Settings > Service Connector**. There's already a connector, which the app creation wizard created for you.
3. Select checkbox next to the connector, then select **Edit**.
4. In the **Basics** tab, set **Client type** to **Node.js**.
5. Select the **Authentication** tab.
6. Select **Store Secret in Key Vault**.
7. Under **Key Vault Connection**, select **Create new**. A **Create connection** dialog is opened on top of the edit dialog.

The screenshot shows the Azure portal interface for managing a service connector. On the left, the sidebar has a red box around the 'Service Connector' item. In the center, there's a 'msdocs-expressjs-mongodb-125 | Service Connector' card with an 'Edit' button highlighted by a red box. To the right, a 'defaultConnector' dialog box is open, with the 'Authentication' tab selected (also highlighted by a red box). Under 'Service type', 'Connection string' is chosen. In the 'Key Vault Connection' section, a dropdown menu shows 'No item available' and a 'Create new' button, which is also highlighted by a red box.

Step 5: In the **Create connection** dialog for the Key Vault connection:

1. In **Key Vault**, select the key vault you created earlier.
2. Select **Review + Create**. You should see that **System assigned managed identity** is set to **Selected**.
3. When validation completes, select **Create**.

Create connection

X

Basics

Networking

Review + Create

Select the service instance and client type.

Service type * ⓘ

Key Vault

Connection name * ⓘ

keyvault_de6bd

Subscription * ⓘ

▼

Key vault * ⓘ

vault09453

[Create new](#)

Client type * ⓘ

Node.js

Next : Networking

Cancel

Report an issue

Step 6: You're back in the edit dialog for **defaultConnector**.

1. In the **Authentication** tab, wait for the key vault connector to be created. When it's finished, the **Key Vault Connection** dropdown automatically selects it.
2. Select **Next: Networking**.
3. Select **Configure firewall rules to enable access to target service**. If you see the message, "No Private Endpoint on the target service," ignore it. The app creation wizard already secured the Cosmos DB database with a private endpoint.
4. Select **Save**. Wait until the **Update succeeded** notification appears.

defaultConnector

Basics **Authentication** Networking

Select the authentication type you'd like to use between your compute service and target service. [Learn more](#)

System assigned managed identity ⓘ
 User assigned managed identity ⓘ
 Connection string ⓘ
 Service principal ⓘ

Permissions for the connection string

Read-Write

Store Secret In Key Vault ⓘ

Key Vault Connection * ⓘ

vault09453 (keyvault_de6bd)

Create new

Store Configuration in App Configuration ⓘ

> Advanced

Next : Networking Previous Cancel

Step 7: To verify your changes:

1. From the left menu, select **Environment variables** again.
2. Next to the app setting **AZURE_COSMOS_CONNECTIONSTRING**, select **Show value**. The value should be `@microsoft.KeyVault(...)`, which means that it's a **key vault reference** because the secret is now managed in the key vault.

The screenshot shows the 'App settings' blade in the Azure portal. On the left, there's a navigation menu with 'Events (preview)', 'Better Together (preview)', 'Deployment' (which includes 'Deployment slots' and 'Deployment Center'), and 'Settings' (which includes 'Environment variables', 'Configuration', 'Authentication', 'Identity', 'Backups', and 'Custom domains'). The 'Environment variables' option is selected and highlighted with a red box. The main area displays three environment variables:

Name	Value	Deployment
AZURE_COSMOS_CONNECTI...	Show value	✓
AZURE_KEYVAULT_RESOURCE...	Show value	✓
AZURE_KEYVAULT_SCOPE	Show value	✓

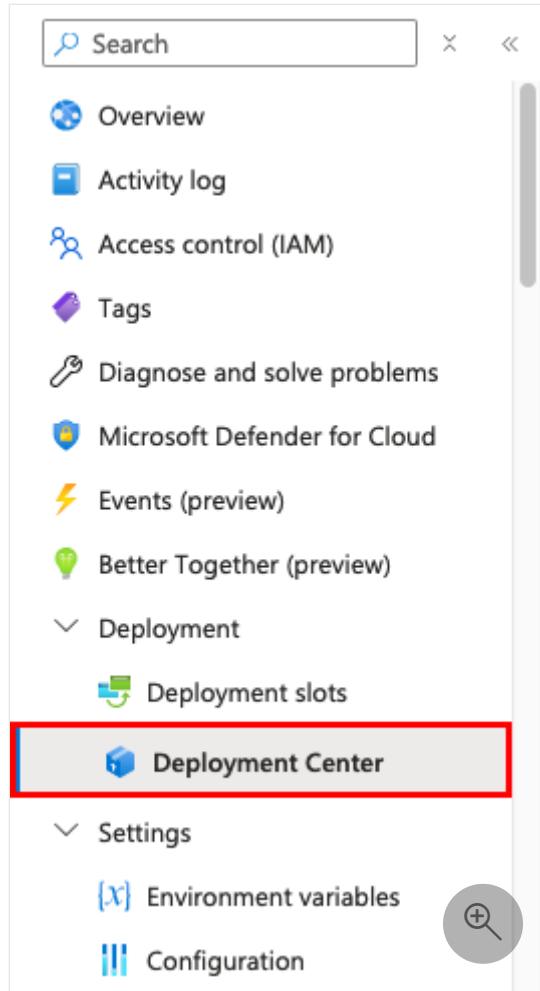
At the bottom, there are 'Apply' and 'Discard' buttons, and a 'Send us your feedback' button.

Having issues? Check the [Troubleshooting section](#).

4. Deploy sample code

In this step, you configure GitHub deployment using GitHub Actions. It's just one of many ways to deploy to App Service, but also a great way to have continuous integration in your deployment process. By default, every `git push` to your GitHub repository kicks off the build and deploy action.

Step 1: In the left menu, select **Deployment > Deployment Center**.



Step 2: In the Deployment Center page:

1. In **Source**, select **GitHub**. By default, **GitHub Actions** is selected as the build provider.
2. Sign in to your GitHub account and follow the prompt to authorize Azure.
3. In **Organization**, select your account.
4. In **Repository**, select **msdocs-nodejs-mongodb-azure-sample-app**.
5. In **Branch**, select **starter-no-infra**. This is the same branch that you worked in with your sample app, without any Azure-related files or configuration.
6. For **Authentication type**, select **User-assigned identity**.
7. In the top menu, select **Save**. App Service commits a workflow file into the chosen GitHub repository, in the `.github/workflows` directory. By default, the deployment center [creates a user-assigned identity](#) for the workflow to authenticate using Microsoft Entra (OIDC authentication). For alternative authentication options, see [Deploy to App Service using GitHub Actions](#).

[Save](#) [Discard](#) [Browse](#) [Manage publish profile](#) [Sync](#) ...

[Settings *](#) [Logs](#) [FTPS credentials](#)

Info You're now in the production slot, which is not recommended for setting up CI/CD. [Learn more](#) [X](#)

Deploy and build code from your preferred source and build provider. [Learn more](#)

Source *

[Change provider](#)

Building with GitHub Actions. [Change provider](#).

GitHub

App Service will place a GitHub Actions workflow in your chosen repository to build and deploy your app whenever there is a commit on the chosen branch. If you can't find an organization or repository, you may need to enable additional permissions on GitHub. You must have write access to your chosen GitHub repository to deploy with GitHub Actions. [Learn more](#)

Signed in as

Icephas [Change Account](#) [\(i\)](#)

Organization *

Repository *

[\(i\)](#)

Branch *

[\(i\)](#)

Build

Runtime stack

Node

Version

Node 20 LTS

Authentication settings

Select how you want your GitHub Action workflow to authenticate to Azure. If you choose user-assigned identity, the identity selected will be federated with GitHub as an authorized client and given write permissions on the app. [Learn more](#)

Authentication type *

User-assigned identity

Basic authentication



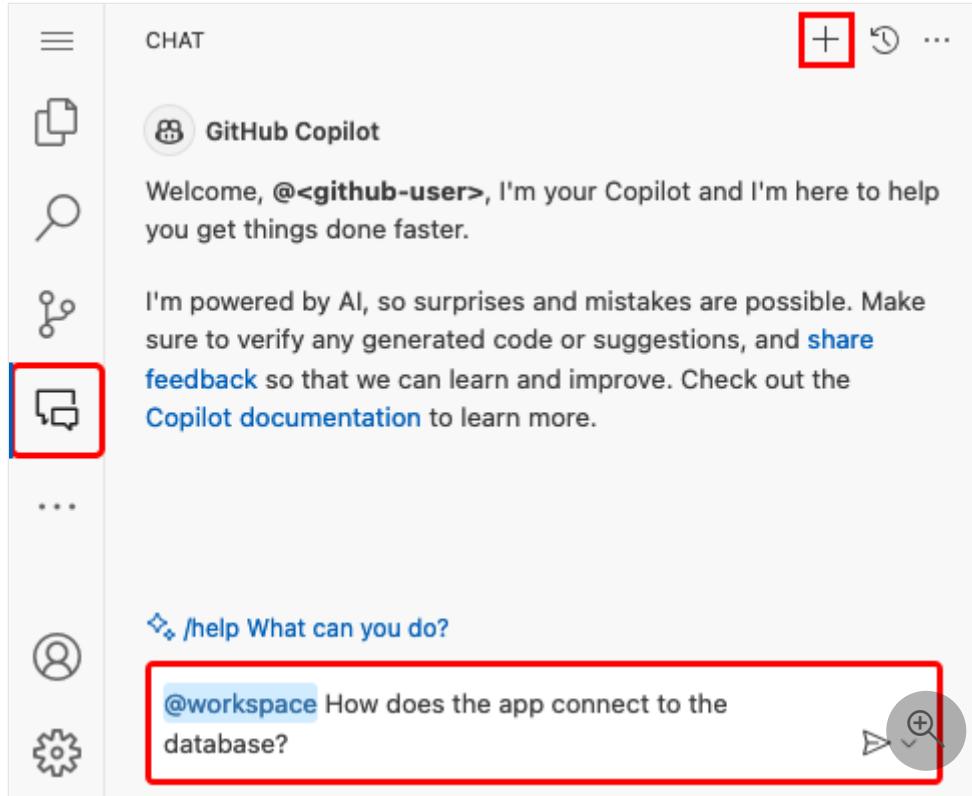
Step 3: Back in the GitHub codespace of your sample fork, run `git pull origin starter-no-infra`. This pulls the newly committed workflow file into your codespace.

The screenshot shows a GitHub Codespace interface. On the left is the Explorer sidebar with project files like .devcontainer, .github, bin, models, node_modules, public, routes, views, .gitignore, app.js, CHANGELOG.md, CONTRIBUTING.md, LICENSE.md, package-lock.json, package.json, and README.md. The README.md file is currently previewed. The main area is a terminal window titled 'bash' showing the command `git pull origin starter-no-infra` being run. The output of the command is displayed, showing the cloning of the repository and the creation of a new branch named 'starter-no-infra'. The status bar at the bottom indicates the workspace is 'urban fiesta' and the branch is 'starter-no-infra'.

Step 4 (Option 1: with GitHub Copilot):

1. Start a new chat session by selecting the **Chat** view, then selecting **+**.
2. Ask, "@workspace How does the app connect to the database?" Copilot might point you to the `app.js` file and the `mongoose.connect` call.
3. Say, "I have a connection string variable in Azure called `AZURE_COSMOS_CONNECTIONSTRING`". Copilot might give you a code suggestion similar to the one in the **Option 2: without GitHub Copilot** steps below and even tell you to make the change in `app.js`.
4. Open `app.js` in the explorer and add the code suggestion in the `getApp` method. GitHub Copilot doesn't give you the same response every time, you might need to

ask more questions to fine-tune its response. For tips, see [What can I do with GitHub Copilot in my codespace?](#).



Step 4 (Option 2: without GitHub Copilot):

1. From the explorer, open `app.js`.
2. Find the line where `mongoose.connect` is called (Line 16) and change

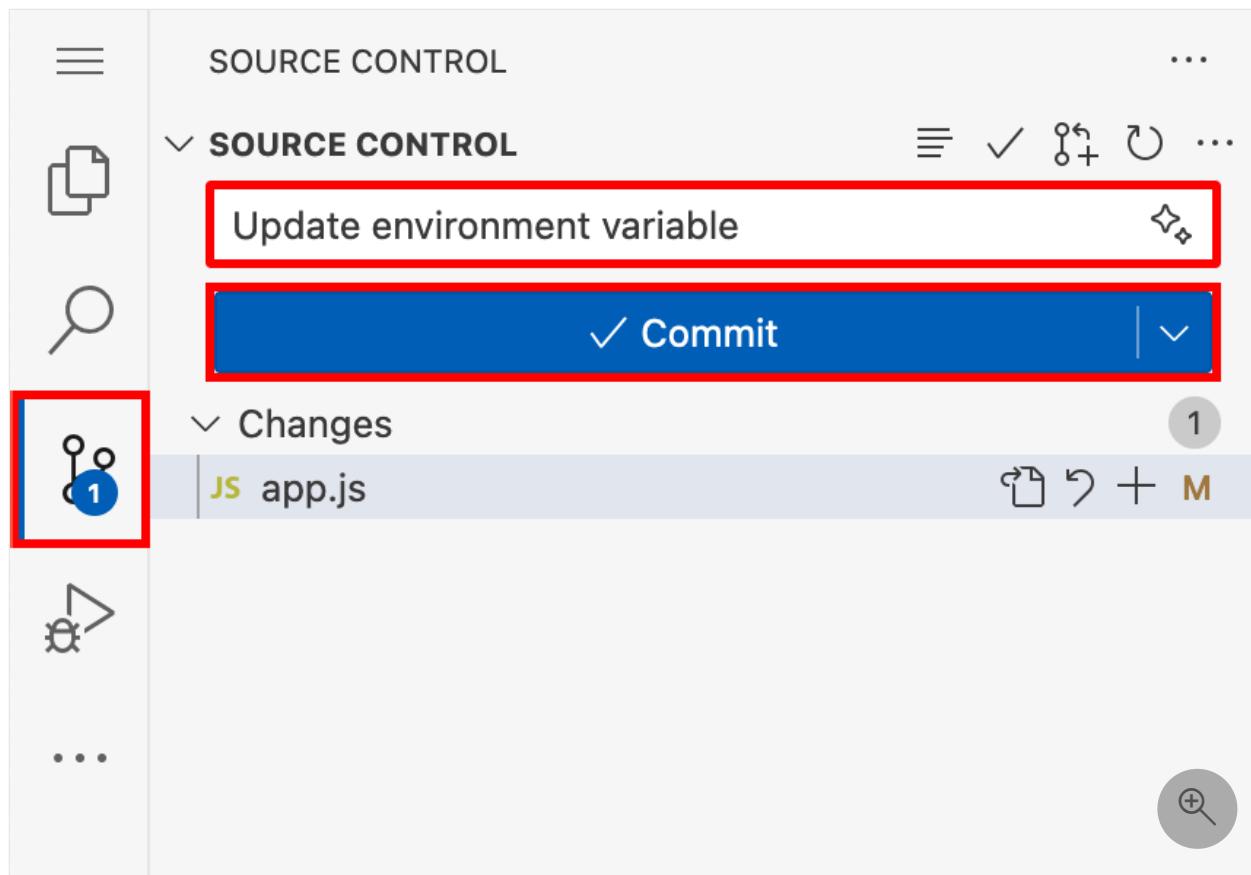
```
process.env.MONGODB_URI to process.env.AZURE_COSMOS_CONNECTIONSTRING ||  
process.env.MONGODB_URI.
```

The screenshot shows the VS Code interface with the file `app.js` open. The code editor highlights the line where the database connection is configured:

```
// Use AZURE_COSMOS_CONNECTIONSTRING if available, otherwise fall back to MONGODB_URI  
const mongoUri = process.env.AZURE_COSMOS_CONNECTIONSTRING || process.env.MONGODB_URI;
```

Step 5:

1. Select the **Source Control** extension.
2. In the textbox, type a commit message like `Update environment variable`. Or, select  and let GitHub Copilot generate a commit message for you.
3. Select **Commit**, then confirm with **Yes**.
4. Select **Sync changes 1**, then confirm with **OK**.



Step 6: Back in the Deployment Center page in the Azure portal:

1. Under the **Logs** tab, select **Refresh**. A new deployment run is already started from your committed changes.
2. In the log item for the deployment run, select the **Build/Deploy Logs** entry with the latest timestamp.

The screenshot shows the Azure portal interface for managing publish profiles. The top navigation bar includes Save, Discard, Browse, Manage publish profile, Sync, and more. Below the navigation, there are tabs for Settings, Logs (which is selected and highlighted with a red box), and FTPS credentials. Under the Logs tab, there are buttons for Refresh (highlighted with a red box) and Delete. The main content area displays a table of log entries. The columns are Time, Commit ID, Logs (with a blue link), and Commit Author. The table shows five entries for Wednesday, September 25, 2024. The third entry has its 'Logs' link highlighted with a red box. A magnifying glass icon is located in the bottom right corner of the table area.

Time	Commit ID	Logs	Commit Author
09/25 2024, 4:26:...	7bcc52d	App Logs	N/A
09/25 2024, 4:25:...	temp-d1	App Logs	N/A
09/25 2024, 4:24:...	37f9f36	Build/Deploy Lo...	Icephas
09/25 2024, 4:03:...	8eb506d	App Logs	N/A
09/25 2024, 4:01:...	139f0eb	Build/Deploy Lo...	Icephas

Step 7: You're taken to your GitHub repository and see that the GitHub action is running. The workflow file defines two separate stages, build and deploy. Wait for the GitHub run to show a status of **Complete**.

The screenshot shows the GitHub Actions interface for a workflow named 'Update environment variable #2'. The top navigation bar includes Code, Pull requests, Actions (which is selected and highlighted with a red box), Projects, Wiki, Security, Insights, and Settings. The main content area shows a summary of the run, which was triggered via push 7 minutes ago by 'Icephas pushed' to branch 'starter-no-infra'. The status is Success, total duration is 7m 10s, and there is 1 artifact. On the left, there are sections for Jobs (build, deploy), Run details, Usage, and Workflow file. The workflow details section shows a timeline of steps: build (12s) followed by deploy (6m 38s). The deploy step is linked to a URL: <http://msdocs-expressjs-mongodb-125...>. A magnifying glass icon is located in the bottom right corner of the workflow details area.

Having issues? Check the [Troubleshooting section](#).

5. Browse to the app

Step 1: In the App Service page:

1. From the left menu, select **Overview**.
2. Select the URL of your app. You can also navigate directly to `https://<app-name>.azurewebsites.net`.

The screenshot shows the Azure App Service Overview page. The left sidebar has a red box around the 'Overview' item. The main area shows the app's details under 'Essentials'. The 'Default domain' field is highlighted with a red box and contains the value 'msdocs-expressjs-mongodb-125-fuaf....'. Other details include: Status (Running), App Service Plan (ASP-msdocsexpressjsmongodbtutorial-be), Operating System (Linux), Health Check (Not Configured), GitHub Project (<https://github.com/.../msdocs-nodejs-...>), and Tags (edit).

Step 2: Add a few tasks to the list. Congratulations, you're running a secure data-driven Node.js app in Azure App Service.

The screenshot shows the 'Azure Todo List' application. At the top, there's a logo and the text 'Azure Todo List' and 'ExpressJS/MongoDB Sample Application'. Below that is a 'New task' input field with placeholder 'Enter new task here' and a green 'Add task' button. The main area is divided into 'Current tasks' and 'Completed tasks'. The 'Current tasks' table has columns: Task, Created, and Action. It lists three tasks: 'Create Node.js app and MongoDB in Azure' (Created 2024-09-25), 'Deploy data-driven app' (Created 2024-09-25), and 'That's it!' (Created 2024-09-25). Each task has a 'Complete' button (blue) and a 'Delete' button (red). The 'Completed tasks' table has columns: Task, Created, and Completed. It lists the same three tasks, all of which are marked as completed.

6. Stream diagnostic logs

Azure App Service captures all messages logged to the console to assist you in diagnosing issues with your application. The sample app outputs console log messages in each of its endpoints to demonstrate this capability. For example, the `get` endpoint outputs a message about the number of tasks retrieved from the database and an error message appears if something goes wrong.

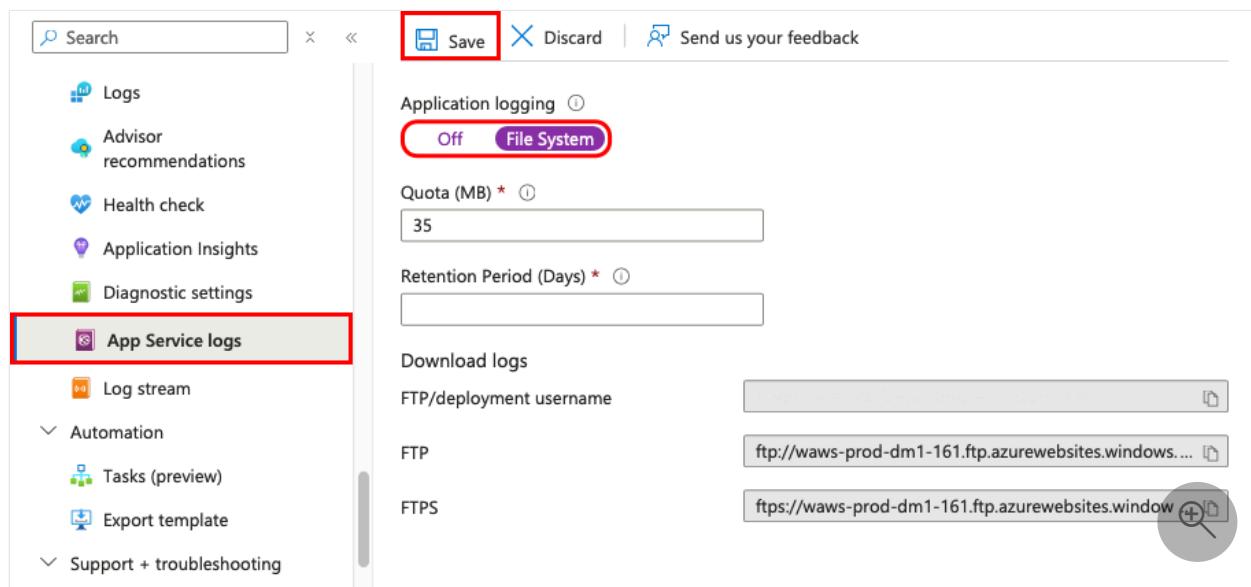
```
JavaScript

router.get('/', function(req, res, next) {
  Task.find()
    .then((tasks) => {
      const currentTasks = tasks.filter(task => !task.completed);
      const completedTasks = tasks.filter(task => task.completed === true);

      console.log(`Total tasks: ${tasks.length}  Current tasks:
${currentTasks.length}  Completed tasks: ${completedTasks.length}`)
      res.render('index', { currentTasks, completedTasks: completedTasks });
    })
    .catch((err) => {
      console.log(err);
      res.send('Sorry! Something went wrong.');
    });
});
```

Step 1: In the App Service page:

1. From the left menu, select **App Service logs**.
2. Under **Application logging**, select **File System**.
3. In the top menu, select **Save**.



Step 2: From the left menu, select **Log stream**. You see the logs for your app, including platform logs and logs from inside the container.

The screenshot shows the Azure App Service Kudu logs interface. On the left, there's a sidebar with various monitoring and diagnostic tools: CORS, Monitoring, Alerts, Metrics, Logs, Advisor recommendations, Health check, Application Insights, Diagnostic settings, App Service logs, and Log stream. The 'Log stream' option is highlighted with a red box. The main area displays a log history with several entries. Two specific log entries are highlighted with red boxes:

```
2024-09-25T14:47:21.4695825Z npm info using node@v20.15.1
2024-09-25T14:47:41.7038078Z
2024-09-25T14:47:41.7039075Z > todolist@0.0.0 start
2024-09-25T14:47:41.7039139Z > node ./bin/www
2024-09-25T14:47:41.7039181Z
2024-09-25T14:48:08.9337699Z Connected to database
2024-09-25T14:48:10.8773088Z [0mGET /robots933456.txt [33m404[0m 1857.556 ms -
1498[0m
2024-09-25T14:48:10.9142072Z [0mGET /robots933456.txt [0m-[0m - ms - -[0m
2024-09-25T14:48:11.2437727Z [0mGET /robots933456.txt [33m404[0m 327.601 ms -
1498[0m
2024-09-25T14:50:27.4356668Z Total tasks: 3 Current tasks: 3 Completed
tasks: 0
2024-09-25T14:50:32.1175548Z [0mGET / [36m304[0m 2739.801 ms - -[0m
2024-09-25T14:50:32.1187265Z [0mGET /stylesheets/style.css [36m304[0m 46.677 ms
- -[0m
2024-09-25T14:50:32.1189729Z [0mGET /css/bootstrap.css [36m304[0m 65.376 ms - -
[0m
2024-09-25T14:50:32.1189886Z [0mGET /js/bootstrap.min.js [36m304[0m 40.191 ms -
-[0m
2024-09-25T14:50:32.1189928Z [0mGET /images/Azure-A-48px-product.svg [32m200[0m
116.503 ms - 1820[0m
2024-09-25T14:56:43.4739352Z Total tasks: 3 Current tasks: 3 Completed
tasks: 0
2024-09-25T14:56:43.9449941Z [0mGET / [36m304[0m 1164.005 ms - -[0m
2024-09-25T14:56:44.1600179Z [0mGET /css/bootstrap.css [36m304[0m 17.665 ms - -
[0m
2024-09-25T14:56:44.4262551Z [0mGET /stylesheets/style.css [36m304[0m 2.982 ms
- -[0m
2024-09-25T14:56:44.4456658Z [0mGET /js/bootstrap.min.js [36m304[0m 6.345 ms -
-[0m
2024-09-25T14:56:44.5733128Z [0mGET /images/Azure-A-48px-product.svg [36m304[0m
2.145 ms - -[0m
2024-09-25T14:56:44.7933181Z [0mGET /favicon.ico [33m404[0m 74.127 ms - 1498[0m
```

7. Inspect deployed files using Kudu

Azure App Service provides a web-based diagnostics console named [Kudu](#) that lets you examine the server hosting environment for your web app. Using Kudu, you can view the files deployed to Azure, review the deployment history of the application, and even open an SSH session into the hosting environment.

Step 1: In the App Service page:

1. From the left menu, select **Advanced Tools**.
2. Select **Go**. You can also navigate directly to <https://<app-name>.scm.azurewebsites.net>.

Search

- Quotas
- Change App Service plan
- Development Tools
 - SSH
 - Advanced Tools

Advanced Tools provides a collection of developer oriented tools and extensibility points for your App Service Apps. [Learn more](#)

Go →

API Management

API definition

CORS

Monitoring

Alerts

Metrics

Step 2: In the Kudu page, select **Deployments**.

Azure App Service Environment SSH Bash Log stream user@example.com

Environment

Build 1.0.0.7 (e59ed50ca2)
Site up time 00.00:38:08
Site folder /home
Temp folder /tmp/

REST API

(works best when using a JSON viewer extension)

- App Settings
- Deployments
- Source control info
- Files
- Current Docker logs ([Download as zip](#))

Browse Directory

- Deployment Logs
- Site wwwroot

More information about Kudu can be found on the [wiki](#).

If you deploy code to App Service using Git or zip deploy, you see a history of deployments of your web app.

```
[{"id": "c6bf4a4f-15ce-41fc-a05b-81bcfc86c9", "status": 4, "status_text": "", "author_email": "N/A", "author": "N/A", "deployer": "GITHUB_ZIP_DEPLOY", "message": "", {"\\"type\\": \"deployment\", \\"sha\\": \"8164af05dc4ab3ccc8412d80a21f5cea63b00e7\", \\"repoName\\": \"lcep has/msdocs-nodejs-mongodb-azure-sample-app\", \\"slotName\\": \"Production\", \\"commitMessage\\": \"Add or update the Azure App Service build and deployment workflow config\"}, \"progress\": \"\", \"received_time\": \"2022-09-06T14:22:33.3162747Z\", \"start_time\": \"2022-09-06T14:22:34.5823201Z\", \"end_time\": \"2022-09-06T14:25:42.4832809Z\", \"last_success_end_time\": \"2022-09-06T14:25:42.4832809Z\", \"complete\": true, \"active\": true, \"is_temp\": false, \"is_READONLY\": true, \"url\": \"https://msdocs-expressjs-mongodb-234.scm.azurewebsites.net/api/deployments/c6bf4a4f-15ce-41fc-a05b-81bcfc86c9\", \"log_url\": \"https://msdocs-expressjs-mongodb-234.scm.azurewebsites.net/api/deployments/c6bf4a4f-15ce-41fc-a05b-81bcfc86c9/log\", \"site_name\": \"msdocs-expressjs-mongodb-234\", \"build_summary\": {\"errors\": [], \"warnings\": []}}]
```



Step 3: Go back to the Kudu homepage and select Site wwwroot.



Environment

Build 1.0.0.7 ([e59ed50ca2](#))

Site up time 00:00:38:10

Site folder /home

Temp folder /tmp/

REST API

(works best when using a JSON viewer extension)

- [App Settings](#)
- [Deployments](#)
- [Source control info](#)
- [Files](#)
- [Current Docker logs](#) ([Download as zip](#))

Browse Directory

- [Deployment Logs](#)
- [Site wwwroot](#)

More information about Kudu can be found on the [wiki](#).



You can see the deployed folder structure and select to browse and view the files.

Index of /wwwroot/

Name	Size	Last Modified
bin/		09/06/2022 14:22:39 +00:00
config/		09/06/2022 14:22:39 +00:00
models/		09/06/2022 14:22:39 +00:00
node_modules/		09/06/2022 14:25:41 +00:00
public/		09/06/2022 14:25:41 +00:00
routes/		09/06/2022 14:25:41 +00:00
views/		09/06/2022 14:25:41 +00:00
app.js	1,977	09/06/2022 14:12:10 +00:00
CHANGELOG.md	135	09/06/2022 14:12:10 +00:00
CONTRIBUTING.md	3,941	09/06/2022 14:12:10 +00:00
LICENSE.md	1,140	09/06/2022 14:12:10 +00:00
package-lock.json	152,081	09/06/2022 14:12:10 +00:00
package.json	480	09/06/2022 14:12:10 +00:00
README.md	453	09/06/2022 14:12:10 +00:00



8. Clean up resources

When you're finished, you can delete all of the resources from your Azure subscription by deleting the resource group.

Step 1: In the search bar at the top of the Azure portal:

1. Enter the resource group name.
2. Select the resource group.

The screenshot shows the Microsoft Azure portal interface. At the top, there's a search bar with the text "msdocs-expressjs-mongodb-tutorial". Below the search bar, there are several navigation tabs: "All" (which is highlighted in blue), "Resource Groups (1)", "Documentation (4)", "Services (0)", "Resources (0)", "Marketplace (0)", and "Azure Active Directory (0)". On the left sidebar, under "Azure services", there are links for "Create a resource", "Subscription", "Azure Cosmos DB API for...", and "Azure Cosmos DB". Under "Resources", there are "Recent" and "Favorite" sections. On the right side, there's a sidebar with "Storage accounts" and a "Give feedback" button.

Step 2: In the resource group page, select **Delete resource group**.

This screenshot shows the "Delete resource group" page for the resource group "msdocs-expressjs-mongodb-tutorial". At the top, there are buttons for "Create", "Manage view", and "Delete resource group" (which is highlighted with a red box). Below that, there's a section titled "Essentials" with the following details:

Subscription (move)	: Visual Studio Enterprise Subsc...
Subscription ID	: 00000000-0000-0000-0000-0...
Deployments	: 7 Succeeded
Location	: UK West
Tags (edit)	: Click here to add tags

Below the essentials section, there are tabs for "Resources" (which is underlined) and "Recommendations (2)". At the bottom, there are filters for "Filter for any field...", "Add filter", "More (2)", and buttons for "Show hidden types" and "List view".

Step 3:

1. Enter the resource group name to confirm your deletion.
2. Select **Delete**.

 Are you sure you want to delete "msdoc... X

! Warning! Deleting the "msdocs-expressjs-mongodb-tutorial" resource group is irreversible. The action you're about to take can't be undone. Going further will delete this resource group and all the resources in it permanently.

TYPE THE RESOURCE GROUP NAME:

msdocs-expressjs-mongodb-tutorial 

AFFECTED RESOURCES

There are 8 resources in this resource group that will be deleted.

Name	Type	Location
 ASP-msdocsexpressjsmongodb...	App Service plan	UK West
 msdocs-expressjs-mongodb-234	App Service	UK West
 msdocs-expressis-monaodb-23...	Private endpoint	UK West

Delete **Cancel**  :

Troubleshooting

- The portal deployment view for Azure Cosmos DB shows a Conflict status
- The browser page of the deployed app says "Something went wrong."

The portal deployment view for Azure Cosmos DB shows a Conflict status

Depending on your subscription and the region you select, you might see the deployment status for Azure Cosmos DB to be `Conflict`, with the following message in Operation details:

`Sorry, we are currently experiencing high demand in <region> region, and cannot fulfill your request at this time.`

The error is most likely caused by a limit on your subscription for the region you select. Try choosing a different region for your deployment.

The browser page of the deployed app says "Something went wrong."

You probably still need to make the connection string changes in your application code. See [4. Deploy sample code](#).

Frequently asked questions

- How much does this setup cost?
- How do I connect to the Azure Cosmos DB server that's secured behind the virtual network with other tools?
- How does local app development work with GitHub Actions?
- Why is the GitHub Actions deployment so slow?
- I don't have permissions to create a user-assigned identity
- What can I do with GitHub Copilot in my codespace?

How much does this setup cost?

Pricing for the created resources is as follows:

- The App Service plan is created in **Basic** tier and can be scaled up or down. See [App Service pricing](#).
- The Azure Cosmos DB server is created in a single region and can be distributed to other regions. See [Azure Cosmos DB pricing](#).
- The virtual network doesn't incur a charge unless you configure extra functionality, such as peering. See [Azure Virtual Network pricing](#).
- The private DNS zone incurs a small charge. See [Azure DNS pricing](#).

How do I connect to the Azure Cosmos DB server that's secured behind the virtual network with other tools?

- For basic access from a command-line tool, you can run `mongosh` from the app's SSH terminal. The app's container doesn't come with `mongosh`, so you must [install it manually](#). Remember that the installed client doesn't persist across app restarts.
- To connect from a MongoDB GUI client, your machine must be within the virtual network. For example, it could be an Azure VM that's connected to one of the subnets, or a machine in an on-premises network that has a [site-to-site VPN](#) connection with the Azure virtual network.
- To connect from the MongoDB shell from the Azure Cosmos DB management page in the portal, your machine must also be within the virtual network. You could instead open the Azure Cosmos DB server's firewall for your local machine's IP address, but it increases the attack surface for your configuration.

How does local app development work with GitHub Actions?

Take the autogenerated workflow file from App Service as an example, each `git push` kicks off a new build and deployment run. From a local clone of the GitHub repository, you make the desired updates push it to GitHub. For example:

terminal

```
git add .
git commit -m "<some-message>"
git push origin main
```

Why is the GitHub Actions deployment so slow?

The autogenerated workflow file from App Service defines build-then-deploy, two-job run. Because each job runs in its own clean environment, the workflow file ensures that the `deploy` job has access to the files from the `build` job:

- At the end of the `build` job, [upload files as artifacts ↗](#).
- At the beginning of the `deploy` job, download the artifacts.

Most of the time taken by the two-job process is spent uploading and download artifacts. If you want, you can simplify the workflow file by combining the two jobs into one, which eliminates the need for the upload and download steps.

I don't have permissions to create a user-assigned identity

See [Set up GitHub Actions deployment from the Deployment Center](#).

What can I do with GitHub Copilot in my codespace?

You might notice that the GitHub Copilot chat view was already there for you when you created the codespace. For your convenience, we include the GitHub Copilot chat extension in the container definition (see `.devcontainer/devcontainer.json`). However, you need a [GitHub Copilot account ↗](#) (30-day free trial available).

A few tips for you when you talk to GitHub Copilot:

- In a single chat session, the questions and answers build on each other and you can adjust your questions to fine-tune the answer you get.
- By default, GitHub Copilot doesn't have access to any file in your repository. To ask questions about a file, open the file in the editor first.

- To let GitHub Copilot have access to all of the files in the repository when preparing its answers, begin your question with `@workspace`. For more information, see [Use the `@workspace` agent](#).
- In the chat session, GitHub Copilot can suggest changes and (with `@workspace`) even where to make the changes, but it's not allowed to make the changes for you. It's up to you to add the suggested changes and test it.

Here are some other things you can say to fine-tune the answer you get:

- `@workspace` Where is MONGODB_URI defined?
- Which file do I make the change in?
- Will this change break my app when running locally?

Next steps

[JavaScript on Azure developer center](#)

[Configure Node.js app in App Service](#)

[Secure with custom domain and certificate](#)

Feedback

Was this page helpful?

 Yes

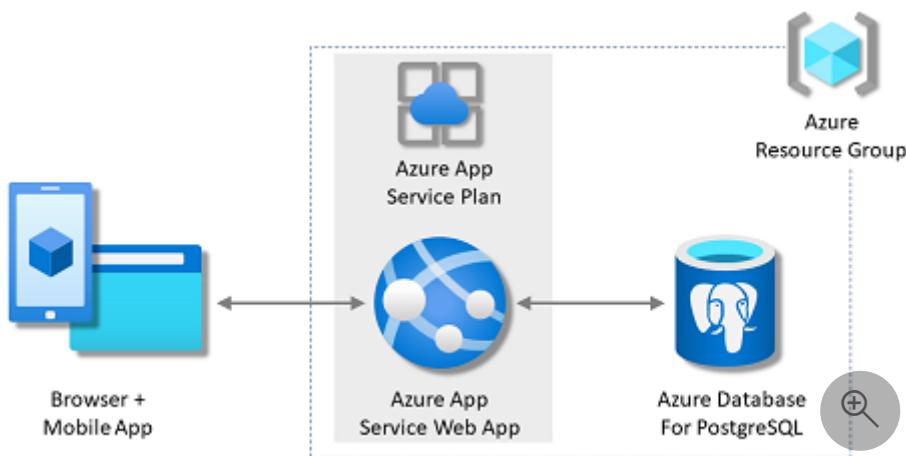
 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

Deploy a Python (Flask) web app with PostgreSQL in Azure

Article • 01/29/2025

In this tutorial, you'll deploy a data-driven Python web app ([Flask](#)) to [Azure App Service](#) with the [Azure Database for PostgreSQL](#) relational database service. Azure App Service supports [Python](#) in a Linux server environment. If you want, see the [Django tutorial](#) or the [FastAPI tutorial](#) instead.



In this tutorial, you learn how to:

- ✓ Create a secure-by-default App Service, PostgreSQL, and Redis cache architecture.
- ✓ Secure connection secrets using a managed identity and Key Vault references.
- ✓ Deploy a sample Python app to App Service from a GitHub repository.
- ✓ Access App Service connection strings and app settings in the application code.
- ✓ Make updates and redeploy the application code.
- ✓ Generate database schema by running database migrations.
- ✓ Stream diagnostic logs from Azure.
- ✓ Manage the app in the Azure portal.
- ✓ Provision the same architecture and deploy by using Azure Developer CLI.
- ✓ Optimize your development workflow with GitHub Codespaces and GitHub Copilot.

Prerequisites

- An Azure account with an active subscription. If you don't have an Azure account, you [can create one for free](#).
- A GitHub account. you can also [get one for free](#).
- Knowledge of Python with Flask development.

- (Optional) To try GitHub Copilot, a [GitHub Copilot account](#). A 30-day free trial is available.

Skip to the end

If you just want to see the sample app in this tutorial running in Azure, just run the following commands in the [Azure Cloud Shell](#), and follow the prompt:

Bash

```
mkdir msdocs-flask-postgresql-sample-app
cd msdocs-flask-postgresql-sample-app
azd init --template msdocs-flask-postgresql-sample-app
azd up
```

1. Run the sample

First, you set up a sample data-driven app as a starting point. For your convenience, the [sample repository](#), includes a [dev container](#) configuration. The dev container has everything you need to develop an application, including the database, cache, and all environment variables needed by the sample application. The dev container can run in a [GitHub codespace](#), which means you can run the sample on any computer with a web browser.

ⓘ Note

If you are following along with this tutorial with your own app, look at the *requirements.txt* file description in [README.md](#) to see what packages you'll need.

Step 1: In a new browser window:

1. Sign in to your GitHub account.
2. Navigate to <https://github.com/Azure-Samples/msdocs-flask-postgresql-sample-app/fork>.
3. Unselect **Copy the main branch only**. You want all the branches.
4. Select **Create fork**.

The screenshot shows the GitHub fork creation interface for the repository `msdocs-flask-postgresql-sample-app`. The top navigation bar includes links for Code, Issues, Pull requests, Actions, Projects, Security, and Insights. The 'Code' tab is selected. The main section is titled 'Create a new fork'. A note explains that a fork is a copy of a repository, allowing experimentation without affecting the original project. It also states that required fields are marked with an asterisk (*). The 'Owner' dropdown is set to the user's account, and the 'Repository name' field contains `msdocs-flask-postgresql-sam`, with a note indicating that the name is available. A note also says that forks are typically named the same as the upstream repository. Below these fields is a 'Description (optional)' input area, which is currently empty. There is a checkbox for 'Copy the main branch only', which is unchecked. A note below it encourages contributing back to the upstream repository by adding your own branch, with a link to 'Learn more'. A note also indicates that the fork is being created in the user's personal account. At the bottom right is a large red-bordered 'Create fork' button.

Step 2: In the GitHub fork:

1. Select `main > starter-no-infra` for the starter branch. This branch contains just the sample project and no Azure-related files or configuration.
2. Select **Code > Create codespace on starter-no-infra**. The codespace takes a few minutes to set up, and it runs `pip install -r requirements.txt` for your repository at the end.

This branch is up to date
Azure-Samples/msdocs-1

Contribute

add copilot ext .devcontainer .github azureproject migrations static templates .env

starter-no-infra

Local Codespaces

Codespaces Your workspaces in the cloud + ...

No codespaces You don't have any codespaces with this repository checked out

Create codespace on starter-no-infra

Learn more about codespaces...

Codespace usage for this repository is paid for by Icephas.

Remove spurious line. 3 years ago convert to service connecto... 20 hours ago

About

No description, website, or topics provided.

Readme MIT license Code of conduct Activity 0 stars 0 watching 0 forks

Releases

No releases published Create a new release

Packages

No packages published

Step 3: In the codespace terminal:

1. Run database migrations with `flask db upgrade`.
2. Run the app with `flask run`.
3. When you see the notification `Your application running on port 5000 is available.`, select **Open in Browser**. You should see the sample application in a new browser tab. To stop the application, type `Ctrl + C`.

💡 Tip

You can ask [GitHub Copilot](#) about this repository. For example:

- @workspace *What does this project do?*
- @workspace *What does the .devcontainer folder do?*

Having issues? Check the [Troubleshooting section](#).

2. Create App Service and PostgreSQL

In this step, you create the Azure resources. The steps used in this tutorial create a set of secure-by-default resources that include App Service and Azure Database for PostgreSQL. For the creation process, you specify:

- The **Name** for the web app. It's used as part of the DNS name for your app in the form of `https://<app-name>-<hash>.azurewebsites.net`.
- The **Region** to run the app physically in the world. It's also used as part of the DNS name for your app.
- The **Runtime stack** for the app. It's where you select the version of Python to use for your app.
- The **Hosting plan** for the app. It's the pricing tier that includes the set of features and scaling capacity for your app.

- The **Resource Group** for the app. A resource group lets you group (in a logical container) all the Azure resources needed for the application.

Sign in to the [Azure portal](#) and follow these steps to create your Azure App Service resources.

Step 1: In the Azure portal:

1. Enter "web app database" in the search bar at the top of the Azure portal.
2. Select the item labeled **Web App + Database** under the **Marketplace** heading. You can also navigate to the [creation wizard](#) directly.

The screenshot shows the Microsoft Azure portal interface. At the top, there's a search bar with the text "web app database". Below the search bar, the "Marketplace" section is visible, featuring a list of services. The "Web App + Database" option is highlighted with a red box. To the left, there's a sidebar with sections for "Azure services" (including "Create a resource", "Resource groups", "Microsoft Entra ID", and "DNS zones") and "Resources" (Recent and Favorite). On the right, there are various icons for AI, Managed Identities, and a feedback button.

Step 2: In the Create Web App + Database page, fill out the form as follows.

1. **Resource Group:** Select **Create new** and use a name of **msdocs-flask-postgres-tutorial**.
2. **Region:** Any Azure region near you.
3. **Name:** **msdocs-python-postgres-XYZ**.
4. **Runtime stack:** **Python 3.12**.
5. **Database:** **PostgreSQL - Flexible Server** is selected by default as the database engine. The server name and database name are also set by default to appropriate values.
6. **Add Azure Cache for Redis?: No.**
7. **Hosting plan:** **Basic**. When you're ready, you can [scale up](#) to a production pricing tier.
8. Select **Review + create**.
9. After validation completes, select **Create**.

Home >

Create Web App + Database

X

Basics Tags Review + create

This template will create a secure by default configuration where the only publicly accessible endpoint will be your app following the recommended security best practices. [Learn more](#)

Project Details

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

Subscription *

Resource Group * [Create new](#)

Region *

Web App Details

Name [More about this update](#)

Secure unique default hostname on. [More about this update](#)

Runtime stack *

Database

Info Database access will be locked down and not exposed to the public internet. This is in compliance with recommended best practices for security.

Engine *

Server name *

Database name *

Azure Cache for Redis

Add Azure Cache for Redis? No

Hosting

Hosting plan * Basic - For hobby or research purposes
 Standard - General purpose production apps

Review + create < Previous Next : Tags > 

Step 3: The deployment takes a few minutes to complete. Once deployment completes, select the **Go to resource** button. You're taken directly to the App Service app, but the following resources are created:

- **Resource group:** The container for all the created resources.

- **App Service plan:** Defines the compute resources for App Service. A Linux plan in the *Basic* tier is created.
- **App Service:** Represents your app and runs in the App Service plan.
- **Virtual network:** Integrated with the App Service app and isolates back-end network traffic.
- **Network interfaces:** Represents private IP addresses, one for each of the private endpoints.
- **Azure Database for PostgreSQL flexible server:** Accessible only from within the virtual network. A database and a user are created for you on the server.
- **Private DNS zones:** Enables DNS resolution of the key vault and the database server in the virtual network.

 Your deployment is complete

 Deployment name : Microsoft.Web-WebAppDatabase-Portal-afa69d9d-97bc
 Subscription :
 Resource group : [msdocs-python-postgres-tutorial](#)
 Start time : 11/29/2023, 10:17:05 AM
 Correlation ID :

Deployment details

Next steps

[Go to resource](#)

Give feedback 

Tell us about your experience with deployment 

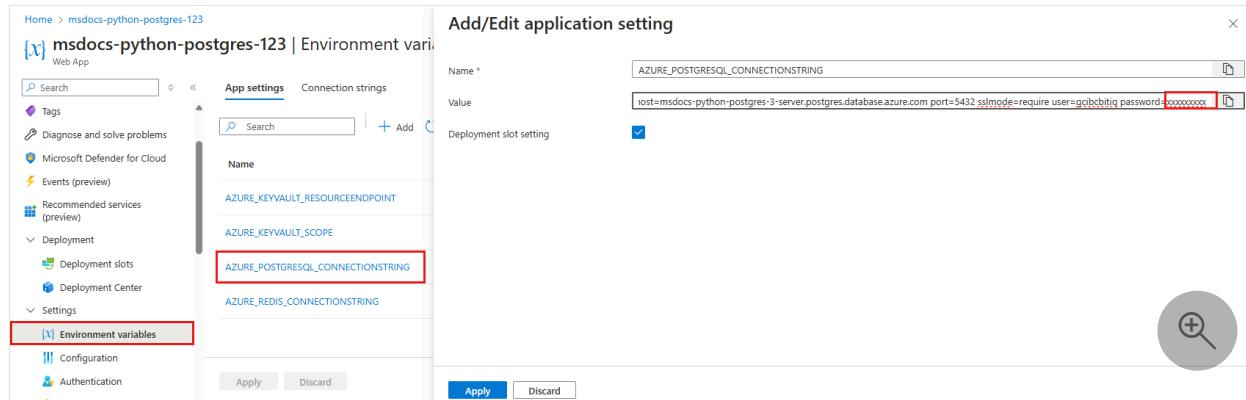
3. Secure connection secrets

The creation wizard generated the connectivity variables for you already as [app settings](#). However, the security best practice is to keep secrets out of App Service completely. You'll move your secrets to a key vault and change your app setting to [Key Vault references](#) with the help of Service Connectors.

Step 1: Retrieve the existing connection string

1. In the left menu of the App Service page, select **Settings > Environment variables**.
2. Select **AZURE_POSTGRESQL_CONNECTIONSTRING**.
3. In **Add/Edit application setting**, in the **Value** field, find the *password=* part at the end of the string.
4. Copy the password string after *Password=* for use later. This app setting lets you connect to the Postgres database secured behind a private endpoint. However, the

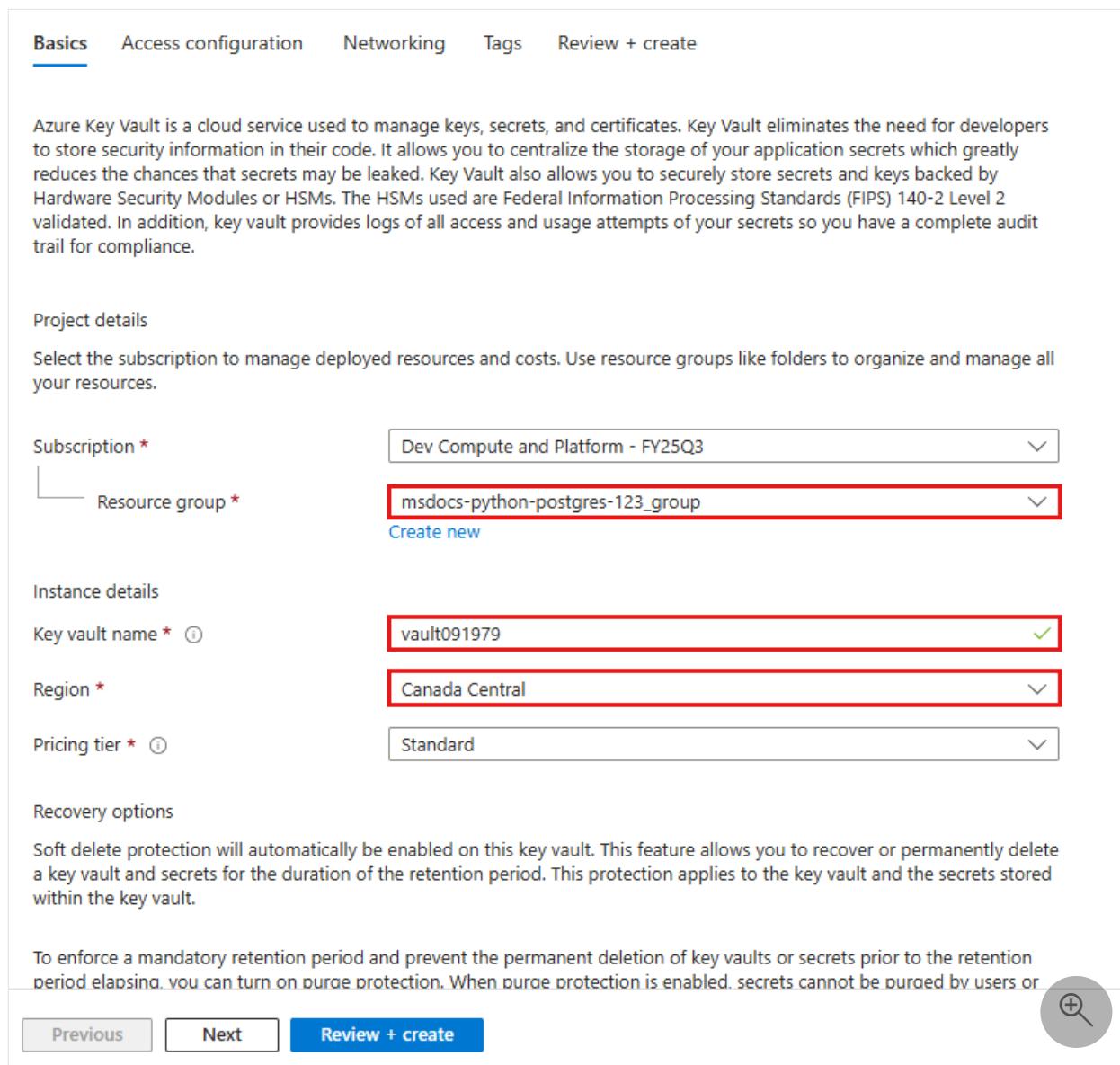
secret is saved directly in the App Service app, which isn't the best. You'll change this.



The screenshot shows the 'Environment variables' section of the Azure App Service configuration. A new variable 'AZURE_POSTGRESQL_CONNECTIONSTRING' is being added with the value 'host=microsoft-123.postgres.database.azure.com;port=5432;sslmode=require;user=gciabc1q;password=xxxxxxxxx'. Other variables like 'AZURE_KEYVAULT_RESOURCEENDPOINT' and 'AZURE_KEYVAULT_SCOPE' are also listed.

Step 2: Create a key vault for secure management of secrets

1. In the top search bar, type "key vault", then select Marketplace > Key Vault.
2. In Resource Group, select msdocs-python-postgres-tutorial.
3. In Key vault name, type a name that consists of only letters and numbers.
4. In Region, set it to the same location as the resource group.



The screenshot shows the 'Basics' step of the Azure Key Vault creation wizard. It includes fields for Subscription (Dev Compute and Platform - FY25Q3), Resource group (msdocs-python-postgres-123_group), Key vault name (vault091979), Region (Canada Central), and Pricing tier (Standard). The 'Review + create' button is at the bottom.

Step 3: Secure the key vault with a Private Endpoint

1. Select the **Networking** tab.
2. Unselect **Enable public access**.
3. Select **Create a private endpoint**.
4. In **Resource Group**, select **msdocs-python-postgres-tutorial**.
5. In the dialog, in **Location**, select the same location as your App Service app.
6. In **Name**, type **msdocs-python-postgres-XYZVaultEndpoint**.
7. In **Virtual network**, select **msdocs-python-postgres-XYZVnet**.
8. In **Subnet**, **msdocs-python-postgres-XYZSubnet**.
9. Select **OK**.
10. Select **Review + create**, then select **Create**. Wait for the key vault deployment to finish. You should see "Your deployment is complete."

The screenshot shows the Azure portal interface for creating a key vault. On the left, the 'Create a key vault' blade is open, showing tabs for Basics, Access configuration, Networking (which is highlighted with a red box), and Tags. Under the Networking tab, there's a note about connecting publicly or privately and a checkbox for 'Enable public access' which is unchecked. Below that is a 'Private endpoint' section with a '+ Create a private endpoint' button (also highlighted with a red box). On the right, a 'Create private endpoint' dialog is displayed. It includes fields for Subscription (set to 'Dev Compute and Platform - FY25Q3'), Resource group (set to 'msdocs-python-postgres-123_group'), Location (set to 'Canada Central'), Name (set to 'msdocs-python-postgres-234VaultEndpoint'), and Target sub-resource (set to 'Vault'). Under 'Networking', it shows the selected Virtual network ('msdocs-python-postgres-123Vnet') and Subnet ('msdocs-python-postgres-123Subnet'). A note indicates that if a Network Security Group (NSG) is enabled on the subnet, it will be disabled for private endpoints. The 'Private DNS integration' section shows a Private DNS Zone ('privatelink.vaultcore.azure.net') and a 'Yes' button for integrating with a private DNS zone. At the bottom of the dialog are 'OK' and 'Discard' buttons, with 'OK' being highlighted with a red box.

Step 4: Configure the PostgreSQL connector

1. In the top search bar, type *msdocs-python-postgres*, then select the App Service resource called **msdocs-python-postgres-XYZ**.
2. In the App Service page, in the left menu, select **Settings > Service Connector**. There's already a connector, which the app creation wizard created for you.
3. Select checkbox next to the PostgreSQL connector, then select **Edit**.
4. In **Client type**, select **Django**. Even though you have a Flask app, the **Django client type in the PostgreSQL service connector** gives you database variables in separate settings instead of one connection string. The separate variables are easier for you to use in your application code, which uses **SQLAlchemy** to connect to the database.

5. Select the **Authentication** tab.
6. In **Password**, paste the password you copied earlier.
7. Select **Store Secret in Key Vault**.
8. Under **Key Vault Connection**, select **Create new**. A **Create connection** dialog is opened on top of the edit dialog.

The screenshot shows the Azure portal interface for creating a Service Connector. On the left, the navigation menu is visible with the 'Service Connector' item selected. The main area shows a list of target services: 'DB for PostgreSQL flexible server' (selected), 'Cache for Redis', and 'Key Vault'. The 'Key Vault' entry has 'mangesh-key-vault-python' listed under 'Resource name'. On the right, the 'defaultConnector' dialog is open, specifically the 'Authentication' tab. It shows the 'Connection string' authentication type selected. Below it, the 'Continue with...' section offers 'Database credentials' and 'Key Vault'. The 'Database credentials' option is selected. The 'Username' field contains 'gibcbcbitiq' and the 'Password' field contains a redacted password. The 'Store Secret In Key Vault' checkbox is checked. The 'Key Vault Connection' dropdown shows 'mangesh-key-vault-python (keyvault_362d5)' and the 'Create new' button is highlighted with a red box.

Step 5: Establish the Key Vault connection

1. In the **Create connection** dialog for the Key Vault connection, in **Key Vault**, select the key vault you created earlier.
2. Select **Review + Create**.
3. When validation completes, select **Create**.

Create connection

Basics Networking Review + Create

Select the service instance and client type.

Service type * ⓘ

Key Vault

Connection name * ⓘ

keyvault_cc22c

Subscription * ⓘ

Dev Compute and Platform - FY25Q3

Key vault * ⓘ

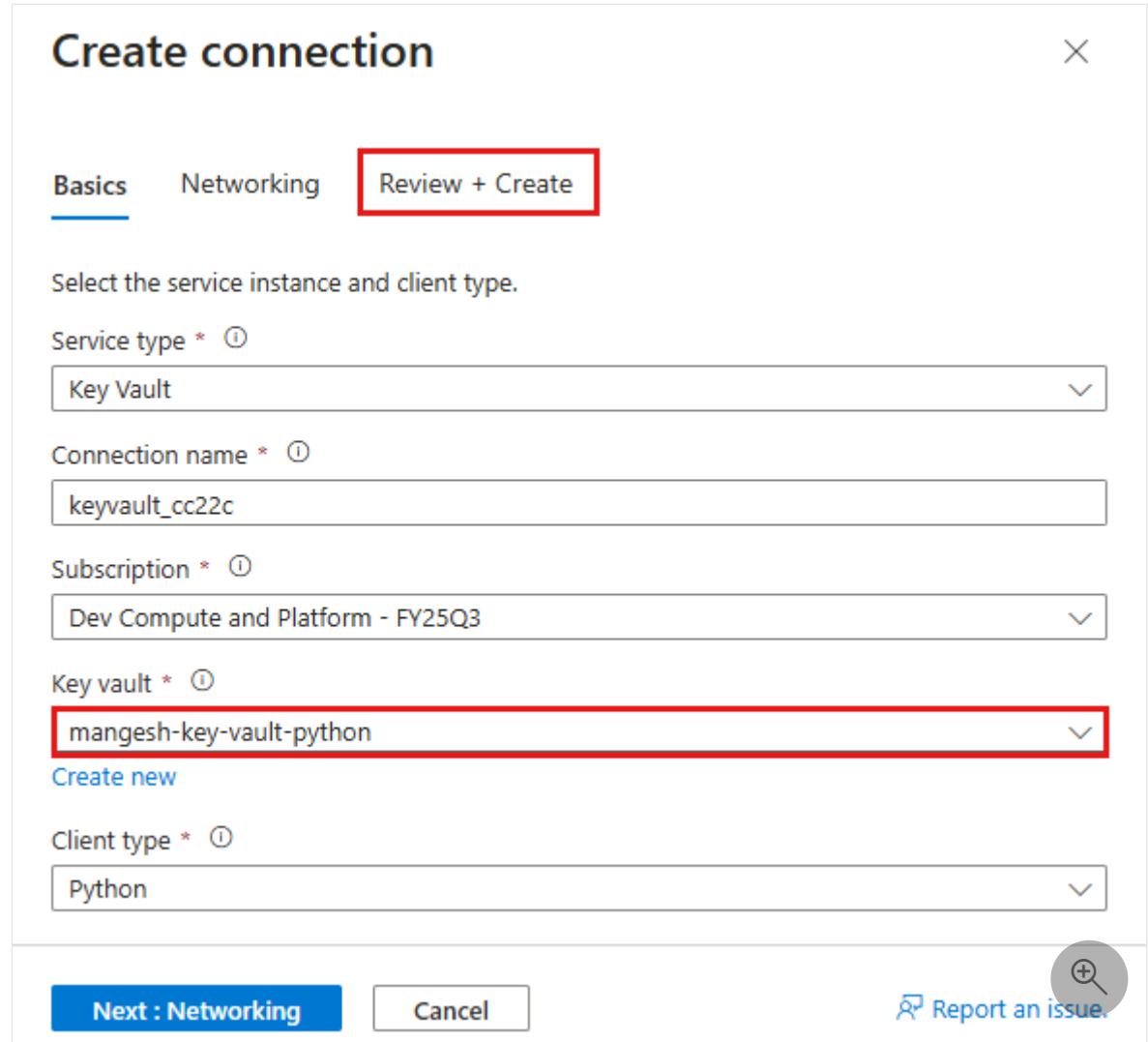
mangesh-key-vault-python

Create new

Client type * ⓘ

Python

Next : Networking Cancel Report an issue.



Step 6: Finalize the PostgreSQL connector settings

1. You're back in the edit dialog for **defaultConnector**. In the **Authentication** tab, wait for the key vault connector to be created. When it's finished, the **Key Vault Connection** dropdown automatically selects it.
2. Select **Next: Networking**.
3. Select **Save**. Wait until the **Update succeeded** notification appears.

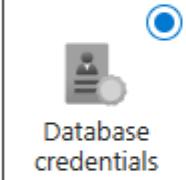
defaultConnector

Basics **Authentication** Networking

Select the authentication type you'd like to use between your compute service and target service. [Learn more about authentication.](#)

System assigned managed identity (Supported via Azure CLI. [Learn more](#)) ⓘ
 User assigned managed identity (Supported via Azure CLI. [Learn more](#)) ⓘ
 Connection string ⓘ
 Service principal (Supported via Azure CLI. [Learn more](#)) ⓘ

Continue with...

 Database credentials  Key Vault

Username *
gcibcbtiq

Password *
.....

[Forgot password?](#)

Store Secret In Key Vault ⓘ

Key Vault Connection * ⓘ
mangesh-key-vault-python (keyvault_362d5) 

[Create new](#)

Store Configuration in App Configuration ⓘ

Next : Networking Previous Cancel 

Step 7: Verify the Key Vault integration

1. From the left menu, select **Settings > Environment variables** again.
2. Next to **AZURE_POSTGRESQL_PASSWORD**, select **Show value**. The value should be `@Microsoft.KeyVault(...)`, which means that it's a **key vault reference** because the secret is now managed in the key vault.

The screenshot shows the 'App settings' section of the Azure App Service configuration. On the left, a sidebar lists various deployment and settings options. The 'Environment variables' option is highlighted with a red box. The main area displays a table of environment variables:

Name	Value	Deployment
AZURE_KEYVAULT_RESOURCE...	Show value	✓
AZURE_KEYVAULT_SCOPE	Show value	✓
AZURE_POSTGRESQL_CONNE...	Show value	✓
AZURE_REDIS_CONNECTIONS...	Show value	✓

At the bottom, there are 'Apply' and 'Discard' buttons, and a 'Send us your feedback' link.

To summarize, the process for securing your connection secrets involved:

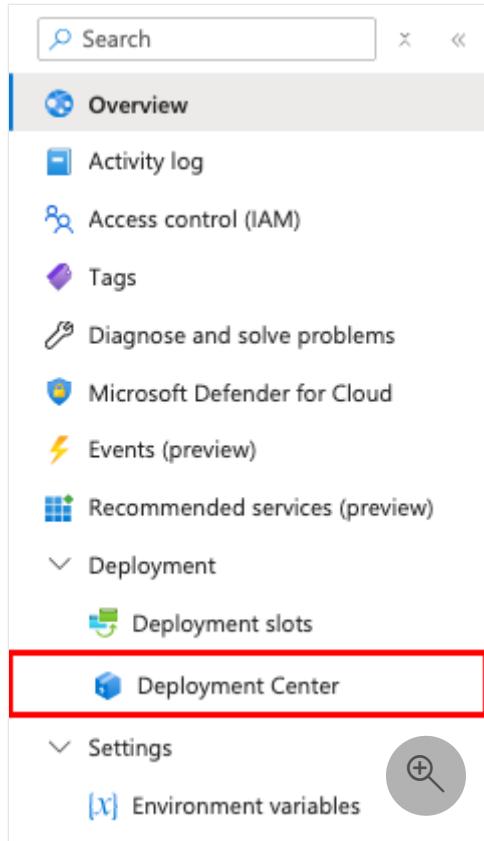
- Retrieving the connection secrets from the App Service app's environment variables.
- Creating a key vault.
- Creating a Key Vault connection with the system-assigned managed identity.
- Updating the service connectors to store the secrets in the key vault.

Having issues? Check the [Troubleshooting section](#).

4. Deploy sample code

In this step, you configure GitHub deployment using GitHub Actions. It's just one of many ways to deploy to App Service, but also a great way to have continuous integration in your deployment process. By default, every `git push` to your GitHub repository kicks off the build and deploy action.

Step 1: In the left menu, select **Deployment > Deployment Center**.



Step 2: In the Deployment Center page:

1. In **Source**, select **GitHub**. By default, **GitHub Actions** is selected as the build provider.
2. Sign in to your GitHub account and follow the prompt to authorize Azure.
3. In **Organization**, select your account.
4. In **Repository**, select **msdocs-flask-postgresql-sample-app**.
5. In **Branch**, select **starter-no-infra**. This is the same branch that you worked in with your sample app, without any Azure-related files or configuration.
6. For **Authentication type**, select **User-assigned identity**.
7. In the top menu, select **Save**. App Service commits a workflow file into the chosen GitHub repository, in the `.github/workflows` directory. By default, the deployment center [creates a user-assigned identity](#) for the workflow to authenticate using Microsoft Entra (OIDC authentication). For alternative authentication options, see [Deploy to App Service using GitHub Actions](#).

[Save](#) [Discard](#) [Browse](#) [Manage publish profile](#) [Sync](#) [Leave Feedback](#)

[Settings *](#) [Logs](#) [FTPS credentials](#)

Info You're now in the production slot, which is not recommended for setting up CI/CD. [Learn more](#) [X](#)

Deploy and build code from your preferred source and build provider. [Learn more](#)

Source* [GitHub](#)

Building with GitHub Actions. [Change provider](#).

GitHub

App Service will place a GitHub Actions workflow in your chosen repository to build and deploy your app whenever there is a commit on the chosen branch. If you can't find an organization or repository, you may need to enable additional permissions on GitHub. You must have write access to your chosen GitHub repository to deploy with GitHub Actions. [Learn more](#)

Signed in as
icephas [Change Account](#) ⓘ

Organization* [msdocs-flask-postgresql-sample-app](#)

Repository* [msdocs-flask-postgresql-sample-app](#)

Branch* [starter-no-infra](#)

Workflow Option*
 Overwrite the workflow: Overwrite the existing workflow file 'starter-no-infra_msdocs-python-postgres-234.yml' in the selected repository and branch.
 Use existing workflow: Use the existing workflow file 'starter-no-infra_msdocs-python-postgres-234.yml' in the selected repository and branch.

Build

Runtime stack
Python

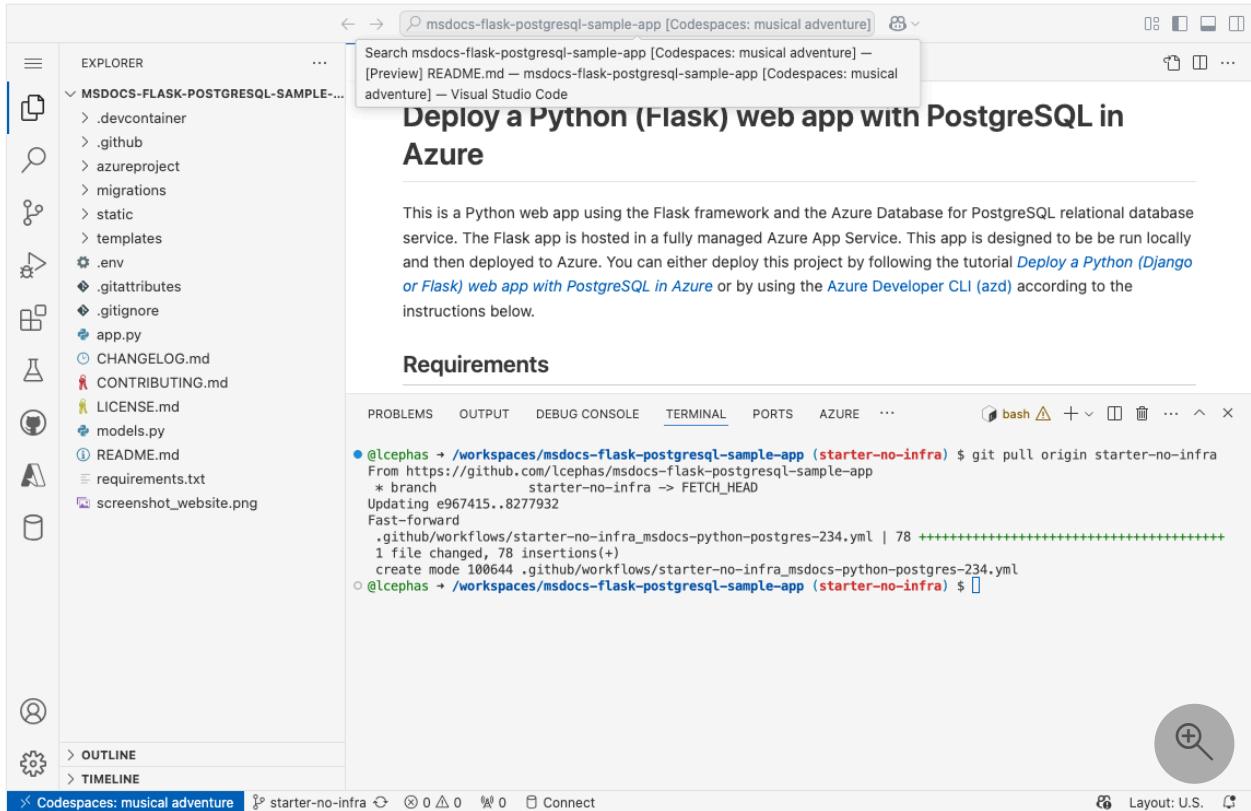
Version
Python 3.12

Authentication settings

Select how you want your GitHub Action workflow to authenticate to Azure. If you choose user-assigned identity, the identity selected will be federated with GitHub as an authorized client and given write permissions on the app. [Learn more](#)

Authentication type*
 User-assigned identity 
 Basic authentication

Step 3: Back in the GitHub codespace of your sample fork, run `git pull origin starter-no-infra`. This pulls the newly committed workflow file into your codespace.



Step 4 (Option 1: with GitHub Copilot):

1. Start a new chat session by selecting the **Chat** view, then selecting **+**.
2. Ask, "*@workspace How does the app connect to the database?*" Copilot might give you some explanation about **SQLAlchemy** how its connection URI is configured in *azureproject/development.py* and *azureproject/production.py*.
3. Ask, "*@workspace In production mode, my app is running in an App Service web app, which uses Azure Service Connector to connect to a PostgreSQL flexible server using the Django client type. What are the environment variable names I need to use?*" Copilot might give you a code suggestion similar to the one in the **Option 2: without GitHub Copilot** steps below and even tell you to make the change in the *azureproject/production.py* file.
4. Open *azureproject/production.py* in the explorer and add the code suggestion. GitHub Copilot doesn't give you the same response every time, and it's not always correct. You might need to ask more questions to fine-tune its response. For tips, see [What can I do with GitHub Copilot in my codespace?](#).

The screenshot shows the GitHub Copilot interface. At the top, there are icons for file operations (copy, paste, etc.) and a red-bordered plus sign button. The title "Ask Copilot" is centered above a message about AI-powered mistakes and telemetry collection. Below this, there are instructions for using context, extensions, and commands. A red box highlights a response from the AI, which asks how the app connects to a database. The AI interface includes input fields for '@' and a user icon, a dropdown for 'GPT 4o', and a send button.

Copilot is powered by AI, so mistakes are possible.
Review output carefully before use.

As an internal user, additional telemetry is collected. If you work on a project that contains customer content, you must [disable telemetry](#).

⌚ or type # to attach context
@ to chat with extensions
Type / to use commands

/help What can you do?

@workspace How does the app connect to the database?

Step 4 (Option 2: without GitHub Copilot):

1. Open `Program.cs` in the explorer.
2. Find the commented code (lines 3-8) and uncomment it. This creates a connection string for SQLAlchemy by using `AZURE_POSTGRESQL_USER`, `AZURE_POSTGRESQL_PASSWORD`, `AZURE_POSTGRESQL_HOST`, and `AZURE_POSTGRESQL_NAME`.

```
import os
DATABASE_URI = 'postgresql+psycopg2://{}:{}@{}{}'.format(dbuser, dbpass, dbhost, dbname)
dbuser=os.getenv('AZURE_POSTGRESQL_USER'),
dbpass=os.getenv('AZURE_POSTGRESQL_PASSWORD'),
dbhost=os.getenv('AZURE_POSTGRESQL_HOST'),
dbname=os.getenv('AZURE_POSTGRESQL_NAME')
```

```
@lcephas ~ /workspaces/msdocs-flask-postgresql-sample-app (starter-no-infra) $ git pull origin starter-no-infra
From https://github.com/lcephas/msdocs-flask-postgresql-sample-app
 * branch            starter-no-infra -> FETCH_HEAD
Updating e967415..8277932
Fast-forward
 .github/workflows/starter-no-infra_msdocs-python-postgres-234.yml | 78 ++++++
+++++
1 file changed, 78 insertions(+)
create mode 100644 .github/workflows/starter-no-infra_msdocs-python-postgres-234.yml
@lcephas ~ /workspaces/msdocs-flask-postgresql-sample-app (starter-no-infra) $
```

Step 5:

1. Select the **Source Control** extension.
2. In the textbox, type a commit message like `Configure Azure database connection`.
Or, select and let GitHub Copilot generate a commit message for you.
3. Select **Commit**, then confirm with **Yes**.
4. Select **Sync changes 1**, then confirm with **OK**.

Save Discard Browse Manage publish profile Sync Leave Feedback

Settings Logs FTPS credentials

Refresh Delete

Time	Commit ID	Logs	Commit Author	Status	Message
Wednesday, January 22, 2025 (4)					
01/22/2025, 3:26:04..	6a55fb8	Build/Deploy Lo...	lcephas	In Progress....	Configure Azure database connection
01/22/2025, 3:08:15..	80cf7f1	App Log	N/A	Failed	OneDeploy
01/22/2025, 3:08:00..	temp-9d	App Log	N/A	Failed	OneDeploy
01/22/2025, 3:05:24..	8277952	Build/Deploy Lo...	lcephas	Failed	Add or update the Azure App Service build and deployment workflow config

Step 6: Back in the Deployment Center page in the Azure portal:

1. Select the **Logs** tab, then select **Refresh** to see the new deployment run.
2. In the log item for the deployment run, select the **Build/Deploy Logs** entry with the latest timestamp.

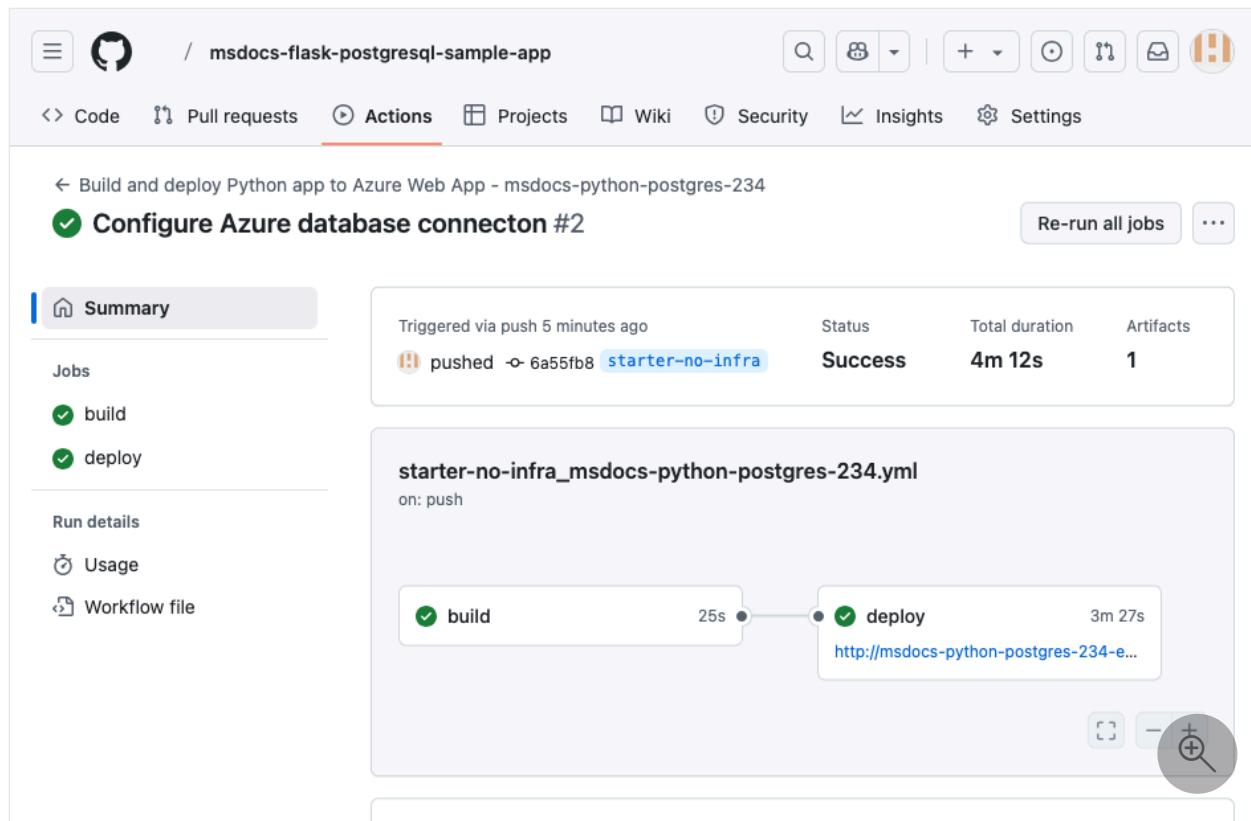
Save Discard Browse Manage publish profile Sync Leave Feedback

Settings Logs FTPS credentials

Refresh Delete

Time	Commit ID	Logs	Commit Author	Status	Message
Wednesday, January 22, 2025 (4)					
01/22/2025, 3:26:04..	6a55fb8	Build/Deploy Lo...	lcephas	In Progress....	Configure Azure database connection
01/22/2025, 3:08:15..	80cf7f1	App Log	N/A	Failed	OneDeploy
01/22/2025, 3:08:00..	temp-9d	App Log	N/A	Failed	OneDeploy
01/22/2025, 3:05:24..	8277952	Build/Deploy Lo...	lcephas	Failed	Add or update the Azure App Service build and deployment workflow config

Step 7: You're taken to your GitHub repository and see that the GitHub action is running. The workflow file defines two separate stages, build and deploy. Wait for the GitHub run to show a status of **Success**. It takes about 5 minutes.



The screenshot shows a GitHub repository page for 'msdocs-flask-postgresql-sample-app'. The 'Actions' tab is selected, showing a recent run titled 'Configure Azure database connecton #2'. The run summary indicates it was triggered via push 5 minutes ago, pushed commit '6a55fb8', and completed successfully ('Success') in a total duration of 4m 12s, producing 1 artifact. The workflow file is 'starter-no-infra_msdocs-python-postgres-234.yml' and it includes a 'push' event trigger. The workflow diagram shows a 'build' step followed by a 'deploy' step, with a link between them. The URL for the deployment is visible as 'http://msdocs-python-postgres-234-e...'. On the left sidebar, there are links for 'Summary', 'Jobs', 'build', 'deploy', 'Run details', 'Usage', and 'Workflow file'.

Having issues? Check the [Troubleshooting guide](#).

5. Generate database schema

With the PostgreSQL database protected by the virtual network, the easiest way to run [Flask database migrations](#) is in an SSH session with the Linux container in App Service.

Step 1: Back in the App Service page, in the left menu,

1. Select **Development Tools > SSH**.
2. Select **Go**.

Microsoft Azure (Preview)  Search resources, services, and docs (G+/)

Home > msdocs-python-postgres-234

msdocs-python-postgres-234 | SSH

App Service

Search

Quotas Change App Service plan

Development Tools

Clone App

SSH 

Advanced Tools Extensions

API API Management

SSH provides a Web SSH console experience for your Linux App code. [Learn more](#)

Go 



Step 2: In the SSH session, run `flask db upgrade`. If it succeeds, App Service is connecting successfully to the database.

```
APP SERVICE ON LINUX

Documentation: http://aka.ms/webapp-linux
Python 3.9.7
Note: Any data outside '/home' is not persisted
(antenv) root@aa2d84bd54c7:/tmp/8daa8347537426e# flask db upgrade
Loading config.production.
INFO [alembic.runtime.migration] Context impl PostgresqlImpl.
INFO [alembic.runtime.migration] Will assume transactional DDL.
INFO [alembic.runtime.migration] Running upgrade  -> 336483b236e3, initial migration
(antenv) root@aa2d84bd54c7:/tmp/8daa8347537426e# 
```



☰ Menu ssh://root@192.0.2.13:2222 | SSH CONNECTION ESTABLISHED |

 Tip

In the SSH session, only changes to files in `/home` can persist beyond app restarts.

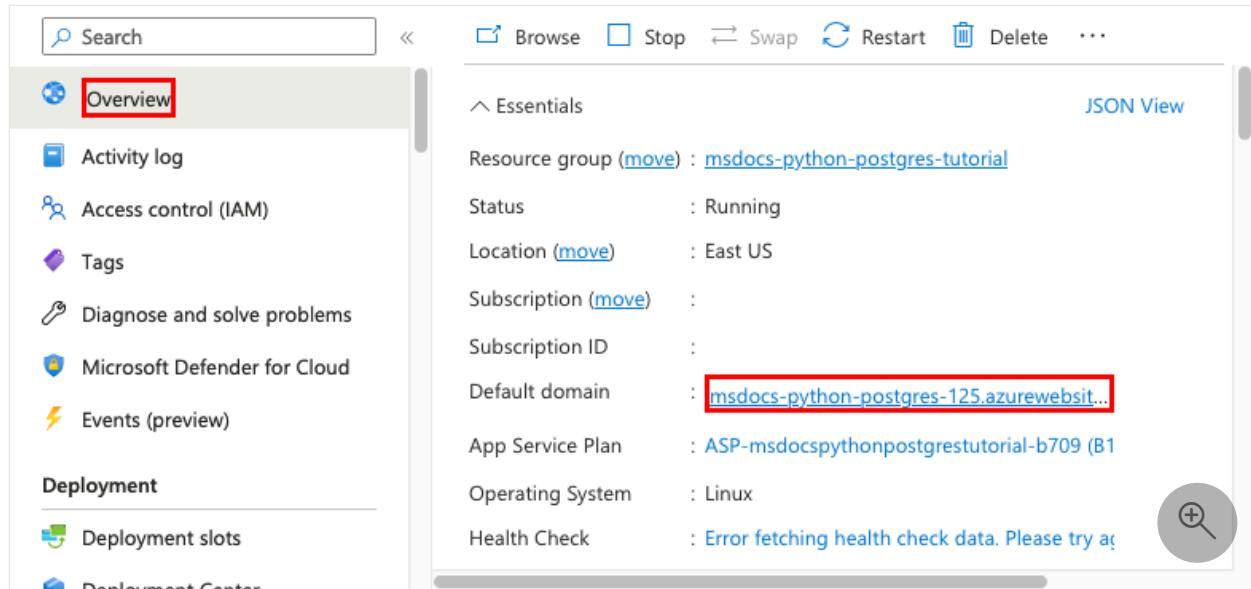
Changes outside of `/home` aren't persisted.

Having issues? Check the [Troubleshooting section](#).

6. Browse to the app

Step 1: In the App Service page:

1. From the left menu, select **Overview**.
2. Select the URL of your app.



The screenshot shows the Azure App Service Overview page. On the left, there's a navigation menu with options like Overview, Activity log, Access control (IAM), Tags, Diagnose and solve problems, Microsoft Defender for Cloud, Events (preview), Deployment, Deployment slots, and Deployment Center. The 'Overview' tab is selected and highlighted with a red box. On the right, there's a summary card with various settings. At the top of the card are buttons for Browse, Stop, Swap, Restart, Delete, and more. Below that is a 'Essentials' section with JSON View. The configuration details listed are:

Resource group (move)	: msdocs-python-postgres-tutorial
Status	: Running
Location (move)	: East US
Subscription (move)	:
Subscription ID	:
Default domain	: msdocs-python-postgres-125.azurewebsit...
App Service Plan	: ASP-msdocspythonpostgrestutorial-b709 (B1)
Operating System	: Linux
Health Check	: Error fetching health check data. Please try again.

Step 2: Add a few restaurants to the list. Congratulations, you're running a web app in Azure App Service, with secure connectivity to Azure Database for PostgreSQL.

Restaurants

Name	Rating	Details
Fourth Coffee	★ ★	2.0 (2 reviews) Details
Contoso Restaurant	★ ★ ★ ★	4.0 (3 reviews) Details

[Add new restaurant](#)

7. Stream diagnostic logs

Azure App Service captures all console logs to help you diagnose issues with your application. The sample app includes `print()` statements to demonstrate this capability as shown below.

Python

```
@app.route('/', methods=['GET'])
def index():
    print('Request for index page received')
    restaurants = Restaurant.query.all()
    return render_template('index.html', restaurants=restaurants)
```

Step 1: In the App Service page:

1. From the left menu, select **Monitoring > App Service logs**.
2. Under **Application logging**, select **File System**.
3. In the top menu, select **Save**.

msdocs-python-postgres-234 | App Service logs

App Service

Search Save Discard Send us your feedback

Metrics

Logs

Advisor recommendations

Health check

Diagnostic settings

App Service logs

Log stream

Log stream (preview)

Process explorer

Automation

Application logging

Off File System

Quota (MB) * 35

Retention Period (Days)

Download logs

FTP/deployment username

FTP

Step 2: From the left menu, select **Log stream**. You see the logs for your app, including platform logs and logs from inside the container.

msdocs-python-postgres-234 | Log stream

App Service

Search Reconnect Copy Pause Clear

Logs

Advisor recommendations

Health check

Diagnostic settings

App Service logs

Log stream

Log stream (preview)

Process explorer

Automation

Tasks (preview)

```
details page received
2022-10-05T18:49:34.756098791Z 169.254.130.1 - -
[05/Oct/2022:18:49:34 +0000] "GET /2/ HTTP/1.1" 200 6858
"https://msdocs-python-postgres-234.azurewebsites.net/2/"
Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/106.0.5249.30 Safari/537.36"
2022-10-05T18:49:35.087646396Z Request for index page
received
2022-10-05T18:49:35.087686798Z 169.254.130.1 - -
[05/Oct/2022:18:49:35 +0000] "GET / HTTP/1.1" 200 5891
"https://msdocs-python-postgres-234.azurewebsites.net/2/"
Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/106.0.5249.30 Safari/537.36"
2022-10-05T18:48:30.785Z INFO - Starting container for
```

Learn more about logging in Python apps in the series on [setting up Azure Monitor for your Python application](#).

8. Clean up resources

When you're finished, you can delete all of the resources from your Azure subscription by deleting the resource group.

Step 1: In the search bar at the top of the Azure portal:

1. Enter the resource group name.
2. Select the resource group.

The screenshot shows the Microsoft Azure portal interface. At the top, there's a search bar with the text "msdocs-flask-postgres-tutorial" and a magnifying glass icon. Below the search bar, the main content area is titled "Azure services". On the left, there's a sidebar with "Create a resource" and "Resource group" buttons. The main content area shows a "Resource Groups" section with one item listed: "msdocs-flask-postgres-tutorial", which is highlighted with a red box. To the right of this, there are sections for "Documentation" with links to tutorials and "Virtual networks". At the bottom of the search results, there's a "Continue searching in Microsoft Entra ID" button and a "Give feedback" link.

Step 2: In the resource group page, select Delete resource group.

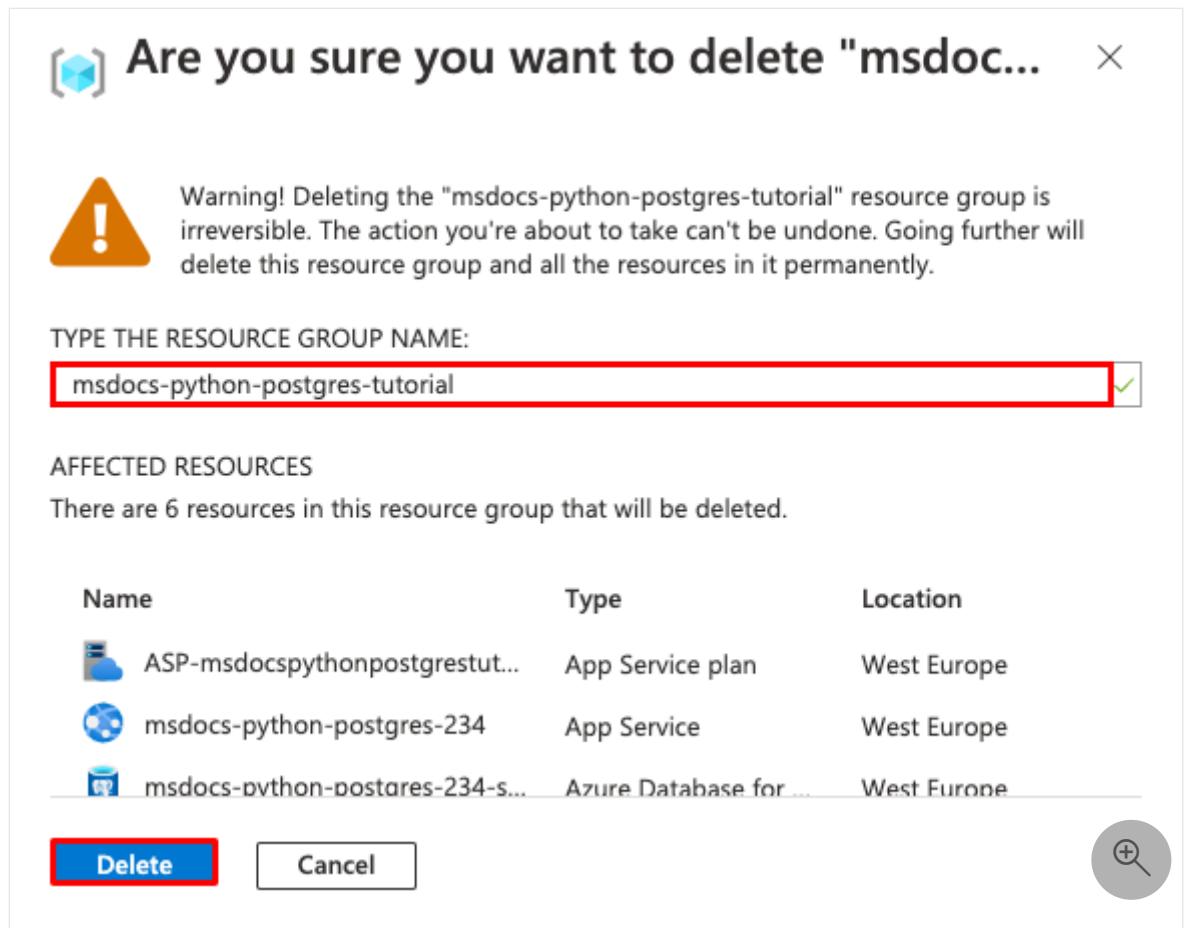
The screenshot shows the "msdocs-flask-postgres-tutorial" resource group page. At the top, there's a navigation bar with "Create", "Manage view", "Delete resource group" (which is highlighted with a red box), and other options. Below the navigation bar, there's a section titled "Essentials" with the following details:

- Subscription ([move](#)) : [Antares-Demo](#)
- Subscription ID : [\(redacted\)](#)
- Deployments : [7 Succeeded](#)
- Location : West Europe
- Tags ([edit](#)) : [Click here to add tags](#)

At the bottom of this section, there are tabs for "Resources" and "Recommendations (1)". The "Resources" tab is selected. At the very bottom, there are filters and pagination controls.

Step 3:

1. Enter the resource group name to confirm your deletion.
2. Select Delete.



Troubleshooting

Listed below are issues you might encounter while trying to work through this tutorial and steps to resolve them.

I can't connect to the SSH session

If you can't connect to the SSH session, then the app itself has failed to start. Check the [diagnostic logs](#) for details. For example, if you see an error like `KeyError: 'AZURE_POSTGRESQL_HOST'`, it might mean that the environment variable is missing (you might have removed the app setting).

I get an error when running database migrations

If you encounter any errors related to connecting to the database, check if the app settings (`AZURE_POSTGRESQL_USER`, `AZURE_POSTGRESQL_PASSWORD`, `AZURE_POSTGRESQL_HOST`,

and `AZURE_POSTGRESQL_NAME`) were changed or deleted. Without that connection string, the migrate command can't communicate with the database.

Frequently asked questions

- How much does this setup cost?
- How do I connect to the PostgreSQL server that's secured behind the virtual network with other tools?
- How does local app development work with GitHub Actions?
- How do I debug errors during the GitHub Actions deployment?
- I don't have permissions to create a user-assigned identity
- What can I do with GitHub Copilot in my codespace?

How much does this setup cost?

Pricing for the created resources is as follows:

- The App Service plan is created in **Basic** tier and can be scaled up or down. See [App Service pricing](#).
- The PostgreSQL flexible server is created in the lowest burstable tier **Standard_B1ms**, with the minimum storage size, which can be scaled up or down. See [Azure Database for PostgreSQL pricing](#).
- The virtual network doesn't incur a charge unless you configure extra functionality, such as peering. See [Azure Virtual Network pricing](#).
- The private DNS zone incurs a small charge. See [Azure DNS pricing](#).

How do I connect to the PostgreSQL server that's secured behind the virtual network with other tools?

- For basic access from a command-line tool, you can run `psql` from the app's SSH session.
- To connect from a desktop tool, your machine must be within the virtual network. For example, it could be an Azure VM that's connected to one of the subnets, or a machine in an on-premises network that has a [site-to-site VPN](#) connection with the Azure virtual network.
- You can also [integrate Azure Cloud Shell](#) with the virtual network.

How does local app development work with GitHub Actions?

Using the autogenerated workflow file from App Service as an example, each `git push` kicks off a new build and deployment run. From a local clone of the GitHub repository, you make the desired updates and push to GitHub. For example:

terminal

```
git add .
git commit -m "<some-message>"
git push origin main
```

How do I debug errors during the GitHub Actions deployment?

If a step fails in the autogenerated GitHub workflow file, try modifying the failed command to generate more verbose output. For example, you can get more output from the `python` command by adding the `-d` option. Commit and push your changes to trigger another deployment to App Service.

I don't have permissions to create a user-assigned identity

See [Set up GitHub Actions deployment from the Deployment Center](#).

What can I do with GitHub Copilot in my codespace?

You might have noticed that the GitHub Copilot chat view was already there for you when you created the codespace. For your convenience, we include the GitHub Copilot chat extension in the container definition (see `.devcontainer/devcontainer.json`). However, you need a [GitHub Copilot account](#) (30-day free trial available).

A few tips for you when you talk to GitHub Copilot:

- In a single chat session, the questions and answers build on each other and you can adjust your questions to fine-tune the answer you get.
- By default, GitHub Copilot doesn't have access to any file in your repository. To ask questions about a file, open the file in the editor first.
- To let GitHub Copilot have access to all of the files in the repository when preparing its answers, begin your question with `@workspace`. For more information, see [Use the @workspace agent](#).
- In the chat session, GitHub Copilot can suggest changes and (with `@workspace`) even where to make the changes, but it's not allowed to make the changes for you. It's up to you to add the suggested changes and test it.

Next steps

Advance to the next tutorial to learn how to secure your app with a custom domain and certificate.

[Secure with custom domain and certificate](#)

Learn how App Service runs a Python app:

[Configure Python app](#)

Feedback

Was this page helpful?

 Yes

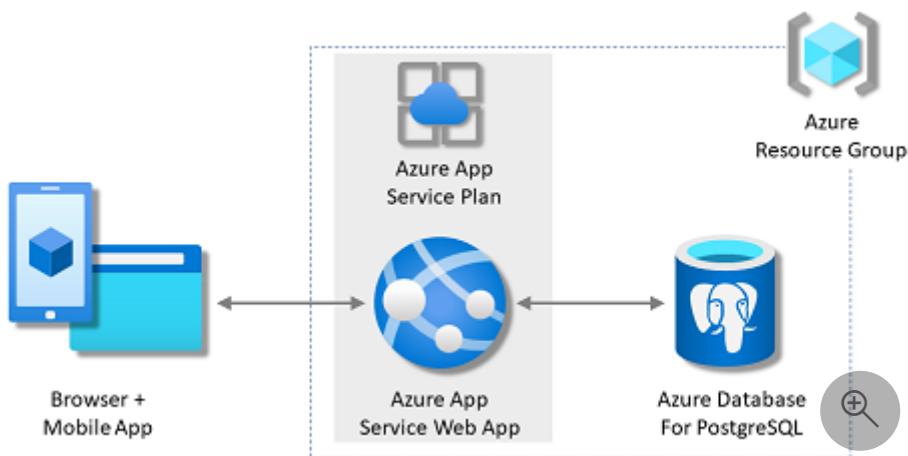
 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

Deploy a Python Django web app with PostgreSQL in Azure

Article • 01/29/2025

In this tutorial, you'll deploy a data-driven Python web app ([Django](#)) to [Azure App Service](#) with the [Azure Database for PostgreSQL](#) relational database service. Azure App Service supports [Python](#) in a Linux server environment. If you want, see the [Flask tutorial](#) or the [FastAPI tutorial](#) instead.



In this tutorial, you learn how to:

- ✓ Create a secure-by-default App Service, PostgreSQL, and Redis cache architecture.
- ✓ Secure connection secrets using a managed identity and Key Vault references.
- ✓ Deploy a sample Python app to App Service from a GitHub repository.
- ✓ Access App Service connection strings and app settings in the application code.
- ✓ Make updates and redeploy the application code.
- ✓ Generate database schema by running database migrations.
- ✓ Stream diagnostic logs from Azure.
- ✓ Manage the app in the Azure portal.
- ✓ Provision the same architecture and deploy by using Azure Developer CLI.
- ✓ Optimize your development workflow with GitHub Codespaces and GitHub Copilot.

Prerequisites

- An Azure account with an active subscription. If you don't have an Azure account, you [can create one for free](#).
- A GitHub account. you can also [get one for free](#).
- Knowledge of [Python with Django development](#).

- (Optional) To try GitHub Copilot, a [GitHub Copilot account](#). A 30-day free trial is available.

Skip to the end

If you just want to see the sample app in this tutorial running in Azure, just run the following commands in the [Azure Cloud Shell](#), and follow the prompt:

Bash

```
mkdir msdocs-django-postgresql-sample-app
cd msdocs-django-postgresql-sample-app
azd init --template msdocs-django-postgresql-sample-app
azd up
```

1. Run the sample

First, you set up a sample data-driven app as a starting point. For your convenience, the [sample repository](#), includes a [dev container](#) configuration. The dev container has everything you need to develop an application, including the database, cache, and all environment variables needed by the sample application. The dev container can run in a [GitHub codespace](#), which means you can run the sample on any computer with a web browser.

ⓘ Note

If you are following along with this tutorial with your own app, look at the *requirements.txt* file description in [README.md](#) to see what packages you'll need.

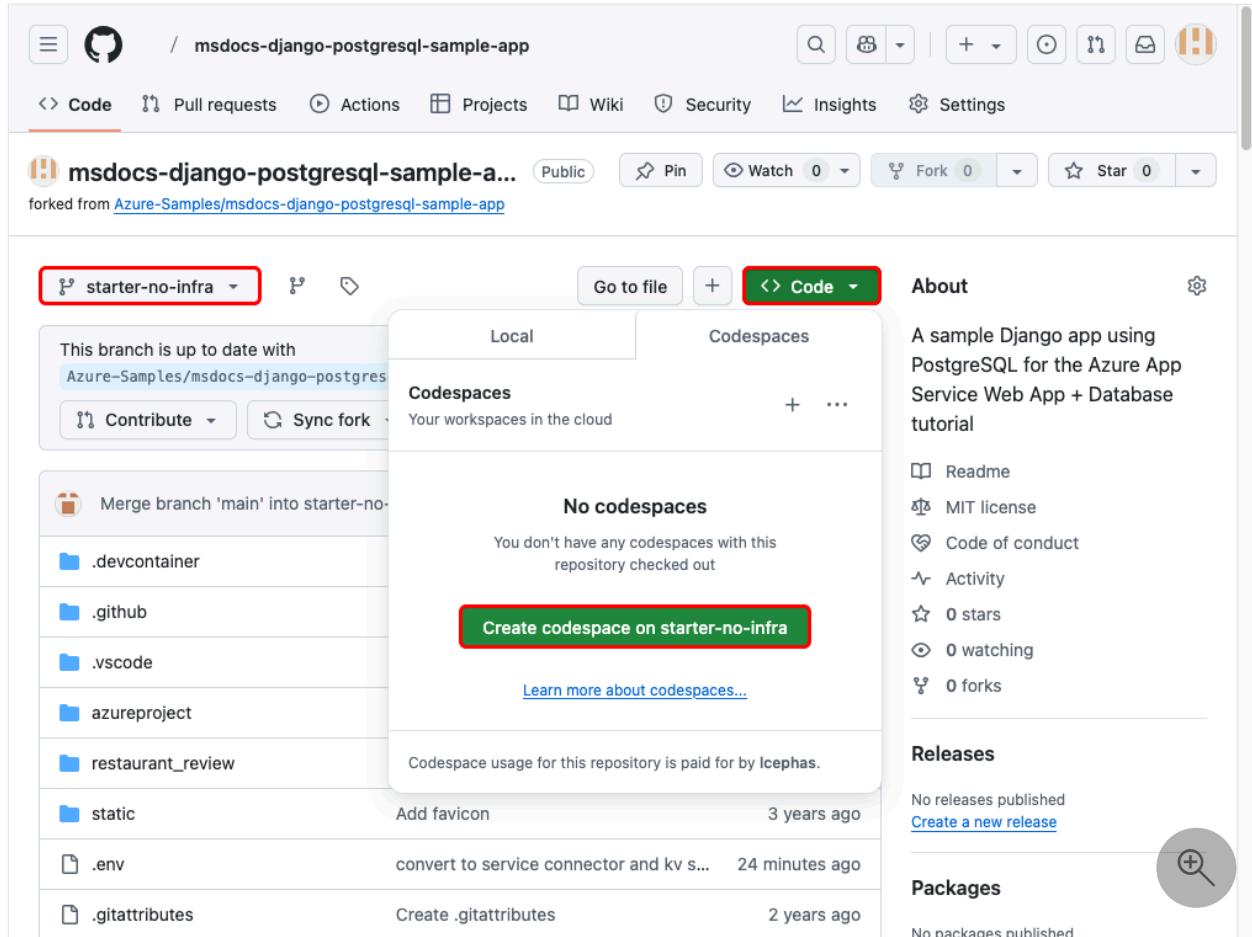
Step 1: In a new browser window:

1. Sign in to your GitHub account.
2. Navigate to <https://github.com/Azure-Samples/msdocs-django-postgresql-sample-app/fork>.
3. Unselect **Copy the main branch only**. You want all the branches.
4. Select **Create fork**.

The screenshot shows the GitHub fork creation interface for the repository `msdocs-django-postgresql-sample-app`. The top navigation bar includes links for Code, Issues, Pull requests, Actions, Projects, Security, and Insights. The main section is titled "Create a new fork". It explains that a fork is a copy of a repository and allows experimentation without affecting the original project. A note indicates that required fields are marked with an asterisk (*). The "Owner" dropdown is set to "I", and the "Repository name" field contains "msdocs-django-postgresql-s-". A message states that the name is available. Below this, a note says that forks are typically named the same as the upstream repository but can be customized. The "Description (optional)" field contains "A sample Django app using PostgreSQL for the Azure App Service Web App + Database tutorial". There is a checkbox for "Copy the main branch only", which is unchecked. A note below it encourages contributing back to the upstream repository by adding your own branch, with a link to "Learn more". A note also states that the fork is being created in the user's personal account. At the bottom right is a red "Create fork" button.

Step 2: In the GitHub fork:

1. Select **main > starter-no-infra** for the starter branch. This branch contains just the sample project and no Azure-related files or configuration.
2. Select **Code > Create codespace on starter-no-infra**. The codespace takes a few minutes to set up, and it runs `pip install -r requirements.txt` for your repository at the end. Also, the provided `.env` file already contains a dummy **SECRET_KEY** variable that **Django needs to run locally**.



Step 3: In the codespace terminal:

1. Run database migrations with `python manage.py migrate`.
2. Run the app with `python manage.py runserver`.
3. When you see the notification `Your application running on port 8000 is available.`, select **Open in Browser**. You should see the sample application in a new browser tab. To stop the application, type `Ctrl + C`.

This is a screenshot of the Microsoft Visual Studio Code interface, showing a Python Django application running in a Codespace. The terminal window displays the command `python manage.py migrate` being run, followed by the output of database migrations. Another command, `python manage.py runserver`, is also shown. The status bar at the bottom right includes buttons for 'Open in Browser' and 'Make Public'.

💡 Tip

You can ask [GitHub Copilot](#) about this repository. For example:

- @workspace *What does this project do?*
- @workspace *What does the .devcontainer folder do?*

Having issues? Check the [Troubleshooting section](#).

2. Create App Service, database, and cache

In this step, you create the Azure resources. The steps used in this tutorial create a set of secure-by-default resources that include App Service, Azure Database for PostgreSQL, and Azure Cache. For the creation process, you specify:

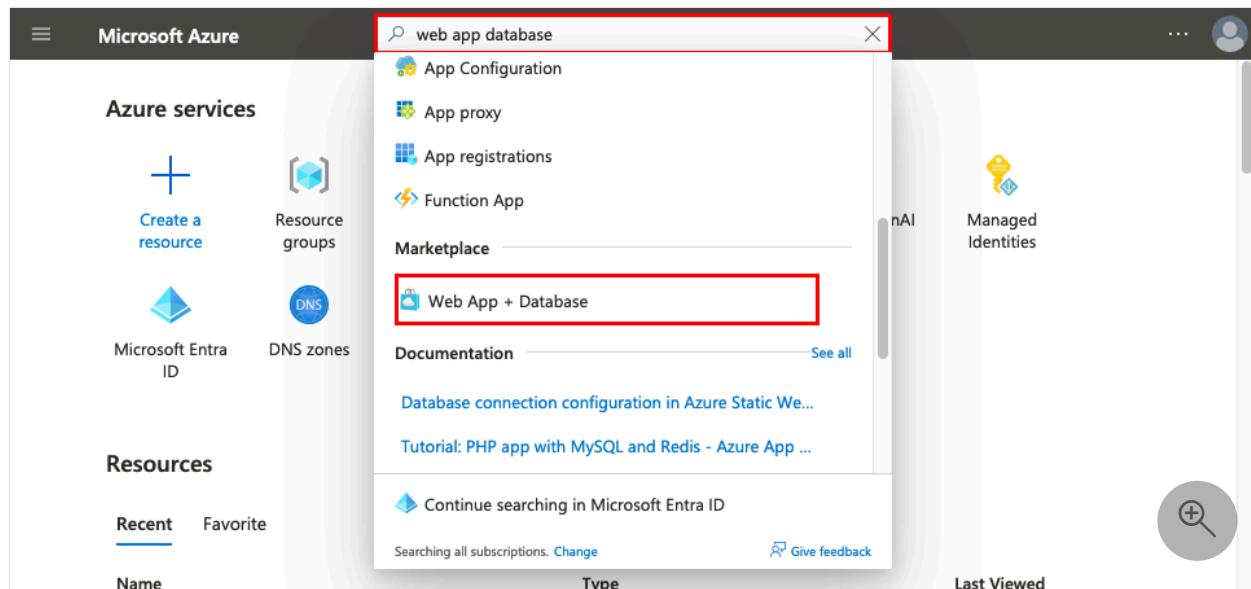
- The **Name** for the web app. It's used as part of the DNS name for your app in the form of `https://<app-name>-<hash>.azurewebsites.net`.
- The **Region** to run the app physically in the world. It's also used as part of the DNS name for your app.

- The **Runtime stack** for the app. It's where you select the version of Python to use for your app.
- The **Hosting plan** for the app. It's the pricing tier that includes the set of features and scaling capacity for your app.
- The **Resource Group** for the app. A resource group lets you group (in a logical container) all the Azure resources needed for the application.

Sign in to the [Azure portal](#) and follow these steps to create your Azure App Service resources.

Step 1: In the Azure portal:

1. Enter "web app database" in the search bar at the top of the Azure portal.
2. Select the item labeled **Web App + Database** under the **Marketplace** heading. You can also navigate to the [creation wizard](#) directly.



Step 2: In the **Create Web App + Database** page, fill out the form as follows.

1. **Resource Group:** Select **Create new** and use a name of **msdocs-django-postgres-tutorial**.
2. **Region:** Any Azure region near you.
3. **Name:** **msdocs-python-postgres-XYZ**.
4. **Runtime stack:** **Python 3.12**.
5. **Database:** **PostgreSQL - Flexible Server** is selected by default as the database engine. The server name and database name are also set by default to appropriate values.
6. **Add Azure Cache for Redis:** **Yes**.
7. **Hosting plan:** **Basic**. When you're ready, you can [scale up](#) to a production pricing tier.
8. Select **Review + create**.

9. After validation completes, select **Create**.

Home >

Create Web App + Database

Basics Tags Review + create

This template will create a secure by default configuration where the only publicly accessible endpoint will be your app following the recommended security best practices. [Learn more](#)

Project Details

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

Subscription * ⓘ

Resource Group * ⓘ [Create new](#)

Region *

Web App Details

Name [Secure unique default hostname on. More about this update](#)

Runtime stack *

Database

Info Database access will be locked down and not exposed to the public internet. This is in compliance with recommended best practices for security.

Engine * ⓘ

Server name *

Database name *

Azure Cache for Redis

Add Azure Cache for Redis? Yes No

Cache name *

Hosting

Hosting plan * Basic - For hobby or research purposes Standard - General purpose production apps

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

Step 3: The deployment takes a few minutes to complete. Once deployment completes, select the **Go to resource** button. You're taken directly to the App Service app, but the following resources are created:

- **Resource group:** The container for all the created resources.
- **App Service plan:** Defines the compute resources for App Service. A Linux plan in the *Basic* tier is created.
- **App Service:** Represents your app and runs in the App Service plan.
- **Virtual network:** Integrated with the App Service app and isolates back-end network traffic.
- **Private endpoint:** Access endpoint for the Redis cache in the virtual network.
- **Network interfaces:** Represents private IP addresses, one for each of the private endpoints.
- **Azure Database for PostgreSQL flexible server:** Accessible only from within the virtual network. A database and a user are created for you on the server.
- **Azure Cache for Redis:** Accessible only from behind its private network.
- **Private DNS zones:** Enables DNS resolution of the database server and the Redis cache in the virtual network.

 Your deployment is complete

 Deployment name : Microsoft.Web-WebAppDatabase-Portal-afa69d9d-97bc
Subscription :
Resource group : [msdocs-python-postgres-tutorial](#)
Start time : 11/29/2023, 10:17:05 AM
Correlation ID :

Deployment details

Next steps

[Go to resource](#)

Give feedback

 Tell us about your experience with deployment 

3. Secure connection secrets and add SECRET_KEY

The creation wizard generated the connectivity variables for you already as [app settings](#). However, the security best practice is to keep secrets out of App Service completely. You'll move your secrets to a key vault and change your app setting to [Key Vault references](#) with the help of Service Connectors.

Step 1: Retrieve the existing connection string

1. In the left menu of the App Service page, select **Settings > Environment variables**.
2. Select **AZURE_POSTGRESQL_CONNECTIONSTRING**.
3. In **Add/Edit application setting**, in the **Value** field, find the *password=* part at the end of the string.
4. Copy the password string after *password=* for use later. This app settings let you connect to the Postgres database and the Redis cache secured behind private endpoints. However, the secrets are saved directly in the App Service app, which isn't the best. You'll change this. In addition, you will add a **SECRET_KEY** setting, which is required by your Django app.

The screenshot shows the Azure portal interface for managing environment variables of an App Service. On the left, the navigation bar includes 'Tags', 'Diagnose and solve problems', 'Microsoft Defender for Cloud', 'Events (preview)', 'Recommended services (preview)', 'Deployment' (with 'Deployment slots' and 'Deployment Center' options), and 'Settings' (with 'Environment variables' selected). The main panel is titled 'Add/Edit application setting' for 'AZURE_POSTGRESQL_CONNECTIONSTRING'. It shows the 'Name' as 'AZURE_POSTGRESQL_CONNECTIONSTRING' and the 'Value' as 'host=msdocs-python-postgres-3-server.postgres.database.azure.com port=5432 sslmode=require user=gjibcbiq password:xxxxxxxxxx'. A 'Deployment slot setting' checkbox is checked. Other settings listed are 'AZURE_KEYVAULT_RESOURCEENDPOINT', 'AZURE_KEYVAULT_SCOPE', and 'AZURE_REDIS_CONNECTIONSTRING'. At the bottom are 'Apply' and 'Discard' buttons, and a magnifying glass icon.

Step 2: Create a key vault for secure management of secrets

1. In the top search bar, type "key vault", then select **Marketplace > Key Vault**.
2. In **Resource Group**, select **msdocs-python-postgres-tutorial**.
3. In **Key vault name**, type a name that consists of only letters and numbers.
4. In **Region**, set it to the same location as the resource group.

Azure Key Vault is a cloud service used to manage keys, secrets, and certificates. Key Vault eliminates the need for developers to store security information in their code. It allows you to centralize the storage of your application secrets which greatly reduces the chances that secrets may be leaked. Key Vault also allows you to securely store secrets and keys backed by Hardware Security Modules or HSMs. The HSMs used are Federal Information Processing Standards (FIPS) 140-2 Level 2 validated. In addition, key vault provides logs of all access and usage attempts of your secrets so you have a complete audit trail for compliance.

Project details

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

Subscription *

Dev Compute and Platform - FY25Q3

Resource group *

msdocs-python-postgres-123_group

[Create new](#)

Instance details

Key vault name * ⓘ

vault091979

Region *

Canada Central

Pricing tier * ⓘ

Standard

Recovery options

Soft delete protection will automatically be enabled on this key vault. This feature allows you to recover or permanently delete a key vault and secrets for the duration of the retention period. This protection applies to the key vault and the secrets stored within the key vault.

To enforce a mandatory retention period and prevent the permanent deletion of key vaults or secrets prior to the retention period elapsing, you can turn on purge protection. When purge protection is enabled, secrets cannot be purged by users or

[Previous](#)

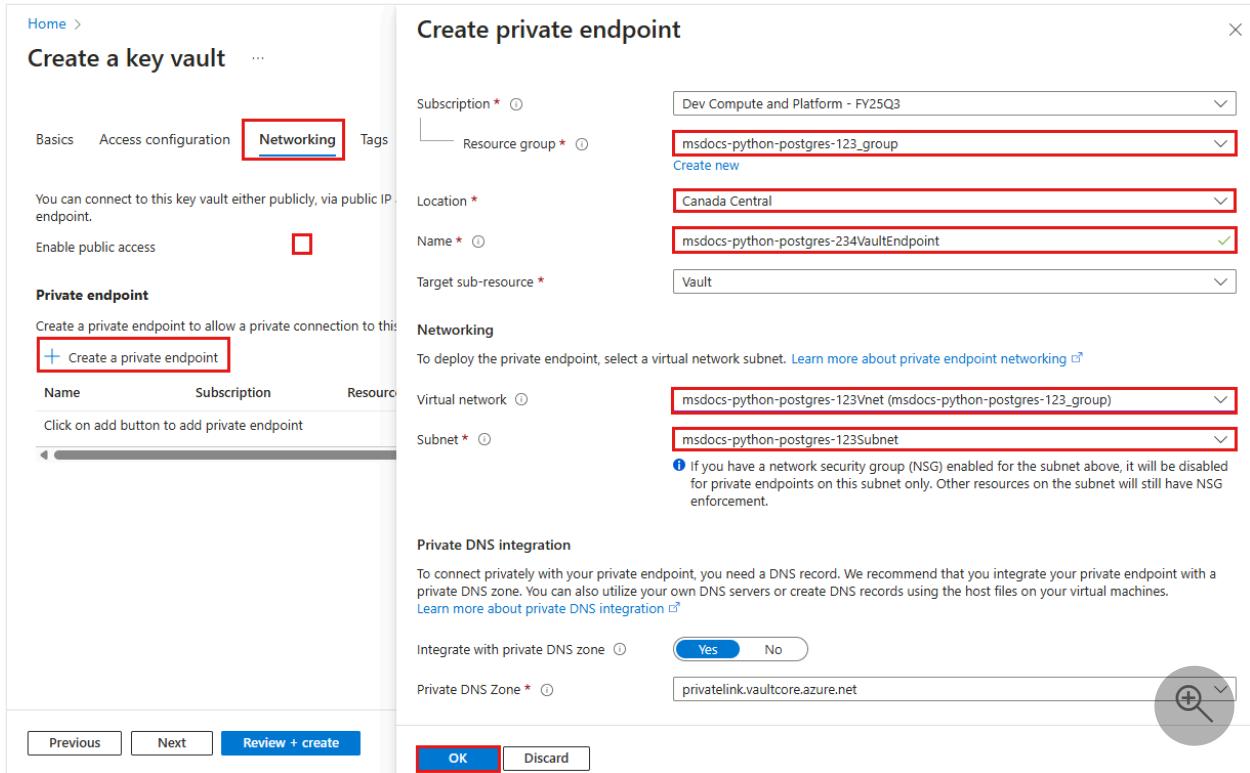
[Next](#)

[Review + create](#)



Step 3: Secure the key vault with a Private Endpoint

1. Select the **Networking** tab.
2. Unselect **Enable public access**.
3. Select **Create a private endpoint**.
4. In **Resource Group**, select **msdocs-python-postgres-tutorial**.
5. In the dialog, in **Location**, select the same location as your App Service app.
6. In **Name**, type **msdocs-python-postgres-XYZVaultEndpoint**.
7. In **Virtual network**, select **msdocs-python-postgres-XYZVnet**.
8. In **Subnet**, **msdocs-python-postgres-XYZSubnet**.
9. Select **OK**.
10. Select **Review + create**, then select **Create**. Wait for the key vault deployment to finish. You should see "Your deployment is complete."



Step 4: Configure the PostgreSQL connector

1. In the top search bar, type `msdocs-python-postgres`, then select the App Service resource called **msdocs-python-postgres-XYZ**.
2. In the App Service page, in the left menu, select **Settings > Service Connector**. There are already two connectors, which the app creation wizard created for you.
3. Select checkbox next to the PostgreSQL connector, then select **Edit**.
4. In **Client type**, select **Django**. The [Django client type in the PostgreSQL service connector](#) gives you database variables in separate settings instead of one connection string. The separate variables are easier for you to use in Django's [database settings](#).
5. Select the **Authentication** tab.
6. In **Password**, paste the password you copied earlier.
7. Select **Store Secret in Key Vault**.
8. Under **Key Vault Connection**, select **Create new**. A **Create connection** dialog is opened on top of the edit dialog.

The screenshot shows the Azure portal interface for managing a Service Connector. On the left, the sidebar has a tree view with various options like Environment variables, Configuration, Authentication, Identity, Backups, Custom domains, Certificates, Networking, Scale up (App Service plan), Scale out (App Service plan), WebJobs (preview), Locks, Performance, App Service plan, Development Tools, API, Monitoring, Automation, and Support + troubleshooting. The 'Service Connector' option is selected and highlighted with a red box.

In the main pane, there's a list of service types: DB for PostgreSQL flexible server, Cache for Redis, and Key Vault. The 'DB for PostgreSQL flexible server' item is checked and expanded, showing its resource name as 'msdocs-python-123'. Below this list, there's a section titled 'Continue with...' with two options: 'Database credentials' and 'Key Vault'. The 'Key Vault' option is selected.

The top navigation bar has tabs for Basics, Authentication (which is selected and highlighted with a red box), and Networking. There are also buttons for Create, Edit, Refresh, Validate, and Sample code.

The 'Authentication' tab contains fields for Username (gcibbitiq) and Password (redacted). There's a link for 'Forgot password?'. A checkbox for 'Store Secret In Key Vault' is checked, and the 'Key Vault Connection' dropdown is set to 'mangesh-key-vault-python (keyvault_362d5)'. A 'Create new' button is highlighted with a red box. At the bottom, there are 'Next : Networking', 'Previous', and 'Cancel' buttons.

Step 5: Establish the Key Vault connection

1. In the **Create connection** dialog for the Key Vault connection, in **Key Vault**, select the key vault you created earlier.
2. Select **Review + Create**.
3. When validation completes, select **Create**.

Create connection

Basics Networking Review + Create

Select the service instance and client type.

Service type * ⓘ

Key Vault

Connection name * ⓘ

keyvault_cc22c

Subscription * ⓘ

Dev Compute and Platform - FY25Q3

Key vault * ⓘ

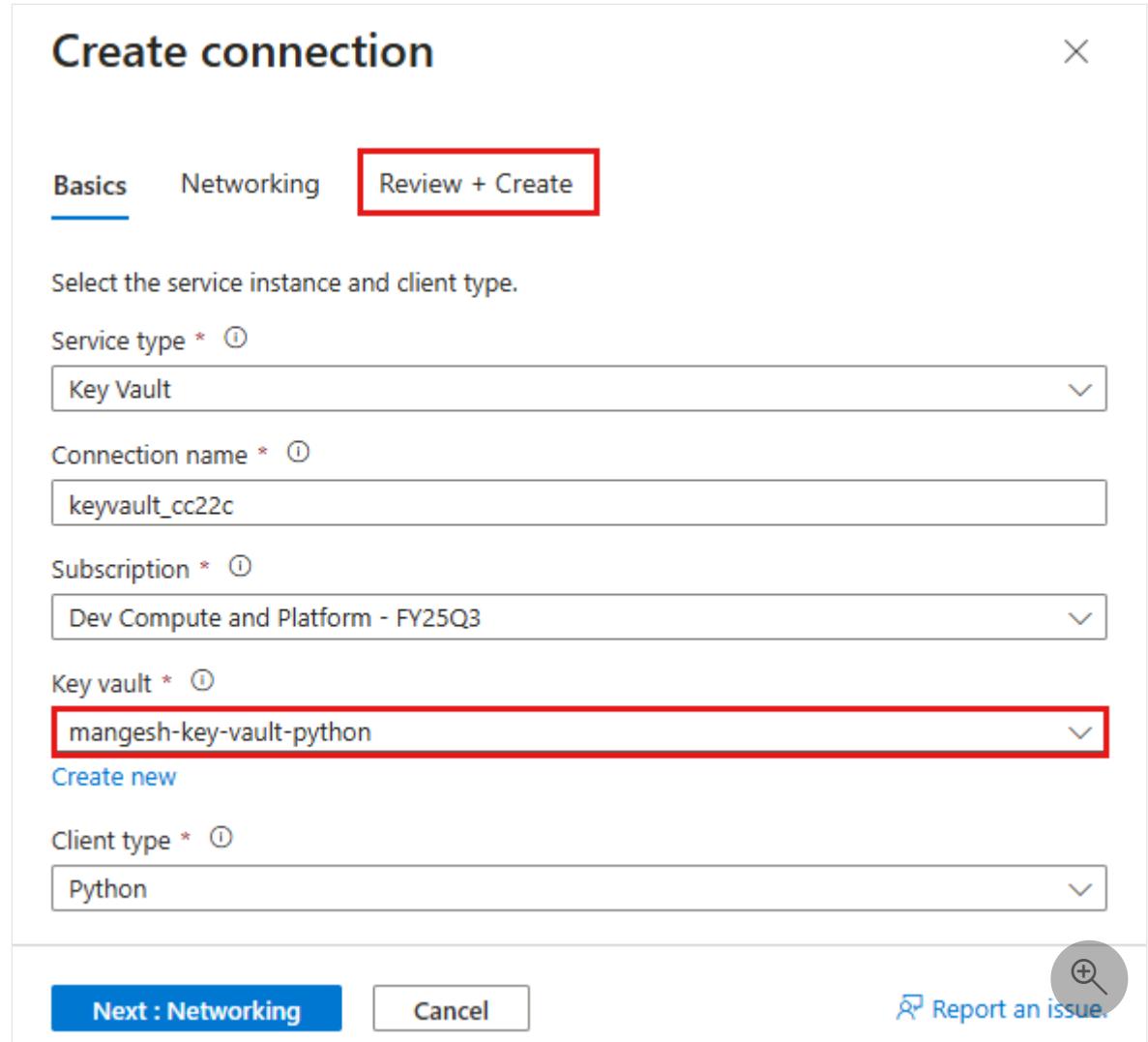
mangesh-key-vault-python

Create new

Client type * ⓘ

Python

Next : Networking Cancel Report an issue.



Step 6: Finalize the PostgreSQL connector settings

1. You're back in the edit dialog for **defaultConnector**. In the **Authentication** tab, wait for the key vault connector to be created. When it's finished, the **Key Vault Connection** dropdown automatically selects it.
2. Select **Next: Networking**.
3. Select **Save**. Wait until the **Update succeeded** notification appears.

defaultConnector

X

Basics

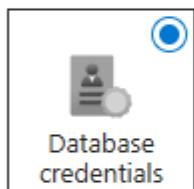
Authentication

Networking

Select the authentication type you'd like to use between your compute service and target service. [Learn more about authentication.](#)

- System assigned managed identity (Supported via Azure CLI. [Learn more](#)) ⓘ
- User assigned managed identity (Supported via Azure CLI. [Learn more](#)) ⓘ
- Connection string ⓘ
- Service principal (Supported via Azure CLI. [Learn more](#)) ⓘ

Continue with...



Username *

gcibcbtiq

Password *

.....

[Forgot password?](#)

Store Secret In Key Vault ⓘ

Key Vault Connection * ⓘ

mangesh-key-vault-python (keyvault_362d5)

▼

[Create new](#)

Store Configuration in App Configuration ⓘ

[Next : Networking](#)

[Previous](#)

[Cancel](#)



Step 7: Configure the Redis connector to use Key Vault secrets

1. In the Service Connectors page, select the checkbox next to the Cache for Redis connector, then select **Edit**.
2. Select the **Authentication** tab.
3. Select **Store Secret in Key Vault**.

4. Under Key Vault Connection, select the key vault you created.
5. Select Next: Networking.
6. Select Configure firewall rules to enable access to target service. The app creation wizard already secured the SQL database with a private endpoint.
7. Select Save. Wait until the Update succeeded notification appears.

The screenshot shows the Azure portal interface for managing a Service Connector. On the left, the navigation menu is open, and the 'Service Connector' item is highlighted with a red box. In the center, a modal window titled 'RedisConnector' is displayed, specifically the 'Authentication' tab. The 'Connection string' radio button is selected. Below it, there's a note about managed identities and a link to update Microsoft Entra authentication. A checkbox for 'Store Secret In Key Vault' is checked, and a dropdown menu shows a selected key vault connection named 'vault03940324 (keyvault_ca5fa)'. At the bottom of the modal, there are 'Next : Networking', 'Previous', and 'Cancel' buttons, with 'Next : Networking' also highlighted with a red box.

Step 8: Verify the Key Vault integration

1. From the left menu, select Settings > Environment variables again.
2. Next to **AZURE_POSTGRESQL_PASSWORD**, select Show value. The value should be `@Microsoft.KeyVault(...)`, which means that it's a **key vault reference** because the secret is now managed in the key vault.
3. To verify the Redis connection string, select Show value next to **AZURE_REDIS_CONNECTIONSTRING**.

Name	Value	Deployment slot setting	Source	Delete
AZURE_KEYVAULT_RESOURCEENDPOINT	Show value	✓	App Service	
AZURE_KEYVAULT_SCOPE	Show value	✓	App Service	
AZURE_POSTGRESQL_HOST	Show value	✓	App Service	
AZURE_POSTGRESQL_NAME	Show value	✓	App Service	
AZURE_POSTGRESQL_PASSWORD	Show value	✓	App Service	
AZURE_POSTGRESQL_USER	Show value	✓	App Service	
AZURE_REDIS_CONNECTIONSTRING	Show value	✓	App Service	

Step 9: The sample application reads the `SECRET_KEY` environment variable to set the required `SECRET_KEY` setting. You create it as an app setting in this step.

1. In the **App settings** tab, select **Add**.
2. Set **Name** to `SECRET_KEY`.
3. Set **Value** to a long random string.
4. Click **Apply**, then **Apply** again, then **Confirm**.

Name *	SECRET_KEY
Value	XX

To summarize, the process for securing your connection secrets involved:

- Retrieving the connection secrets from the App Service app's environment variables.
- Creating a key vault.
- Creating a Key Vault connection with the system-assigned managed identity.
- Updating the service connectors to store the secrets in the key vault.

⚠ Note

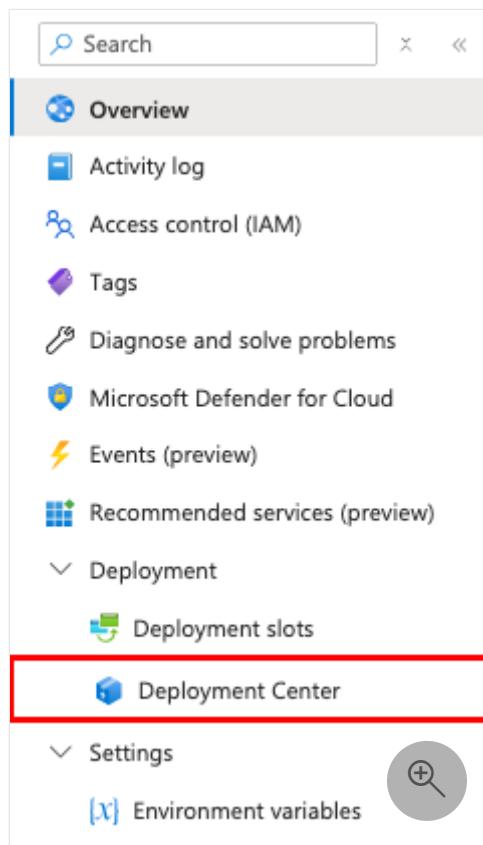
Ideally, the `SECRET_KEY` app setting should be configured as a key vault reference too, which is a multi-step process. For more information, see [How do I change the SECRET KEY app setting to a Key Vault reference?](#)

Having issues? Check the [Troubleshooting section](#).

4. Deploy sample code

In this step, you configure GitHub deployment using GitHub Actions. It's just one of many ways to deploy to App Service, but also a great way to have continuous integration in your deployment process. By default, every `git push` to your GitHub repository kicks off the build and deploy action.

Step 1: In the left menu, select Deployment > Deployment Center.



Step 2: In the Deployment Center page:

1. In **Source**, select **GitHub**. By default, **GitHub Actions** is selected as the build provider.
2. Sign in to your GitHub account and follow the prompt to authorize Azure.
3. In **Organization**, select your account.
4. In **Repository**, select `msdocs-django-postgresql-sample-app`.

5. In **Branch**, select **starter-no-infra**. This is the same branch that you worked in with your sample app, without any Azure-related files or configuration.
6. For **Authentication type**, select **User-assigned identity**.
7. In the top menu, select **Save**. App Service commits a workflow file into the chosen GitHub repository, in the `.github/workflows` directory. By default, the deployment center [creates a user-assigned identity](#) for the workflow to authenticate using Microsoft Entra (OIDC authentication). For alternative authentication options, see [Deploy to App Service using GitHub Actions](#).

Save Discard Browse Manage publish profile Sync Leave Feedback

Settings * Logs FTPS credentials

You're now in the production slot, which is not recommended for setting up CI/CD. [Learn more](#)

Deploy and build code from your preferred source and build provider. [Learn more](#)

Source * GitHub

Building with GitHub Actions. [Change provider](#).

GitHub

App Service will place a GitHub Actions workflow in your chosen repository to build and deploy your app whenever there is a commit on the chosen branch. If you can't find an organization or repository, you may need to enable additional permissions on GitHub. You must have write access to your chosen GitHub repository to deploy with GitHub Actions.

[Learn more](#)

Signed in as <github-alias> [Change Account](#)

Organization * [empty dropdown]

Repository * msdocs-django-postgresql-sample- ...

Branch * starter-no-infra

Workflow Option *

- Add a workflow: Add a new workflow file 'starter-no-infra_msdocs-python-postgres-235.yml' in the selected repository and branch.
- Use available workflow: Use one of the workflow files available in the selected repository and branch.

Build

Runtime stack Python

Version Python 3.12

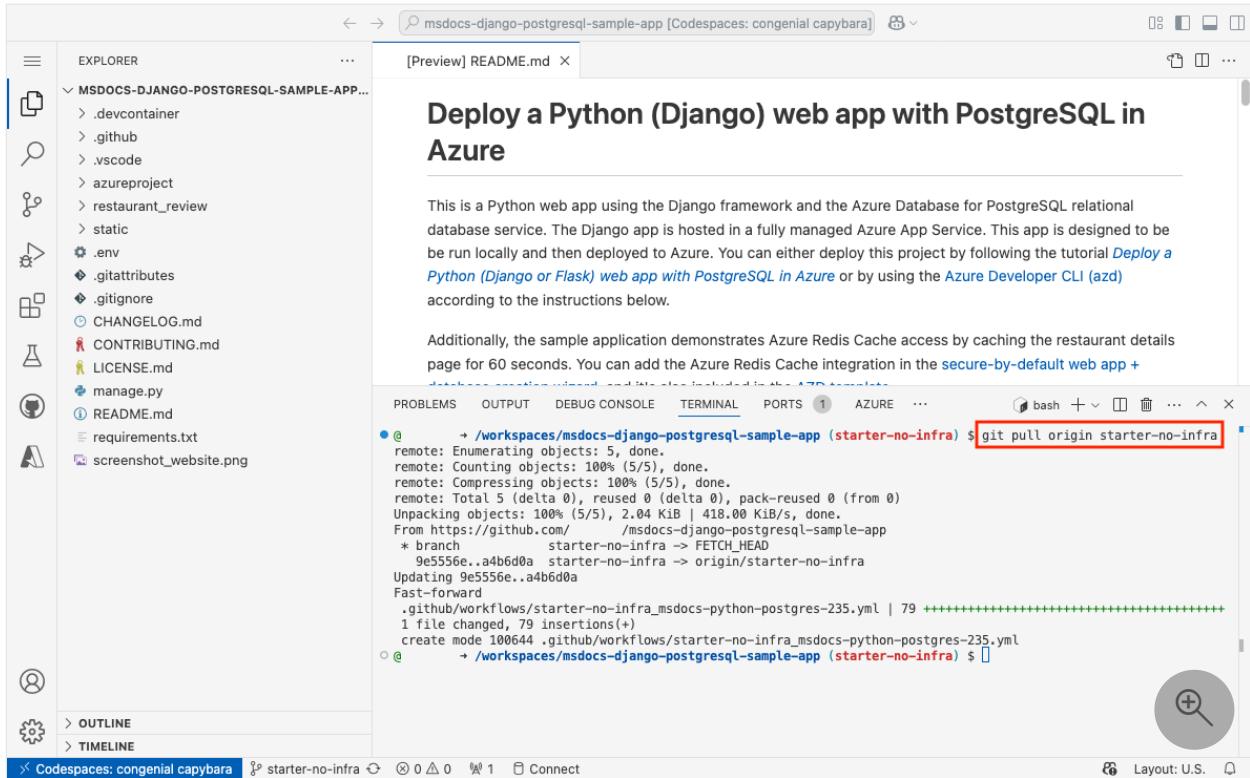
Authentication settings

Select how you want your GitHub Action workflow to authenticate to Azure. If you choose user-assigned identity, the identity selected will be federated with GitHub as an authorized client and given write permissions on the app. [Learn more](#)

Authentication type *

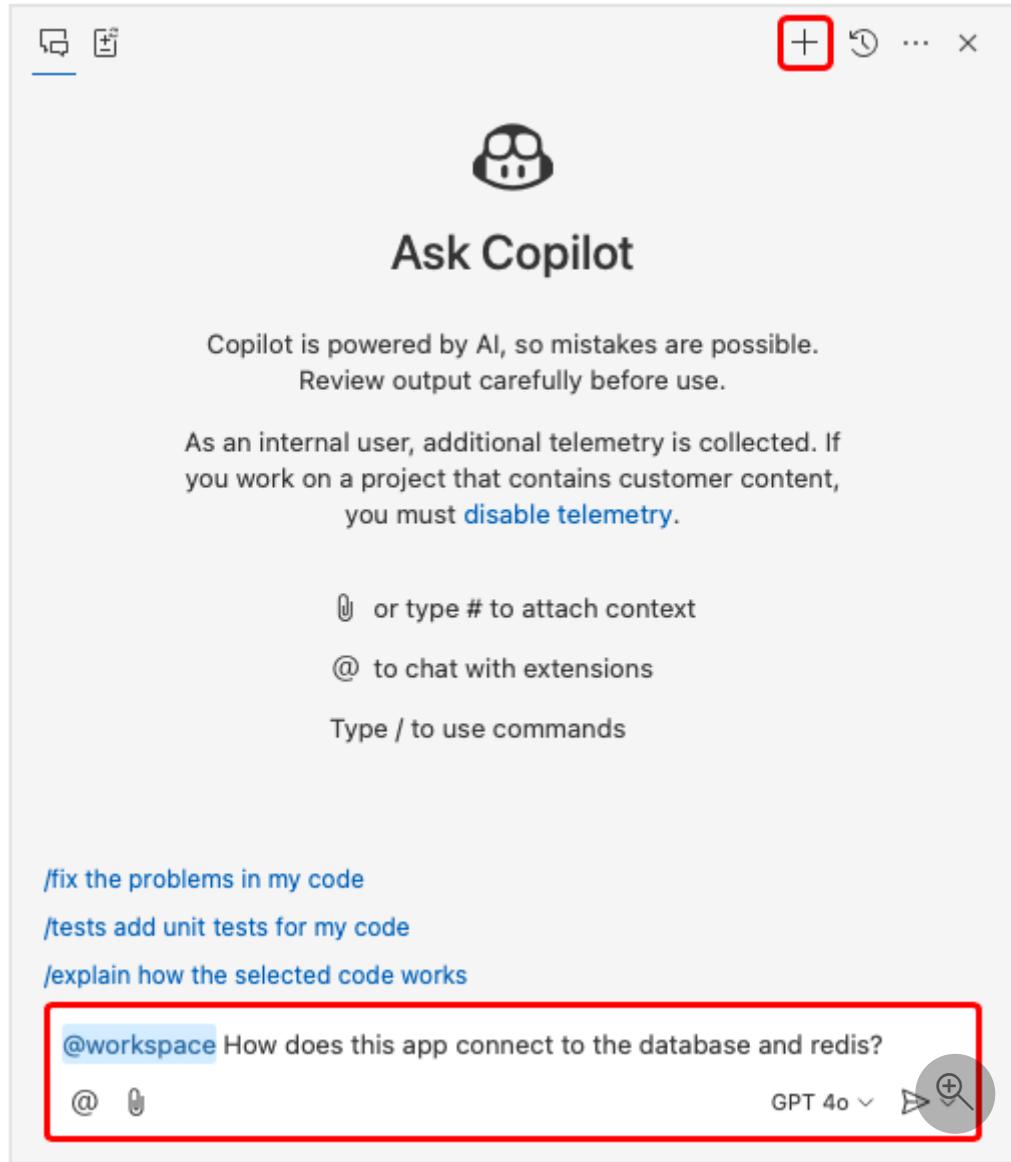
- User-assigned identity
- Basic authentication

Step 3: Back in the GitHub codespace of your sample fork, run `git pull origin starter-no-infra`. This pulls the newly committed workflow file into your codespace.



Step 4 (Option 1: with GitHub Copilot):

1. Start a new chat session by selecting the **Chat** view, then selecting **+**.
2. Ask, "*@workspace How does the app connect to the database and redis?*" Copilot might give you some explanation about how the settings are configured in *azureproject/development.py* and *azureproject/production.py*.
3. Ask, "*@workspace In production mode, my app is running in an App Service web app, which uses Azure Service Connector to connect to a PostgreSQL flexible server using the Django client type. What are the environment variable names I need to use?*" Copilot might give you a code suggestion similar to the one in the **Option 2: without GitHub Copilot** steps below and even tell you to make the change in the *azureproject/production.py* file.
4. Open *azureproject/production.py* in the explorer and add the code suggestion.
5. Ask, "*@workspace My App Service app also uses Azure Service Connector to connect to a Cache for Redis using the Django client type. What are the environment variable names I need to use?*" Copilot might give you a code suggestion similar to the one in the **Option 2: without GitHub Copilot** steps below and even tell you to make the change in the *azureproject/production.py* file.
6. Add the code suggestion. GitHub Copilot doesn't give you the same response every time, and it's not always correct. You might need to ask more questions to fine-tune its response. For tips, see [What can I do with GitHub Copilot in my codespace?](#).



Step 4 (Option 2: without GitHub Copilot):

1. Open `Program.cs` in the explorer.
2. Find the commented code (lines 29-48) and uncomment it. This creates PostgreSQL and Redis connections by using `AZURE_POSTGRESQL_USER`, `AZURE_POSTGRESQL_PASSWORD`, `AZURE_POSTGRESQL_HOST`, `AZURE_POSTGRESQL_NAME`, and `AZURE_REDIS_CONNECTIONSTRING`.

```

21     'django.contrib.messages.middleware.MessageMiddleware',
22     'django.middleware.clickjacking.XFrameOptionsMiddleware',
23   ],
24
25 SESSION_ENGINE = "django.contrib.sessions.backends.cache"
26 STATICFILES_STORAGE = 'whitenoise.storage.CompressedManifestStaticFilesStorage'
27 STATIC_ROOT = os.path.join(BASE_DIR, 'staticfiles')
28
29
30 DATABASES = {
31     'default': {
32         'ENGINE': 'django.db.backends.postgresql',
33         'NAME': os.environ['AZURE_POSTGRESOL_NAME'],
34         'HOST': os.environ['AZURE_POSTGRESOL_HOST'],
35         'USER': os.environ['AZURE_POSTGRESOL_USER'],
36         'PASSWORD': os.environ['AZURE_POSTGRESOL_PASSWORD'],
37     }
38
39
40 CACHES = {
41     'default': {
42         'BACKEND': "django_redis.cache.RedisCache",
43         'LOCATION': os.environ['AZURE_REDIS_CONNECTIONSTRING'],
44         'OPTIONS': {
45             'CLIENT_CLASS': "django_redis.client.DefaultClient",
46             'COMPRESSOR': "django_redis.compressors.zlib.ZlibCompressor",
47         },
48     }
49

```

Step 5:

1. Select the **Source Control** extension.
2. In the textbox, type a commit message like `Configure Azure database and cache connections`. Or, select and let GitHub Copilot generate a commit message for you.
3. Select **Commit**, then confirm with **Yes**.
4. Select **Sync changes 1**, then confirm with **OK**.

The screenshot shows the Azure Deployment Center interface. The 'Logs' tab is selected, indicated by a red box around the 'Logs' tab name. Below it, the 'Refresh' button is also highlighted with a red box. The main area displays a table of deployment logs. The first log entry, dated 01/24/2025 at 1:35:07 PM, has its 'Build/Deploy Logs' link highlighted with a red box. The second log entry, dated 01/24/2025 at 1:16:57 PM, also has its 'Build/Deploy Logs' link highlighted with a red box.

Time	Commit ID	Logs	Commit Author	Status	Message
Friday, January 24, 2025 (2)					
01/24/2025, 1:35:07 PM +01:00	dd051b4	Build/Deploy Logs	Icephas	In Progress... ...	Configure Azure database and cache connectons
01/24/2025, 1:16:57 PM +01:00	a4b6d0a	Build/Deploy Logs	Icephas	Failed	Add or update the Azure App Service build and deployment workflow config

Step 6: Back in the Deployment Center page in the Azure portal:

1. Select the **Logs** tab, then select **Refresh** to see the new deployment run.
2. In the log item for the deployment run, select the **Build/Deploy Logs** entry with the latest timestamp.

The screenshot shows the Azure Deployment Center interface, identical to the previous one but with a slight delay in the log entries. The 'Logs' tab is selected, indicated by a red box around the 'Logs' tab name. Below it, the 'Refresh' button is also highlighted with a red box. The main area displays a table of deployment logs. The first log entry, dated 01/24/2025 at 1:35:07 PM, has its 'Build/Deploy Logs' link highlighted with a red box. The second log entry, dated 01/24/2025 at 1:16:57 PM, also has its 'Build/Deploy Logs' link highlighted with a red box.

Time	Commit ID	Logs	Commit Author	Status	Message
Friday, January 24, 2025 (2)					
01/24/2025, 1:35:07 PM +01:00	dd051b4	Build/Deploy Logs	Icephas	In Progress... ...	Configure Azure database and cache connectons
01/24/2025, 1:16:57 PM +01:00	a4b6d0a	Build/Deploy Logs	Icephas	Failed	Add or update the Azure App Service build and deployment workflow config

Step 7: You're taken to your GitHub repository and see that the GitHub action is running. The workflow file defines two separate stages, build and deploy. Wait for the GitHub run to show a status of **Success**. It takes about 5 minutes.

The screenshot shows a GitHub Actions run summary for a repository named 'msdocs-flask-postgresql-sample-app'. The run was triggered via push 2 days ago and is labeled '#2'. The status is 'Success' with a total duration of '4m 12s' and 1 artifact. The workflow file is 'starter-no-infra_msdocs-python-postgres-235.yml' and it runs on push. The workflow consists of two jobs: 'build' (25s) and 'deploy' (3m 27s). The 'build' job is successful, and the 'deploy' job is also successful, leading to a deployed URL: <http://msdocs-python-postgres-235-e...>. The left sidebar shows other options like 'Code', 'Pull requests', 'Actions', 'Projects', 'Wiki', 'Security', 'Insights', and 'Settings'.

Having issues? Check the [Troubleshooting guide](#).

5. Generate database schema

With the PostgreSQL database protected by the virtual network, the easiest way to run [Django database migrations](#) is in an SSH session with the Linux container in App Service.

Step 1: Back in the App Service page, in the left menu,

1. Select **Development Tools > SSH**.
2. Select **Go**.

The screenshot shows the Azure portal's left sidebar with a search bar at the top. Below it are several menu items: 'Quotas', 'Change App Service plan', 'Development Tools' (which is currently selected), 'SSH' (which is highlighted with a red box), and 'Advanced Tools'. Under 'Development Tools', there are three sub-options: 'API Management', 'API definition', and 'CORS'. To the right of the sidebar, the main content area has a title 'SSH' with a blue arrow icon. Below the title, a sub-instruction reads 'SSH provides a web-based SSH console interface for your app. [Learn more](#)'. A red box highlights the 'Go' button next to the title. In the bottom right corner of the main area, there is a magnifying glass icon inside a circle.

Step 2: In the SSH session, run `python manage.py migrate`. If it succeeds, App Service is connecting successfully to the database.

The screenshot shows an SSH terminal window with a black background and white text. The output of the command `python manage.py migrate` is displayed, showing the progress of applying migrations across various apps and models. The output is as follows:

```
Documentation: http://aka.ms/webapp-linux
Python 3.12.0
Note: Any data outside '/home' is not persisted
(antenv) root@d316c094963d:/tmp/8dbf0da5234ace0# python manage.py migrate
Operations to perform:
  Apply all migrations: admin, auth, contenttypes, restaurant_review, sessions
Running migrations:
  Applying contenttypes.0001_initial... OK
  Applying auth.0001_initial... OK
  Applying admin.0001_initial... OK
  Applying admin.0002_logentry_remove_auto_add... OK
  Applying admin.0003_logentry_add_action_flag_choices... OK
  Applying contenttypes.0002_remove_content_type_name... OK
  Applying auth.0002_alter_permission_name_max_length... OK
  Applying auth.0003_alter_user_email_max_length... OK
  Applying auth.0004_alter_user_username_opts... OK
  Applying auth.0005_alter_user_last_login_null... OK
  Applying auth.0006_require_contenttypes_0002... OK
  Applying auth.0007_alter_validators_add_error_messages... OK
  Applying auth.0008_alter_user_username_max_length... OK
  Applying auth.0009_alter_user_last_name_max_length... OK
  Applying auth.0010_alter_group_name_max_length... OK
  Applying auth.0011_update_proxy_permissions... OK
  Applying auth.0012_alter_user_first_name_max_length... OK
  Applying restaurant_review.0001_initial... OK
  Applying restaurant_review.0002_alter_review_rating... OK
  Applying sessions.0001_initial... OK
(antenv) root@d316c094963d:/tmp/8dbf0da5234ace0#
```

At the bottom of the terminal window, there is a green status bar with the text 'SSH CONNECTION ESTABLISHED'.

Tip

In the SSH session, only changes to files in `/home` can persist beyond app restarts. Changes outside of `/home` aren't persisted. The SSH session is useful for running common `python manage.py` commands, such as user creation with the `python manage.py createsuperuser`. For more information, see the documentation for

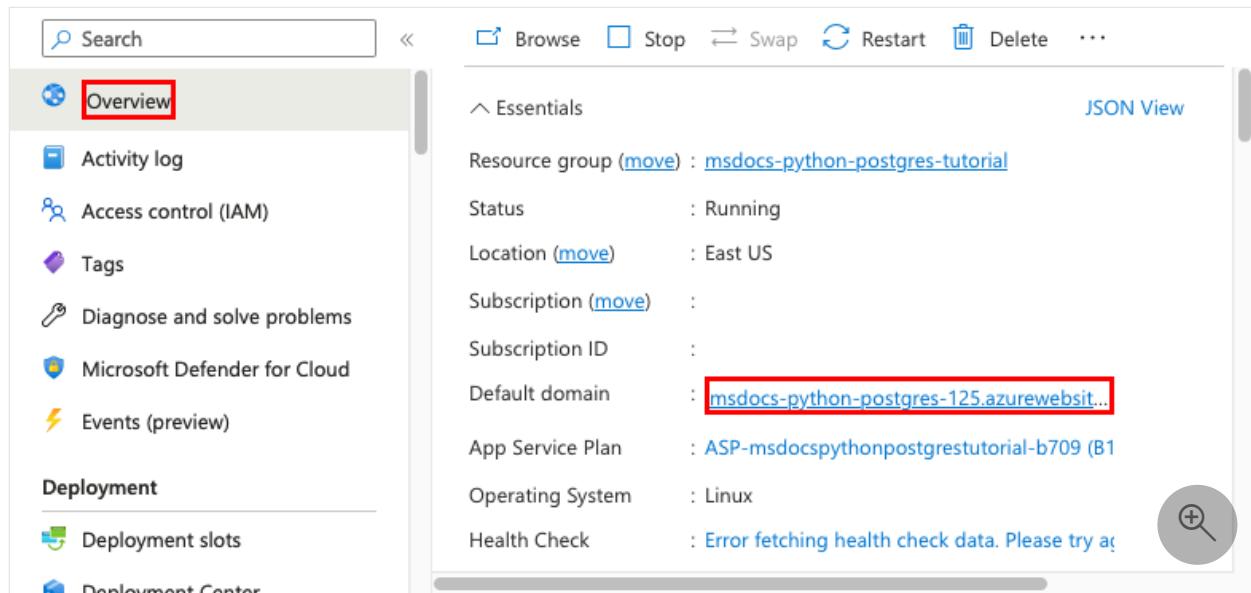
[django django-admin and manage.py](#). Use the superuser account to access the `/admin` portion of the web site.

Having issues? Check the [Troubleshooting section](#).

6. Browse to the app

Step 1: In the App Service page:

1. From the left menu, select **Overview**.
2. Select the URL of your app.



The screenshot shows the Azure App Service Overview page. On the left, there's a navigation menu with links like 'Activity log', 'Access control (IAM)', 'Tags', 'Diagnose and solve problems', 'Microsoft Defender for Cloud', 'Events (preview)', 'Deployment slots', and 'Deployment Center'. The 'Overview' link is highlighted with a red box. On the right, there's a summary card with various details: Resource group (move) : msdocs-python-postgres-tutorial, Status : Running, Location (move) : East US, Subscription (move) : , Subscription ID : , Default domain : msdocs-python-postgres-125.azurewebsites.net (highlighted with a red box), App Service Plan : ASP-msdocspythonpostgrestutorial-b709 (B1), Operating System : Linux, and Health Check : Error fetching health check data. Please try again. There's also a 'JSON View' button and a magnifying glass icon.

Step 2: Add a few restaurants to the list. Congratulations, you're running a web app in Azure App Service, with secure connectivity to Azure Database for PostgreSQL.

Last viewed restaurant (saved in cache): Contoso Restaurant Restaurants

Name	Rating	Details
Contoso Restaurant	★★★★★	4.0 (3 reviews) Details
Fourth Coffee	★★	2.0 (2 reviews) Details

[Add new restaurant](#)

7. Stream diagnostic logs

Azure App Service captures all console logs to help you diagnose issues with your application. The sample app includes `print()` statements to demonstrate this capability as shown below.

Python

```
def index(request):
    print('Request for index page received')
    restaurants =
        Restaurant.objects.annotate(avg_rating=Avg('review__rating')).annotate(
            review_count=Count('review'))
    lastViewedRestaurant = request.session.get("lastViewedRestaurant",
        False)
```

Step 1: In the App Service page:

1. From the left menu, select **Monitoring > App Service logs**.
2. Under **Application logging**, select **File System**.
3. In the top menu, select **Save**.

The screenshot shows the 'msdocs-python-postgres-234 | App Service logs' configuration page in the Azure portal. The left sidebar has 'App Service logs' selected. At the top right are 'Save' (highlighted), 'Discard', and 'Send us your feedback' buttons. Under 'Application logging', 'File System' is selected (highlighted). Other settings include 'Quota (MB)' set to 35, 'Retention Period (Days)' (empty), and download options for 'FTP/deployment username' and 'FTP'. A 'Log stream' button is also visible.

Step 2: From the left menu, select **Log stream**. You see the logs for your app, including platform logs and logs from inside the container.

The screenshot shows the 'msdocs-python-postgres-234 | Log stream' page. The left sidebar has 'Log stream' selected. The main area displays log entries. One entry, '2022-10-05T18:49:35.087646396Z Request for index page received', is highlighted with a red box.

Learn more about logging in Python apps in the series on [setting up Azure Monitor for your Python application](#).

8. Clean up resources

When you're finished, you can delete all of the resources from your Azure subscription by deleting the resource group.

Step 1: In the search bar at the top of the Azure portal:

1. Enter the resource group name.
2. Select the resource group.

The screenshot shows the Microsoft Azure portal interface. At the top, there is a search bar with the text 'msdocs-django-postgres-tutorial' and a Copilot button. On the left, there's a sidebar with 'Azure services' and 'Resources' sections. Under 'Azure services', there are 'Create a resource', 'Resource group', 'Key vaults', and 'Azure Data for My...'. Under 'Resources', there are 'Recent' and 'Favorite' tabs. The main content area shows 'Resource Groups' with one item listed: 'msdocs-django-postgres-tutorial'. To the right, there are links for 'Virtual networks' and a search icon.

Step 2: In the resource group page, select Delete resource group.

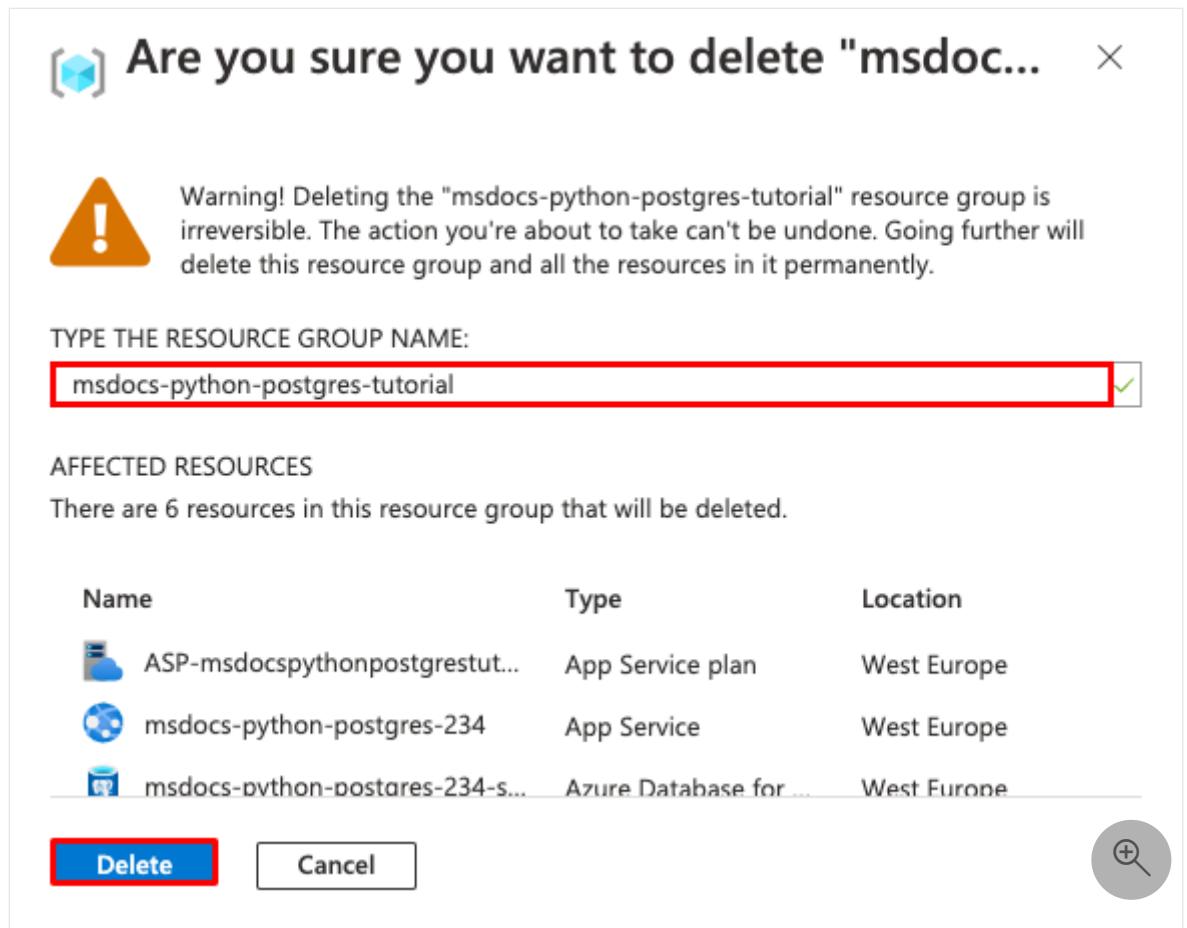
The screenshot shows the 'msdocs-django-postgres-tutorial' resource group page. At the top, there are buttons for '+ Create', 'Manage view', 'Delete resource group' (which is highlighted with a red box), and '...' (ellipsis). Below this, there's an 'Essentials' section with the following details:

- Subscription ([move](#)) : [Antares-Demo](#)
- Subscription ID : [\[REDACTED\]](#)
- Deployments : [7 Succeeded](#)
- Location : West Europe
- Tags ([edit](#)) : [Click here to add tags](#)

At the bottom, there are sections for 'Resources' and 'Recommendations (1)'. There are also filters and sorting options at the very bottom.

Step 3:

1. Enter the resource group name to confirm your deletion.
2. Select Delete.



Troubleshooting

Listed below are issues you might encounter while trying to work through this tutorial and steps to resolve them.

I can't connect to the SSH session

If you can't connect to the SSH session, then the app itself has failed to start. Check the [diagnostic logs](#) for details. For example, if you see an error like `KeyError: 'AZURE_POSTGRESQL_HOST'`, it might mean that the environment variable is missing (you might have removed the app setting).

I get an error when running database migrations

If you encounter any errors related to connecting to the database, check if the app settings (`AZURE_POSTGRESQL_USER`, `AZURE_POSTGRESQL_PASSWORD`, `AZURE_POSTGRESQL_HOST`,

and `AZURE_POSTGRESQL_NAME`) were changed or deleted. Without that connection string, the migrate command can't communicate with the database.

Frequently asked questions

- How much does this setup cost?
- How do I connect to the PostgreSQL server that's secured behind the virtual network with other tools?
- How does local app development work with GitHub Actions?
- How is the Django sample configured to run on Azure App Service?
- How do I change the SECRET_KEY app setting to a Key Vault reference?
- How do I debug errors during the GitHub Actions deployment?
- I don't have permissions to create a user-assigned identity
- What can I do with GitHub Copilot in my codespace?

How much does this setup cost?

Pricing for the created resources is as follows:

- The App Service plan is created in **Basic** tier and can be scaled up or down. See [App Service pricing](#).
- The PostgreSQL flexible server is created in the lowest burstable tier **Standard_B1ms**, with the minimum storage size, which can be scaled up or down. See [Azure Database for PostgreSQL pricing](#).
- The virtual network doesn't incur a charge unless you configure extra functionality, such as peering. See [Azure Virtual Network pricing](#).
- The private DNS zone incurs a small charge. See [Azure DNS pricing](#).

How do I connect to the PostgreSQL server that's secured behind the virtual network with other tools?

- For basic access from a command-line tool, you can run `psql` from the app's SSH session.
- To connect from a desktop tool, your machine must be within the virtual network. For example, it could be an Azure VM that's connected to one of the subnets, or a machine in an on-premises network that has a [site-to-site VPN](#) connection with the Azure virtual network.
- You can also [integrate Azure Cloud Shell](#) with the virtual network.

How does local app development work with GitHub Actions?

Using the autogenerated workflow file from App Service as an example, each `git push` kicks off a new build and deployment run. From a local clone of the GitHub repository, you make the desired updates and push to GitHub. For example:

terminal

```
git add .
git commit -m "<some-message>"
git push origin main
```

How is the Django sample configured to run on Azure App Service?

The [Django sample application](#) configures settings in the `azureproject/production.py` file so that it can run in Azure App Service. These changes are common to deploying Django to production, and not specific to App Service.

- Django validates the `HTTP_HOST` header in incoming requests. The sample code uses the [WEBSITE_HOSTNAME environment variable in App Service](#) to add the app's domain name to Django's [ALLOWED_HOSTS](#) setting.

Python

```
# Configure the domain name using the environment variable
# that Azure automatically creates for us.
ALLOWED_HOSTS = [os.environ['WEBSITE_HOSTNAME']] if 'WEBSITE_HOSTNAME'
in os.environ else []
```

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

Python

```
# WhiteNoise configuration
MIDDLEWARE = [
    'django.middleware.security.SecurityMiddleware',
    # Add whitenoise middleware after the security middleware
    'whitenoise.middleware.WhiteNoiseMiddleware',
```

Then the static file settings are configured according to the Django documentation.

Python

```
SESSION_ENGINE = "django.contrib.sessions.backends.cache"
STATICFILES_STORAGE =
'whitenoise.storage.CompressedManifestStaticFilesStorage'
```

For more information, see [Production settings for Django apps](#).

How do I change the SECRET_KEY app setting to a Key Vault reference?

From the portal steps above, you can change `SECRET_KEY` to a Key Vault reference by running the following Azure CLI commands in the [cloud shell](#):

Azure CLI

```
# Change the following variables to match your environment
SUBSCRIPTION_ID=<subscription-id>
RESOURCE_GROUP=<resource-group-name>
KEY_VAULT_NAME=<key-vault-name>
APP_SERVICE_NAME=<app-name>
SECRET_NAME=djangoSecretKey

# Set the subscription ID
az account set --subscription $SUBSCRIPTION_ID

# Assign 'Key Vault Secrets Officer' role to your user at the scope of the
# key vault
az role assignment create \
    --assignee $(az ad signed-in-user show --query id -o tsv) \
    --role $(az role definition list --name "Key Vault Secrets Officer" -- \
    query "[].id" -o tsv) \
    --scope $(az keyvault show --name $KEY_VAULT_NAME --resource-group \
$RESOURCE_GROUP --query id --output tsv)

# Add the secret to the key vault
az keyvault secret set \
    --vault-name $KEY_VAULT_NAME \
    --name $SECRET_NAME \
    --value $(python -c 'import secrets; print(secrets.token_hex())')

# Add Key Vault reference to the App Service configuration
az webapp config appsettings set \
    --resource-group $RESOURCE_GROUP \
    --name $APP_SERVICE_NAME \
    --settings
"SECRET_KEY=@Microsoft.KeyVault(SecretUri=https://$KEY_VAULT_NAME.vault.azure.net/secrets/$SECRET_NAME)"
```

You can also do the same thing in the portal. For more information, see:

1. [Key Vault scope role assignment](#)
2. [Add a secret to Key Vault](#)
3. [Retrieve a secret from Key Vault](#)
4. [Configure app settings](#)

How do I debug errors during the GitHub Actions deployment?

If a step fails in the autogenerated GitHub workflow file, try modifying the failed command to generate more verbose output. For example, you can get more output from the `python` command by adding the `-d` option. Commit and push your changes to trigger another deployment to App Service.

I don't have permissions to create a user-assigned identity

See [Set up GitHub Actions deployment from the Deployment Center](#).

What can I do with GitHub Copilot in my codespace?

You might have noticed that the GitHub Copilot chat view was already there for you when you created the codespace. For your convenience, we include the GitHub Copilot chat extension in the container definition (see `.devcontainer/devcontainer.json`). However, you need a [GitHub Copilot account](#) (30-day free trial available).

A few tips for you when you talk to GitHub Copilot:

- In a single chat session, the questions and answers build on each other and you can adjust your questions to fine-tune the answer you get.
- By default, GitHub Copilot doesn't have access to any file in your repository. To ask questions about a file, open the file in the editor first.
- To let GitHub Copilot have access to all of the files in the repository when preparing its answers, begin your question with `@workspace`. For more information, see [Use the @workspace agent](#).
- In the chat session, GitHub Copilot can suggest changes and (with `@workspace`) even where to make the changes, but it's not allowed to make the changes for you. It's up to you to add the suggested changes and test it.

Next steps

Advance to the next tutorial to learn how to secure your app with a custom domain and certificate.

Secure with custom domain and certificate

Learn how App Service runs a Python app:

Configure Python app

Feedback

Was this page helpful?

 Yes

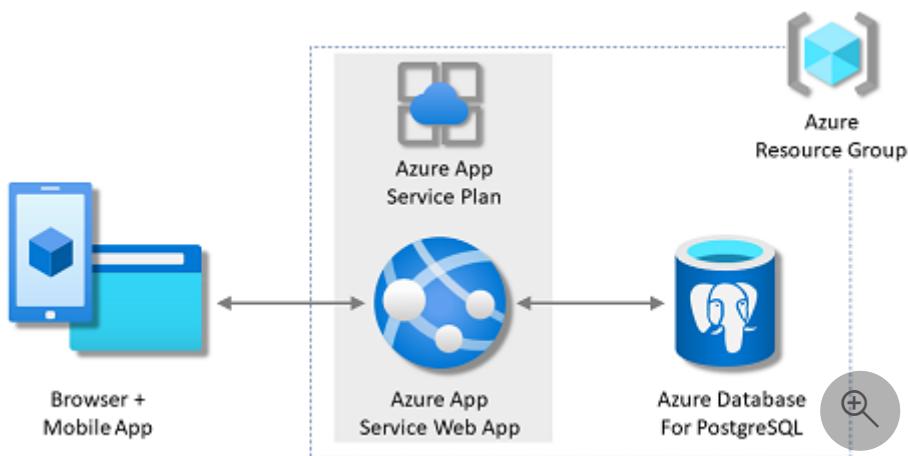
 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

Deploy a Python FastAPI web app with PostgreSQL in Azure

Article • 07/26/2024

In this tutorial, you deploy a data-driven Python web app ([FastAPI](#)) to [Azure App Service](#) with the [Azure Database for PostgreSQL](#) relational database service. Azure App Service supports [Python](#) in a Linux server environment. If you want, see the [Flask tutorial](#) or the [Django tutorial](#) instead.



To complete this tutorial, you'll need:

- An Azure account with an active subscription. If you don't have an Azure account, you [can create one for free](#).
- Knowledge of Python with FastAPI development

Skip to the end

With [Azure Developer CLI](#) installed, you can skip to the end of the tutorial by running the following commands in an empty working directory:

```
Bash
```

```
azd auth login  
azd init --template msdocs-fastapi-postgresql-sample-app  
azd up
```

Sample application

A sample Python application using FastAPI framework is provided to help you follow along with this tutorial. To deploy it without running it locally, skip this part.

To run the application locally, make sure you have [Python 3.8 or higher](#) and [PostgreSQL](#) installed locally. Then, clone the sample repository's `starter-no-infra` branch and change to the repository root.

Bash

```
git clone -b starter-no-infra https://github.com/Azure-Samples/msdocs-fastapi-postgresql-sample-app  
cd msdocs-fastapi-postgresql-sample-app
```

Create an `.env` file as shown below using the `.env.sample` file as a guide. Set the value of `DBNAME` to the name of an existing database in your local PostgreSQL instance. Set the values of `DBHOST`, `DBUSER`, and `DBPASS` as appropriate for your local PostgreSQL instance.

```
DBNAME=<database name>  
DBHOST=<database-hostname>  
DBUSER=<db-user-name>  
DBPASS=<db-password>
```

Create a virtual environment for the app:

Windows

Console

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

Install the dependencies:

Bash

```
python3 -m pip install -r src/requirements.txt
```

Install the app as an editable package:

Bash

```
python3 -m pip install -e src
```

Run the sample application with the following commands:

Bash

```
# Run database migration
python3 src/fastapi_app/seed_data.py
# Run the app at http://127.0.0.1:8000
python3 -m uvicorn fastapi_app:app --reload --port=8000
```

1. Create App Service and PostgreSQL

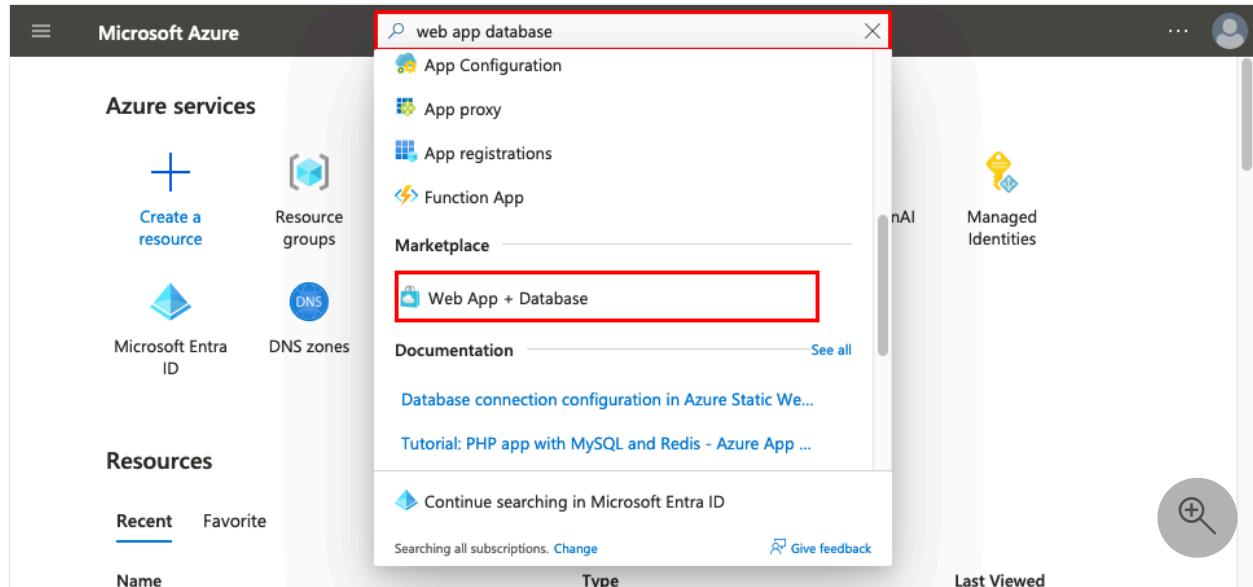
In this step, you create the Azure resources. The steps used in this tutorial create a set of secure-by-default resources that include App Service and Azure Database for PostgreSQL. For the creation process, you specify:

- The **Name** for the web app. It's the name used as part of the DNS name for your webapp in the form of `https://<app-name>.azurewebsites.net`.
- The **Region** to run the app physically in the world.
- The **Runtime stack** for the app. It's where you select the version of Python to use for your app.
- The **Hosting plan** for the app. It's the pricing tier that includes the set of features and scaling capacity for your app.
- The **Resource Group** for the app. A resource group lets you group (in a logical container) all the Azure resources needed for the application.

Sign in to the [Azure portal](#) and follow these steps to create your Azure App Service resources.

Step 1: In the Azure portal:

1. Enter "web app database" in the search bar at the top of the Azure portal.
2. Select the item labeled **Web App + Database** under the **Marketplace** heading. You can also navigate to the [creation wizard](#) directly.



Step 2: In the **Create Web App + Database** page, fill out the form as follows.

1. *Resource Group* → Select **Create new** and use a name of **msdocs-python-postgres-tutorial**.
2. *Region* → Any Azure region near you.
3. *Name* → **msdocs-python-postgres-XYZ** where **XYZ** is any three random characters.
This name must be unique across Azure.
4. *Runtime stack* → **Python 3.12**.
5. *Database* → **PostgreSQL - Flexible Server** is selected by default as the database engine. The server name and database name are also set by default to appropriate values.
6. *Hosting plan* → **Basic**. When you're ready, you can **scale up** to a production pricing tier later.
7. Select **Review + create**.
8. After validation completes, select **Create**.

Create Web App + Database

X

Basics Tags Review + create

This template will create a secure by default configuration where the only publicly accessible endpoint will be your app following the recommended security best practices. [Learn more](#)

Project Details

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

Subscription * ⓘ

Antares-Demo

Resource Group * ⓘ

(New) msdocs-python-postgres-tutorial

[Create new](#)

Region *

West Europe

Web App Details

Name *

msdocs-python-postgres-234



.azurewebsites.net

Runtime stack *

Python 3.9

Database

i Database access will be locked down and not exposed to the public internet. This is in compliance with recommended best practices for security.

Engine * ⓘ

PostgreSQL - Flexible Server (recommended)



Server name *

msdocs-python-postgres-234-server



Database name *

msdocs-python-postgres-234-database



Hosting

Hosting plan *

- Basic - For hobby or research purposes
 Standard - General purpose production apps

[Review + create](#)

[< Previous](#)

[Next : Tags >](#)



Step 3: The deployment takes a few minutes to complete. Once deployment completes, select the **Go to resource** button. You're taken directly to the App Service app, but the following resources are created:

- **Resource group** → The container for all the created resources.
- **App Service plan** → Defines the compute resources for App Service. A Linux plan in the *Basic* tier is created.

- **App Service** → Represents your app and runs in the App Service plan.
- **Virtual network** → Integrated with the App Service app and isolates back-end network traffic.
- **Azure Database for PostgreSQL flexible server** → Accessible only from within the virtual network. A database and a user are created for you on the server.
- **Private DNS zone** → Enables DNS resolution of the PostgreSQL server in the virtual network.

 Your deployment is complete

 Deployment name : Microsoft.Web-WebAppDatabase-Portal-afa69d9d-97bc
Subscription :
Resource group : [msdocs-python-postgres-tutorial](#)
Start time : 11/29/2023, 10:17:05 AM
Correlation ID :

> Deployment details

▽ Next steps

[Go to resource](#)

Give feedback 

 Tell us about your experience with deployment

Step 4: For FastAPI apps, you must enter a startup command so App service can start your app. On the App Service page:

1. In the left menu, under **Settings**, select **Configuration**.
2. In the **General settings** tab of the **Configuration** page, enter `src/entrypoint.sh` in the **Startup Command** field under **Stack settings**.
3. Select **Save**. When prompted, select **Continue**. To learn more about app configuration and startup in App Service, see [Configure a Linux Python app for Azure App Service](#).

The screenshot shows the 'Configuration' tab selected in the left sidebar of the Azure App Service configuration interface. In the main pane, under 'Stack settings', the 'Startup Command' field contains the value 'src/entrypoint.sh', which is highlighted with a red box. A tooltip below the field reads: 'Provide an optional startup command that will be run as part of container startup. [Learn more](#)'.

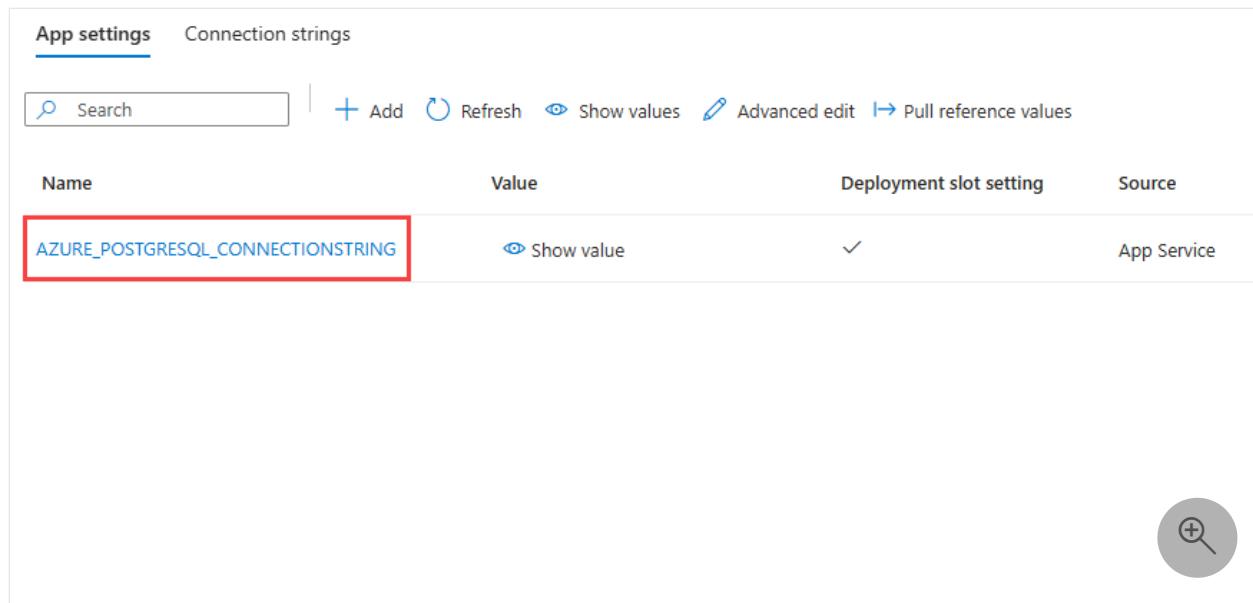
2. Verify connection settings

The creation wizard generated the connectivity variables for you already as [app settings](#). App settings are one way to keep connection secrets out of your code repository. When you're ready to move your secrets to a more secure location, here's an [article on storing in Azure Key Vault](#).

Step 1: In the App Service page, in the left menu, select **Environment variables**.

The screenshot shows the 'Settings' section of the Azure App Service configuration interface. The 'Environment variables' tab is highlighted with a red box in the left sidebar. Other tabs visible include Configuration, Authentication, Application Insights, Identity, Backups, Custom domains, and Certificates. A search bar and a magnifying glass icon are also present.

Step 2: In the App settings tab of the Environment variables page, verify that `AZURE_POSTGRESQL_CONNECTIONSTRING` is present. The connection string will be injected into the runtime environment as an environment variable.



The screenshot shows the 'App settings' tab of the Azure Environment variables page. At the top, there are tabs for 'App settings' (which is selected) and 'Connection strings'. Below the tabs is a search bar and a set of actions: 'Add', 'Refresh', 'Show values', 'Advanced edit', and 'Pull reference values'. A table follows, with columns: 'Name', 'Value', 'Deployment slot setting', and 'Source'. A single row is present: 'AZURE_POSTGRESQL_CONNECTIONSTRING' with a value of '(empty)' and 'App Service' as the source. The 'Value' column contains a link labeled 'Show value'. A red box highlights the 'AZURE_POSTGRESQL_CONNECTIONSTRING' row. In the bottom right corner of the table area, there is a circular button with a plus sign and a magnifying glass icon.

Name	Value	Deployment slot setting	Source
AZURE_POSTGRESQL_CONNECTIONSTRING	(empty)	✓	App Service

3. Deploy sample code

In this step, you configure GitHub deployment using GitHub Actions. It's just one of many ways to deploy to App Service, but also a great way to have continuous integration in your deployment process. By default, every `git push` to your GitHub repository will kick off the build and deploy action.

Step 1: In a new browser window:

1. Sign in to your GitHub account.
2. Navigate to <https://github.com/Azure-Samples/msdocs-fastapi-postgresql-sample-app>.
3. Select **Fork**.
4. Select **Create fork**.

A screenshot of a GitHub repository page for 'msdocs-fastapi-postgresql-sample-app'. The page shows a list of pull requests and a sidebar with repository details like 'About', 'Readme', and 'Code of conduct'. The 'Fork' button in the top right corner is highlighted with a red box.

Step 2: In the GitHub page, open Visual Studio Code in the browser by pressing the `.` key.

A screenshot of a GitHub repository page for a forked repository, showing the message 'Press the '.' key' in the top navigation bar. The rest of the interface is identical to the previous screenshot, including the list of pull requests and the sidebar.

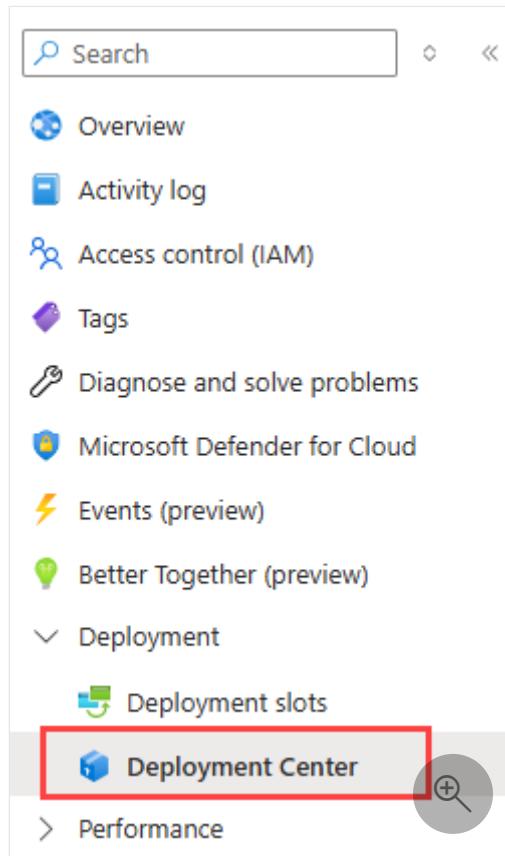
Step 3: In Visual Studio Code in the browser, open `src/fastapi/models.py` in the explorer. See the environment variables being used in the production environment, including the app settings that you saw in the configuration page.

```

src > fastapi_app > models.py > ...
1 import os
2 import typing
3 from datetime import datetime
4 from urllib.parse import quote_plus
5
6 from dotenv import load_dotenv
7 from sqlmodel import Field, SQLModel, create_engine
8
9 sql_url = ""
10 if os.getenv("WEBSITE_HOSTNAME"):
11     print("Connecting to Azure PostgreSQL Flexible server based on AZURE_POSTGRESQL_CONNECTIONSTRING...")
12     env_connection_string = os.getenv("AZURE_POSTGRESQL_CONNECTIONSTRING")
13     if env_connection_string is None:
14         print("Missing environment variable AZURE_POSTGRESQL_CONNECTIONSTRING")
15     else:
16         # Parse the connection string
17         details = dict(item.split('=') for item in env_connection_string.split())
18
19         # Properly format the URL for SQLAlchemy
20         sql_url = (
21             f"postgresql://{{quote_plus(details['user'])}}:{quote_plus(details['password'])}"
22             f"@{{details['host']}:{details['port']}}/{{details['dbname']}}?sslmode={{details['sslmode']}}"
23         )
24
25 else:
26     print("Connecting to local PostgreSQL server based on .env file...")
27     load_dotenv()
28     POSTGRES_USERNAME = os.environ.get("DBUSER")
29     POSTGRES_PASSWORD = os.environ.get("DBPASS")
30     POSTGRES_HOST = os.environ.get("DBHOST")
31     POSTGRES_DATABASE = os.environ.get("DBNAME")
32     POSTGRES_PORT = os.environ.get("DBPORT", 5432)
33
34     sql_url = f"postgresql://{{POSTGRES_USERNAME}}:{POSTGRES_PASSWORD}@{{POSTGRES_HOST}}:{{POSTGRES_PORT}}/{{POSTGRES_}}

```

Step 4: Back in the App Service page, in the left menu, under **Deployment**, select **Deployment Center**.



Step 5: In the Deployment Center page:

1. In **Source**, select **GitHub**. By default, **GitHub Actions** is selected as the build provider.
2. Sign in to your GitHub account and follow the prompt to authorize Azure.

3. In **Organization**, select your account.
4. In **Repository**, select `msdocs-fastapi-postgresql-sample-app`.
5. In **Branch**, select `main`.
6. Keep the default option selected to **Add a workflow**.
7. Under **Authentication type**, select **User-assigned identity**.
8. In the top menu, select **Save**. App Service commits a workflow file into the chosen GitHub repository, in the `.github/workflows` directory.

Save Discard Browse Manage publish profile Sync Leave Feedback

Settings * Logs FTPS credentials

Deploy and build code from your preferred source and build provider. [Learn more](#)

Source* GitHub

Building with GitHub Actions. [Change provider](#).

GitHub

App Service will place a GitHub Actions workflow in your chosen repository to build and deploy your app whenever there is a commit on the chosen branch. If you can't find an organization or repository, you may need to enable additional permissions on GitHub. You must have write access to your chosen GitHub repository to deploy with GitHub Actions. [Learn more](#)

Signed in as Someone [Change Account](#)

Organization* <github-alias>

Repository* msdocs-fastapi-postgresql-sample-...

Branch* main

Workflow Option* Add a workflow: Add a new workflow file 'main_msdocs-python-postgres-235.yml' in the selected repository and branch. Use available workflow: Use one of the workflow files available in the selected repository and branch.

Build

Runtime stack Python

Version Python 3.12

Authentication settings

Select how you want your GitHub Action workflow to authenticate to Azure. If you choose user-assigned identity, the identity selected will be federated with GitHub as an authorized client and given write permissions on the app. [Learn more](#)

Authentication type* User-assigned identity Basic authentication

Step 6: In the Deployment Center page:

1. Select **Logs**. A deployment run is already started.
2. In the log item for the deployment run, select **Build/Deploy Logs**.

The screenshot shows the Azure portal's deployment logs for an Azure App Service. The 'Logs' tab is selected. The table has columns: Time, Commit ID, Log Type, Commit Author, Status, and Message. There are two entries for Tuesday, July 23, 2024. The second entry, which is highlighted with a red box, shows a 'Log Type' of 'Build/Deploy Lo...', an author of 'someone', and a message about updating the build and deployment workflow config.

Step 7: You're taken to your GitHub repository and see that the GitHub action is running. The workflow file defines two separate stages, build and deploy. Wait for the GitHub run to show a status of **Complete**. It takes about 5 minutes.

The screenshot shows a GitHub Actions run page for a workflow named 'main_msdocs-python-postgres-235.yml'. The 'Actions' tab is selected. A single job titled 'Build and deploy Python app to Azure Web App - msdocs-python-postgres-235 #2' is shown in progress. The summary indicates it was triggered manually 4 minutes ago and is currently 'In progress'. The status table shows 1 artifact. The workflow file name 'main_msdocs-python-postgres-235.yml' and the trigger 'on: workflow_dispatch' are also visible.

Having issues? Check the [Troubleshooting guide](#).

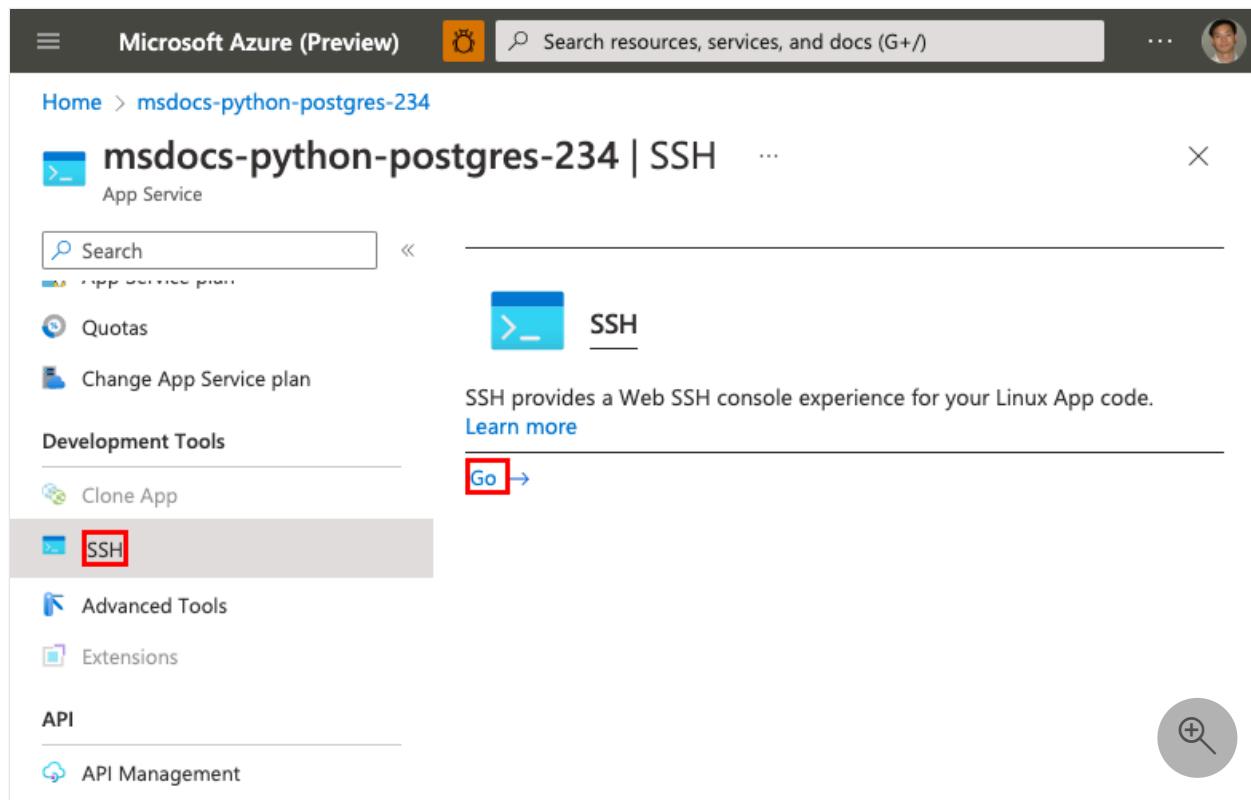
4. Generate database schema

In previous section, you added `src/entrypoint.sh` as the startup command for your app. `entrypoint.sh` contains the following line: `python3 src/fastapi_app/seed_data.py`. This command migrates your database. In the sample app, it only ensures that the correct tables are created in your database. It doesn't populate these tables with any data.

In this section, you'll run this command manually for demonstration purposes. With the PostgreSQL database protected by the virtual network, the easiest way to run the command is in an SSH session with the App Service container.

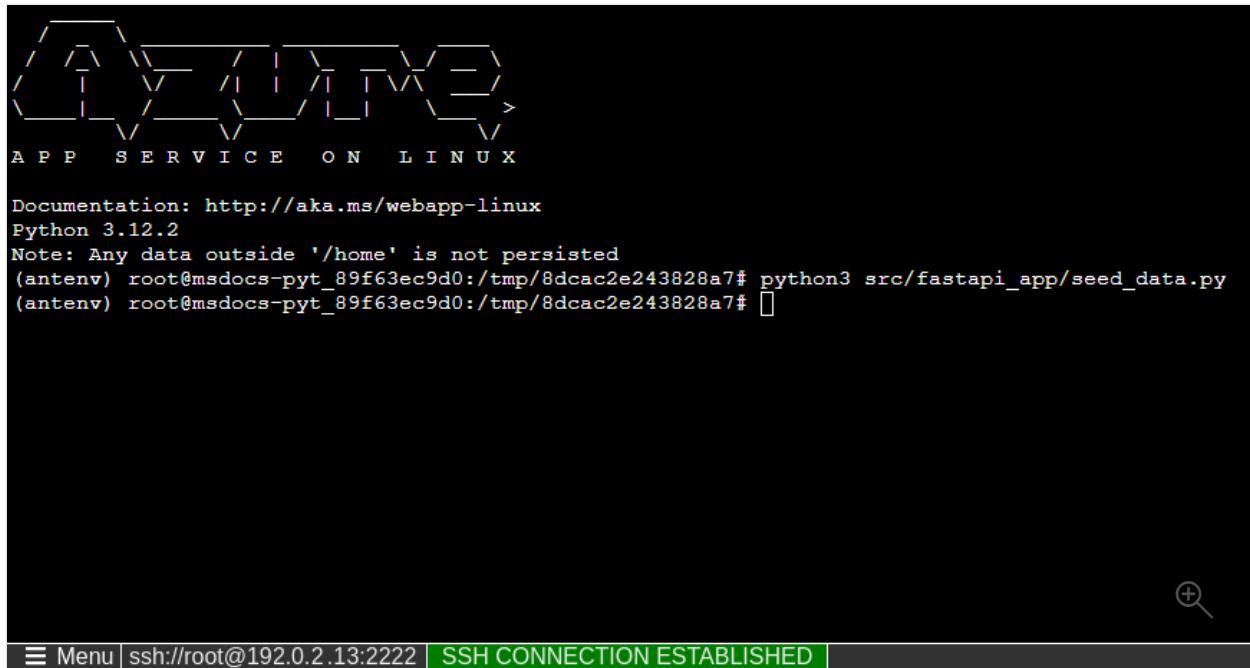
Step 1: Back in the App Service page, in the left menu,

1. Select SSH.
2. Select Go.



Step 2: In the SSH terminal, run `python3 src/fastapi_app/seed_data.py`. If it succeeds, App Service is [connecting successfully to the database](#). Only changes to files in `/home` can persist beyond app restarts. Changes outside of `/home` aren't persisted.

```
APP SERVICE ON LINUX
Documentation: http://aka.ms/webapp-linux
Python 3.12.2
Note: Any data outside '/home' is not persisted
(antenv) root@msdocs-pyt_89f63ec9d0:/tmp/8dcac2e243828a7# python3 src/fastapi_app/seed_data.py
(antenv) root@msdocs-pyt_89f63ec9d0:/tmp/8dcac2e243828a7# 
```

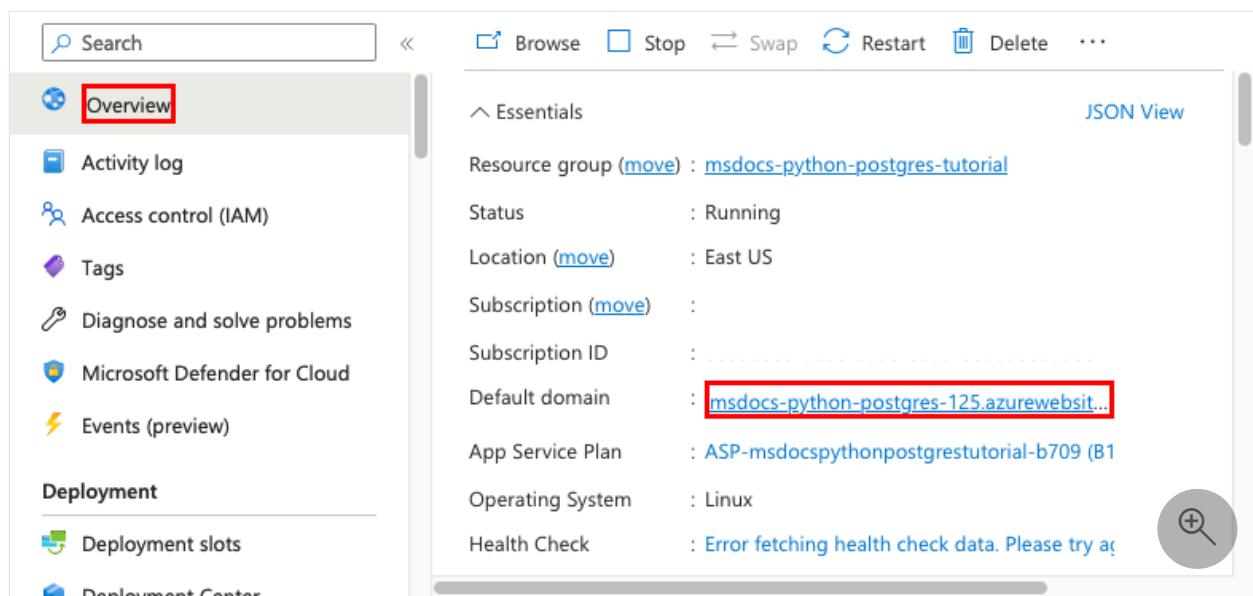


The screenshot shows an SSH session with a black background. At the top, there's a decorative logo consisting of various symbols like brackets and arrows. Below it, the text "APP SERVICE ON LINUX" is displayed in a monospaced font. A message about documentation and Python version follows. A note about data persistence is present, along with a command history showing the execution of "seed_data.py". The bottom of the terminal window has a dark bar with the text "☰ Menu ssh://root@192.0.2.13:2222 SSH CONNECTION ESTABLISHED" and a magnifying glass icon.

5. Browse to the app

Step 1: In the App Service page:

1. From the left menu, select **Overview**.
2. Select the URL of your app. You can also navigate directly to `https://<app-name>.azurewebsites.net`.



The screenshot shows the Azure App Service Overview page. On the left, a sidebar lists navigation options: Search, Overview (which is selected and highlighted with a red box), Activity log, Access control (IAM), Tags, Diagnose and solve problems, Microsoft Defender for Cloud, Events (preview), Deployment (with sub-options Deployment slots and Deployment Center), and a plus sign icon. The main content area is titled "Essentials" and shows resource group, status, location, subscription, default domain, app service plan, operating system, and health check details. The "Default domain" field contains the URL "msdocs-python-postgres-125.azurewebsites.net", which is also highlighted with a red box. A "JSON View" link is located in the top right corner of the essentials section. A magnifying glass icon is in the bottom right corner of the main content area.

Step 2: Add a few restaurants to the list. Congratulations, you're running a web app in Azure App Service, with secure connectivity to Azure Database for PostgreSQL.

Restaurants

Name	Rating	Details
Fourth Coffee	★ ★	2.0 (2 reviews) Details
Contoso Restaurant	★ ★ ★ ★	4.0 (3 reviews) Details

[Add new restaurant](#)

6. Stream diagnostic logs

The sample app uses the Python Standard Library logging module to help you diagnose issues with your application. The sample app includes calls to the logger as shown in the following code.

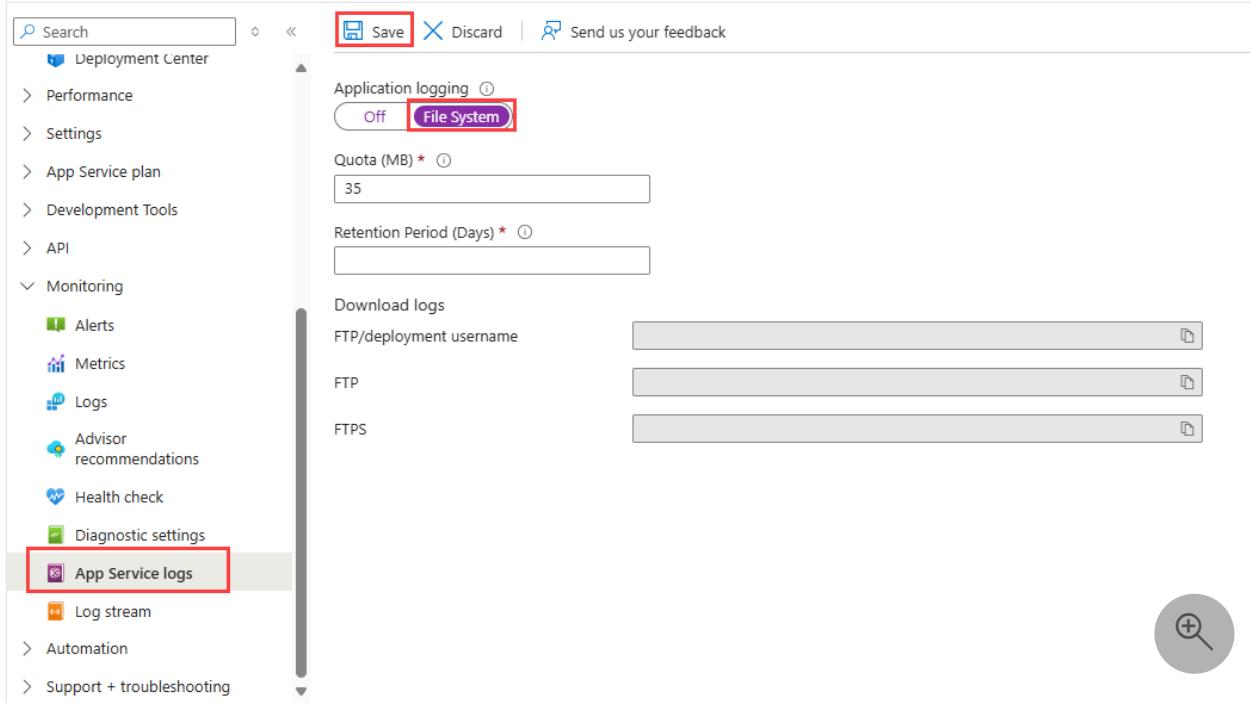
Python

```
@app.get("/", response_class=HTMLResponse)
async def index(request: Request, session: Session =
Depends(get_db_session)):
    logger.info("root called")
    statement = (
        select(Restaurant, func.avg(Review.rating).label("avg_rating"),
func.count(Review.id).label("review_count"))
        .outerjoin(Review, Review.restaurant == Restaurant.id)
        .group_by(Restaurant.id)
    )
```

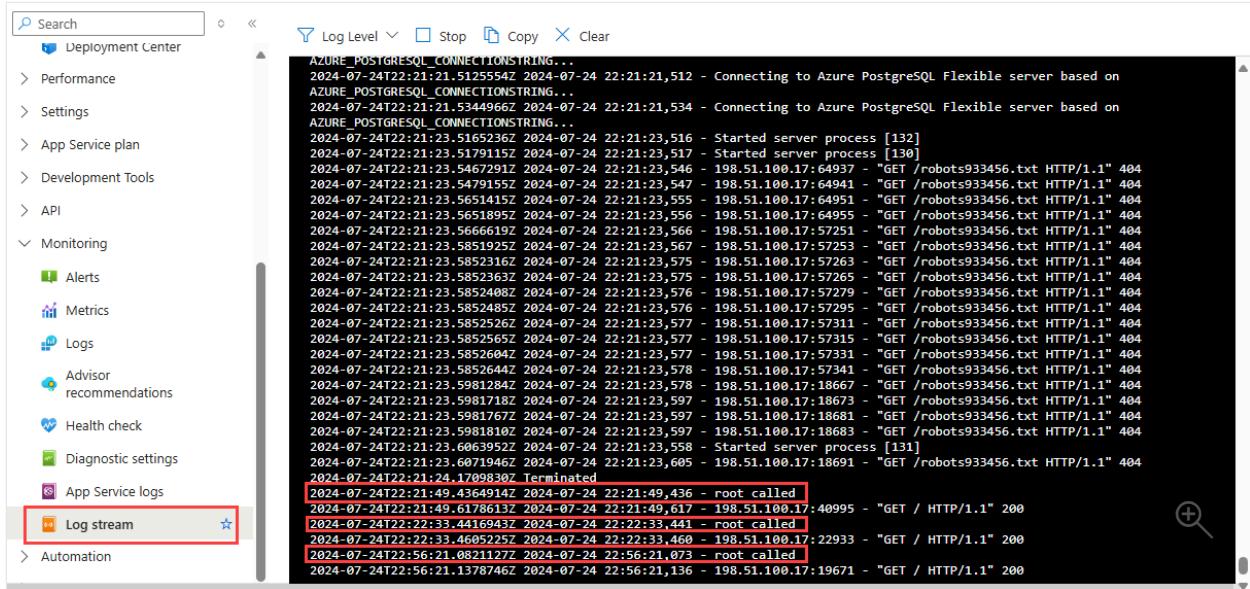
Step 1: In the App Service page:

1. From the left menu, under **Monitoring**, select **App Service logs**.
2. Under **Application logging**, select **File System**.

3. In the top menu, select **Save**.



Step 2: From the left menu, select **Log stream**. You see the logs for your app, including platform logs and logs from inside the container.



Events can take several minutes to show up in the diagnostic logs. Learn more about logging in Python apps in the series on [setting up Azure Monitor for your Python application](#).

7. Clean up resources

When you're finished, you can delete all of the resources from your Azure subscription by deleting the resource group.

Step 1: In the search bar at the top of the Azure portal:

1. Enter the resource group name.
2. Select the resource group.

The screenshot shows the Microsoft Azure (Preview) interface. At the top, there is a search bar with the text "msdocs-python-postgres-tutorial" entered. Below the search bar, there are several navigation links: "All", "Resource Groups (1)", "Documentation (16)", "Services (0)", "Resources (0)", "Marketplace (0)", and "Azure Active Directory (0)". A red box highlights the search term in the search bar and the resulting resource group entry in the list. On the left side, there is a sidebar with sections for "Azure services" (Create a resource, App Service plans, Microsoft Dev Box, Subscriptions) and "Resources" (Recent, Favorite). At the bottom right, there is a "Give feedback" link and a magnifying glass icon.

Step 2: In the resource group page, select Delete resource group.

The screenshot shows the Azure Resource Group details page for the "msdocs-python-postgres-tutorial" group. At the top, there is a toolbar with "Create", "Manage view", "Delete resource group" (which is highlighted with a red box), and other options. Below the toolbar, there is a section for "Essentials" with the following details:

- Subscription ([move](#)) : [Antares-Demo](#)
- Subscription ID : 00000000-0000-0000-0000-0...
- Deployments : [7 Succeeded](#)
- Location : West Europe
- Tags ([edit](#)) : [Click here to add tags](#)

At the bottom, there is a "Resources" tab selected, showing "Recommendations (1)". There are also filters, a search bar, and pagination controls.

Step 3:

1. Enter the resource group name to confirm your deletion.
2. Select Delete.

 Are you sure you want to delete "msdoc... X

! Warning! Deleting the "msdocs-python-postgres-tutorial" resource group is irreversible. The action you're about to take can't be undone. Going further will delete this resource group and all the resources in it permanently.

TYPE THE RESOURCE GROUP NAME:

msdocs-python-postgres-tutorial 

AFFECTED RESOURCES

There are 6 resources in this resource group that will be deleted.

Name	Type	Location
 ASP-msdocspythonpostgrestut...	App Service plan	West Europe
 msdocs-python-postgres-234	App Service	West Europe
 msdocs-python-postares-234-s...	Azure Database for ...	West Europe

Delete **Cancel**  :

Troubleshooting

Listed below are issues you might encounter while trying to work through this tutorial and steps to resolve them.

I can't connect to the SSH session

If you can't connect to the SSH session, then the app itself has failed to start. Check the [diagnostic logs](#) for details. For example, if you see an error like `KeyError: 'AZURE_POSTGRESQL_CONNECTIONSTRING'`, it might mean that the environment variable is missing (you might have removed the app setting).

I get an error when running database migrations

If you encounter any errors related to connecting to the database, check if the app settings (`AZURE_POSTGRESQL_CONNECTIONSTRING`) have been changed. Without that connection string, the migrate command can't communicate with the database.

Frequently asked questions

- How much does this setup cost?
- How do I connect to the PostgreSQL server that's secured behind the virtual network with other tools?
- How does local app development work with GitHub Actions?

How much does this setup cost?

Pricing for the created resources is as follows:

- The App Service plan is created in **Basic** tier and can be scaled up or down. See [App Service pricing ↗](#).
- The PostgreSQL flexible server is created in the lowest burstable tier **Standard_B1ms**, with the minimum storage size, which can be scaled up or down. See [Azure Database for PostgreSQL pricing ↗](#).
- The virtual network doesn't incur a charge unless you configure extra functionality, such as peering. See [Azure Virtual Network pricing ↗](#).
- The private DNS zone incurs a small charge. See [Azure DNS pricing ↗](#).

How do I connect to the PostgreSQL server that's secured behind the virtual network with other tools?

- For basic access from a command-line tool, you can run `psql` from the app's SSH terminal.
- To connect from a desktop tool, your machine must be within the virtual network. For example, it could be an Azure VM that's connected to one of the subnets, or a machine in an on-premises network that has a [site-to-site VPN](#) connection with the Azure virtual network.
- You can also [integrate Azure Cloud Shell](#) with the virtual network.

How does local app development work with GitHub Actions?

Using the autogenerated workflow file from App Service as an example, each `git push` kicks off a new build and deployment run. From a local clone of the GitHub repository, you make the desired updates and push to GitHub. For example:

terminal

```
git add .
git commit -m "<some-message>"
git push origin main
```

Next steps

Advance to the next tutorial to learn how to secure your app with a custom domain and certificate.

[Secure with custom domain and certificate](#)

Learn how App Service runs a Python app:

[Configure Python app](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

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

Article • 01/31/2025

In this tutorial, you learn how to build, configure, and deploy a secure Spring Boot application in Azure App Service that connects to a MongoDB database in Azure (actually, a Cosmos DB database with MongoDB API). When you're finished, you'll have a Java SE application running on Azure App Service on Linux.

The screenshot shows a user interface for managing tasks or steps. At the top right is a blue 'Add' button. Below it is a list of three items, each with a checkbox and 'Delete' and 'Edit' buttons:

- Create Spring Boot app an MongoDB in Azure Delete Edit
- Deploy data-driven app Delete Edit
- That's it! Delete Edit

In this tutorial, you learn how to:

- ✓ Create a secure-by-default architecture for Azure App Service and Azure Cosmos DB with MongoDB API.
- ✓ Secure connection secrets using a managed identity and Key Vault references.
- ✓ Deploy a Spring Boot sample app to App Service from a GitHub repository.
- ✓ Access App Service app settings in the application code.
- ✓ Make updates and redeploy the application code.
- ✓ Stream diagnostic logs from App Service.
- ✓ Manage the app in the Azure portal.
- ✓ Provision the same architecture and deploy by using Azure Developer CLI.
- ✓ Optimize your development workflow with GitHub Codespaces and GitHub Copilot.

Prerequisites

- An Azure account with an active subscription. If you don't have an Azure account, you [can create one for free ↗](#).
- A GitHub account. you can also [get one for free ↗](#).
- Knowledge of Java with Spring Framework development.
- **(Optional)** To try GitHub Copilot, a [GitHub Copilot account ↗](#). A 30-day free trial is available.

Skip to the end

You can quickly deploy the sample app in this tutorial and see it running in Azure. Just run the following commands in the [Azure Cloud Shell ↗](#), and follow the prompt:

Bash

```
mkdir msdocs-spring-boot-mongodb-sample-app
cd msdocs-spring-boot-mongodb-sample-app
azd init --template msdocs-spring-boot-mongodb-sample-app
azd up
```

1. Run the sample

First, you set up a sample data-driven app as a starting point. For your convenience, the [sample repository ↗](#), includes a [dev container ↗](#) configuration. The dev container has everything you need to develop an application, including the MongoDB database, cache, and all environment variables needed by the sample application. The dev container can run in a [GitHub codespace ↗](#), which means you can run the sample on any computer with a web browser.

Step 1: In a new browser window:

1. Sign in to your GitHub account.
2. Navigate to <https://github.com/Azure-Samples/msdocs-spring-boot-mongodb-sample-app/fork>.
3. Unselect **Copy the main branch only**. You want all the branches.
4. Select **Create fork**.

Create a new fork

A *fork* is a copy of a repository. Forking a repository allows you to freely experiment with changes without affecting the original project.

Required fields are marked with an asterisk (*).

Owner *



Repository name *



/ msdocs-spring-boot-mongoc

msdocs-spring-boot-mongodb-sample-app is available.

By default, forks are named the same as their upstream repository. You can customize the name to distinguish it further.

Description (optional)

A Spring Boot CRUD sample application for Azure App Service and Azure Cosmos DB.

Copy the `main` branch only

Contribute back to Azure-Samples/msdocs-spring-boot-mongodb-sample-app by adding your own branch. [Learn more.](#)

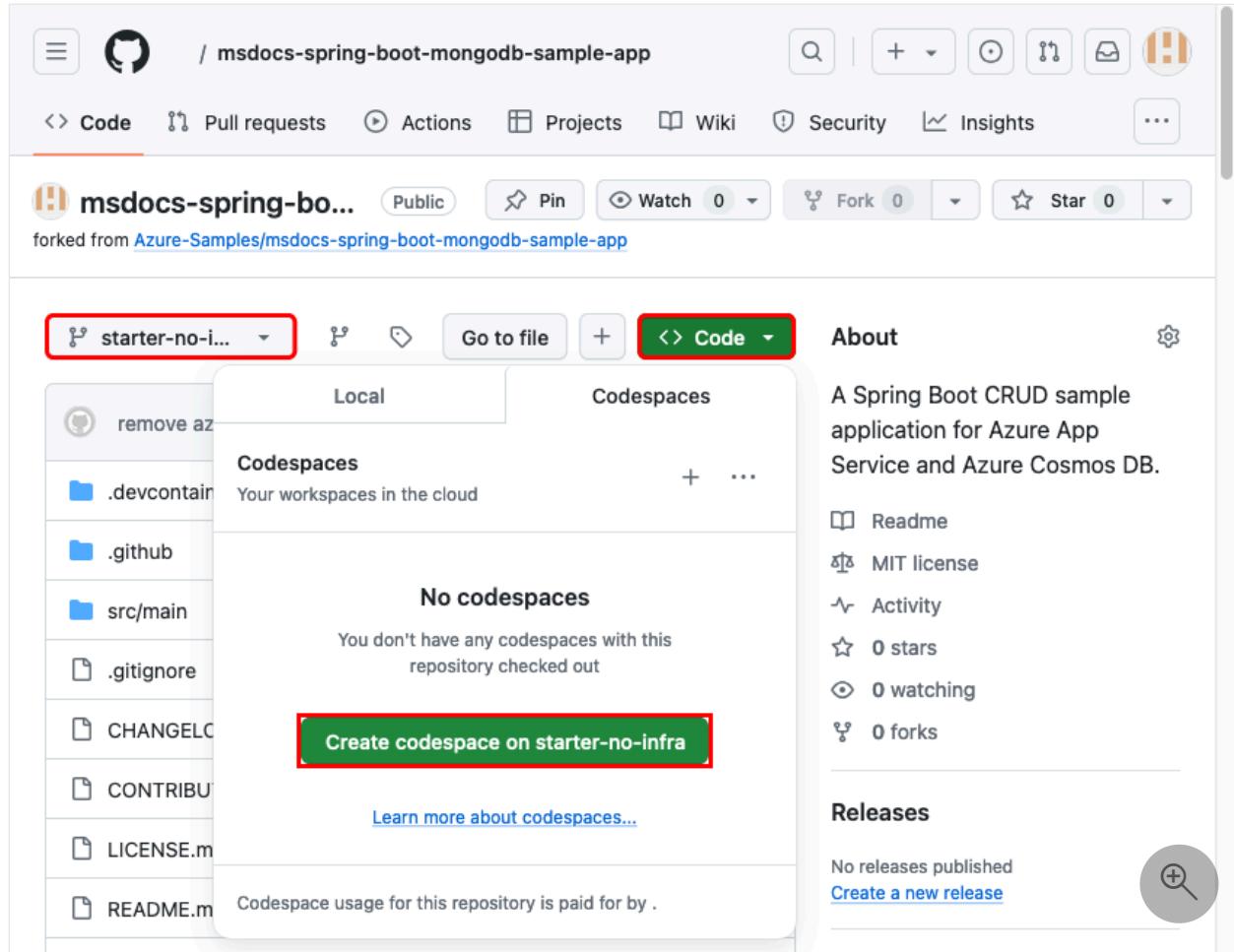
You are creating a fork in your personal account.

Create fork



Step 2: In the GitHub fork:

1. Select **main > starter-no-infra** for the starter branch. This branch contains just the sample project and no Azure-related files or configuration.
2. Select **Code > Create codespace on starter-no-infra**. The codespace takes a few minutes to set up.



Step 3: In the codespace terminal:

1. Run `mvn package spring-boot:run`.
2. When you see the notification `Your application running on port 8080 is available.`, select **Open in Browser**. You should see the sample application in a new browser tab. To stop the Jetty server, type `ctrl + c`.

This screenshot shows the Microsoft Visual Studio Code interface running in a Codespace. The left sidebar displays the project structure for 'MSDOCS-SPRING-BOOT-MONGODB-SA...', including files like .devcontainer, .github, src, .gitignore, CHANGELOG.md, CONTRIBUTING.md, LICENSE.md, pom.xml, and README.md. The main area shows a terminal window with the command 'mvn package spring-boot:run' highlighted in red. The output of the command shows INFO logs related to the MongoDB driver connecting to a database. A tooltip provides information about the application being available on port 8080. At the bottom right of the terminal, there are buttons for 'Open in Browser' and 'Make Public'. The status bar at the bottom shows 'Layout: U.S.'.

💡 Tip

You can ask [GitHub Copilot](#) about this repository. For example:

- @workspace *What does this project do?*
- @workspace *How does the app connect to the database?*
- @workspace *What does the .devcontainer folder do?*

Having issues? Check the [Troubleshooting section](#).

2. Create App Service and Cosmos DB

First, you create the Azure resources. The steps used in this tutorial create a set of secure-by-default resources that include App Service and Azure Cosmos DB. For the creation process, you specify:

- The **Name** for the web app. It's used as part of the DNS name for your app in the form of `https://<app-name>-<hash>.azurewebsites.net`.
- The **Region** to run the app physically in the world. It's also used as part of the DNS name for your app.
- The **Runtime stack** for the app. It's where you select the version of Java to use for your app.
- The **Hosting plan** for the app. It's the pricing tier that includes the set of features and scaling capacity for your app.

- The **Resource Group** for the app. A resource group lets you group (in a logical container) all the Azure resources needed for the application.

Sign in to the [Azure portal](#) and follow these steps to create your Azure App Service resources.

Step 1: In the Azure portal:

1. Enter "web app database" in the search bar at the top of the Azure portal.
2. Select the item labeled **Web App + Database** under the **Marketplace** heading. You can also navigate to the [creation wizard](#) directly.

The screenshot shows the Microsoft Azure portal interface. At the top, there's a search bar with the text "web app database". Below the search bar, the "Marketplace" section is visible, showing various service categories like "App Services", "SQL databases", "Azure Database for MySQL servers", and "Static Web Apps". Under the "Marketplace" heading, the "Web App + Database" item is listed and highlighted with a red box. To the right of the search bar, there's a user profile icon and some navigation links. On the left side, there's a sidebar with options like "Create a resource", "Azure DevOps organizations", and "Resources".

Step 2: In the **Create Web App + Database** page, fill out the form as follows.

1. **Resource Group:** Select **Create new** and use a name of **msdocs-spring-cosmosdb-tutorial**.
2. **Region:** Any Azure region near you.
3. **Name:** **msdocs-spring-cosmosdb-XYZ** where **XYZ** is any three random characters. This name must be unique across Azure.
4. **Runtime stack:** **Java 21**.
5. **Java web server stack:** **Java SE (Embedded Web Server)**.
6. **Engine:** **Cosmos DB API for MongoDB**. Cosmos DB is a fully managed NoSQL, relational, and vector database as a service on Azure.
7. **Hosting plan:** **Basic**. When you're ready, you can [scale up](#) to a production pricing tier.
8. Select **Review + create**.
9. After validation completes, select **Create**.

Basics Tags Review + create

This template will create a secure by default configuration where the only publicly accessible endpoint will be your app following the recommended security best practices. [Learn more](#)

Project Details

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

Subscription * ⓘ

Visual Studio Enterprise Subscription

Resource Group * ⓘ

(New) msdocs-spring-cosmosdb-tutorial

[Create new](#)

Region *

Central US

Web App Details

Name *

msdocs-spring-cosmosdb-123

-b8b0hudna5efbnhf.centralus-01.azurewebsites.net

Unique default hostname (preview) on. [More about this update](#)

Runtime stack *

Java 21

Java web server stack *

Java SE (Embedded Web Server)

Database

 Database access will be locked down and not exposed to the public internet. This is in compliance with recommended best practices for security.

Engine * ⓘ

Cosmos DB API for MongoDB

Account name *

msdocs-spring-cosmosdb-123-server

Database name *

msdocs-spring-cosmosdb-123-database

Azure Cache for Redis

Add Azure Cache for Redis?

Yes
 No

Hosting

Hosting plan *

Basic - For hobby or research purposes
 Standard - General purpose production apps

[Review + create](#)

[< Previous](#)

[Next : Tags >](#)



Step 3: The deployment takes a few minutes to complete. Once deployment completes, select the **Go to resource** button. You're taken directly to the App Service app, but the

following resources are created:

- **Resource group:** The container for all the created resources.
- **App Service plan:** Defines the compute resources for App Service. A Linux plan in the *Basic* tier is created.
- **App Service:** Represents your app and runs in the App Service plan.
- **Virtual network:** Integrated with the App Service app and isolates back-end network traffic.
- **Azure Cosmos DB:** Accessible only from behind its private endpoint. A database is created for you on the database account.
- **Private endpoints:** Access endpoints for the database server and the Redis cache in the virtual network.
- **Private DNS zones:** Enable DNS resolution of the database server and the Redis cache in the virtual network.

The screenshot shows the Azure portal deployment summary page. At the top, there are buttons for Delete, Cancel, Redeploy, Download, and Refresh. Below that, a green checkmark icon indicates 'Your deployment is complete'. Deployment details are listed:

- Deployment name : Microsoft.Web-WebAppDatabase-Portal-b0fb9389-be25
- Subscription :
- Resource group : [msdocs-spring-cosmosdb-tutorial](#)
- Start time : 8/27/2024, 2:25:05 PM
- Correlation ID :

Below the details are two expandable sections: 'Deployment details' and 'Next steps'. A blue button labeled 'Go to resource' is visible. At the bottom, there are links for 'Give feedback' and 'Tell us about your experience with deployment', along with a magnifying glass icon.

Having issues? Check the [Troubleshooting section](#).

3. Secure connection secrets

The creation wizard generated the connectivity string for you already as an [app setting](#). However, the security best practice is to keep secrets out of App Service completely. You'll move your secrets to a key vault and change your app setting to a [Key Vault reference](#) with the help of Service Connectors.

Step 1: In the App Service page:

1. In the left menu, select **Settings > Environment variables**.
2. Next to **AZURE_COSMOS_CONNECTIONSTRING**, select **Show value**. This connection string lets you connect to the Cosmos DB database secured behind a private endpoint. However, the secret is saved directly in the App Service app, which isn't the best. You'll change this.

The screenshot shows the Azure App Settings blade. On the left, there's a navigation menu with items like Deployment, Performance, Settings, and Environment variables (which is highlighted with a red box). The main area is titled "App settings" and contains a table of environment variables. One row is selected, showing "AZURE_COSMOS_CONNECTIONSTRING" in the Name column and a "Show value" button in the Value column, also highlighted with a red box. At the bottom, there are Apply and Discard buttons, and a "Send us your feedback" button.

Name	Value	Deploy
AZURE_COSMOS_CONNECTIONSTRING	Show value	✓

Step 2: Create a key vault for secure management of secrets.

1. In the top search bar, type "*key vault*", then select **Marketplace > Key Vault**.
2. In **Resource Group**, select **msdocs-spring-cosmosdb-tutorial**.
3. In **Key vault name**, type a name that consists of only letters and numbers.
4. In **Region**, set it to the sample location as the resource group.

Azure Key Vault is a cloud service used to manage keys, secrets, and certificates. Key Vault eliminates the need for developers to store security information in their code. It allows you to centralize the storage of your application secrets which greatly reduces the chances that secrets may be leaked. Key Vault also allows you to securely store secrets and keys backed by Hardware Security Modules or HSMs. The HSMs used are Federal Information Processing Standards (FIPS) 140-2 Level 2 validated. In addition, key vault provides logs of all access and usage attempts of your secrets so you have a complete audit trail for compliance.

Project details

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

Subscription *

Visual Studio Enterprise Subscription

Resource group *

msdocs-spring-cosmosdb-tutorial

[Create new](#)

Instance details

Key vault name * ⓘ

vault039485

Region *

Central US

Pricing tier * ⓘ

Standard

[Previous](#)

[Next](#)

[Review + create](#)

[Give feedback](#)

Step 3:

1. Select the **Networking** tab.
2. Unselect **Enable public access**.
3. Select **Create a private endpoint**.
4. In **Resource Group**, select **msdocs-spring-cosmosdb-tutorial**.
5. In **Key vault name**, type a name that consists of only letters and numbers.
6. In **Region**, set it to the sample location as the resource group.
7. In the dialog, in **Location**, select the same location as your App Service app.
8. In **Resource Group**, select **msdocs-spring-cosmosdb-tutorial**.
9. In **Name**, type **msdocs-spring-cosmosdb-XYZVaultEndpoint**.
10. In **Virtual network**, select **msdocs-spring-cosmosdb-XYZVnet**.
11. In **Subnet**, **msdocs-spring-cosmosdb-XYZSubnet**.
12. Select **OK**.
13. Select **Review + create**, then select **Create**. Wait for the key vault deployment to finish. You should see "Your deployment is complete."

Step 4:

1. In the top search bar, type *msdocs-spring-cosmosdb*, then the App Service resource called **msdocs-spring-cosmosdb-XYZ**.
2. In the App Service page, in the left menu, select **Settings > Service Connector**. There's already a connector, which the app creation wizard created for you.
3. Select checkbox next to the connector, then select **Edit**.
4. In the **Basics** tab, set **Client type** to **SpringBoot**. This option creates the Spring Boot specific environment variables for you.
5. Select the **Authentication** tab.
6. Select **Store Secret in Key Vault**.
7. Under **Key Vault Connection**, select **Create new**. A **Create connection** dialog is opened on top of the edit dialog.

The screenshot shows the Azure portal interface for managing a Service Connector. On the left, the navigation menu is open, showing various options like Identity, Backups, Custom domains, Certificates, Networking, and Service Connector. The Service Connector option is highlighted with a red box. On the right, a modal window titled 'defaultConnector' is displayed. The 'Authentication' tab is selected and highlighted with a red box. Inside the modal, there are sections for Basics, Authentication (which is selected), and Networking. The Authentication section asks for the authentication type, with 'Connection string' selected (highlighted with a red box). It also includes fields for permissions ('Read-Write') and key vault storage ('Store Secret In Key Vault' checked). A dropdown for 'Key Vault Connection' shows 'No item available' with a 'Create new' button highlighted with a red box. At the bottom of the modal are buttons for 'Next : Networking', 'Previous', and 'Cancel'.

Step 5: In the Create connection dialog for the Key Vault connection:

1. In **Key Vault**, select the key vault you created earlier.
2. Select **Review + Create**. You should see that **System assigned managed identity** is set to **Selected**.
3. When validation completes, select **Create**.

Create connection

X

Basics

Networking

Review + Create

Select the service instance and client type.

Service type * ⓘ

Key Vault

Connection name * ⓘ

keyvault_8ae7a

Subscription * ⓘ

Visual Studio Enterprise Subscription

Key vault * ⓘ

vault039485

[Create new](#)

Client type * ⓘ

SpringBoot

[Next : Networking](#)

[Cancel](#)

[Report an issue](#)

Step 6: You're back in the edit dialog for **defaultConnector**.

1. In the **Authentication** tab, wait for the key vault connector to be created. When it's finished, the **Key Vault Connection** dropdown automatically selects it.
2. Select **Next: Networking**.
3. Select **Configure firewall rules to enable access to target service**. If you see the message, "No Private Endpoint on the target service," ignore it. The app creation wizard already secured the Cosmos DB database with a private endpoint.
4. Select **Save**. Wait until the **Update succeeded** notification appears.

defaultConnector

Basics **Authentication** Networking

Select the authentication type you'd like to use between your compute service and target service. [Learn more](#)

System assigned managed identity(Not supported by selected client type) ⓘ

User assigned managed identity(Not supported by selected client type) ⓘ

Connection string ⓘ

Service principal(Not supported by selected client type) ⓘ

Permissions for the connection string

Read-Write

Store Secret In Key Vault ⓘ

Key Vault Connection * ⓘ

vault039485 (keyvault_8ae7a)

Create new

Store Configuration in App Configuration ⓘ

Next : Networking Previous Cancel

Step 7: To verify your changes:

1. From the left menu, select **Environment variables** again.
2. Make sure that the app setting **spring.data.mongodb.uri** exists. The default connector generated it for you, and your Spring Boot application already uses the variable.
3. Next to the app setting, select **Show value**. The value should be `@Microsoft.KeyVault(...)`, which means that it's a **key vault reference** because the secret is now managed in the key vault.

The screenshot shows the 'App settings' blade in the Azure portal. On the left, there's a sidebar with various links like Tags, Diagnose and solve problems, Microsoft Defender for Cloud, Events (preview), Better Together (preview), Deployment, Performance, and Settings. Under Settings, 'Environment variables' is selected and highlighted with a red box. The main area shows a table of environment variables:

Name	Value
azure.keyvault.scope	Show value
azure.keyvault.uri	Show value
spring.cloud.azure.keyvault.secret.credent...	Show value
spring.cloud.azure.keyvault.secret.endpoint	Show value
spring.data.mongodb.database	Show value
spring.data.mongodb.uri	@Microsoft.KeyVault(SecretUri=https://...)

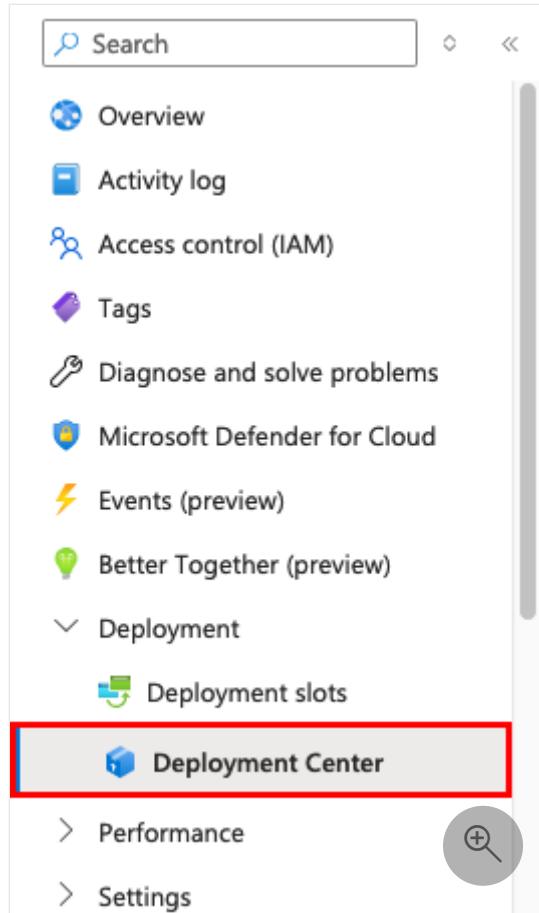
At the bottom, there are 'Apply' and 'Discard' buttons, and a 'Send us your feedback' button.

Having issues? Check the [Troubleshooting section](#).

4. Deploy sample code

In this step, you configure GitHub deployment using GitHub Actions. It's just one of many ways to deploy to App Service, but also a great way to have continuous integration in your deployment process. By default, every `git push` to your GitHub repository kicks off the build and deploy action.

Step 1: In the left menu, select **Deployment > Deployment Center**.



Step 2: In the Deployment Center page:

1. In **Source**, select **GitHub**. By default, **GitHub Actions** is selected as the build provider.
2. Sign in to your GitHub account and follow the prompt to authorize Azure.
3. In **Organization**, select your account.
4. In **Repository**, select **msdocs-spring-boot-mongodb-sample-app**.
5. In **Branch**, select **starter-no-infra**. This is the same branch that you worked in with your sample app, without any Azure-related files or configuration.
6. For **Authentication type**, select **User-assigned identity**.
7. In the top menu, select **Save**. App Service commits a workflow file into the chosen GitHub repository, in the `.github/workflows` directory. By default, the deployment center [creates a user-assigned identity](#) for the workflow to authenticate using Microsoft Entra (OIDC authentication). For alternative authentication options, see [Deploy to App Service using GitHub Actions](#).

Save **Discard** **Browse** **Manage publish profile** **Sync** **Leave Feedback**

Settings * **Logs** **FTPS credentials**

Info You're now in the production slot, which is not recommended for setting up CI/CD. [Learn more](#) **X**

Deploy and build code from your preferred source and build provider. [Learn more](#)

Source *

GitHub

Building with GitHub Actions. [Change provider](#).

GitHub

App Service will place a GitHub Actions workflow in your chosen repository to build and deploy your app whenever there is a commit on the chosen branch. If you can't find an organization or repository, you may need to enable additional permissions on GitHub. You must have write access to your chosen GitHub repository to deploy with GitHub Actions.

[Learn more](#)

Signed in as

<github-alias> [Change Account](#) ⓘ

Organization *

msdocs-spring-boot-mongodb-sam... ▾

Repository *

msdocs-spring-boot-mongodb-sam... ▾

Branch *

starter-no-infra ▾

Build

Runtime stack

Java

Version

java21

Java web server stack

Java SE

Authentication settings

Select how you want your GitHub Action workflow to authenticate to Azure. If you choose user-assigned identity, the identity selected will be federated with GitHub as an authorized client and given write permissions on the app. [Learn more](#)

Authentication type *

User-assigned identity 

Basic authentication

Step 3:

1. Select the **Logs** tab. See that a new deployment already ran, but the status is **Failed**.

2. Select Build/Deploy Logs. A browser tab opens to the Actions tab of your forked repository in GitHub. In Annotations, you see the error `The string 'java21' is not valid SeVer notation for a Java version`. If you want, select the failed build step in the page to get more information.

The screenshot shows the GitHub Actions Logs page. At the top, there are buttons for Save, Discard, Browse, Manage publish profile, Sync, and Leave Feedback. Below these are tabs for Settings and Logs, with Logs selected and highlighted by a red box. There are also Refresh and Delete buttons. The main area has columns for Time, Commit, Logs, Commit Author, and Status. A dropdown menu is open under Time, showing 'Wednesday, August 28, 2024 (1)'. Under this, a log entry is shown with the timestamp '08/28/2024, 4:21:...', the commit hash 'ecc4dc...', and the status 'Failed'. The 'Logs' column for this entry is highlighted by a red box. To the right of the logs, it says '<github-alias>'. A search icon is located in the bottom right corner of the logs section.

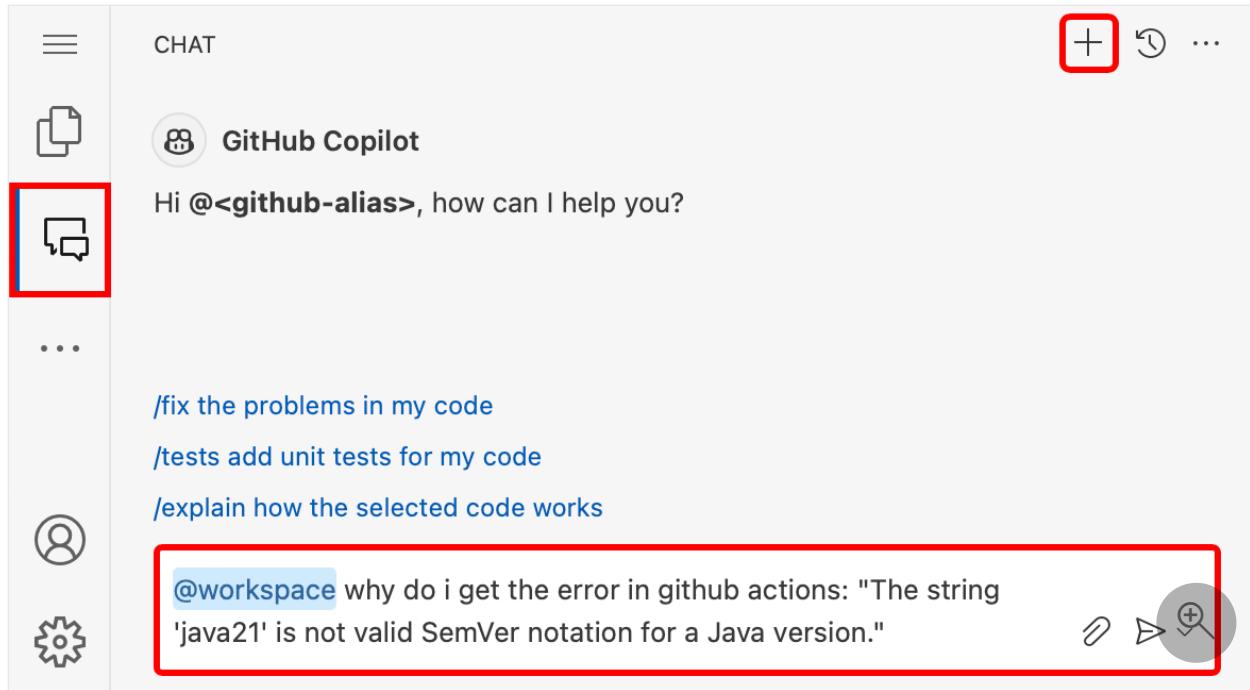
Step 4: The error shows that something went wrong during the GitHub workflow. To fix it, pull the latest changes into your codespace first. Back in the GitHub codespace of your sample fork, run `git pull origin starter-no-infra`. This pulls the newly committed workflow file into your codespace.

The screenshot shows the GitHub Codespaces interface. On the left is the Explorer sidebar with project files like '.devcontainer', 'docker-compose.yml', 'README.md', 'CHANGELOG.md', 'CONTRIBUTING.md', 'LICENSE.md', 'pom.xml', and another 'README.md'. The main area is a terminal window titled '[Preview] README.md' with the heading 'Deploy a Spring Boot web app with MongoDB in Azure'. It contains instructions for deploying a Spring Boot app with MongoDB using Azure App Service. Below this, a terminal session is shown with the command `git pull origin starter-no-infra` highlighted by a red box. The terminal output shows the process of cloning the repository and updating files.

Step 5 (Option 1: with GitHub Copilot):

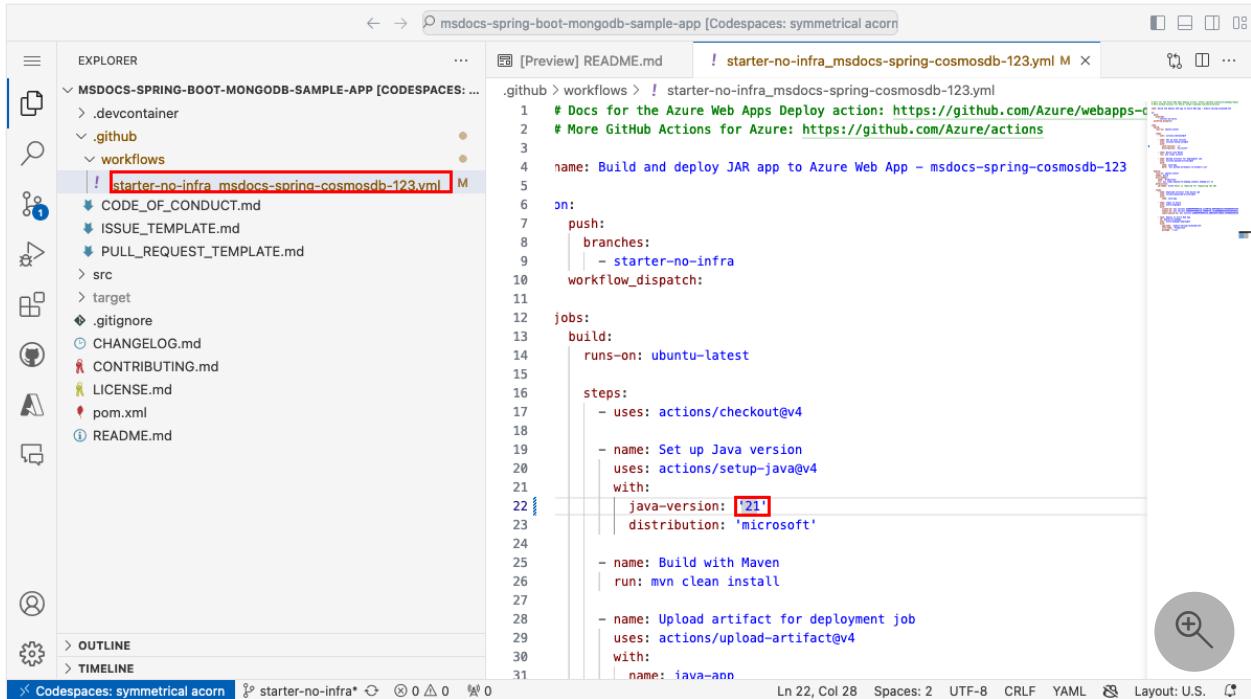
1. Start a new chat session by selecting the Chat view, then selecting +.

2. Ask, "@workspace Why do I get the error in GitHub actions: The string 'java21' is not valid SemVer notation for a Java version." Copilot might give you an explanation and even give you the link to the workflow file that you need to fix.
3. Open `.github/workflows/starter-no-infra_msdocs-spring-cosmosdb-123.yaml` in the explorer and make the suggested fix. GitHub Copilot doesn't give you the same response every time, you might need to ask more questions to fine-tune its response. For tips, see [What can I do with GitHub Copilot in my codespace?](#).



Step 5 (Option 2: without GitHub Copilot):

1. Open `.github/workflows/starter-no-infra_msdocs-spring-cosmosdb-123.yaml` in the explorer and find the `setup-java@v4` action.
2. Change the value of `java-version` to `'21'`.

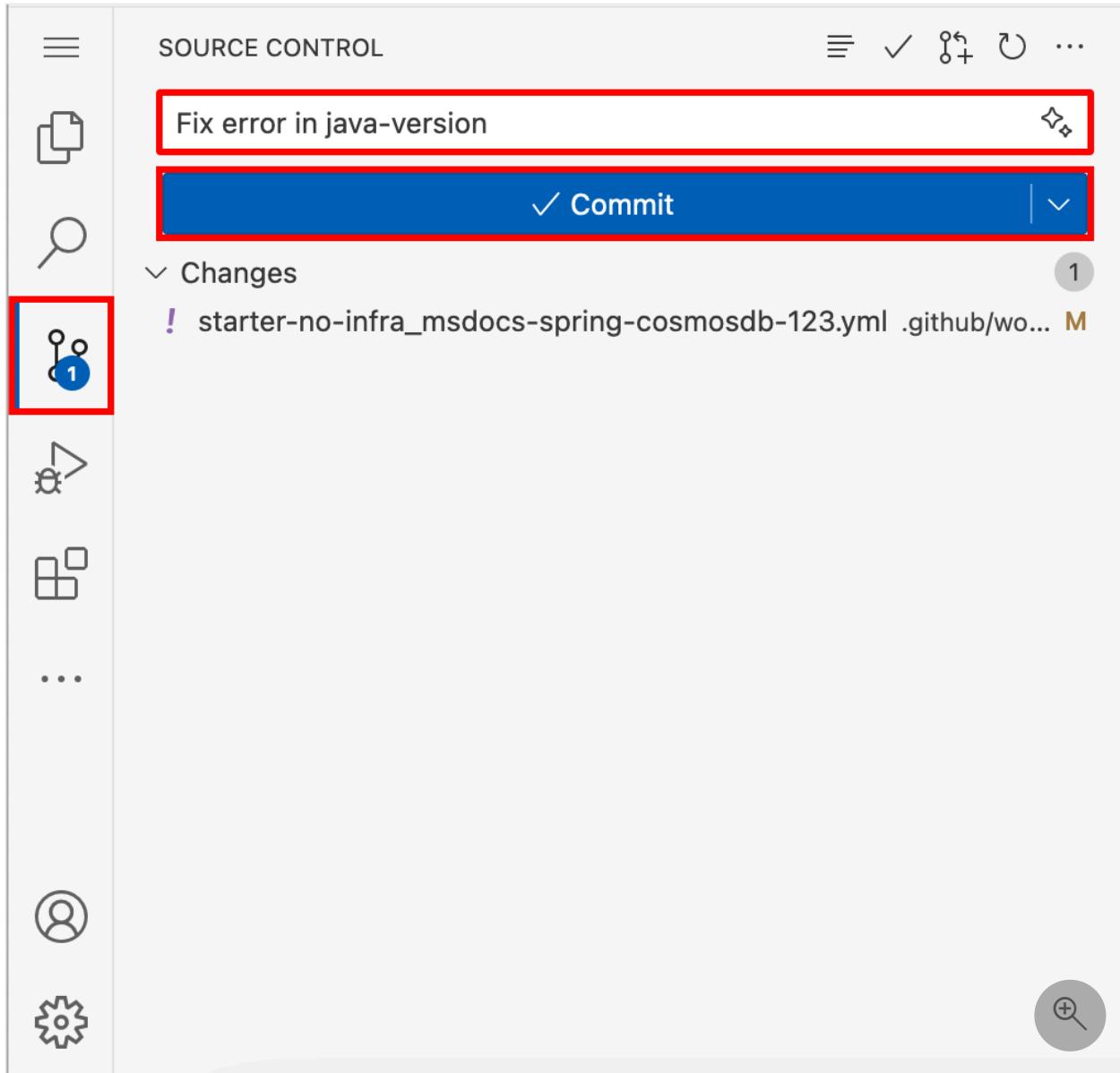


The screenshot shows the GitHub Codespaces interface with a workflow file open. The file is named `starter-no-infra_msdocs-spring-cosmosdb-123.yml`. A specific line of code, which defines a Java version step, is highlighted with a red box. The code snippet is as follows:

```
 1  # Docs for the Azure Web Apps Deploy action: https://github.com/Azure/webapps-C
 2  # More GitHub Actions for Azure: https://github.com/Azure/actions
 3
 4  name: Build and deploy JAR app to Azure Web App - msdocs-spring-cosmosdb-123
 5
 6  on:
 7    push:
 8      branches:
 9        - starter-no-infra
10
11  jobs:
12    build:
13      runs-on: ubuntu-latest
14
15    steps:
16      - uses: actions/checkout@v4
17
18      - name: Set up Java version
19        uses: actions/setup-java@v4
20        with:
21          java-version: '21'
22          distribution: 'microsoft'
23
24      - name: Build with Maven
25        run: mvn clean install
26
27      - name: Upload artifact for deployment job
28        uses: actions/upload-artifact@v4
29        with:
30          name: iava-ann
31
```

Step 6:

1. Select the **Source Control** extension.
2. In the textbox, type a commit message like `Fix error in java-version`. Or, select  and let GitHub Copilot generate a commit message for you.
3. Select **Commit**, then confirm with **Yes**.
4. Select **Sync changes 1**, then confirm with **OK**.



Step 7: Back in the Deployment Center page in the Azure portal:

1. Under the **Logs** tab, select **Refresh**. A new deployment run is already started from your committed changes.
2. In the log item for the deployment run, select the **Build/Deploy Logs** entry with the latest timestamp.

Save Discard Browse Manage publish profile Sync Leave Feedback

Settings Logs FTPS credentials

Refresh Delete

Time	Com...	Logs	Commit Author	Status
08/29 2024, 12:0...	f0d78e7	App Logs	N/A	Success (Active)
08/29 2024, 12:0...	72da328	Build/Deploy Lo...		Success
08/29 2024, 11:3...	5d3b96a	Build/Deploy Lo...		Failed

🔍

Step 8: You're taken to your GitHub repository and see that the GitHub action is running. The workflow file defines two separate stages, build and deploy. Wait for the GitHub run to show a status of **Complete**.

/ msdocs-spring-boot-mongodb-sample-app

Code Pull requests Actions Projects Wiki Security Insights Settings

← Build and deploy JAR app to Azure Web App - msdocs-spring-cosmosdb-123

Fix error in java-version #2 Re-run all jobs ⋮

Summary

Triggered via push 13 minutes ago Status Success Total duration 1m 26s

pushed → 72da328 starter-no-infra

Jobs: build, deploy

Artifacts: 1

Run details Usage Workflow file

starter-no-infra_msdocs-spring-cosmosdb-123.yml
on: push

build 18s deploy 48s

http://msdocs-spring-cosmosdb-123-b... 🔍

Having issues? Check the [Troubleshooting section](#).

5. Browse to the app

Step 1: In the App Service page:

1. From the left menu, select **Overview**.
2. Select the URL of your app.

Search

Overview

Activity log

Access control (IAM)

Tags

Diagnose and solve problems

Microsoft Defender for Cloud

Events (preview)

Better Together (preview)

Deployment

Deployment slots

Deployment Center

Resource group (move)
msdocs-spring-cosmosdb-tutorial

Status
Running

Location (move)
Central US

Subscription (move)

Subscription ID

Tags (edit)
Add tags

Default domain
msdocs-spring-cosmosdb-123-b8b0hu...

App Service Plan
ASP-msdocsspringcosmosdbtutorial-9b68

Operating System
Linux

Health Check
Not Configured

Github Project

JSON View

+

Step 2: Add a few tasks to the list. Congratulations, you're running a web app in Azure App Service, with secure connectivity to Azure Cosmos DB.

		Add
<input type="checkbox"/>	Create Spring Boot app an MongoDB in Azure	Delete Edit
<input type="checkbox"/>	Deploy data-driven app	Delete Edit
<input type="checkbox"/>	That's it!	Delete Edit

+

Having issues? Check the [Troubleshooting section](#).

6. Stream diagnostic logs

Azure App Service captures all messages output to the console to help you diagnose issues with your application. The sample application includes standard Log4j logging statements to demonstrate this capability, as shown in the following snippet:

```
Java

private static Logger logger =
LoggerFactory.getLogger(TodoListController.class);

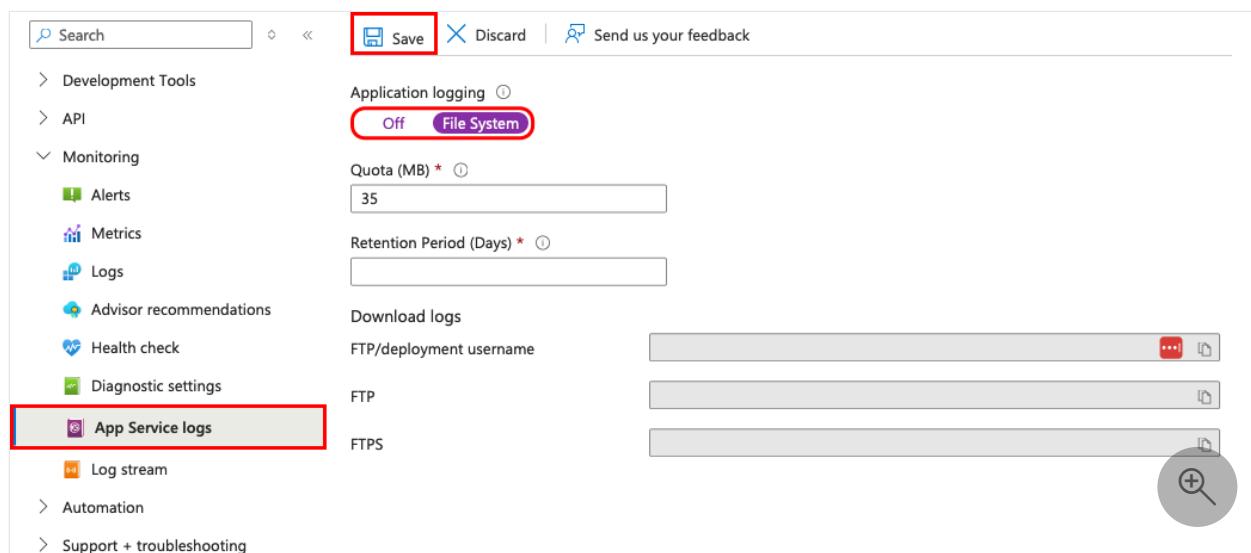
@Autowired
private TodoItemRepository todoItemRepository;

public TodoListController() {
}

/**
 * HTTP GET
 */
@GetMapping(path = "/api/todolist/{index}", produces =
{MediaType.APPLICATION_JSON_VALUE})
public TodoItem getTodoItem(@PathVariable("index") String index) {
    logger.info("GET request access '/api/todolist/{}' path.", index);
    return todoItemRepository.findById(index).get();
}
```

Step 1: In the App Service page:

1. From the left menu, select **App Service logs**.
2. Under **Application logging**, select **File System**.
3. In the top menu, select **Save**.



Step 2: From the left menu, select **Log stream**. You see the logs for your app, including platform logs and logs from inside the container.

The screenshot shows the Azure Monitor Log Stream interface. On the left, there's a sidebar with various monitoring and troubleshooting options. The 'Log stream' option is highlighted with a red box. The main area displays a log history for a Java application. A specific log entry is highlighted with a red box: '2024-08-29T10:49:30.007332954Z: [INFO] 2024-08-29T10:49:30.006Z INFO 84 --- [p-nio-80-exec-8] c.m.a.e.s.c.TodoListController : GET request access '/api/todolist' path.'. This log entry corresponds to the first log entry in the provided code snippet.

```
static/index.html]
2024-08-29T10:48:57.441748935Z: [INFO] 2024-08-29T10:48:57.441Z INFO 84 --- [
main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port 80 (http) with
context path '/'
2024-08-29T10:48:57.657232594Z: [INFO] 2024-08-29T10:48:57.656Z INFO 84 --- [
main] c.m.a.e.s.TodoApplication : Started TodoApplication in 46.794
seconds (process running for 58.392)
2024-08-29T10:48:58.699576246Z: [INFO] 2024-08-29T10:48:58.699Z INFO 84 --- [p-nio-80-
exec-2] o.s.web.servlet.DispatcherServlet : Initializing Servlet
'dispatcherServlet'
2024-08-29T10:48:58.702547473Z: [INFO] 2024-08-29T10:48:58.701Z INFO 84 --- [p-nio-80-
exec-2] o.s.web.servlet.DispatcherServlet : Completed initialization in 2 ms
2024-08-29T10:49:00.911582391Z: [INFO] Received signal SIGTERM
2024-08-29T10:49:00.914199616Z: [INFO] Sending SIGTERM to main process. Child Process
ID: 79
2024-08-29T10:49:30.007332954Z: [INFO] 2024-08-29T10:49:30.006Z INFO 84 --- [p-nio-80-
exec-8] c.m.a.e.s.c.TodoListController : GET request access '/api/todolist'
path.
2024-08-29T10:49:43.512332830Z: [INFO] 2024-08-29T10:49:43.482Z INFO 84 --- [p-nio-80-
exec-1] c.m.a.e.s.c.TodoListController : POST request access '/api/todolist'
path with item: ssss
2024-08-29T10:49:44.171320839Z: [INFO] 2024-08-29T10:49:44.170Z INFO 84 --- [p-nio-80-
exec-9] c.m.a.e.s.c.TodoListController : GET request access '/api/todolist'
path.
```

Learn more about logging in Java apps in the series on [Enable Azure Monitor OpenTelemetry for .NET, Node.js, Python and Java applications](#).

Having issues? Check the [Troubleshooting section](#).

7. Clean up resources

When you're finished, you can delete all of the resources from your Azure subscription by deleting the resource group.

Step 1: In the search bar at the top of the Azure portal:

1. Enter the resource group name.
2. Select the resource group.

The screenshot shows the Microsoft Azure portal interface. At the top, there's a search bar with the text "msdocs-spring-cosmosdb-tutorial". Below the search bar, the "Resource Groups" section is visible, showing one item: "msdocs-spring-cosmosdb-tutorial", which is highlighted with a red box. To the right, there's a "Microsoft Entra ID" section with a blue icon and the text "Microsoft Entra ID". On the left sidebar, under "Azure services", there are options like "Create a resource", "Subscriptions", and "Resources". Under "Resources", there are tabs for "Recent" (which is selected) and "Favorite". Below these tabs, there are sections for "Name" and "Type". A "Give feedback" button is also present. At the bottom right of the main content area, there's a circular icon with a magnifying glass and a plus sign.

Step 2: In the resource group page, select **Delete resource group**.

The screenshot shows the "Resource Groups" page in the Azure portal. At the top, there's a toolbar with buttons for "Create", "Manage view", "Delete resource group" (which is highlighted with a red box), "Refresh", "Export to CSV", "Open query", and more. Below the toolbar, there's a section titled "Essentials" with a "Resources" tab selected. The main area displays a list of resources with columns for "Name", "Type", and "Location". Each resource entry includes a checkbox, a preview icon, the resource name, its type, location, and a three-dot menu icon. At the bottom right of the list area, there's a circular icon with a magnifying glass and a plus sign.

Step 3:

1. Confirm your deletion by typing the resource group name.
2. Select **Delete**.
3. Confirm with **Delete** again.

Delete a resource group

X

The following resource group and all its dependent resources will be permanently deleted.

Resource group to be deleted



Dependent resources to be deleted (16)

All dependent resources, including hidden types, are shown

Name	Resource type
ASP-msdocsspringcosmosdbtutorial-9b68	App Service plan
lcg7qhqn65jo (privatelink.vaultcore.azure.net	Microsoft.Network/privateDnsZones/v...
msdocs-spring-co-id-8b69	Managed Identity
msdocs-spring-cosmosdb-123	App Service
msdocs-spring-cosmosdb-123-server	Azure Cosmos DB for MongoDB acco...
msdocs-spring-cosmosdb-123-vaultEndpoint	Private endpoint

Enter resource group name to confirm deletion *

msdocs-spring-cosmosdb-tutorial

Delete

Cancel



Troubleshooting

- The portal deployment view for Azure Cosmos DB shows a Conflict status
- The deployed sample app doesn't show the tasks list app

The portal deployment view for Azure Cosmos DB shows a Conflict status

Depending on your subscription and the region you select, you might see the deployment status for Azure Cosmos DB to be `Conflict`, with the following message in Operation details:

Sorry, we are currently experiencing high demand in <region> region, and cannot fulfill your request at this time.

The error is most likely caused by a limit on your subscription for the region you select. Try choosing a different region for your deployment.

The deployed sample app doesn't show the tasks list app

If you see a `Hey, Java developers!` page instead of the tasks list app, App Service is most likely still loading the updated container from your most recent code deployment. Wait a few minutes and refresh the page.

Frequently asked questions

- [How much does this setup cost?](#)
- [How do I run database migration with the Cosmos DB database behind the virtual network?](#)
- [How does local app development work with GitHub Actions?](#)
- [I don't have permissions to create a user-assigned identity](#)

How much does this setup cost?

Pricing for the created resources is as follows:

- The App Service plan is created in **Basic** tier and can be scaled up or down. See [App Service pricing](#).
- The Azure Cosmos DB account is created in **Serverless** tier and there's a small cost associated with this tier. See [Azure Cosmos DB pricing](#).
- The Azure Cache for Redis is created in **Basic** tier with the minimum cache size. There's a small cost associated with this tier. You can scale it up to higher performance tiers for higher availability, clustering, and other features. See [Azure Cache for Redis pricing](#).
- The virtual network doesn't incur a charge unless you configure extra functionality, such as peering. See [Azure Virtual Network pricing](#).
- The private DNS zone incurs a small charge. See [Azure DNS pricing](#).

How do I run database migration with the Cosmos DB database behind the virtual network?

The Java SE container in App Service already has network connectivity to Cosmos DB, but doesn't contain any migration tools or other MongoDB tools. You have a few options:

- Run database migrations automatically at app start, such as with Hibernate and or Flyway.
- In the app's [SSH session](#), install a migration tool like Flyway, then run the migration script. Remember that the installed tool won't persist after an app restart unless it's in the `/home` directory.
- [Integrate the Azure cloud shell](#) with the virtual network and run database migrations from there.

How does local app development work with GitHub Actions?

Using the autogenerated workflow file from App Service as an example, each `git push` kicks off a new build and deployment run. From a local clone of the GitHub repository, you make the desired updates and push to GitHub. For example:

terminal

```
git add .
git commit -m "<some-message>"
git push origin main
```

I don't have permissions to create a user-assigned identity

See [Set up GitHub Actions deployment from the Deployment Center](#).

What can I do with GitHub Copilot in my codespace?

You might notice that the GitHub Copilot chat view was already there for you when you created the codespace. For your convenience, we include the GitHub Copilot chat extension in the container definition (see `.devcontainer/devcontainer.json`). However, you need a [GitHub Copilot account](#) (30-day free trial available).

A few tips for you when you talk to GitHub Copilot:

- In a single chat session, the questions and answers build on each other and you can adjust your questions to fine-tune the answer you get.
- By default, GitHub Copilot doesn't have access to any file in your repository. To ask questions about a file, open the file in the editor first.

- To let GitHub Copilot have access to all of the files in the repository when preparing its answers, begin your question with `@workspace`. For more information, see [Use the `@workspace` agent ↗](#).
- In the chat session, GitHub Copilot can suggest changes and (with `@workspace`) even where to make the changes, but it's not allowed to make the changes for you. It's up to you to add the suggested changes and test it.

Next steps

- [Azure for Java Developers](#)
- [Spring Boot ↗](#)
- [Spring Data for Azure Cosmos DB](#)
- [Azure Cosmos DB and -App Service Linux](#)

Learn more about running Java apps on App Service in the developer guide.

[Configure a Java app in Azure App Service](#)

Learn how to secure your app with a custom domain and certificate.

[Secure with custom domain and certificate](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

Tutorial: Build a Tomcat web app with Azure App Service on Linux and MySQL

Article • 09/15/2024

This tutorial shows how to build, configure, and deploy a secure Tomcat application in Azure App Service that connects to a MySQL database (using [Azure Database for MySQL](#)). Azure App Service is a highly scalable, self-patching, web-hosting service that can easily deploy apps on Windows or Linux. When you're finished, you have a Tomcat app running on [Azure App Service on Linux](#).

The screenshot displays a user interface for managing tasks. At the top, there is a "New Task" section with a "Task" input field containing "Enter task description" and a blue "Add Task" button. Below this is a "Current Tasks" section containing three items:

- Create Tomcat app and MySQL in Azure Delete
- Deploy data-driven app Delete
- That's it! Delete

In this tutorial, you learn how to:

- ✓ Create a secure-by-default architecture for Azure App Service and Azure Database for MySQL.
- ✓ Secure connection secrets using a managed identity and Key Vault references.
- ✓ Deploy a Tomcat sample app to App Service from a GitHub repository.
- ✓ Access App Service app settings in the application code.
- ✓ Make updates and redeploy the application code.
- ✓ Stream diagnostic logs from App Service.

- ✓ Manage the app in the Azure portal.
- ✓ Provision the same architecture and deploy by using Azure Developer CLI.
- ✓ Optimize your development workflow with GitHub Codespaces and GitHub Copilot.

Prerequisites

- An Azure account with an active subscription. If you don't have an Azure account, you [can create one for free](#).
- A GitHub account. you can also [get one for free](#).
- Knowledge of Java with Tomcat development.
- **(Optional)** To try GitHub Copilot, a [GitHub Copilot account](#). A 30-day free trial is available.

Skip to the end

You can quickly deploy the sample app in this tutorial and see it running in Azure. Just run the following commands in the [Azure Cloud Shell](#), and follow the prompt:

Bash

```
mkdir msdocs-tomcat-mysql-sample-app
cd msdocs-tomcat-mysql-sample-app
azd init --template msdocs-tomcat-mysql-sample-app
azd up
```

1. Run the sample

First, you set up a sample data-driven app as a starting point. For your convenience, the [sample repository](#), includes a [dev container](#) configuration. The dev container has everything you need to develop an application, including the database, cache, and all environment variables needed by the sample application. The dev container can run in a [GitHub codespace](#), which means you can run the sample on any computer with a web browser.

Step 1: In a new browser window:

1. Sign in to your GitHub account.
2. Navigate to <https://github.com/Azure-Samples/msdocs-tomcat-mysql-sample-app/fork>.
3. Unselect **Copy the main branch only**. You want all the branches.
4. Select **Create fork**.

Create a new fork

A *fork* is a copy of a repository. Forking a repository allows you to freely experiment with changes without affecting the original project. [View existing forks.](#)

Required fields are marked with an asterisk (*).

Owner * Repository name *



/ msdocs-tomcat-mysql-sampl

msdocs-tomcat-mysql-sample-app is available.

By default, forks are named the same as their upstream repository. You can customize the name to distinguish it further.

Description (optional)

This is a Tomcat CRUD sample application for Azure App Service and the Azure Database for MySQL flexible...

Copy the `main` branch only

Contribute back to Azure-Samples/msdocs-tomcat-mysql-sample-app by adding your own branch. [Learn more.](#)

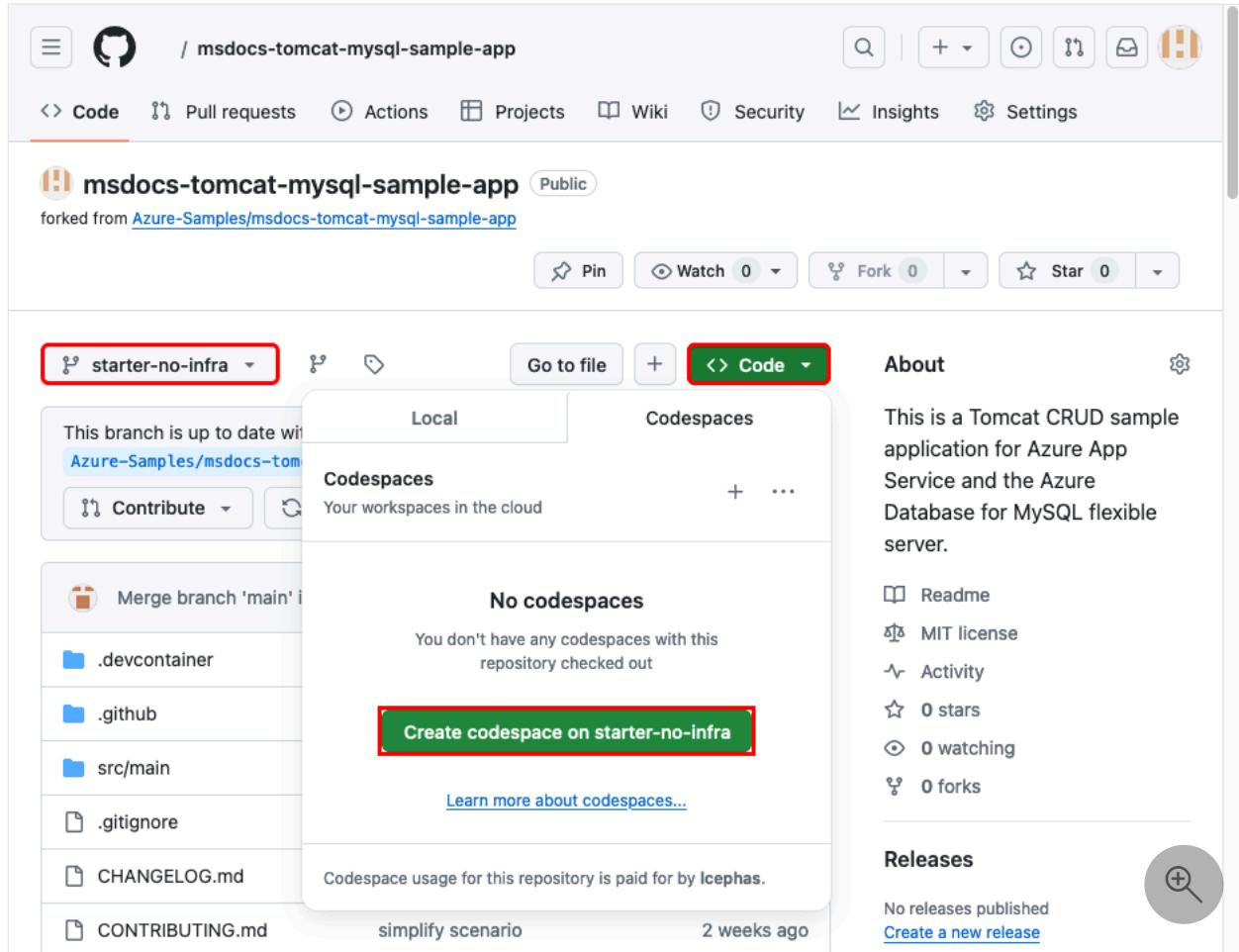
You are creating a fork in your personal account.

Create fork



Step 2: In the GitHub fork:

1. Select **main > starter-no-infra** for the starter branch. This branch contains just the sample project and no Azure-related files or configuration.
2. Select **Code > Create codespace on starter-no-infra**. The codespace takes a few minutes to set up.



Step 3: In the codespace terminal:

1. Run `mvn jetty:run`.
2. When you see the notification `Your application running on port 80 is available.`, select **Open in Browser**. You should see the sample application in a new browser tab. To stop the Jetty server, type `Ctrl+C`.

This is a Tomcat 10.1 web app using Servlet 6.0 and the Azure Database for MySQL flexible server. The Tomcat app is hosted in a fully managed Azure App Service. This app is designed to be run locally with a Jetty plugin and then deployed to a Tomcat container in Azure App Service. You can either deploy this project by following the tutorial [Deploy a Tomcat web app with MySQL in Azure](#) or by using the [Azure Developer CLI \(azd\)](#) according to the instructions below.

```
@ _      + /workspaces/msdocs-tomcat-mysql-sample-app (starter-no-infra) $ mvn jetty:run
[INFO] Scanning for projects...
[INFO] 
[INFO] --- com.microsoft.azure.appservice.examples:tomcatmysql 1.0-SNAPSHOT
[INFO] Building A Jakarta Servlet @WebServlet example 10---SNAPSHOT
[INFO]   from pom.xml
[INFO]   [ war ]
[INFO] 
[INFO] >>> jetty:11.0.20:run (default-cli) > test-compile @ tomcatmysql >>>
[WARNING] org.apache.taglibs.standard.tlv.JstlXmlTLV scanned from multiple locations: jar:file:///home/vscode/.m2/repository/org/glassfish/web/jakarta.servlet.jsp.jstl/2.0.0/jakarta.servlet.jsp.jstl-2.0.0.jar!/org/apache/taglibs/standard/tlv/JstlXmlTLV.class, jar:file:///home/vscode/.m2/repository/org/glassfish/web/jakarta.servlet.jsp.jstl/2.0.0/jakarta.servlet.jsp.jstl-2.0.0.jar!/org/apache/taglibs/standard/tlv/JstlXmlTLV.class
[INFO] Session workerName=node0
Initializing WebListener.
HHH000204: Processing PersistenceUnitInfo [name: defaultpu]
HHH000412: Hibernate ORM core version 5.6.15.Final
HHH000400: Using dialect: org.hibernate.dialect.MySQL8Dialect
HHH000490: Using JtaPlatform implementation: [org.hibernate.dialect.MySQL8Dialect]
[INFO] Started o.e.j.m.p.MavenWebAppContext@43cb5f38{/,file:/VAILABLE}{file:///workspaces/msdocs-tomcat-mysql-sample-app/src/main/webapp}
[INFO] Started ServerConnector@248ba4fc[HTTP/1.1]{http://127.0.0.1:8080}
[INFO] Started Server@32f1afe[STARTING][11.0.20,stop=0] @10571
[INFO] Automatic redeployment disabled, see 'mvn jetty:help' 1
```

Your application running on port 80 is available. See all forwarded ports

[Open in Browser](#) [Make Public](#)

Tip

You can ask [GitHub Copilot](#) about this repository. For example:

- @workspace *What does this project do?*
- @workspace *What does the .devcontainer folder do?*

Having issues? Check the [Troubleshooting section](#).

2. Create App Service and MySQL

First, you create the Azure resources. The steps used in this tutorial create a set of secure-by-default resources that include App Service and Azure Database for MySQL. For the creation process, you specify:

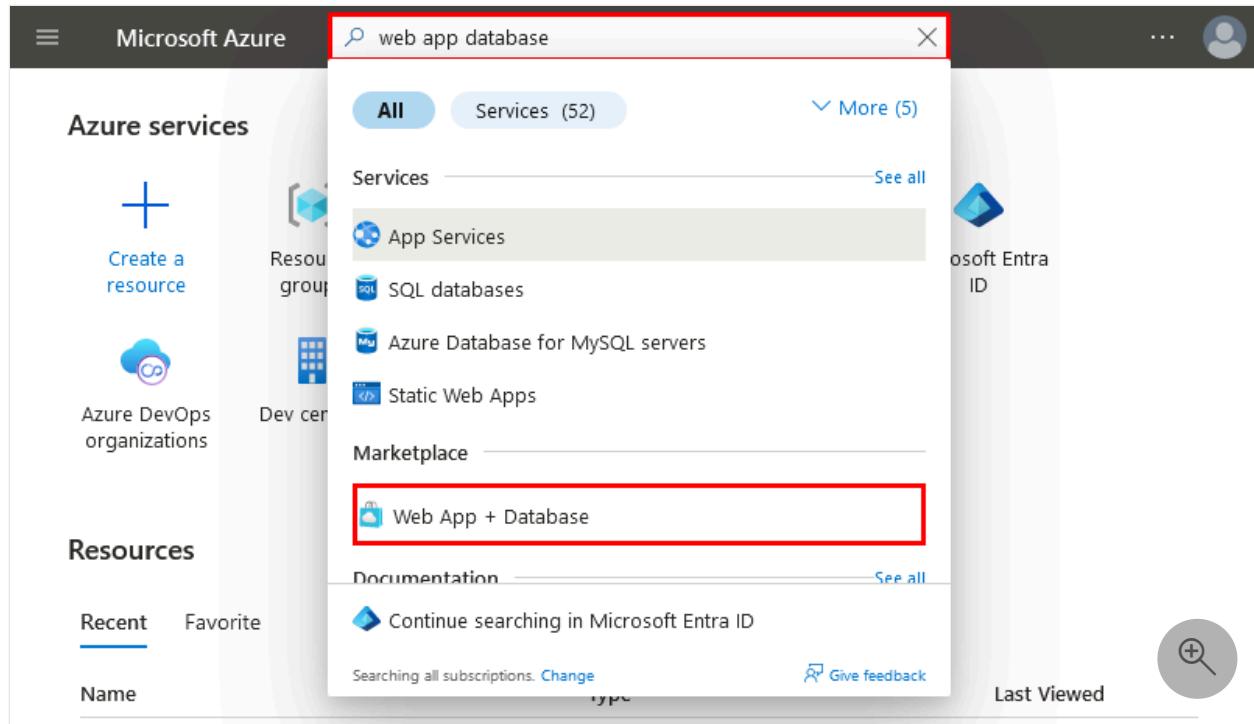
- The **Name** for the web app. It's used as part of the DNS name for your app in the form of `https://<app-name>-<hash>.<region>.azurewebsites.net`.
- The **Region** to run the app physically in the world. It's also used as part of the DNS name for your app.
- The **Runtime stack** for the app. It's where you select the version of Java to use for your app.
- The **Hosting plan** for the app. It's the pricing tier that includes the set of features and scaling capacity for your app.

- The **Resource Group** for the app. A resource group lets you group (in a logical container) all the Azure resources needed for the application.

Sign in to the [Azure portal](#) and follow these steps to create your Azure App Service resources.

Step 1: In the Azure portal:

1. Enter "web app database" in the search bar at the top of the Azure portal.
2. Select the item labeled **Web App + Database** under the **Marketplace** heading. You can also navigate to the [creation wizard](#) directly.



Step 2: In the **Create Web App + Database** page, fill out the form as follows.

1. **Resource Group:** Select **Create new** and use a name of **msdocs-tomcat-mysql-tutorial**.
2. **Region:** Any Azure region near you.
3. **Name:** **msdocs-tomcat-mysql-XYZ**, where **XYZ** is any three random characters.
4. **Runtime stack:** **Java 17**.
5. **Java web server stack:** **Apache Tomcat 10.1**.
6. **Engine:** **MySQL - Flexible Server** is selected for you by default as the database engine. If not, select it. Azure Database for MySQL - Flexible Server is a fully managed MySQL database as a service on Azure, compatible with the latest community editions.
7. **Hosting plan:** **Basic**. When you're ready, you can [scale up](#) to a production pricing tier.
8. Select **Review + create**.
9. After validation completes, select **Create**.

Basics Tags Review + create

This template will create a secure by default configuration where the only publicly accessible endpoint will be your app following the recommended security best practices. [Learn more](#)

Project Details

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

Subscription * ⓘ

Resource Group * ⓘ

(New) msdocs-tomcat-mysql-tutorial

[Create new](#)

Region *

East US

Web App Details

Name *

msdocs-tomcat-mysql-123



.azurewebsites.net

Runtime stack *

Java 17

Java web server stack *

Apache Tomcat 10.1



Database



Database access will be locked down and not exposed to the public internet. This is in compliance with recommended best practices for security.

Engine * ⓘ

MySQL - Flexible Server (recommended)



Server name *

msdocs-tomcat-mysql-123-server



Database name *

msdocs-tomcat-mysql-123-database

Azure Cache for Redis

Add Azure Cache for Redis?

Yes

No

Hosting

Hosting plan *

Basic - For hobby or research purposes

Standard - General purpose production apps

[Review + create](#)

[< Previous](#)

[Next : Tags >](#)



Step 3: The deployment takes a few minutes to complete. Once deployment completes, select the **Go to resource** button. You're taken directly to the App Service app, but the following resources are created:

- **Resource group:** The container for all the created resources.
- **App Service plan:** Defines the compute resources for App Service. A Linux plan in the *Basic* tier is created.

- **App Service:** Represents your app and runs in the App Service plan.
- **Virtual network:** Integrated with the App Service app and isolates back-end network traffic.
- **Azure Database for MySQL Flexible Server:** Accessible only from the virtual network. A database and a user are created for you on the server.
- **Private DNS zones:** Enable DNS resolution of the database server in the virtual network.

 Your deployment is complete

 Deployment name : Microsoft.Web-WebAppDatabase-Portal-b990d412-babf
Subscription : [Visual Studio Enterprise Subscription](#)
Resource group : [msdocs-tomcat-mysql-tutorial](#)
Start time : 3/8/2024, 7:07:05 AM
Correlation ID : 6981cfe5-2ad7-4c1f-bab5-bf45280606ea

> Deployment details

▽ Next steps

[Go to resource](#)

[Give feedback](#)

 [Tell us about your experience with deployment](#)



Having issues? Check the [Troubleshooting section](#).

3. Secure connection secrets

The creation wizard generated the database connectivity string for you already as an [app setting](#). However, the security best practice is to keep secrets out of App Service completely. You move your secrets to a key vault and change your app setting to a [Key Vault reference](#) with the help of Service Connectors.

Step 1: Retrieve the existing connection string

1. In the left menu of the App Service page, select **Settings > Environment variables**.
2. Select **AZURE_MYSQL_CONNECTIONSTRING**. It contains a JDBC connection string.
If you add an app setting that contains a valid Oracle, SQL Server, PostgreSQL, or MySQL connection string, App Service injects it as a Java Naming and Directory Interface (JNDI) data source in the Tomcat server's *context.xml* file.
3. In **Add/Edit application setting**, in the **Value** field, find the *password=* part at the end of the string.

4. Copy the password string after *Password=* for use later. This app setting lets you connect to the MySQL database secured behind a private endpoint. However, the secret is saved directly in the App Service app, which isn't the best. You'll change this.

The screenshot shows the Azure portal interface for managing an App Service application. The left sidebar lists various settings like Activity log, Access control (IAM), Tags, Diagnose and solve problems, Microsoft Defender for Cloud, Events (preview), Better Together (preview), Deployment, and Settings. Under Settings, the 'Environment variables' section is selected and highlighted with a red box. On the right, the 'Add/Edit application setting' dialog is open. It shows a table with one row: Name 'AZURE_MYSQL_CONNECTIONSTRING' and Value 'n:3306/msdocs-tomcat-mysql-123-database?serverTimezone=UTC&sslmode=require&user=bfvjagfei&password=xxxxxxxxxxxxxx'. The password part of the value is also highlighted with a red box. There are 'Apply' and 'Discard' buttons at the bottom of the dialog.

Step 2: Create a key vault for secure management of secrets

1. In the top search bar, type "key vault", then select Marketplace > Key Vault.
2. In Resource Group, select msdocs-tomcat-mysql-tutorial.
3. In Key vault name, type a name that consists of only letters and numbers.
4. In Region, set it to the same location as the resource group.

[Basics](#) [Access configuration](#) [Networking](#) [Tags](#) [Review + create](#)

Azure Key Vault is a cloud service used to manage keys, secrets, and certificates. Key Vault eliminates the need for developers to store security information in their code. It allows you to centralize the storage of your application secrets which greatly reduces the chances that secrets may be leaked. Key Vault also allows you to securely store secrets and keys backed by Hardware Security Modules or HSMs. The HSMs used are Federal Information Processing Standards (FIPS) 140-2 Level 2 validated. In addition, key vault provides logs of all access and usage attempts of your secrets so you have a complete audit trail for compliance.

Project details

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

Subscription * [Dev Compute and Platform](#)

Resource group * [msdocs-tomcat-mysql-tutorial](#) [Create new](#)

Instance details

Key vault name * [vault029345](#)

Region * [Central US](#)

Pricing tier * [Standard](#)

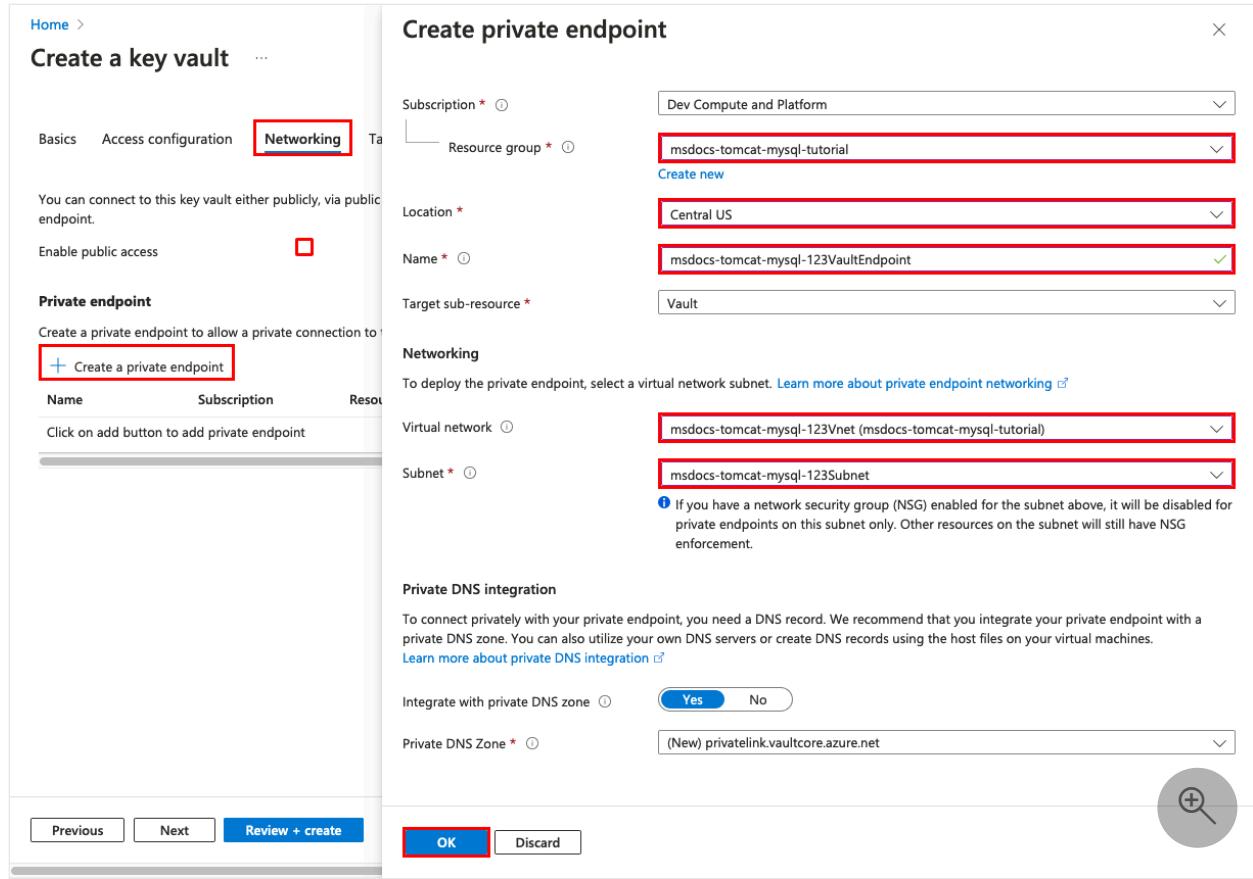
Recovery options

Soft delete protection will automatically be enabled on this key vault. This feature allows you to recover or permanently delete a key vault and secrets for the duration of the retention period. This protection applies to the key vault and the secrets stored

[Previous](#) [Next](#) [Review + create](#) [Give feedback](#)

Step 3: Secure the key vault with a Private Endpoint

1. Select the **Networking** tab.
2. Unselect **Enable public access**.
3. Select **Create a private endpoint**.
4. In **Resource Group**, select **msdocs-tomcat-mysql-tutorial**.
5. In **Name**, type a name for the private endpoint that consists of only letters and numbers.
6. In **Region**, set it to the same location as the resource group.
7. In the dialog, in **Location**, select the same location as your App Service app.
8. In **Resource Group**, select **msdocs-tomcat-mysql-tutorial**.
9. In **Name**, type **msdocs-tomcat-mysql-XYZVaultEndpoint**.
10. In **Virtual network**, select **msdocs-tomcat-mysql-XYZVnet**.
11. In **Subnet**, **msdocs-tomcat-mysql-XYZSubnet**.
12. Select **OK**.
13. Select **Review + create**, then select **Create**. Wait for the key vault deployment to finish. You should see "Your deployment is complete."



Step 4: Configure the Service Connector

1. In the top search bar, type *msdocs-tomcat-mysql*, then the App Service resource called **msdocs-tomcat-mysql-XYZ**.
2. In the App Service page, in the left menu, select **Settings > Service Connector**. There's already a connector, which the app creation wizard created for you.
3. Select checkbox next to the connector, then select **Edit**.
4. In the **Basics** tab, set **Client type** to **Java**.
5. Select the **Authentication** tab.
6. In **Password**, paste the password you copied earlier.
7. Select **Store Secret in Key Vault**.
8. Under **Key Vault Connection**, select **Create new**. A **Create connection** dialog is opened on top of the edit dialog.

Step 5: Establish the Key Vault connection

1. In the **Create connection** dialog for the Key Vault connection, in **Key Vault**, select the key vault you created earlier.
2. Select **Review + Create**. You should see that **System assigned managed identity** is set to **Selected**.
3. When validation completes, select **Create**.

Create connection

X

Basics

Networking

Review + Create

Select the service instance and client type.

Service type * ⓘ

Key Vault

Connection name * ⓘ

keyvault_e1c26

Subscription * ⓘ

Dev Compute and Platform

Key vault * ⓘ

vault029345

[Create new](#)

Client type * ⓘ

None

[Next : Networking](#)

[Cancel](#)

[Report an issue](#)

Step 6: Finalize the Service Connector configuration

1. You're back in the edit dialog for **defaultConnector**. In the **Authentication** tab, wait for the key vault connector to be created. When it's finished, the **Key Vault Connection** dropdown automatically selects it.
2. Select **Next: Networking**.
3. Select **Save**. Wait until the **Update succeeded** notification appears.

defaultConnector

X

Basics

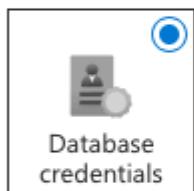
Authentication

Networking

Select the authentication type you'd like to use between your compute service and target service. [Learn more](#)

- System assigned managed identity(Supported via Azure CLI. [Learn more](#)) ⓘ
- User assigned managed identity(Supported via Azure CLI. [Learn more](#)) ⓘ
- Connection string ⓘ
- Service principal(Supported via Azure CLI. [Learn more](#)) ⓘ

Continue with...



Database credentials



Key Vault

Username *

bvfjagfei ...

Password *

***** ...

[Forgot password?](#)

Store Secret In Key Vault ⓘ

Key Vault Connection * ⓘ

vault029345 (keyvault_e1c26) ▼

[Create new](#)

Store Configuration in App Configuration ⓘ

[Next : Networking](#)

[Previous](#)

[Cancel](#)



Step 7: Verify the Key Vault integration

- From the left menu, select **Settings > Environment variables** again.

2. Next to AZURE_MYSQL_CONNECTIONSTRING, select Show value. The value should be @Microsoft.KeyVault(...), which means that it's a key vault reference because the secret is now managed in the key vault.

The screenshot shows the 'App settings' blade in the Azure portal. On the left, there's a sidebar with various options like 'Diagnose and solve problems', 'Deployment', and 'Settings'. Under 'Settings', 'Environment variables' is selected and highlighted with a red box. The main area lists three environment variables:

Name	Value	Deployment slot setting
AZURE_KEYVAULT_RESOURCEENDPOINT	Show value	✓
AZURE_KEYVAULT_SCOPE	Show value	✓
AZURE_MYSQL_CONNECTIONSTRING	Show value	✓

At the bottom of the blade, there are 'Apply' and 'Discard' buttons, and a 'Send us your feedback' button.

To summarize, the process involved retrieving the MySQL connection string from the App Service's environment variables, creating an Azure Key Vault for secure secret management with private access, and updating the service connector to store the password in the key vault. A secure connection between the App Service app and key vault was established using a system-assigned managed identity, and the setup was verified by confirming the connection string uses a Key Vault reference.

Having issues? Check the [Troubleshooting section](#).

4. Confirm JNDI data source

If you add an app setting that contains a valid JDBC connection string for Oracle, SQL Server, PostgreSQL, or MySQL, App Service adds a Java Naming and Directory Interface (JNDI) data source for it in the Tomcat server's *context.xml* file. In this step, you use the SSH connection to the app container to verify the JNDI data source. In the process, you learn how to access the SSH shell for the Tomcat container.

Step 1: Back in the App Service page:

1. In the left menu, select **SSH**.
2. Select **Go**.

The screenshot shows the Azure portal's 'Development Tools' section. On the left, there's a sidebar with icons for 'App Service plan', 'Quotas', 'Change App Service plan', 'Development Tools' (which is selected and highlighted with a red box), and 'Advanced Tools'. Below that is the 'API' section with 'API Management', 'API definition', and 'CORS'. In the center, there's a large 'SSH' button with a magnifying glass icon and a 'Go' button next to it. A red box highlights the 'SSH' button.

Step 2: In the SSH terminal, run `cat /usr/local/tomcat/conf/context.xml`. You should see that a JNDI resource called `jdbc/AZURE_MYSQL_CONNECTIONSTRING_DS` was added. You use this data source later.

```
Last login: Fri Mar  8 06:41:58 2024 from 169.254.129.3
[REDACTED]
JAVA ON APP SERVICE
Documentation: https://aka.ms/appservice

**NOTE**: No files or system changes outside of /home will persist beyond your application's current session. /home is your application's persistent storage and is shared across all the server instances.

root@3ee1f29776d3:/# cat /usr/local/tomcat/conf/context.xml
<?xml version="1.0"?>
<!--
Licensed to the Apache Software Foundation (ASF) under one or more
contributor license agreements. See the NOTICE file distributed with
this work for additional information regarding copyright ownership.
The ASF licenses this file to You under the Apache License, Version 2.0
(the "License"); you may not use this file except in compliance with
the License. You may obtain a copy of the License at
http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and
limitations under the License.
-->
<!-- The contents of this file will be loaded for each web application -->
<Context>
    <!--In AppService we need to disable session persistence across Tomcat restarts-->
    <Manager pathname="/">
<Resource name="jdbc/AZURE_MYSQL_CONNECTIONSTRING_DS" auth="Container" type="javax.sql.DataSource" driverClassName="com.mysql.cj.jdbc.Driver" url="${APPSETTING.AZURE_MYSQL_CONNECTIONSTRING.url}"/><Resource name="jdbc/APPSETTING_AZURE_MYSQL_CONNECTIONSTRING_DS" auth="Container" type="javax.sql.DataSource" driverClassName="com.mysql.cj.jdbc.Driver" url="${APPSETTING.AZURE_MYSQL_CONNECTIONSTRING.url}"/></Context>
root@3ee1f29776d3:/# 
```

① Note

Only changes to files in `/home` can persist beyond app restarts. For example, if you edit `/usr/local/tomcat/conf/server.xml`, the changes won't persist beyond an app restart.

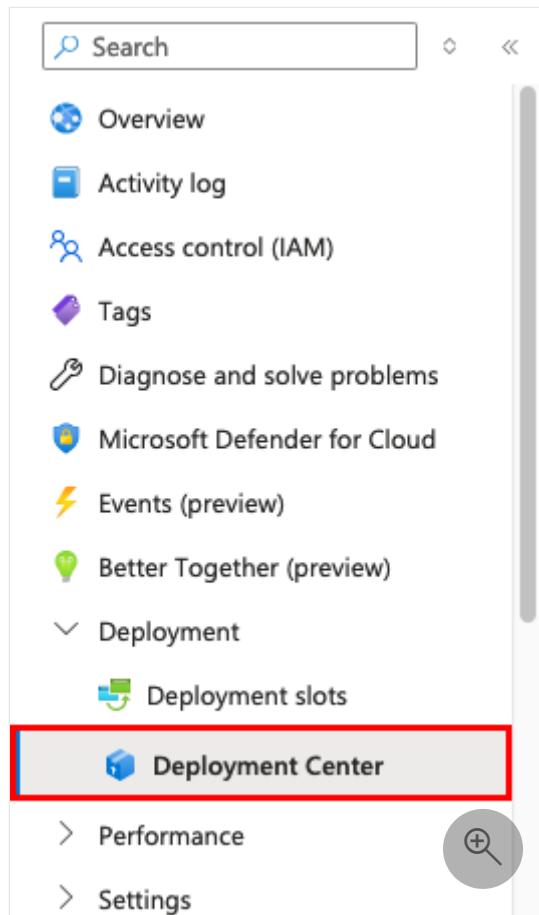
Having issues? Check the [Troubleshooting section](#).

5. Deploy sample code

In this step, you configure GitHub deployment using GitHub Actions. It's just one of many ways to deploy to App Service, but also a great way to have continuous integration in your deployment process. By default, every `git push` to your GitHub repository kicks off the build and deploy action.

Like the Tomcat convention, if you want to deploy to the root context of Tomcat, name your built artifact `ROOT.war`.

Step 1: Back in the App Service page, in the left menu, select **Deployment Center**.



Step 2: In the Deployment Center page:

1. In **Source**, select **GitHub**. By default, **GitHub Actions** is selected as the build provider.

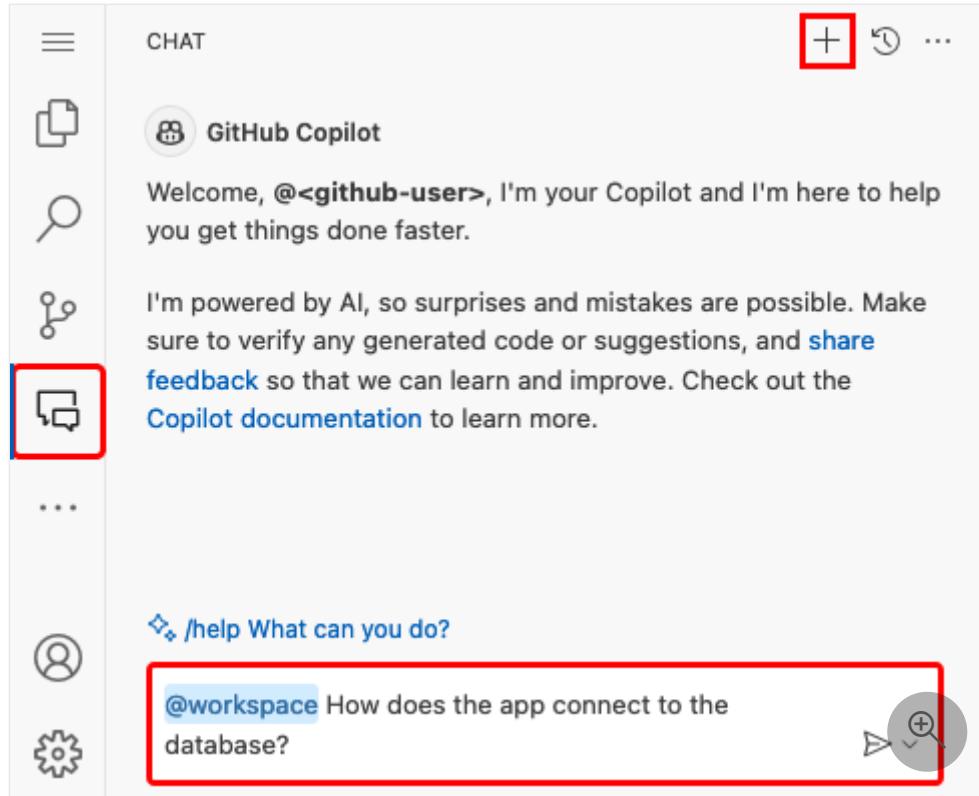
2. Sign in to your GitHub account and follow the prompt to authorize Azure.
3. In **Organization**, select your account.
4. In **Repository**, select **msdocs-tomcat-mysql-sample-app**.
5. In **Branch**, select **starter-no-infra**. This is the same branch that you worked in with your sample app, without any Azure-related files or configuration.
6. For **Authentication type**, select **User-assigned identity**.
7. In the top menu, select **Save**. App Service commits a workflow file into the chosen GitHub repository, in the `.github/workflows` directory. By default, the deployment center **creates a user-assigned identity** for the workflow to authenticate using Microsoft Entra (OIDC authentication). For alternative authentication options, see [Deploy to App Service using GitHub Actions](#).

The screenshot shows the 'Settings' blade for an Azure App Service. At the top, there are navigation links: Save (highlighted with a red box), Discard, Browse, Manage publish profile, Sync, and Leave Feedback. Below these are tabs for Settings, Logs, and FTPS credentials, with 'Settings' being the active tab. A blue banner at the top states: 'You're now in the production slot, which is not recommended for setting up CI/CD. Learn more' with a close button. The main content area is titled 'Deploy and build code from your preferred source and build provider. Learn more'. Under 'Source*', a dropdown menu is set to 'GitHub' (highlighted with a red box). Below it is a link 'Change provider.' Under the 'GitHub' section, there's a sub-section titled 'Signed in as' with a 'Change Account' link. Fields for 'Organization*' (highlighted with a red box), 'Repository*' (highlighted with a red box), and 'Branch*' (highlighted with a red box) are filled with their respective values. Below this is a 'Build' section with 'Runtime stack', 'Version', and 'Java web server stack' fields. At the bottom, under 'Authentication settings', there's a note: 'Select how you want your GitHub Action workflow to authenticate to Azure. If you choose user-assigned identity, the identity selected will be federated with GitHub as an authorized client and given write permissions on the app. Learn more'. Two radio buttons are shown: 'User-assigned identity' (selected and highlighted with a red box) and 'Basic authentication'. A circular icon with a plus sign is located in the bottom right corner of the blade.

Step 3: Back in the GitHub codespace of your sample fork, run `git pull origin starter-no-infra`. This pulls the newly committed workflow file into your codespace.

Step 4 (Option 1: with GitHub Copilot):

1. Start a new chat session by clicking the **Chat** view, then clicking **+**.
2. Ask, "*@workspace How does the app connect to the database?*" Copilot might give you some explanation about the `jdbc/MYSQLDS` data source and how it's configured.
3. Ask, "*@workspace I want to replace the data source defined in persistence.xml with an existing JNDI data source in Tomcat but I want to do it dynamically.*". Copilot might give you a code suggestion similar to the one in the **Option 2: without GitHub Copilot** steps below and even tell you to make the change in the `ContextListener` class.
4. Open `src/main/java/com/microsoft/azure/appservice/examples/tomcatmysql/ContextListener.java` in the explorer and add the code suggestion in the `contextInitialized` method. GitHub Copilot doesn't give you the same response every time, you might need to ask more questions to fine-tune its response. For tips, see [What can I do with GitHub Copilot in my codespace?](#).



Step 4 (Option 2: without GitHub Copilot):

1. Open

`src/main/java/com/microsoft/azure/appservice/examples/tomcatmysql/ContextListener.java` in the explorer. When the application starts, this class loads the database settings in `src/main/resources/META-INF/persistence.xml`.

2. In the `contextInitialized()` method, find the commented code (lines 29-33) and uncomment it. This code checks to see if the `AZURE_MYSQL_CONNECTIONSTRING` app setting exists, and changes the data source to `java:comp/env/jdbc/AZURE_MYSQL_CONNECTIONSTRING_DS`, which is the data source you found earlier in `context.xml` in the SSH shell.

The screenshot shows the Microsoft Visual Studio Code interface. The Explorer sidebar on the left lists files and folders for the 'MSDOCS-TOMCAT-MYSQL-SAMPLE-APP [CODESPACES: FICTITIONAL ADVENTURE...]' project. The Editor tab at the top has 'ContextListener.java M X' selected. The code editor displays Java code for a Tomcat MySQL sample app, specifically the ContextListener.java file. The Source Control tab at the bottom shows a commit message 'Configure Azure data source' with a red box around it.

Step 5:

1. Select the **Source Control** extension.
2. In the textbox, type a commit message like `Configure Azure data source`.
3. Select **Commit**, then confirm with **Yes**.
4. Select **Sync changes 1**, then confirm with **OK**.

The screenshot shows the Microsoft Visual Studio Code interface with the Source Control tab active. A commit message 'Configure Azure data source' is entered in the commit box, highlighted with a red box. The commit button is shown with a checkmark and the word 'Commit'. The code editor shows the same Java code as the previous screenshot.

Step 6: Back in the Deployment Center page in the Azure portal:

1. Select **Logs**. A new deployment run is already started from your committed changes.
2. In the log item for the deployment run, select the **Build/Deploy Logs** entry with the latest timestamp.

Save Discard Browse Manage publish profile Sync Leave Feedback

Settings Logs FTPS credentials

Refresh Delete

Time	Com...	Logs	Commit Author	Status	Message
Friday, March 8, 2024 (4)					
03/8 2024, 9:14:21 AM...	46bb0b6	App Logs	N/A	Success (Active)	Configure Azure data source
03/8 2024, 9:12:52 AM...	93f5e54	Build/Deploy Lo...	Icepheas	Success	Configure Azure data source
03/8 2024, 8:51:01 AM...	77a2985	App Logs	N/A	Success	Add or update the Azure App Service build and deployment workflow config
03/8 2024, 8:49:50 AM...	34bc6dc	Build/Deploy Lo...	Icepheas	Success	Add or update the Azure App Service build and deployment workflow config



Step 7: You're taken to your GitHub repository and see that the GitHub action is running. The workflow file defines two separate stages, build and deploy. Wait for the GitHub run to show a status of **Complete**. It takes about 5 minutes.

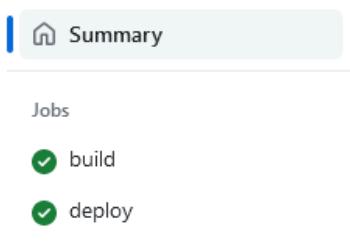
<github-alias> / msdocs-tomcat-mysql-sample-app

Code Pull requests Actions Projects Wiki Security Insights

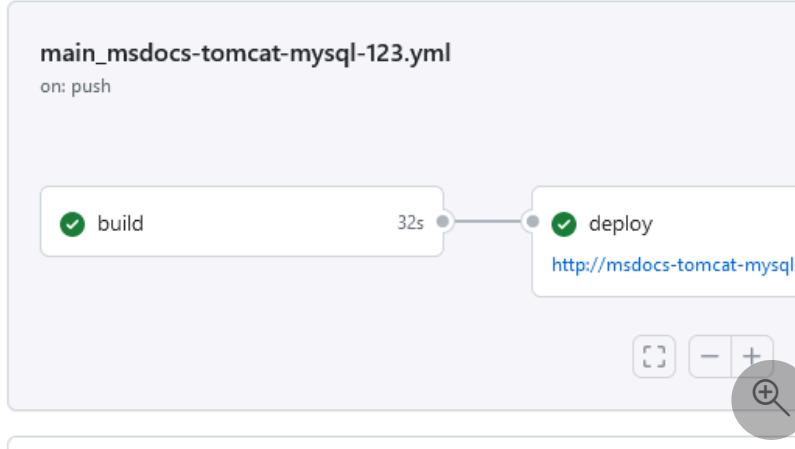
← Build and deploy WAR app to Azure Web App - msdocs-tomcat-mysql-123

✓ Configure Azure data source #2

Re-run all jobs


Summary
Triggered via push 7 minutes ago
Jobs
build
deploy


Status Success
<github-alias> pushed main
Total duration 1m 48s Artifacts 1

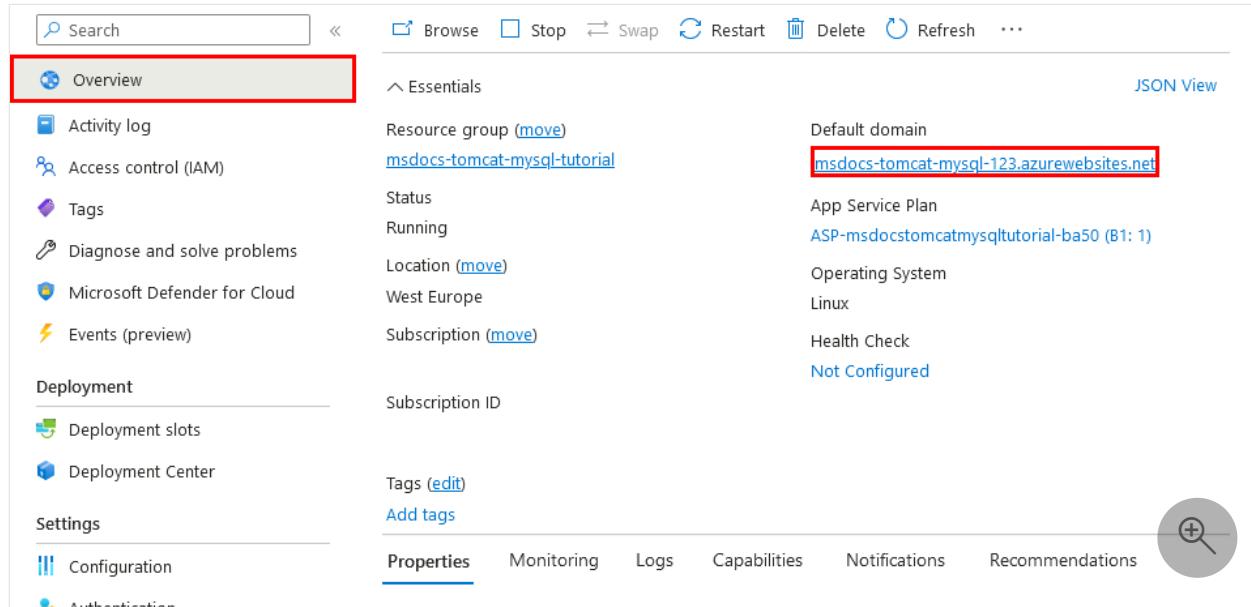

main_msdocs-tomcat-mysql-123.yml
on: push
build --> 32s --> deploy
<http://msdocs-tomcat-mysql-123>

Having issues? Check the [Troubleshooting section](#).

6. Browse to the app

Step 1: In the App Service page:

1. From the left menu, select **Overview**.
2. Select the URL of your app.



The screenshot shows the Azure App Service Overview page. The left sidebar has a red box around the 'Overview' item. The main content area shows various details about the app, including its resource group, status, location, and subscription. The 'Default domain' field contains the URL 'msdocs-tomcat-mysql-123.azurewebsites.net', which is also highlighted with a red box. The bottom navigation bar has tabs for Properties, Monitoring, Logs, Capabilities, Notifications, and Recommendations, with 'Properties' being the active tab. A magnifying glass icon is in the bottom right corner.

Step 2: Add a few tasks to the list. Congratulations, you're running a web app in Azure App Service, with secure connectivity to Azure Database for MySQL.

The screenshot shows a user interface for managing tasks. At the top left is a 'New Task' form with fields for 'Task' and 'Description' (labeled 'Enter task description'). A blue 'Add Task' button is below the form. To the right is a 'Current Tasks' list containing three items: 'Create Tomcat app and MySQL in Azure', 'Deploy data-driven app', and 'That's it!'. Each item has a red 'Delete' button to its right. In the bottom right corner of the tasks area, there is a circular icon with a magnifying glass and a plus sign.

Task	Description	Action
Create Tomcat app and MySQL in Azure	Enter task description	Delete
Deploy data-driven app		Delete
That's it!		Delete

Having issues? Check the [Troubleshooting section](#).

7. Stream diagnostic logs

Azure App Service captures all messages output to the console to help you diagnose issues with your application. The sample application includes standard Log4j logging statements to demonstrate this capability, as shown in the following snippet:

```
Java

@WebServlet(urlPatterns = "/")
public class ViewServlet extends HttpServlet {
    private static Logger logger =
LogManager.getLogger(ViewServlet.class.getName());

    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp)
throws ServletException, IOException {

    logger.info("GET /");
}
```

```
EntityManagerFactory emf = (EntityManagerFactory)
req.getServletContext().getAttribute("EMFactory");
```

Step 1: In the App Service page:

1. From the left menu, select **App Service logs**.
2. Under **Application logging**, select **File System**.
3. In the top menu, select **Save**.

The screenshot shows the 'Monitoring' section of the Azure App Service configuration. On the left, there's a sidebar with various monitoring options: Alerts, Metrics, Logs, Advisor recommendations, Health check, Diagnostic settings, App Service logs (which is highlighted with a red box), Log stream, Tasks (preview), Export template, and Support + troubleshooting. At the top right, there are 'Save', 'Discard', and 'Send us your feedback' buttons. Below the sidebar, the 'Application logging' section is set to 'File System' (also highlighted with a red box). It includes fields for 'Quota (MB)' (set to 35) and 'Retention Period (Days)' (empty). There are also sections for 'Download logs', 'FTP/deployment username' (empty), 'FTP' (set to 'ftp://waws-prod-am2-705.ftp.azurewebsites.windows.net/site/wwwroot'), and 'FTPS' (set to 'ftps://waws-prod-am2-705.ftp.azurewebsites.windows.net/site/wwwroot'). A search bar at the top left and a magnifying glass icon on the right are also visible.

Step 2: From the left menu, select **Log stream**. You see the logs for your app, including platform logs and logs from inside the container.

The screenshot shows the 'Log stream' section of the Azure App Service configuration. The left sidebar is identical to the previous screenshot, with 'Log stream' highlighted by a red box. The main area displays a log history. The first few entries are:
2024-03-20T15:52:54.060149445Z GET /
2024-03-20T15:52:54.060149445Z GET /
- - [20/Mar/2024:15:52:53 +0000] "POST /
create HTTP/1.1" 302 - 467969 7686b715-a65f-4878-8767-
cc61b5832f17 - - [20/Mar/2024:15:52:53 +0000]
"GET / HTTP/1.1" 200 791 226173 03875010-c768-4a7c-ba14-
a79d44723adc - - [20/Mar/2024:15:52:54 +0000]
"GET /favicon.ico HTTP/1.1" 200 791 88796 3793c9fb-
cf3c-40c5-b097-6b74e1514e25
- - [20/Mar/2024:15:53:55 +0000] "GET /
HTTP/1.1" 200 791 230654 019fb295-d0de-434c-
b0c4-2893ed9e22bf - - [20/Mar/2024:15:53:55
+0000] "GET /favicon.ico HTTP/1.1" 200 791 129635
97a15233-2d56-4fba-9f89-4cbefc86d289
2024-03-20T15:53:54.952178862Z GET /
2024-03-20T15:53:55.254795426Z GET /
2024-03-20T15:55:01.231469111Z GET /
- - [20/Mar/2024:15:55:01 +0000] "GET /
HTTP/1.1" 200 2983 120977 fc82ce31-9cda-438a-
a290-74f1c6af2fec

A search bar and a magnifying glass icon are located at the bottom right of the log viewer.

Learn more about logging in Java apps in the series on [Enable Azure Monitor OpenTelemetry for .NET, Node.js, Python, and Java applications](#).

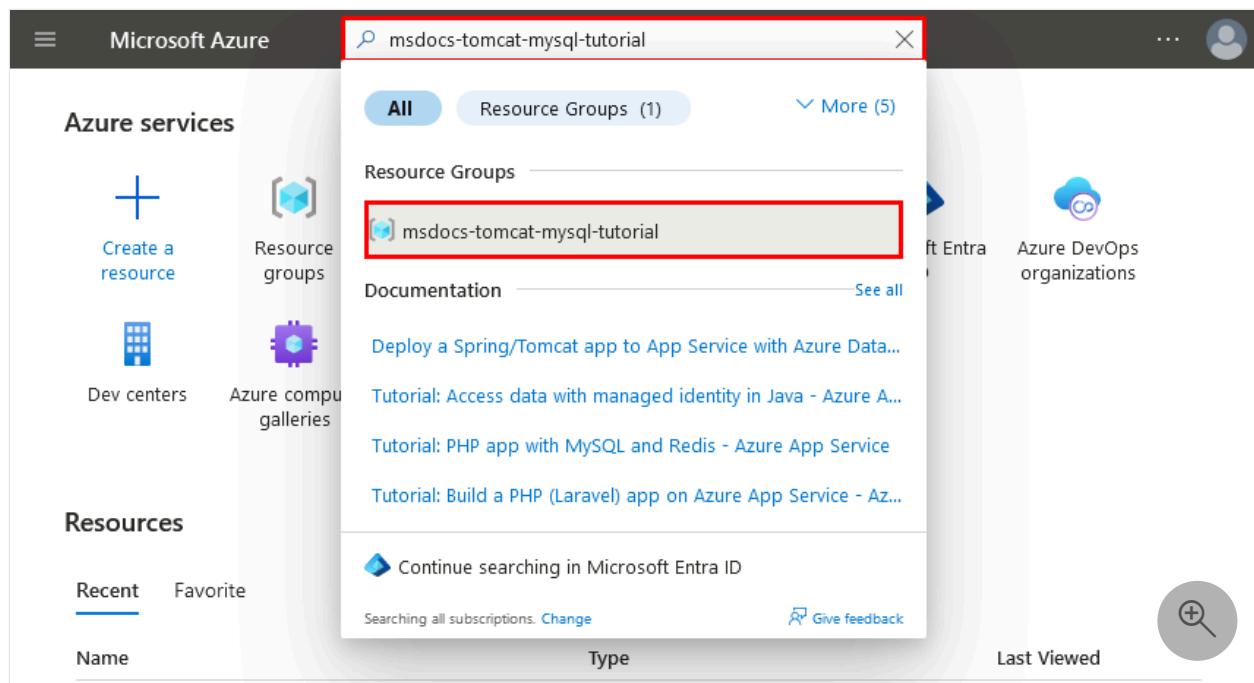
Having issues? Check the [Troubleshooting section](#).

8. Clean up resources

When you're finished, you can delete all of the resources from your Azure subscription by deleting the resource group.

Step 1: In the search bar at the top of the Azure portal:

1. Enter the resource group name.
2. Select the resource group.



The screenshot shows the Microsoft Azure portal interface. At the top, there's a search bar with the text "msdocs-tomcat-mysql-tutorial". Below the search bar, the "Azure services" sidebar is visible, showing options like "Create a resource", "Dev centers", and "Resource groups". The "Resource groups" option is selected. The main content area displays a "Resource Groups" section with a single item: "msdocs-tomcat-mysql-tutorial", which is highlighted with a red box. Below this, there's a "Documentation" section with several links related to Azure App Service and Azure DevOps. At the bottom of the search results, there are buttons for "Continue searching in Microsoft Entra ID" and "Give feedback".

Step 2: In the resource group page, select **Delete resource group**.

The screenshot shows the Azure portal interface for the resource group 'msdocs-tomcat-mysql-tutorial'. At the top, there are navigation links for 'Home >' and the resource group name. Below the name, it says 'Resource group'. On the left, there's a sidebar with 'Essentials' expanded. At the top right, there are buttons for 'Create', 'Manage view', 'Delete resource group' (which is highlighted with a red box), 'Refresh', 'Export to CSV', 'Open query', and more. A 'JSON View' link is also present. The main area shows a table of resources under the 'Resources' tab. The table has columns for Name, Type, and Location. It lists two items: 'ASP-msdocstomcatmysqltutorial-ba50' (App Service plan, West Europe) and 'msdocs-tomcat-my-id-aaf3' (Managed Identity, West Europe). There are filters at the top of the table: 'Type equals all' and 'Location equals all', both with 'X' buttons. Below the table are buttons for 'No grouping' and 'List view'. On the far right of the table, there's a circular icon with a magnifying glass and three dots.

Name	Type	Location
ASP-msdocstomcatmysqltutorial-ba50	App Service plan	West Europe
msdocs-tomcat-my-id-aaf3	Managed Identity	West Europe

Step 3:

1. Confirm your deletion by typing the resource group name.
2. Select **Delete**.
3. Confirm with **Delete** again.

Delete a resource group

X

The following resource group and all its dependent resources will be permanently deleted.

Resource group to be deleted



Dependent resources to be deleted (9)

All dependent resources, including hidden types, are shown

Name	Resource type
ASP-msdocstomcatmysqltutorial-ba50	App Service plan
msdocs-tomcat-my-id-aaf3	Managed Identity
msdocs-tomcat-mysql-123	App Service
msdocs-tomcat-mysql-123-server	Azure Database for MySQL flexible se...
msdocs-tomcat-mysql-123Vnet	Virtual network
privatelink.mysql.database.azure.com	Private DNS zone
privatelink.mysql.database.azure.com-dblink (Microsoft.Network/privateDnsZones/v...	
privatelink.redis.cache.windows.net	Private DNS zone
privatelink.redis.cache.windows.net-applink (Microsoft.Network/privateDnsZones/v...	

Enter resource group name to confirm deletion *

msdocs-tomcat-mysql-tutorial

Delete

Cancel



Troubleshooting

- I see many <Class> scanned from multiple locations warnings with mvn jetty:run
- The portal deployment view for Azure Database for MySQL Flexible Server shows a Conflict status
- The deployed sample app doesn't show the tasks list app
- I see a 404 Page Not Found error in the deployed sample app

I see many <Class> scanned from multiple locations warnings with mvn jetty:run

You can ignore the warnings. The Maven Jetty plugin shows the warnings because the app's `pom.xml` contains the dependency for `jakarta.servlet.jsp.jstl`, which the Jetty already provides out of the box. You need the dependency for Tomcat.

The portal deployment view for Azure Database for MySQL Flexible Server shows a Conflict status

Depending on your subscription and the region you select, you might see the deployment status for Azure Database for MySQL Flexible Server to be `Conflict`, with the following message in Operation details:

`InternalServerError: An unexpected error occurred while processing the request.`

This error is most likely caused by a limit on your subscription for the region you select. Try choosing a different region for your deployment.

The deployed sample app doesn't show the tasks list app

If you see a `Hey, Java developers!` page instead of the tasks list app, App Service is most likely still loading the updated container from your most recent code deployment. Wait a few minutes and refresh the page.

I see a 404 Page Not Found error in the deployed sample app

Make sure that you made the code changes to use the `java:comp/env/jdbc/AZURE_MYSQL_CONNECTIONSTRING_DS` data source. If you made the changes and redeployed your code, App Service is most likely still loading the updated container. Wait a few minutes and refresh the page.

Frequently asked questions

- How much does this setup cost?
- How do I connect to the MySQL server behind the virtual network with other tools?
- How does local app development work with GitHub Actions?
- I don't have permissions to create a user-assigned identity
- What can I do with GitHub Copilot in my codespace?

How much does this setup cost?

Pricing for the created resources is as follows:

- The App Service plan is created in **Basic** tier and can be scaled up or down. See [App Service pricing ↗](#).
- The MySQL flexible server is created in **B1ms** tier and can be scaled up or down. With an Azure free account, **B1ms** tier is free for 12 months, up to the monthly limits. See [Azure Database for MySQL pricing ↗](#).
- The Azure Cache for Redis is created in **Basic** tier with the minimum cache size. There's a small cost associated with this tier. You can scale it up to higher performance tiers for higher availability, clustering, and other features. See [Azure Cache for Redis pricing ↗](#).
- The virtual network doesn't incur a charge unless you configure extra functionality, such as peering. See [Azure Virtual Network pricing ↗](#).
- The private DNS zone incurs a small charge. See [Azure DNS pricing ↗](#).

How do I connect to the MySQL server behind the virtual network with other tools?

- The Tomcat container currently doesn't have the `mysql-client` terminal too. If you want, you must manually install it. Remember that anything you install doesn't persist across app restarts.
- To connect from a desktop tool like MySQL Workbench, your machine must be within the virtual network. For example, it could be an Azure VM in one of the subnets, or a machine in an on-premises network that has a [site-to-site VPN](#) connection with the Azure virtual network.
- You can also [integrate Azure Cloud Shell](#) with the virtual network.

How does local app development work with GitHub Actions?

Using the autogenerated workflow file from App Service as an example, each `git push` kicks off a new build and deployment run. From a local clone of the GitHub repository, you make the desired updates and push to GitHub. For example:

```
terminal

git add .
git commit -m "<some-message>"
git push origin main
```

I don't have permissions to create a user-assigned identity

See [Set up GitHub Actions deployment from the Deployment Center](#).

What can I do with GitHub Copilot in my codespace?

You might notice that the GitHub Copilot chat view was already there for you when you created the codespace. For your convenience, we include the GitHub Copilot chat extension in the container definition (see `.devcontainer/devcontainer.json`). However, you need a [GitHub Copilot account](#) (30-day free trial available).

A few tips for you when you talk to GitHub Copilot:

- In a single chat session, the questions and answers build on each other and you can adjust your questions to fine-tune the answer you get.
- By default, GitHub Copilot doesn't have access to any file in your repository. To ask questions about a file, open the file in the editor first.
- To let GitHub Copilot have access to all of the files in the repository when preparing its answers, begin your question with `@workspace`. For more information, see [Use the @workspace agent](#).
- In the chat session, GitHub Copilot can suggest changes and (with `@workspace`) even where to make the changes, but it's not allowed to make the changes for you. It's up to you to add the suggested changes and test it.

Here are some other things you can say to fine-tune the answer you get:

- Change this code to use the data source `jdbc/AZURE_MYSQL_CONNECTIONSTRING_DS`.
- Some imports in your code are using `javax` but I have a Jakarta app.
- I want this code to run only if the environment variable `AZURE_MYSQL_CONNECTIONSTRING` is set.
- I want this code to run only in Azure App Service and not locally.

Next steps

- [Azure for Java Developers](#)

Learn more about running Java apps on App Service in the developer guide.

[Configure a Java app in Azure App Service](#)

Learn how to secure your app with a custom domain and certificate.

Secure with custom domain and certificate

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

Tutorial: Build a JBoss web app with Azure App Service on Linux and MySQL

Article • 12/13/2024

This tutorial shows how to build, configure, and deploy a secure JBoss application in Azure App Service that connects to a MySQL database (using [Azure Database for MySQL](#)). Azure App Service is a highly scalable, self-patching, web-hosting service that can easily deploy apps on Windows or Linux. When you're finished, you'll have a JBoss app running on [Azure App Service on Linux](#).

The screenshot displays a user interface for managing tasks. At the top, there is a "New Task" form with fields for "Task" and "Enter task description", and a blue "Add Task" button. Below this is a "Current Tasks" section containing three items, each with a "Delete" button:

- Create JBoss app and MySQL in Azure
- Deploy data-driven app
- That's it!

In this tutorial, you learn how to:

- ✓ Create a secure-by-default architecture for Azure App Service and Azure Database for MySQL Flexible Server.
- ✓ Secure database connectivity using a passwordless connection string.
- ✓ Verify JBoss data sources in App Service using JBoss CLI.
- ✓ Deploy a JBoss sample app to App Service from a GitHub repository.
- ✓ Access App Service app settings in the application code.
- ✓ Make updates and redeploy the application code.
- ✓ Stream diagnostic logs from App Service.
- ✓ Manage the app in the Azure portal.
- ✓ Provision the same architecture and deploy by using Azure Developer CLI.
- ✓ Optimize your development workflow with GitHub Codespaces and GitHub Copilot.

Prerequisites

- An Azure account with an active subscription. If you don't have an Azure account, you [can create one for free](#).
- A GitHub account. you can also [get one for free](#).
- Knowledge of Java with JBoss development.
- **(Optional)** To try GitHub Copilot, a [GitHub Copilot account](#). A 30-day free trial is available.

Skip to the end

You can quickly deploy the sample app in this tutorial and see it running in Azure. Just run the following commands in the [Azure Cloud Shell](#), and follow the prompt:

Bash

```
mkdir msdocs-jboss-mysql-sample-app
cd msdocs-jboss-mysql-sample-app
azd init --template msdocs-jboss-mysql-sample-app
azd up
```

1. Run the sample

First, you set up a sample data-driven app as a starting point. For your convenience, the [sample repository](#), includes a [dev container](#) configuration. The dev container has everything you need to develop an application, including the database, cache, and all environment variables needed by the sample application. The dev container can run in a [GitHub codespace](#), which means you can run the sample on any computer with a web browser.

Step 1: In a new browser window:

1. Sign in to your GitHub account.
2. Navigate to <https://github.com/Azure-Samples/msdocs-jboss-mysql-sample-app/fork>.
3. Unselect **Copy the main branch only**. You want all the branches.
4. Select **Create fork**.

The screenshot shows the GitHub fork creation interface for the `msdocs-jboss-mysql-sample-app` repository. At the top, there's a navigation bar with links for Code, Issues, Pull requests, Actions, Projects, Security, and Insights. Below the navigation bar, the title "Create a new fork" is displayed. A note explains that a fork is a copy of a repository, allowing experimentation without affecting the original project, with a link to "View existing forks". A note also states that required fields are marked with an asterisk (*). The "Owner" dropdown is set to "I", and the "Repository name" field contains "msdocs-jboss-mysql-sample-". A green checkmark indicates that the name is available. A note says that by default, forks are named the same as their upstream repository, but you can customize it. There's a "Description (optional)" field containing "This is a JBoss CRUD sample application for Azure App Service and the Azure Database for MySQL flexible". A checkbox for "Copy the main branch only" is unchecked. A note below it encourages contributing back to the upstream repository. A message informs the user that they are creating a fork in their personal account. At the bottom right, there's a red "Create fork" button with a magnifying glass icon.

Create a new fork

A *fork* is a copy of a repository. Forking a repository allows you to freely experiment with changes without affecting the original project. [View existing forks](#).

Required fields are marked with an asterisk (*).

Owner * Repository name *

I / msdocs-jboss-mysql-sample-

msdocs-jboss-mysql-sample-app is available.

By default, forks are named the same as their upstream repository. You can customize the name to distinguish it further.

Description (optional)

This is a JBoss CRUD sample application for Azure App Service and the Azure Database for MySQL flexible

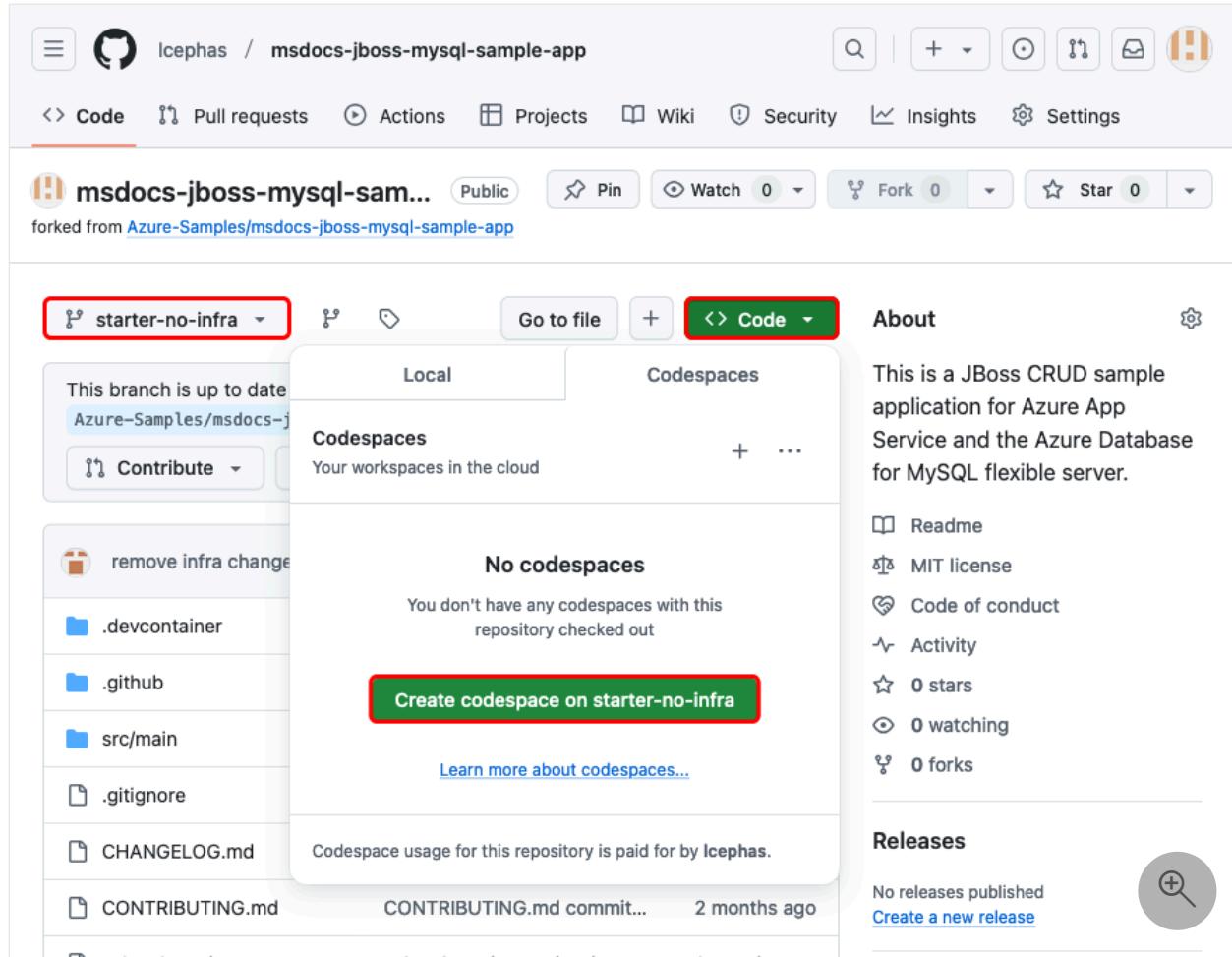
Copy the main branch only
Contribute back to Azure-Samples/msdocs-jboss-mysql-sample-app by adding your own branch. [Learn more](#).

ⓘ You are creating a fork in your personal account.

Create fork

Step 2: In the GitHub fork:

1. Select **main > starter-no-infra** for the starter branch. This branch contains just the sample project and no Azure-related files or configuration. Select **Code > Create codespace on starter-no-infra**. The codespace takes a few minutes to set up.



Step 3: In the codespace terminal:

1. Run `mvn clean wildfly:run`.
2. When you see the notification `Your application running on port 8080 is available.`, wait a few seconds longer for the WildFly server to finish loading the application. Then, select **Open in Browser**. You should see the sample application in a new browser tab. To stop the WildFly server, type `Ctrl + C`.

The screenshot shows the Microsoft Visual Studio Code interface. The left sidebar has icons for Explorer, Search, Open, and others. The Explorer panel shows a folder named 'MSDOCS-JBOSS-MYSQL-SAMPLE-APP [CODESPACES: ...]' containing '.devcontainer', '.github', 'src', '.gitignore', 'pom.xml', and 'README.md'. The main area is titled 'Preview README.md' and contains the following text:

Deploy a JBoss EAP web app with MySQL in Azure

This is a Java EE 10 web app that uses the Azure Database for MySQL flexible server. This app is designed to be run locally with the WildFly Maven plugin and then deployed to a JBoss 8 EAP container in Azure App Service. You can either deploy this project by following the tutorial [Deploy a JBoss web app with MySQL in Azure](#).

Run the sample

TERMINAL

```
@ [WARNING] Unable to find the root directory. Create a .mvn directory in the root directory or add the root="true" attribute on the root project's model to identify it.
[INFO] Scanning for projects...
[INFO] [INFO] Building jbossmysql 0.1-SNAPSHOT
[INFO] [INFO] [INFO] [INFO] --- clean:3.2.0:clean (default-clean) @ jbossmysql ---
[INFO] [INFO] Deleting /workspaces/msdocs-jboss-mysql-sample-app/target
[INFO] [INFO] >>> wildfly:5.0.1.Final:run (default-cli) > package @ jbossmysql >>>
[INFO] [INFO] --- resources:3.3.1:resources (default-resources) @ jbossmysql ---
[INFO] [INFO] Copying 1 resource from src/main/resources to target/classes
14:28:01,015 INFO  [jakarta.enterprise.resource.webcontainer.faces.config] (ServerService Thread Pool -- 10) Initializing!
14:28:01,419 INFO  [org.wildf] ① Your application running on port 8080 is available. See all  ⚙️ ×
021: Registered web context: forwarded ports
14:28:01,456 INFO  [org.jboss.d "jbossmysql.war" (runtime-n
14:28:01,471 INFO  [org.jboss.v interrupting thread Thread[

[...] ②
```

STATUS BAR: Codespaces: cuddly space halibut | Java: Ready

Tip

You can ask [GitHub Copilot](#) about this repository. For example:

- @workspace What does this project do?
- @workspace What does the .devcontainer folder do?

Having issues? Check the [Troubleshooting section](#).

2. Create App Service and MySQL

First, you create the Azure resources. The steps used in this tutorial create a set of secure-by-default resources that include App Service and Azure Database for MySQL. For the creation process, you specify:

- The **Name** for the web app. It's used as part of the DNS name for your app in the form of `https://<app-name>-<hash>.azurewebsites.net`.
- The **Region** to run the app physically in the world. It's also used as part of the DNS name for your app.
- The **Runtime stack** for the app. It's where you select the version of Java to use for your app.

- The **Hosting plan** for the app. It's the pricing tier that includes the set of features and scaling capacity for your app.
- The **Resource Group** for the app. A resource group lets you group (in a logical container) all the Azure resources needed for the application.

Sign in to the [Azure portal](#) and follow these steps to create your Azure App Service resources.

Step 1: In the Azure portal:

1. In the top search bar, type *app service*.
2. Select the item labeled **App Service** under the **Services** heading.
3. Select **Create > Web App**. You can also navigate to the [creation wizard](#) directly.

Step 2: In the **Create Web App** page, fill out the form as follows.

1. **Name:** `msdocs-jboss-mysql`. A resource group named `msdocs-jboss-mysql_group` will be generated for you.
2. **Runtime stack:** **Java 17**.
3. **Java web server stack:** **Red Hat JBoss EAP 8**. If you configured your Red Hat subscription with Azure already, select **Red Hat JBoss EAP 8 BYO License**.
4. **Region:** Any Azure region near you.
5. **Linux Plan:** **Create new** and use the name `msdocs-jboss-mysql`.
6. **Pricing plan:** **Premium V3 P0V3**. When you're ready, you can [scale up](#) to a different pricing tier.
7. **Deploy with your app:** Select **Database**. Azure Database for MySQL - Flexible Server is selected for you by default. It's a fully managed MySQL database as a service on Azure, compatible with the latest community editions.
8. Select **Review + create**.
9. After validation completes, select **Create**.

Create Web App

X

Resource Group * ⓘ

(New) msdocs-jboss-mysql_group

[Create new](#)

Instance Details

Name

msdocs-jboss-mysql



-g2 cwd2dvcxf5seyfd.canadacentral-01.azurewebsites.net

Unique default hostname (preview) on. [More about this update ↗](#)

Publish *

Code Container Static Web App

Runtime stack *

Java 17

Java web server stack *

Red Hat JBoss EAP 8

Operating System *

Linux Windows

Region *

Canada Central



i Not finding your App Service Plan? Try a different region or select your App Service Environment.

Pricing plans

App Service plan pricing tier determines the location, features, cost and compute resources associated with your app.

[Learn more ↗](#)

Linux Plan (Canada Central) * ⓘ

(New) msdocs-jboss-mysql



[Create new](#)

Pricing plan

Premium V3 P0V3 (195 minimum ACU/vCPU, 4 GB memory, 1 vCPU)



[Explore pricing plans](#)

Zone redundancy

An App Service plan can be deployed as a zone redundant service in the regions that support it. This is a deployment time only decision. You can't make an App Service plan zone redundant after it has been deployed [Learn more ↗](#)

Zone redundancy

Enabled: Your App Service plan and the apps in it will be zone redundant. The minimum App Service plan instance count will be three.

Disabled: Your App Service Plan and the apps in it will not be zone redundant. The minimum App Service plan instance count will be one.

Recommended services (preview)

Within a new App Service plan, you can deploy these commonly-used services with your app. Services you select will be deployed secure-by-default. [Learn More ↗](#)

Deploy with your app

Database (optimized based on your runtime)

Engine: MySQL - Flexible Server; **DB name:** msdocs-jboss-mysql-database; **Server name:** msdocs-jboss-mysql-server ([Customize](#))

Azure Cache for Redis (boost your app's performance)

[Review + create](#)

[< Previous](#)

[Next : Deployment >](#)



Step 3: The deployment takes a few minutes to complete. Once deployment completes, select the **Go to resource** button. You're taken directly to the App Service app, but the following resources are created:

- **Resource group:** The container for all the created resources.
- **App Service plan:** Defines the compute resources for App Service. A Linux plan in the *Basic* tier is created.
- **App Service:** Represents your app and runs in the App Service plan.
- **Virtual network:** Integrated with the App Service app and isolates back-end network traffic.
- **Azure Database for MySQL Flexible Server:** Accessible only from the virtual network. A database and a user are created for you on the server.
- **Private DNS zones:** Enable DNS resolution of the database server in the virtual network.
- **Private endpoints:** Access endpoints for the database server in the virtual network.

Create Web App ... X

Basics Deployment Networking Monitor + secure Tags Review + create

Continuous deployment settings

Set up continuous deployment to easily deploy code from your GitHub repository via GitHub Actions. [Learn more ↗](#)

Continuous deployment Disable Enable

GitHub settings

Set up GitHub Actions to push content to your app whenever there are code changes made to your repository. Note: Your GitHub account must have write access to the selected repository in order to add a workflow file which manages deployments to your app.

GitHub account [Change account](#) ⓘ

Organization * <github-alias>

Repository * msdocs-jboss-mysql-sample-app

Branch * main

Workflow configuration

Click the button below to preview what the GitHub Actions workflow file will look like before setting up continuous deployment.

[Preview file](#)

Authentication settings

Choose if you would like to allow basic authentication to deploy code to your app. [Learn more ↗](#)

Basic authentication Disable Enable

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

Having issues? Check the [Troubleshooting section](#).

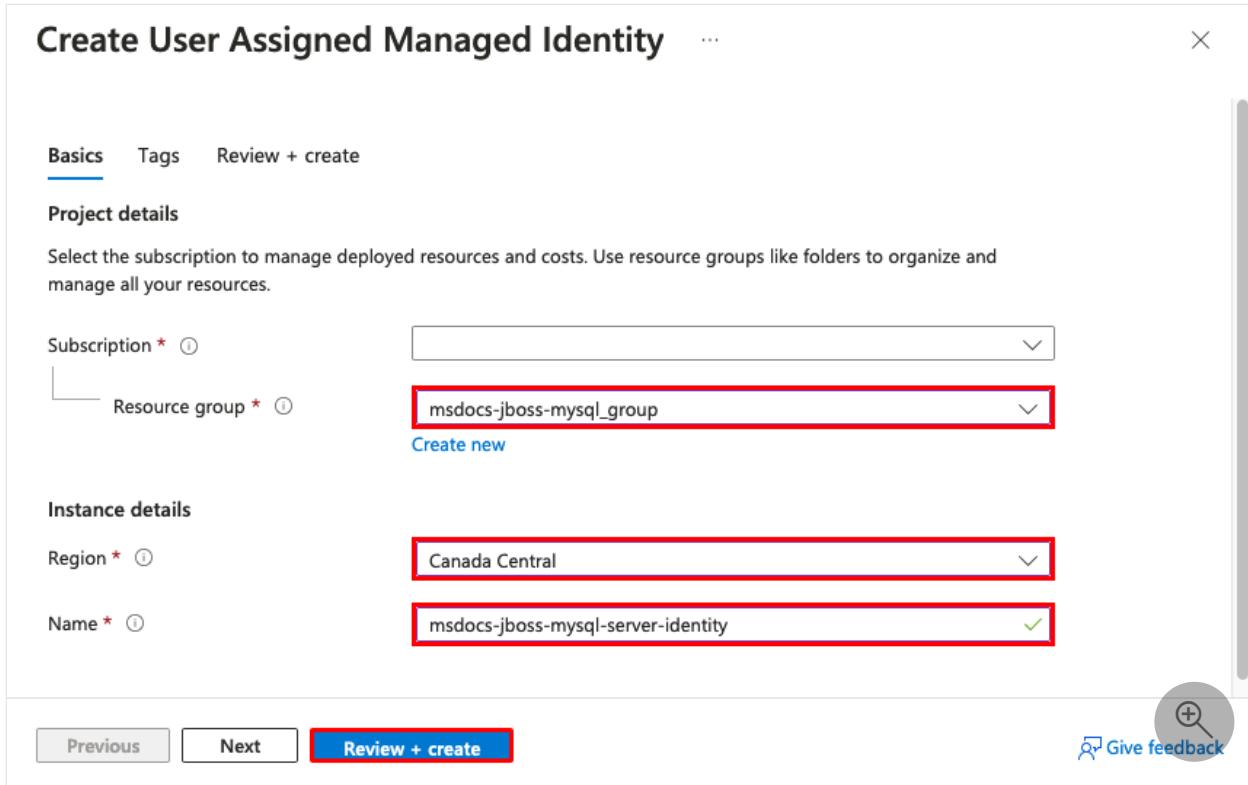
3. Create a passwordless connection

In this step, you generate a managed identity based service connection, which you can later use to create a data source in your JBoss server. By using a managed identity to connect to the MySQL database, your code is safe from accidental secrets leakage.

Step 1: Create a managed identity.

1. In the top search bar, type *managed identity*.
2. Select the item labeled **Managed Identities** under the **Services** heading.
3. Select **Create**.
4. In **Resource group**, select **msdocs-jboss-mysql_group**.

5. In **Region**, select the same region that you used for your web app.
6. In **Name**, type `msdocs-jboss-mysql-server-identity`.
7. Select **Review + create**.
8. Select **Create**.



Step 2: Enable Microsoft Entra authentication in the MySQL server.

1. In the top search bar, type `msdocs-jboss-mysql-server`.
2. Select the Azure Database for MySQL Flexible Server resource called `msdocs-jboss-mysql-server`.
3. From the left menu, select **Security > Authentication**.
4. In **Assign access to**, select **Microsoft Entra authentication only**.
5. In **User assigned managed identity**, select **Select**.
6. Select `msdocs-jboss-mysql-server-identity`, then select **Add**. It takes a moment for the identity to be assigned to the MySQL server.
7. In **Microsoft Entra Admin Name**, select **Select**.
8. Find your Azure account and select it, then select **Select**.
9. Select **Save** and wait for the operation to complete.

The screenshot shows the Azure portal interface for managing a MySQL server's authentication. The left sidebar shows various service links like Overview, Activity log, Access control (IAM), Tags, Diagnose and solve problems, Learning center, Settings, Power Platform, Security, Identity, Data encryption, and Authentication. The Authentication link is highlighted with a red box. At the top, there are Save, Discard, and Feedback buttons. The main content area is titled 'Authentication' and includes a note about Azure Active Directory (now Microsoft Entra ID) and how to select authentication methods (MySQL authentication only, Microsoft Entra authentication only, or both). A radio button for 'Microsoft Entra authentication only' is selected and highlighted with a red box. Below this is a 'Select Identity' section where a user assigned managed identity is selected. The 'User assigned managed identity' dropdown shows 'msdocs-jboss-mysql-server-identity' and has a 'Select' button highlighted with a red box. There is also a 'Create' button. Further down is a 'Microsoft Entra Admins' section with a text input field containing 'azure.account.com#EXT#@azureaccount.onmicrosoft.com' which is also highlighted with a red box. A magnifying glass icon is at the bottom right.

Step 3: Add a managed identity-based service connector.

1. In the top search bar, type *msdocs-jboss-mysql*.
2. Select the App Service resource called **msdocs-jboss-mysql**.
3. In the App Service page, in the left menu, select **Settings > Service Connector**.
4. Select **Create**.
5. In the **Basics** tab:
 6. Set Service type to **DB for MySQL flexible server**.
 7. Set MySQL flexible server to **msdocs-jboss-mysql-server**.
 8. Set MySQL database to **msdocs-jboss-mysql-database**.
 9. Set Client type to **Java**.
10. Select the **Authentication** tab.
11. Select **System assigned managed identity**.
12. Select the **Review + Create** tab.
13. When validation completes, select **Create on Cloud Shell** and wait for the operation to complete in the Cloud Shell.
14. When you see the output JSON, you can close the Cloud Shell. Also, close the **Create connection** dialog.
15. Select **Refresh** to show the new service connector.

Create connection

Basics Authentication Networking **Review + Create**

Validation passed.

Details

Connection name	mysql_3b953
Subscription	Visual Studio Enterprise Subscription
Client type	Java
Service type	DB for MySQL flexible server
MySQL flexible server	msdocs-jboss-mysql-server
MySQL database	msdocs-jboss-mysql-database

Networking

Network solution	Firewall rules of the FlexibleMySQL will be configured to allow access from the WebApp.
------------------	---

CLI command

Please click Create On Cloud Shell or execute the following commands on local.

Create On Cloud Shell Previous Cancel

Step 4: Add authentication plugins to the connection string.

1. From the left menu, select **Environment variables**.
2. Select **AZURE_MYSQL_CONNECTIONSTRING**. The **Value** field should contain a `user` but no `password`. The user is a managed identity.
3. The JBoss server in your App Service app has the authentication plugins authenticate the managed identity, but you still need to add it to the connection string. Scroll to the end of the value and append
`&defaultAuthenticationPlugin=com.azure.identity.extensions.jdbc.mysql.AzureMys
qlAuthenticationPlugin&authenticationPlugins=com.azure.identity.extensions.jdb
c.mysql.AzureMysqlAuthenticationPlugin.`
4. Select **Apply**.
5. Select **Apply**, then **Confirm**.

The screenshot shows the Azure App Service configuration interface. On the left, under 'App settings', the 'Environment variables' section is selected. A modal window titled 'Add/Edit application setting' is open, showing a single entry for 'AZURE_MYSQL_CONNECTIONSTRING' with the value 'ed8user=aad_mysql_3b953&defaultAuthenticationPlugin=com.azure.identity.extensions.jdbc.mysql.AzureMySQLAuthenticationPlug...'. The 'Name' field has 'AZURE_MYSQL_CONNECTIONSTRING' and the 'Value' field has the connection string. The 'Deployment slot setting' checkbox is checked.

Having issues? Check the [Troubleshooting section](#).

4. Confirm JNDI data source

If you add an app setting that contains a valid JDBC connection string for Oracle, SQL Server, PostgreSQL, or MySQL, App Service adds a Java Naming and Directory Interface (JNDI) data source for it in the JBoss server. In this step, you use the SSH connection to the app container to verify the JNDI data source. In the process, you learn how to access the SSH shell and run the JBoss CLI.

Step 1: Back in the App Service page:

1. In the left menu, select **Development Tools > SSH**.
2. Select **Go**.

The screenshot shows the Azure App Service configuration interface. On the left, under 'Development Tools', the 'SSH' option is selected and highlighted with a red box. To the right, there is a section titled 'SSH' with the subtext 'SSH provides a web-based SSH console interface for your app. [Learn more](#)' and a 'Go' button which is also highlighted with a red box.

Step 2: In the SSH terminal:

1. Run `$JBoss_HOME/bin/jboss-cli.sh --connect`.
 2. In the JBoss CLI connection, run `ls subsystem=datasources/data-source`. You should see the automatically generated data source called `AZURE_MYSQL_CONNECTIONSTRING_DS`.
 3. Get the JNDI name of the data source with `/subsystem=datasources/data-source=AZURE_MYSQL_CONNECTIONSTRING_DS:read-attribute(name=jndi-name)`. You now have a JNDI name `java:jboss/env/jdbc/AZURE_MYSQL_CONNECTIONSTRING_DS`, which you can use in your application code later.

! Note

Only changes to files in `/home` can persist beyond app restarts. For example, if you edit `/opt/eap/standalone/configuration/standalone.xml` or change server configuration in the JBoss CLI, the changes won't persist beyond an app restart. To persist your changes, use a startup script, such as demonstrated in [Configure data sources for a Tomcat, JBoss, or Java SE app in Azure App Service](#)

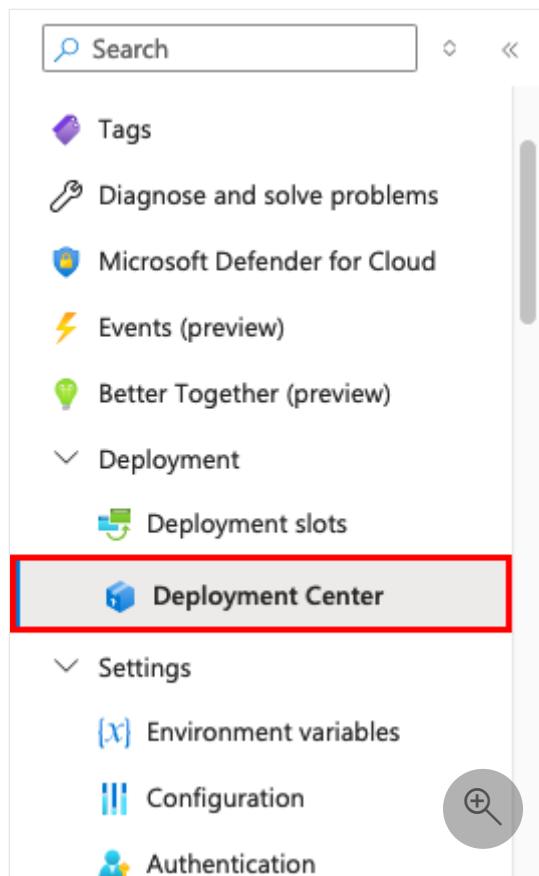
Having issues? Check the [Troubleshooting section](#).

5. Deploy sample code

In this step, you configure GitHub deployment using GitHub Actions. It's just one of many ways to deploy to App Service, but also a great way to have continuous integration in your deployment process. By default, every `git push` to your GitHub repository kicks off the build and deploy action.

Like the JBoss convention, if you want to deploy to the root context of JBoss, name your built artifact `ROOT.war`.

Step 1: Back in the App Service page, in the left menu, select **Deployment > Deployment Center**.



Step 2: In the Deployment Center page:

1. In **Source**, select **GitHub**. By default, **GitHub Actions** is selected as the build provider.
2. Sign in to your GitHub account and follow the prompt to authorize Azure.
3. In **Organization**, select your account.
4. In **Repository**, select `msdocs-jboss-mysql-sample-app`.
5. In **Branch**, select `starter-no-infra`. This is the same branch that you worked in with your sample app, without any Azure-related files or configuration.
6. For **Authentication type**, select **User-assigned identity**.
7. In the top menu, select **Save**. App Service commits a workflow file into the chosen GitHub repository, in the `.github/workflows` directory. By default, the deployment center creates a **user-assigned identity** for the workflow to authenticate using

Microsoft Entra (OIDC authentication). For alternative authentication options, see [Deploy to App Service using GitHub Actions](#).

The screenshot shows the Azure portal interface for configuring GitHub Actions. Key elements include:

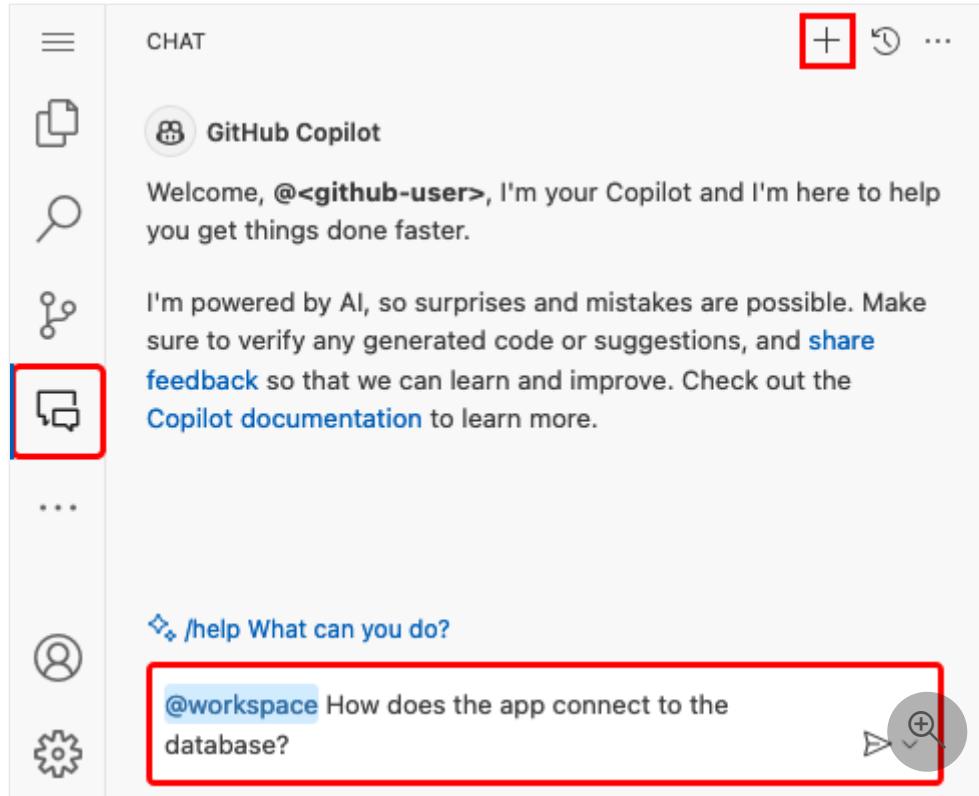
- Save** button highlighted with a red box.
- Logs** and **FTPS credentials** tabs.
- A message bar: "You're now in the production slot, which is not recommended for setting up CI/CD. Learn more" with a close button.
- Source ***: GitHub (highlighted with a red box).
- Building with GitHub Actions. [Change provider](#).
- GitHub** section:
 - Signed in as: <github-alias> [Change Account](#)
 - Organization *: [redacted]
 - Repository *: msdocs-jboss-mysql-sample-app
 - Branch *: starter-no-infra
- Build** section:
 - Runtime stack: Java
 - Version: 17.0
 - Java web server stack: JBoss EAP
- Authentication settings** section:
 - Select how you want your GitHub Action workflow to authenticate to Azure. If you choose user-assigned identity, the identity selected will be federated with GitHub as an authorized client and given write permissions on the app. [Learn more](#)
 - Authentication type ***:
 - User-assigned identity (highlighted with a red box)
 - Basic authentication

Step 3: Back in the GitHub codespace of your sample fork, run `git pull origin starter-no-infra`. This pulls the newly committed workflow file into your codespace. You can modify it according to your needs at `.github/workflows/starter-no-infra_msdocs-jboss-mysql.yml`.

This screenshot shows the GitHub Codespaces interface for a Java EE 10 web application. The left sidebar displays the project structure under 'EXPLORER'. The main area shows a preview of the README.md file, which contains instructions for deploying a JBoss EAP web app with MySQL in Azure. Below the preview, a section titled 'Run the sample' explains how to use a dev container configuration to develop and deploy the app. The bottom half of the screen shows a terminal window where a user is running a 'git pull' command. The command output shows the cloning of the repository from GitHub, including the fetching of the 'starter-no-infra' branch and updating the local repository. The terminal interface includes tabs for PROBLEMS, OUTPUT, DEBUG CONSOLE, TERMINAL, PORTS, AZURE, and COMMENTS, along with various icons for file operations.

Step 4 (Option 1: with GitHub Copilot):

1. Start a new chat session by clicking the **Chat** view, then clicking **+**.
2. Ask, "*@workspace How does the app connect to the database?*" Copilot might give you some explanation about the `java:jboss/MySQLDS` data source and how it's configured.
3. Say, "*The data source in JBoss in Azure uses the JNDI name `javajboss/env/jdbc/AZURE_MYSQL_CONNECTIONSTRING_DS`.*" Copilot might give you a code suggestion similar to the one in the **Option 2: without GitHub Copilot** steps below and even tell you to make the change in the class. GitHub Copilot doesn't give you the same response every time, you might need to ask more questions to fine-tune its response. For tips, see [What can I do with GitHub Copilot in my codespace?](#).



Step 4 (Option 2: without GitHub Copilot):

1. Open `src/main/resources/META-INF/persistence.xml` in the explorer. When the application starts, it loads the database settings in this file.
2. Change the value of `<jta-data-source>` from `java:jboss/MySQLDS` to `java:jboss/env/jdbc/AZURE_MYSQL_CONNECTIONSTRING_DS`, which is the data source you found with JBoss CLI earlier in the SSH shell.

The screenshot shows the Eclipse IDE interface. The left side has an Explorer view showing a project structure with files like `.devcontainer`, `.github`, `src/main`, and `persistence.xml` (which is selected). The right side is a code editor with the XML file `persistence.xml` open. The code is as follows:

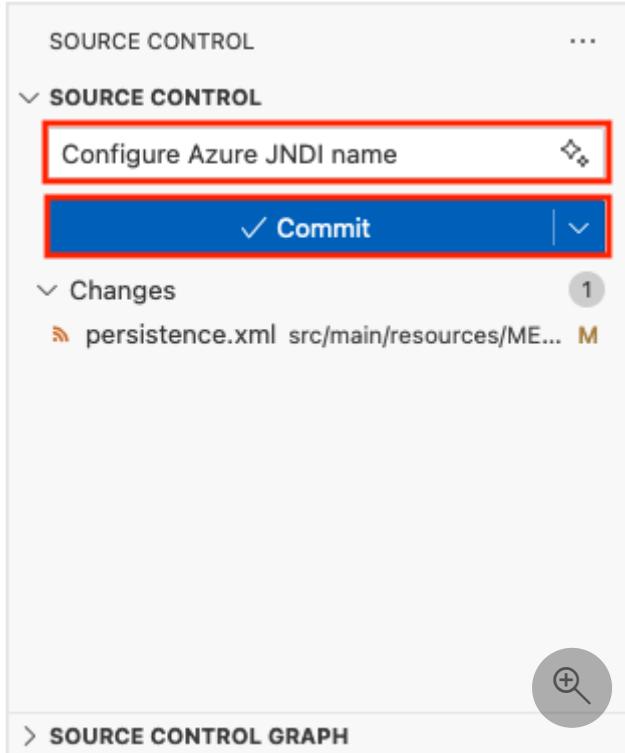
```

<?xml version="1.0" encoding="UTF-8"?>
<persistence version="2.1"
    xmlns="http://xmlns.jcp.org/xml/ns/persistence"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/persistence http://xmlns.jcp.org/xml/ns/persistence_2_1.xsd">
    <persistence-unit name="defaultPU">
        <jta-data-source>java:jboss/env/jdbc/AZURE_MYSQL_CONNECTIONSTRING_DS</jta-data-source>
        <properties>
            <property name="jakarta.persistence.schema-generation.database.action" value="create" />
            <property name="openjpa.jdbc.SynchronizeMappings" value="buildSchema" />
            <property name="eclipselink.logging.level.sql" value="FINE" />
            <property name="eclipselink.logging.parameters" value="true" />
            <property name="hibernate.show_sql" value="true" />
        </properties>
    </persistence-unit>
</persistence>

```

Step 5:

1. Select the **Source Control** extension.
2. In the textbox, type a commit message like `Configure Azure JNDI name`.
3. Select **Commit**, then confirm with **Yes**.
4. Select **Sync changes 1**, then confirm with **OK**.



Step 6: Back in the Deployment Center page in the Azure portal:

1. Select **Logs**. A new deployment run is already started from your committed changes.
2. In the log item for the deployment run, select the **Build/Deploy Logs** entry with the latest timestamp.

Time	Com...	Logs	Commit Author	Status	Message
Tuesday, October 29, 2024 (10)					
10/29/2024, 3:51:00...	f7e1b46	App Logs	N/A	Success (Active)	OneDeploy
10/29/2024, 3:50:00...	2f2f3c2	Build/Deploy Lo...	Icephas	Success	Configure A
10/29/2024, 3:37:00...	135eec2	App Logs	N/A	Success	OneDeploy
10/29/2024, 3:37:00...	a69b3a4	Build/Deploy Lo...	Icephas	Success	Add or upd build and config

Step 7: You're taken to your GitHub repository and see that the GitHub action is running. The workflow file defines two separate stages, build and deploy. Wait for the GitHub run to show a status of **Complete**. It takes about 5 minutes.

Build and deploy WAR app to Azure Web App - msdocs-jboss-mysql

Configure Azure JNDI name #2

Re-run all jobs

Summary

Triggered via push 2 minutes ago

Icephas pushed → 45fc372 [starter-no-infra](#)

Status: Success | Total duration: 1m 25s

Artifacts: 1

starter-no-infra_msdocs-jboss-mysql.yml

on: push

```

graph LR
    build[build] -- "19s" --> deploy[deploy]
    subgraph "starter-no-infra_msdocs-jboss-mysql.yml"
        build
        deploy
        link(( ))
    end
    link -- "47s" --> deploy
    https://msdocs-jboss-mysql-eubeg8he...

```

Having issues? Check the [Troubleshooting section](#).

6. Browse to the app

Step 1: In the App Service page:

1. From the left menu, select **Overview**.
2. In **Default domain**, select the URL of your app.

Search

Overview

Activity log

Access control (IAM)

Tags

Diagnose and solve problems

Microsoft Defender for Cloud

Events (preview)

Better Together (preview)

Deployment

Deployment slots

Deployment Center

Settings

Environment variables

Configuration

Resource group (move)
msdocs-jboss-mysql_group

Default domain
msdocs-jboss-mysql-g2 cwd2dvcxf5ey...

Status
Running

Location (move)
Canada Central

Subscription (move)
msdocs-jboss-mysql (P0v3: 1)

Operating System
Linux

Health Check
Not Configured

GitHub Project
<https://github.com/<github-alias>/ms...>

Tags (edit)
Add tags

Properties Monitoring Logs Capabilities Notifications Recommendations

Web app

Step 2: Add a few tasks to the list. Congratulations, you're running a web app in Azure App Service, with secure connectivity to Azure Database for MySQL.

New Task

Task

Enter task description

Add Task

Current Tasks

Create JBoss app and MySQL in Azure

Delete

Deploy data-driven app

Delete

That's it!

Delete

Having issues? Check the [Troubleshooting section](#).

7. Stream diagnostic logs

Azure App Service captures all messages output to the console to help you diagnose issues with your application. The sample application includes standard Log4j logging statements to demonstrate this capability, as shown in the following snippet:

```
Java

private static final Logger logger =
Logger.getLogger(MethodHandles.lookup().lookupClass().getName());

@PersistenceContext
private EntityManager entityManager;

public List<Task> getAllTasks() {
    logger.log(Level.INFO, "Finding all tasks. ");

    return this.entityManager.createNamedQuery("findAllTasks",
Task.class).getResultList();
}
```

In the App Service page, from the left menu, select **Log stream**. You see the logs for your app, including platform logs and logs from inside the container.



Learn more about logging in Java apps in the series on [Enable Azure Monitor OpenTelemetry for .NET, Node.js, Python, and Java applications](#).

Having issues? Check the [Troubleshooting section](#).

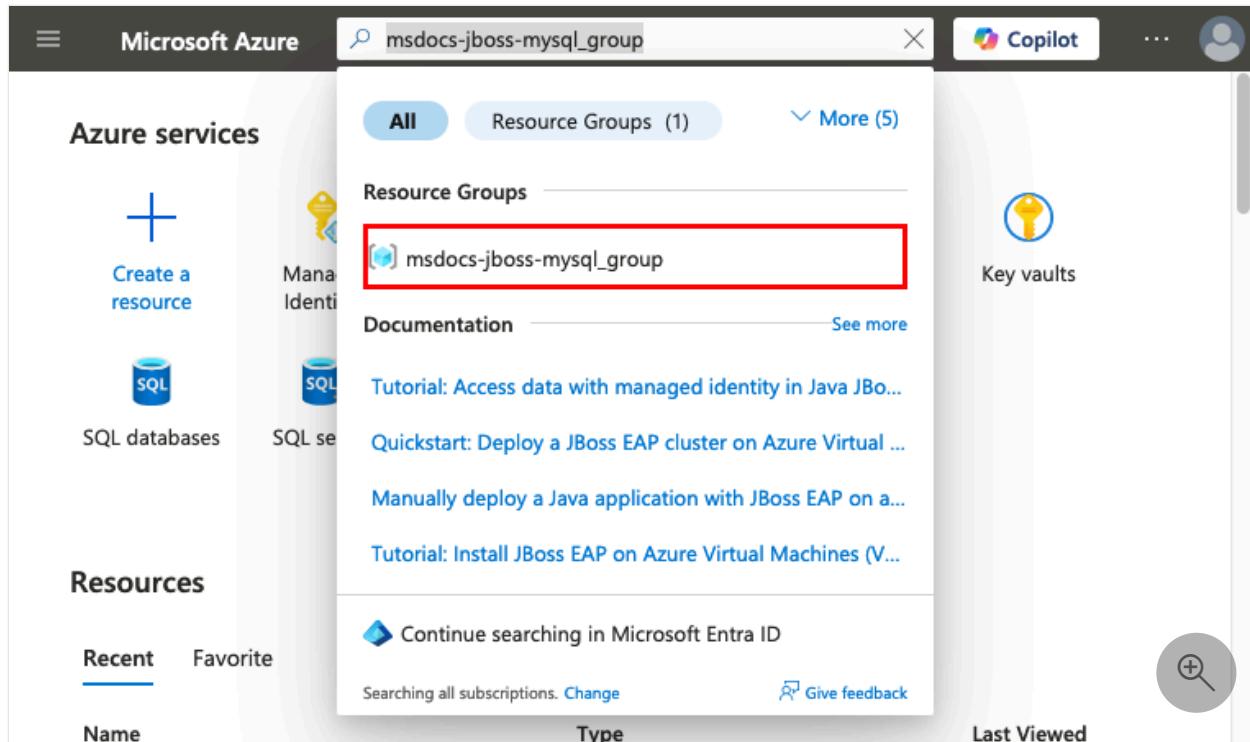
8. Clean up resources

When you're finished, you can delete all of the resources from your Azure subscription by deleting the resource group.

Step 1: In the search bar at the top of the Azure portal:

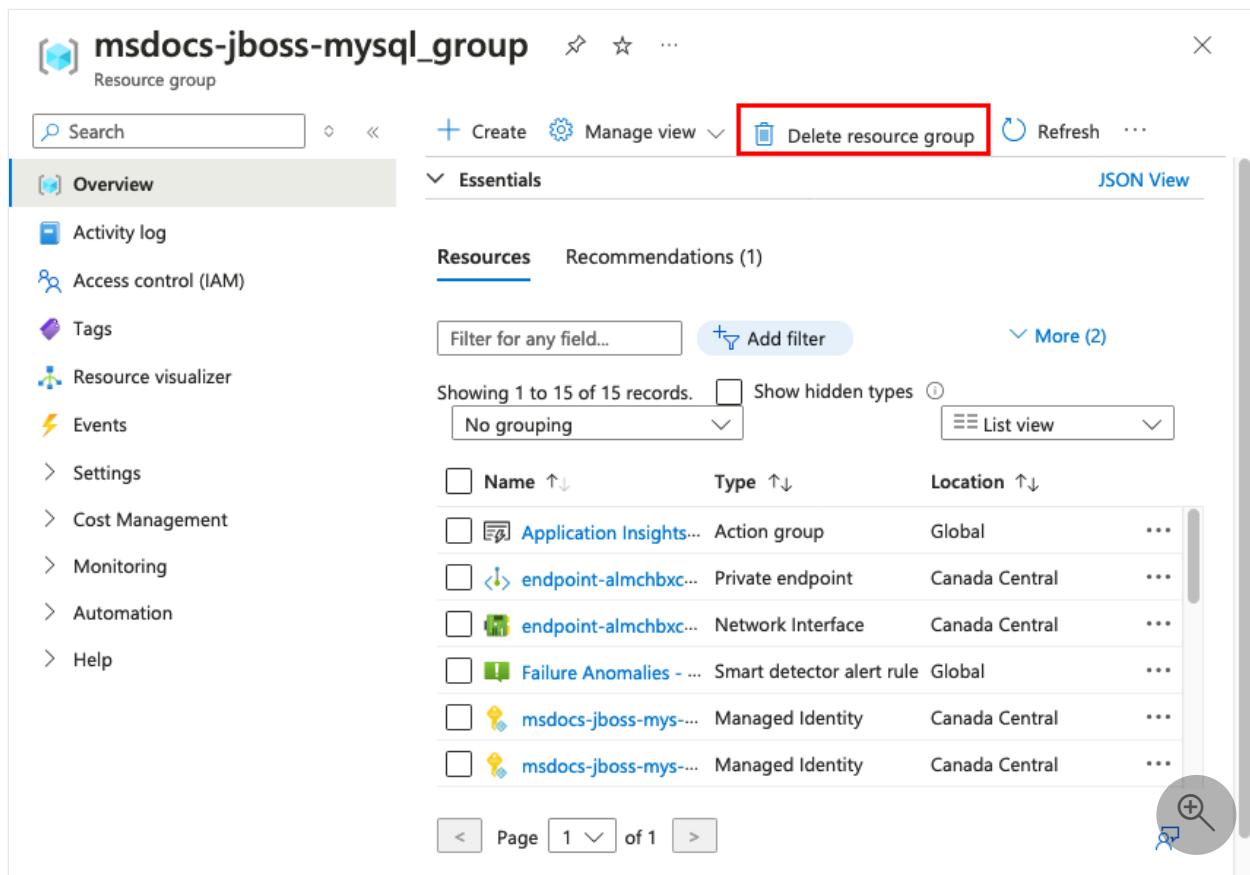
1. Enter the resource group name *msdocs-jboss-mysql_group*.

2. Select the resource group.



The screenshot shows the Microsoft Azure portal interface. In the top navigation bar, the search bar contains the text "msdocs-jboss-mysql_group". Below the search bar, there are three tabs: "All", "Resource Groups (1)", and "More (5)". The "Resource Groups (1)" tab is selected. A single result, "msdocs-jboss-mysql_group", is listed under "Resource Groups" and is highlighted with a red box. To the right of the search results, there is a sidebar with a "Key vaults" section, which includes a key icon and the text "Key vaults". At the bottom of the search results, there is a "Continue searching in Microsoft Entra ID" button and a "Give feedback" link.

Step 2: In the resource group page, select Delete resource group.



The screenshot shows the "msdocs-jboss-mysql_group" Resource Group page. The top navigation bar includes a search bar, a "Create" button, a "Manage view" dropdown, and a "Delete resource group" button, which is highlighted with a red box. Below the navigation bar, there is a sidebar with links like "Overview", "Activity log", "Access control (IAM)", "Tags", "Resource visualizer", "Events", "Settings", "Cost Management", "Monitoring", "Automation", and "Help". The main content area is titled "Resources" and shows a list of 15 records. The list includes items such as "Application Insights...", "Private endpoint", "Network Interface", "Smart detector alert rule", "Managed Identity", and "Managed Identity". Each item has a delete icon next to it. At the bottom of the list, there are buttons for "Page", "1", "of 1", and "More (2)".

Step 3:

1. Confirm your deletion by typing the resource group name.

2. Select Delete.

3. Confirm with Delete again.

Delete a resource group

The following resource group and all its dependent resources will be permanently deleted.

Resource group to be deleted

 msdocs-jboss-mysql_group 

Dependent resources to be deleted (16)

All dependent resources, including hidden types, are shown

Name	Resource type
 Application Insights Smart Detection	Action group
 endpoint-almchbxcgqdf	Private endpoint
 endpoint-almchbxcgqdf.nic.bc336b1c-37fb-4	Network interface
 Failure Anomalies - msdocs-jboss-mysql	Smart detector alert rule
 link-almchbxcgqdf (privatelink.mysql.database...)	Microsoft.Network/privateDnsZone...

Enter resource group name to confirm deletion *

Delete **Cancel** 

Troubleshooting

- I see the error 'not entitled to use the Bring Your Own License feature' in the creation wizard.
- The portal deployment view for Azure Database for MySQL Flexible Server shows a Conflict status.
- The Create connection dialog shows a Create On Cloud Shell button but it's not enabled.
- My app failed to start, and I see 'Access denied for user... (using password: NO)' in the logs.
- The deployed sample app doesn't show the tasks list app.
- I see a "Table 'Task' already exists" error in the diagnostic logs.

I see the error 'not entitled to use the Bring Your Own License feature' in the creation wizard.

If you see the error: `The subscription '701ea799-fb46-4407-bb67-9cbcf289f1c7' is not entitled to use the Bring Your Own License feature when creating the application`, it means that you selected **Red Hat JBoss EAP 7/8 BYO License** in Java web server stack but haven't set up your Azure account in Red Hat Cloud Access or don't have an active JBoss EAP license in Red Hat Cloud Access.

The portal deployment view for Azure Database for MySQL Flexible Server shows a Conflict status.

Depending on your subscription and the region you select, you might see the deployment status for Azure Database for MySQL Flexible Server to be `Conflict`, with the following message in Operation details:

`InternalServerError: An unexpected error occurred while processing the request.`

This error is most likely caused by a limit on your subscription for the region you select. Try choosing a different region for your deployment.

The Create connection dialog shows a Create On Cloud Shell button but it's not enabled.

You might also see an error message in the dialog: `The database server is in Virtual Network and Cloud Shell can't connect to it. Please copy the commands and execute on an environment which can connect to the database server in Virtual Network.`

The service connector automation needs network access to the MySQL server. Look in the networking settings of your MySQL server resource and make sure **Allow public access to this resource through the internet using a public IP address** is selected at a minimum. Service Connector can take it from there.

If you don't see this checkbox, you might have created the deployment using the [Web App + Database wizard ↗](#) instead, and the deployment locks down all public network access to the MySQL server. There's no way to modify the configuration. Since app's Linux container can access MySQL through the virtual network integration, you could install Azure CLI in the app's SSH session and run the supplied Cloud Shell commands there.

The deployed sample app doesn't show the tasks list app.

If you see the JBoss splash page instead of the tasks list app, App Service is most likely still loading the updated container from your most recent code deployment. Wait a few minutes and refresh the page.

My app failed to start, and I see 'Access denied for user... (using password: NO)' in the logs.

This error is most likely because you didn't add the passwordless authentication plugin to the connection string (see the Java sample code for [Integrate Azure Database for MySQL with Service Connector](#)). Change the MySQL connection string by following the instructions in [3. Create a passwordless connection](#).

I see a "Table 'Task' already exists" error in the diagnostic logs.

You can ignore this Hibernate error because it indicates that the application code is connected to the MySQL database. The application is configured to create the necessary tables when it starts (see *src/main/resources/META-INF/persistence.xml*). When the application starts the first time, it should create the tables successfully, but on subsequent restarts, you would see this error because the tables already exist.

Frequently asked questions

- [How much does this setup cost?](#)
- [How do I connect to the MySQL server behind the virtual network with other tools?](#)
- [How do I get a valid access token for the MySQL connection using Microsoft Entra authentication?](#)
- [How does local app development work with GitHub Actions?](#)
- [I don't have permissions to create a user-assigned identity](#)
- [What can I do with GitHub Copilot in my codespace?](#)

How much does this setup cost?

Pricing for the created resources is as follows:

- The App Service plan is created in **P0v3** tier and can be scaled up or down. See [App Service pricing ↗](#).
- The MySQL flexible server is created in **D2ds** tier and can be scaled up or down. See [Azure Database for MySQL pricing ↗](#).
- The Azure Cache for Redis is created in **Basic** tier with the minimum cache size. There's a small cost associated with this tier. You can scale it up to higher

performance tiers for higher availability, clustering, and other features. See [Azure Cache for Redis pricing](#).

- The virtual network doesn't incur a charge unless you configure extra functionality, such as peering. See [Azure Virtual Network pricing](#).
- The private DNS zone incurs a small charge. See [Azure DNS pricing](#).

How do I connect to the MySQL server behind the virtual network with other tools?

In this tutorial, the App Service app is already has network connectivity to the MySQL server and can authenticate with Microsoft Entra by using its system-assigned managed identity. You can connect to MySQL directly from within the app container by running the following commands in the SSH session (get your <server>, <user>, and <database> values from the `AZURE_SQL_CONNECTIONSTRING` app setting):

Bash

```
apt-get update
apt-get install curl less mysql-client jq -y
mysql -h <server> --user <user> --database <database> --enable-cleartext-
plugin --password=`curl "${IDENTITY_ENDPOINT}?resource=https://osrdbms-
aad.database.windows.net&api-version=2019-08-01" -H "X-IDENTITY-HEADER:
$IDENTITY_HEADER" -s | jq -r '.access_token'`
```

A few considerations:

- The tools you install in the SSH session don't persist across app restarts.
- If you followed the portal steps and configured MySQL using your Microsoft Entra user as the administrator, you can connect to MySQL using the Microsoft Entra user.
- To connect from a desktop tool like MySQL Workbench, your machine must be within the virtual network, such as an Azure VM deployed into the same virtual network. You must also configure authentication separately, either with a managed identity or with a Microsoft Entra user.
- To connect from a machine in an on-premises network that has a [site-to-site VPN](#) connection with the Azure virtual network, you can't configure authentication with a managed identity, but you can configure authentication by using a Microsoft Entra user.
- You can also [integrate Azure Cloud Shell](#) and connect using Azure CLI or the MySQL CLI. To authenticate, you can configure a Microsoft Entra user.

How do I get a valid access token for the MySQL connection using Microsoft Entra authentication?

For a Microsoft Entra user, a system-assigned managed identity, or a user-assigned managed identity that's authorized to access the MySQL database, Azure CLI can help you generate an access token. In case of a managed identity, the identity must be configured on the App Service app or VM where you run Azure CLI.

Azure CLI

```
# Sign in as a Microsoft Entra user
az login
# Sign in as the system-assigned managed identity
az login --identity
# Sign in as a user-assigned managed identity
az login --identity --username <client-id-of-user-assigned-identity>

# Get an access token
az account get-access-token --resource-type oss-rdbms
```

If you want, you can also use the [az mysql flexible-server connect](#) Azure CLI command to connect to MySQL. When prompted, use the access token as the password.

Azure CLI

```
az mysql flexible-server connect -n <server-name-only> -u <user> -d
<database> --interactive
```

For more information, see:

- [How to use managed identities for App Service and Azure Functions](#)
- [Authenticate to Azure using Azure CLI](#)
- [Connect to Azure Database for MySQL Flexible Server using Microsoft Entra ID](#)

How does local app development work with GitHub Actions?

Using the autogenerated workflow file from App Service as an example, each `git push` kicks off a new build and deployment run. From a local clone of the GitHub repository, you make the desired updates and push to GitHub. For example:

terminal

```
git add .
git commit -m "<some-message>"
git push origin starter-no-infra
```

I don't have permissions to create a user-assigned identity

See [Set up GitHub Actions deployment from the Deployment Center](#).

What can I do with GitHub Copilot in my codespace?

You might notice that the GitHub Copilot chat view was already there for you when you created the codespace. For your convenience, we include the GitHub Copilot chat extension in the container definition (see `.devcontainer/devcontainer.json`). However, you need a [GitHub Copilot account](#) (30-day free trial available).

A few tips for you when you talk to GitHub Copilot:

- In a single chat session, the questions and answers build on each other and you can adjust your questions to fine-tune the answer you get.
- By default, GitHub Copilot doesn't have access to any file in your repository. To ask questions about a file, open the file in the editor first.
- To let GitHub Copilot have access to all of the files in the repository when preparing its answers, begin your question with `@workspace`. For more information, see [Use the @workspace agent](#).
- In the chat session, GitHub Copilot can suggest changes and (with `@workspace`) even where to make the changes, but it's not allowed to make the changes for you. It's up to you to add the suggested changes and test it.

Here are some other things you can say to fine-tune the answer you get:

- Change this code to use the data source `jdbc/AZURE_MYSQL_CONNECTIONSTRING_DS`.
- Some imports in your code are using `javax` but I have a Jakarta app.
- I want this code to run only if the environment variable `AZURE_MYSQL_CONNECTIONSTRING` is set.
- I want this code to run only in Azure App Service and not locally.

Next steps

- [Azure for Java Developers](#)

Learn more about running Java apps on App Service in the developer guide.

[Configure a Java app in Azure App Service](#)

Learn how to secure your app with a custom domain and certificate.

Secure with custom domain and certificate

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

Configure an App Service app

Article • 10/03/2024

ⓘ Note

Starting June 1, 2024, all newly created App Service apps will have the option to generate a unique default hostname using the naming convention `<app-name>-<random-hash>. <region>.azurewebsites.net`. Existing app names will remain unchanged.

Example: `myapp-ds27dh7271aah175.westus-01.azurewebsites.net`

For further details, refer to [Unique Default Hostname for App Service Resource ↗](#).

This article explains how to configure common settings for web apps, mobile back end, or API app. For Azure Functions, see [App settings reference for Azure Functions](#).

Configure app settings

ⓘ Note

- App settings names can only contain letters, numbers (0-9), periods ("."), and underscores ("_")
- Special characters in the value of an App Setting must be escaped as needed by the target OS

For example to set an environment variable in App Service Linux with the value `"pa$$w0rd\"` the string for the app setting should be: `"pa\$\$w0rd\\\"`

In App Service, app settings are variables passed as environment variables to the application code. For Linux apps and custom containers, App Service passes app settings to the container using the `--env` flag to set the environment variable in the container. In either case, they're injected into your app environment at app startup. When you add, remove, or edit app settings, App Service triggers an app restart.

For ASP.NET and ASP.NET Core developers, setting app settings in App Service are like setting them in `<appSettings>` in `Web.config` or `appsettings.json`, but the values in App Service override the ones in `Web.config` or `appsettings.json`. You can keep development settings (for example, local MySQL password) in `Web.config` or `appsettings.json` and

production secrets (for example, Azure MySQL database password) safely in App Service. The same code uses your development settings when you debug locally, and it uses your production secrets when deployed to Azure.

Other language stacks, likewise, get the app settings as environment variables at runtime. For language-stack specific steps, see:

- [ASP.NET Core](#)
- [Node.js](#)
- [PHP](#)
- [Python](#)
- [Java](#)
- [Custom containers](#)

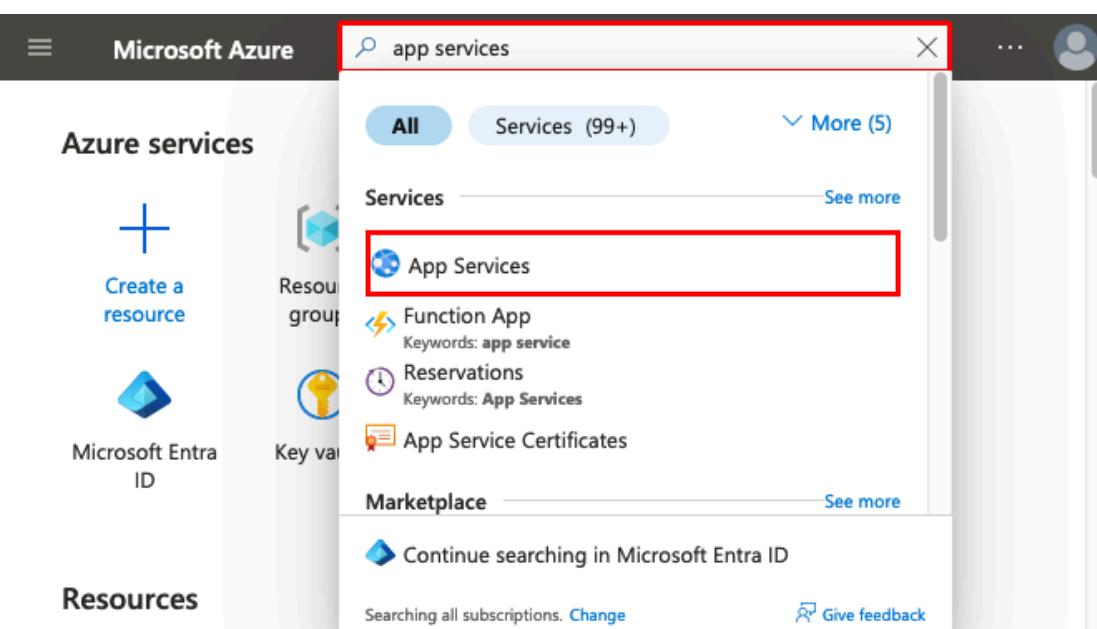
App settings are always encrypted when stored (encrypted-at-rest).

ⓘ Note

If you store secrets in app settings, consider using [Key Vault references](#). If your secrets are for connectivity to back-end resources, consider more secure connectivity options that don't require secrets at all. For more information, see [Secure connectivity to Azure services and databases from Azure App Service](#).

Azure portal

1. In the [Azure portal](#), search for and select **App Services**, and then select your app.



The screenshot shows the Microsoft Azure portal interface. At the top, there's a search bar with the text "app services". Below the search bar, there are three tabs: "All", "Services (99+)", and "More (5)". Under the "Services" tab, there's a list of items. The "App Services" item is highlighted with a red rectangle. Other items listed include "Function App", "Reservations", "App Service Certificates", and "Marketplace". On the left side of the portal, there's a sidebar with sections for "Azure services" (including "Create a resource", "Resource groups", "Microsoft Entra ID", and "Key vault"), "Resources", and "Marketplace".

2. In the app's left menu, select **Environment variables** > **App settings**.

The screenshot shows the Azure portal interface for managing app settings. The left sidebar has a tree view with items like Deployment Center, Performance, Load Testing, Settings, Configuration, Authentication, Application Insights, Identity, and Backups. The 'Settings' node is expanded, and the 'Environment variables' item under it is selected and highlighted with a red box. The main content area is titled 'App settings' and contains a table for managing environment variables. The table has columns for 'Name', 'Value', and 'Deployment s'. At the top of this section are buttons for 'Search', 'Add', 'Refresh', 'Show values', and 'Advanced edit'. At the bottom are 'Apply' and 'Discard' buttons, along with a 'Send us your feedback' link.

By default, values for app settings are hidden in the portal for security. To see a hidden value of an app setting, select its **Value** field. To see the hidden values of all app settings, select the **Show values** button.

3. To add a new app setting, select **Add**. To edit a setting, click the setting.
4. In the dialog, you can [stick the setting to the current slot](#).

(!) Note

In a default Linux app service or a custom Linux container, any nested JSON key structure in the app setting name like `ApplicationInsights:InstrumentationKey` needs to be configured in App Service as `ApplicationInsights__InstrumentationKey` for the key name. In other words, any `:` should be replaced by `_` (double underscore). Any periods in the app setting name will be replaced with a `_` (single underscore).

5. When finished, select **Apply**. Don't forget to select **Apply** back in the **Environment variables** page.

Edit app settings in bulk

Azure portal

Select the **Advanced edit** button. Edit the settings in the text area. When finished, select **OK**. Don't forget to select **Apply** back in the **Environment variables** page.

App settings have the following JSON formatting:

JSON

```
[  
  {  
    "name": "<key-1>",  
    "value": "<value-1>",  
    "slotSetting": false  
  },  
  {  
    "name": "<key-2>",  
    "value": "<value-2>",  
    "slotSetting": false  
  },  
  ...  
]
```

Configure connection strings

ⓘ Note

Consider more secure connectivity options that don't require connection secrets at all. For more information, see [Secure connectivity to Azure services and databases from Azure App Service](#).

For ASP.NET and ASP.NET Core developers, setting connection strings in App Service are like setting them in `<connectionStrings>` in *Web.config*, but the values you set in App Service override the ones in *Web.config*. You can keep development settings (for example, a database file) in *Web.config* and production secrets (for example, SQL Database credentials) safely in App Service. The same code uses your development settings when you debug locally, and it uses your production secrets when deployed to Azure.

For other language stacks, it's better to use [app settings](#) instead, because connection strings require special formatting in the variable keys in order to access the values.

ⓘ Note

There is one case where you may want to use connection strings instead of app settings for non-.NET languages: certain Azure database types are backed up along with the app *only* if you configure a connection string for the database in your App

Service app. For more information, see [Create a custom backup](#). If you don't need this automated backup, then use app settings.

At runtime, connection strings are available as environment variables, prefixed with the following connection types:

- SQLServer: `SQLCONNSTR_`
- MySQL: `MYSQLCONNSTR_`
- SQLAzure: `SQLAZURECONNSTR_`
- Custom: `CUSTOMCONNSTR_`
- PostgreSQL: `POSTGRESQLCONNSTR_`
- Notification Hub: `NOTIFICATIONHUBCONNSTR_`
- Service Bus: `SERVICEBUSCONNSTR_`
- Event Hub: `EVENTHUBCONNSTR_`
- Document DB: `DOCDBCONNSTR_`
- Redis Cache: `REDISCACHECONNSTR_`

Note

.NET apps targeting PostgreSQL, Notification Hub, Service Bus, Event Hub, Document Db and Redis Cache should set the connection string to **Custom** as workaround for a [known issue in .NET EnvironmentVariablesConfigurationProvider](#)

For example, a MySQL connection string named *connectionstring1* can be accessed as the environment variable `MYSQLCONNSTR_connectionString1`. For language-stack specific steps, see:

- [ASP.NET Core](#)
- [Node.js](#)
- [PHP](#)
- [Python](#)
- [Java](#)
- [Custom containers](#)

Connection strings are always encrypted when stored (encrypted-at-rest).

Note

Connection strings can also be resolved from [Key Vault](#) using [Key Vault references](#).

1. In the [Azure portal](#), search for and select App Services, and then select your app.

The screenshot shows the Microsoft Azure portal's search interface. At the top, there is a search bar with the text "app services". Below the search bar, there are two tabs: "All" (selected) and "Services (99+)". To the right of these tabs is a "More (5)" link. The main area is titled "Services" and contains several items: "App Services" (which is highlighted with a red box), "Function App", "Reservations", "App Service Certificates", and a "Marketplace" section. On the left side of the screen, there is a sidebar with sections for "Azure services" (including "Create a resource", "Resource groups", "Microsoft Entra ID", and "Key vault"), "Resources" (with a "Load Testing" icon), and "Settings" (with "Environment variables" highlighted with a red box). At the bottom of the search results, there is a note "Searching all subscriptions. Change" and a "Give feedback" link.

2. In the app's left menu, select Environment variables > Connection strings.

The screenshot shows the "App settings" page for an Azure app. The left sidebar has sections for "Deployment Center", "Performance" (with "Load Testing" listed), and "Settings" (with "Environment variables" highlighted with a red box). The main content area has tabs for "App settings" and "Connection strings" (which is highlighted with a red box). Below the tabs is a search bar and a row of buttons: "Add", "Refresh", "Show values", and "...". A table follows, with columns for "Name", "Value", and "Deployment slot settir". At the bottom of the table are "Apply" and "Discard" buttons, and a "Send us your feedback" link.

By default, values for connection strings are hidden in the portal for security. To see a hidden value of a connection string, select its **Value** field. To see the hidden values of all connection strings, select the **Show value** button.

3. To add a new connection string, select **Add**. To edit a connection string, select the connection string.
4. In the dialog, you can [stick the connection string to the current slot](#).

- When finished, select **Apply**. Don't forget to select **Apply** back in the **Environment variables** page.

Edit connection strings in bulk

Azure portal

Select the **Advanced edit** button. Edit the connection strings in the text area. When finished, select **Apply**. Don't forget to select **Apply** back in the **Environment variables** page.

Connection strings have the following JSON formatting:

JSON

```
[  
  {  
    "name": "name-1",  
    "value": "conn-string-1",  
    "type": "SQLServer",  
    "slotSetting": false  
  },  
  {  
    "name": "name-2",  
    "value": "conn-string-2",  
    "type": "PostgreSQL",  
    "slotSetting": false  
  },  
  ...  
]
```

Configure language stack settings

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

Configure general settings

In the [Azure portal](#), search for and select **App Services**, and then select your app.

In the app's left menu, select **Configuration > General settings**.

The screenshot shows the Azure App Service configuration interface. On the left, there is a sidebar with various options: Deployment Center, Performance, Load Testing, Settings, Environment variables, Configuration (which is highlighted with a red box), Authentication, Application Insights, Identity, and Backups. The main area has a header with a search bar, refresh, save, discard, and leave feedback buttons. A message at the top states: "Custom Error pages requires a premium App Service Plan." Below this, a blue bar says: "View and edit your application settings and connection strings from Environment variables. Click here to go to Environment Variables menu". The "General settings" tab is selected and highlighted with a blue border. Other tabs include "Default documents", "Path mappings", and "...". Under "Stack settings", there is a dropdown for "Stack" set to "Java" and another dropdown for "Java version" set to "Java 17".

Here, you can configure some common settings for the app. Some settings require you to [scale up to higher pricing tiers](#).

- **Stack settings:** The software stack to run the app, including the language and SDK versions.

For Linux apps, you can select the language runtime version and set an optional **Startup command** or a startup command file.

The screenshot shows the "Stack settings" configuration page for a Python app. It includes fields for "Stack" (set to "Python"), "Major version" (set to "Python 3"), "Minor version" (set to "Python 3.8"), and a "Startup Command" input field which is currently empty. A note below the command field says: "Provide an optional startup command that will be run as part of container startup. [Learn more](#)".

- **Platform settings:** Lets you configure settings for the hosting platform, including:
 - **Platform bitness:** 32-bit or 64-bit. For Windows apps only.
 - **FTP state:** Allow only FTPS or disable FTP altogether.
 - **HTTP version:** Set to 2.0 to enable support for [HTTPS/2](#) protocol.

Note

Most modern browsers support HTTP/2 protocol over TLS only, while non-encrypted traffic continues to use HTTP/1.1. To ensure that client browsers connect to your app with HTTP/2, secure your custom DNS name. For more information, see [Secure a custom DNS name with a TLS/SSL binding in Azure App Service](#).

- **Web sockets:** For [ASP.NET SignalR](#) or [socket.io](#), for example.
- **Always On:** Keeps the app loaded even when there's no traffic. When **Always On** isn't turned on (default), the app is unloaded after 20 minutes without any incoming requests. The unloaded app can cause high latency for new requests because of its warm-up time. When **Always On** is turned on, the front-end load balancer sends a GET request to the application root every five minutes. The continuous ping prevents the app from being unloaded.

Always On is required for continuous WebJobs or for WebJobs that are triggered using a CRON expression.
- **Session affinity:** In a multi-instance deployment, ensure that the client is routed to the same instance for the life of the session. You can set this option to **Off** for stateless applications.
- **Session affinity proxy:** Session affinity proxy can be turned on if your app is behind a reverse proxy (like Azure Application Gateway or Azure Front Door) and you are using the default host name. The domain for the session affinity cookie will align with the forwarded host name from the reverse proxy.
- **HTTPS Only:** When enabled, all HTTP traffic is redirected to HTTPS.
- **Minimum TLS version:** Select the minimum TLS encryption version required by your app.
- **Debugging:** Enable remote debugging for [ASP.NET](#), [ASP.NET Core](#), or [Node.js](#) apps. This option turns off automatically after 48 hours.
- **Incoming client certificates:** require client certificates in [mutual authentication](#).

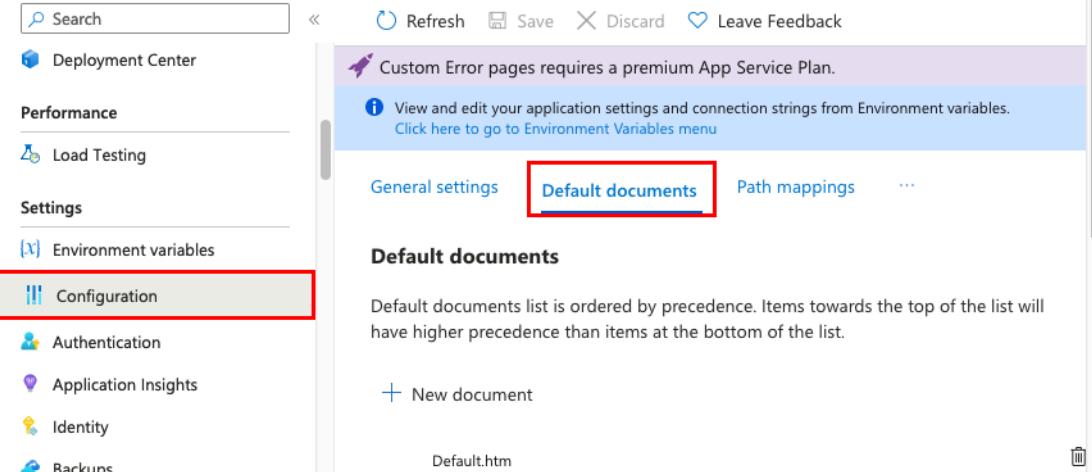
Configure default documents

This setting is only for Windows apps.

The default document is the web page that's displayed at the root URL of an App Service app. The first matching file in the list is used. If the app uses modules that route based on URL instead of serving static content, there's no need for default documents.

Azure portal

1. In the [Azure portal](#), search for and select **App Services**, and then select your app.
2. In the app's left menu, select **Configuration > Default documents**.



3. To add a default document, select **New document**. To remove a default document, select **Delete** to its right.

Map a URL path to a directory

By default, App Service starts your app from the root directory of your app code. But certain web frameworks don't start in the root directory. For example, [Laravel](#) starts in the `public` subdirectory. Such an app would be accessible at

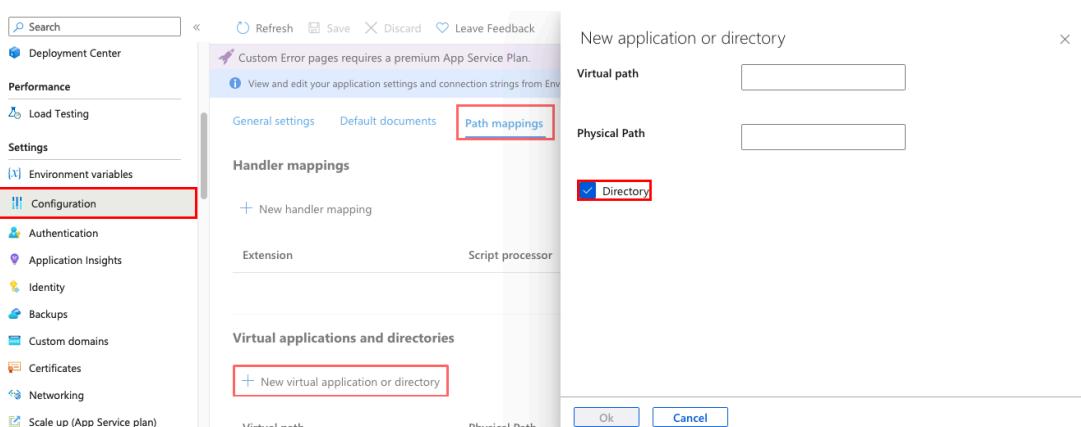
`http://contoso.com/public`, for example, but you typically want to direct `http://contoso.com` to the `public` directory instead. If your app's startup file is in a different folder, or if your repository has more than one application, you can edit or add virtual applications and directories.

Important

Virtual directory to a physical path feature is only available on Windows apps.

Azure portal

1. In the [Azure portal](#), search for and select **App Services**, and then select your app.
2. In the app's left menu, select **Configuration > Path mappings**
3. Select **New virtual application or directory**.
 - To map a virtual directory to a physical path, leave the **Directory** check box selected. Specify the virtual directory and the corresponding relative (physical) path to the website root (`D:\home`).
 - To mark a virtual directory as a web application, clear the **Directory** check box.



4. Select **OK**. Don't forget to select **Save** in the Configuration page.

Configure handler mappings

For Windows apps, you can customize the IIS handler mappings and virtual applications and directories. Handler mappings let you add custom script processors to handle requests for specific file extensions.

To add a custom handler:

1. In the [Azure portal](#), search for and select **App Services**, and then select your app.
2. In the app's left menu, select **Configuration > Path mappings**.

The screenshot shows the 'Configuration' page in the Azure App Service portal. The left sidebar lists various settings like Deployment Center, Performance, Load Testing, and Configuration (which is selected and highlighted with a red box). The main content area shows a message about custom error pages requiring a premium plan, followed by tabs for General settings, Default documents, Path mappings (which is also highlighted with a red box), and Error pages (preview). Below these tabs is a section titled 'Handler mappings' with a 'New handler mapping' button. A table below lists 'Extension', 'Script processor', and 'Arguments'. A note at the bottom says '(no handler mappings to display)'. At the bottom of the page is a 'Virtual applications and directories' section.

3. Select **New handler mapping**. Configure the handler as follows:

- **Extension.** The file extension you want to handle, such as `*.php` or `handler.cgi`.
- **Script processor.** The absolute path of the script processor to you. Requests to files that match the file extension are processed by the script processor. Use the path `D:\home\site\wwwroot` to refer to your app's root directory.
- **Arguments.** Optional command-line arguments for the script processor.

4. Select **OK**. Don't forget to select **Save** in the **Configuration** page.

Configure custom containers

- Configure a custom container for Azure App Service
- Add custom storage for your containerized app

Next steps

- Environment variables and app settings reference
- Configure a custom domain name in Azure App Service
- Set up staging environments in Azure App Service
- Secure a custom DNS name with a TLS/SSL binding in Azure App Service
- Enable diagnostic logs
- Scale an app in Azure App Service
- Monitoring basics in Azure App Service
- Change applicationHost.config settings with applicationHost.xdt ↗

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

Use Key Vault references as app settings in Azure App Service and Azure Functions

Article • 09/30/2024

ⓘ Note

Starting June 1, 2024, all newly created App Service apps will have the option to generate a unique default hostname using the naming convention <app-name>-<random-hash>. <region>.azurewebsites.net. Existing app names will remain unchanged.

Example: myapp-ds27dh7271aah175.westus-01.azurewebsites.net

For further details, refer to [Unique Default Hostname for App Service Resource ↗](#).

This article shows you how to use secrets from Azure Key Vault as values of [app settings](#) or [connection strings](#) in your App Service or Azure Functions apps.

[Azure Key Vault](#) is a service that provides centralized secrets management, with full control over access policies and audit history. When an app setting or connection string is a key vault reference, your application code can use it like any other app setting or connection string. This way, you can maintain secrets apart from your app's configuration. App settings are securely encrypted at rest, but if you need secret management capabilities, they should go into a key vault.

Grant your app access to a key vault

In order to read secrets from a key vault, you need to have a vault created and give your app permission to access it.

1. Create a key vault by following the [Key Vault quickstart](#).
2. Create a [managed identity](#) for your application.

Key vault references use the app's system-assigned identity by default, but you can [specify a user-assigned identity](#).

3. Authorize [read access to secrets in your key vault](#) for the managed identity you created earlier. How you do it depends on the permissions model of your key

vault:

- **Azure role-based access control:** Assign the **Key Vault Secrets User** role to the managed identity. For instructions, see [Provide access to Key Vault keys, certificates, and secrets with an Azure role-based access control](#).
- **Vault access policy:** Assign the **Get** secrets permission to the managed identity. For instructions, see [Assign a Key Vault access policy](#).

Access network-restricted vaults

If your vault is configured with [network restrictions](#), ensure that the application has network access. Vaults shouldn't depend on the app's public outbound IPs because the origin IP of the secret request could be different. Instead, the vault should be configured to accept traffic from a virtual network used by the app.

1. Make sure the application has outbound networking capabilities configured, as described in [App Service networking features](#) and [Azure Functions networking options](#).

Linux applications that connect to private endpoints must be explicitly configured to route all traffic through the virtual network. This requirement will be removed in a forthcoming update. To configure this setting, run the following command:



The screenshot shows the Azure CLI interface. A command line window is open with the text "az webapp config set --subscription <sub> -g <group-name> -n <app-name> --generic-configurations '{"vnetRouteAllEnabled": true}'". The command "az webapp config set" is highlighted in blue, while the configuration parameters and their values are in red and green respectively.

2. Make sure that the vault's configuration allows the network or subnet that your app uses to access it.

Access vaults with a user-assigned identity

Some apps need to reference secrets at creation time, when a system-assigned identity isn't available yet. In these cases, a user-assigned identity can be created and given access to the vault in advance.

Once you have granted permissions to the user-assigned identity, follow these steps:

1. [Assign the identity](#) to your application if you haven't already.

2. Configure the app to use this identity for key vault reference operations by setting the `keyVaultReferenceIdentity` property to the resource ID of the user-assigned identity.

```
Azure CLI

identityResourceId=$(az identity show --resource-group <group-name>
--name <identity-name> --query id -o tsv)
az webapp update --resource-group <group-name> --name <app-name> --
set keyVaultReferenceIdentity=${identityResourceId}
```

This setting applies to all key vault references for the app.

Rotation

If the secret version isn't specified in the reference, the app uses the latest version that exists in the key vault. When newer versions become available, such as with a rotation event, the app automatically updates and begins using the latest version within 24 hours. The delay is because App Service caches the values of the key vault references and refetches it every 24 hours. Any configuration change to the app causes an app restart and an immediate refetch of all referenced secrets.

Source app settings from key vault

To use a key vault reference, set the reference as the value of the setting. Your app can reference the secret through its key as normal. No code changes are required.

Tip

Most app settings using key vault references should be marked as slot settings, as you should have separate vaults for each environment.

A key vault reference is of the form `@Microsoft.KeyVault({referenceString})`, where `{referenceString}` is in one of the following formats:

[+] Expand table

Reference string	Description
SecretUri= <i>secretUri</i>	The SecretUri should be the full data-plane URI of a secret in the vault, optionally including a version, e.g., <code>https://myvault.vault.azure.net/secrets/mysecret/</code> or <code>https://myvault.vault.azure.net/secrets/mysecret/ec96f02080254f109c51a1f14cdb1931</code>
VaultName= <i>vaultName</i> ;SecretName= <i>secretName</i> ;SecretVersion= <i>secretVersion</i>	The VaultName is required and is the vault name. The SecretName is required and is the secret name. The SecretVersion is optional but if present indicates the version of the secret to use.

For example, a complete reference would look like the following string:

```
@Microsoft.KeyVault(SecretUri=https://myvault.vault.azure.net/secrets/mysecret/)
```

Alternatively:

```
@Microsoft.KeyVault(VaultName=myvault;SecretName=mysecret)
```

Considerations for Azure Files mounting

Apps can use the `WEBSITE_CONTENTAZUREFILECONNECTIONSTRING` application setting to mount [Azure Files](#) as the file system. This setting has validation checks to ensure that the app can be properly started. The platform relies on having a content share within Azure Files, and it assumes a default name unless one is specified via the `WEBSITE_CONTENTSHARE` setting. For any requests that modify these settings, the platform validates if this content share exists, and attempts to create it if not. If it can't locate or create the content share, it blocks the request.

When you use key vault references in this setting, the validation check fails by default, because the secret itself can't be resolved while processing the incoming request. To avoid this issue, you can skip the validation by setting

`WEBSITE_SKIP_CONTENTSHARE_VALIDATION` to "1". This setting tells App Service to bypass all checks, and doesn't create the content share for you. You should ensure that it's created in advance.

 **Caution**

If you skip validation and either the connection string or content share are invalid, the app will be unable to start properly and will only serve HTTP 500 errors.

As part of creating the app, attempted mounting of the content share could fail due to managed identity permissions not being propagated or the virtual network integration not being set up. You can defer setting up Azure Files until later in the deployment template to accommodate this. See [Azure Resource Manager deployment](#) to learn more. In this case, App Service uses a default file system until Azure Files is set up, and files aren't copied over. You must ensure that no deployment attempts occur during the interim period before Azure Files is mounted.

Considerations for Application Insights instrumentation

Apps can use the `APPINSIGHTS_INSTRUMENTATIONKEY` or `APPLICATIONINSIGHTS_CONNECTION_STRING` application settings to integrate with [Application Insights](#). The portal experiences for App Service and Azure Functions also use these settings to surface telemetry data from the resource. If these values are referenced from Key Vault, these experiences aren't available, and you instead need to work directly with the Application Insights resource to view the telemetry. However, these values are [not considered secrets](#), so you might alternatively consider configuring them directly instead of using key vault references.

Azure Resource Manager deployment

When automating resource deployments through Azure Resource Manager templates, you may need to sequence your dependencies in a particular order to make this feature work. Be sure to define your app settings as their own resource, rather than using a `siteConfig` property in the app definition. This is because the app needs to be defined first so that the system-assigned identity is created with it and can be used in the access policy.

The following pseudo-template is an example of what a function app might look like:

JSON

```
{  
  //...  
  "resources": [  
    {  
      "type": "Microsoft.Storage/storageAccounts",  
      "name": "[variables('storageAccountName')]",  
      //...  
    },
```

```

    },
    "type": "Microsoft.Insights/components",
    "name": "[variables('appInsightsName')]",
    //...
},
{
    "type": "Microsoft.Web/sites",
    "name": "[variables('functionAppName')]",
    "identity": {
        "type": "SystemAssigned"
    },
    //...
    "resources": [
        {
            "type": "config",
            "name": "appsettings",
            //...
            "dependsOn": [
                "[resourceId('Microsoft.Web/sites',
variables('functionAppName'))]",
                "[resourceId('Microsoft.KeyVault/vaults/',
variables('keyVaultName'))]",
                "[resourceId('Microsoft.KeyVault/vaults/secrets',
variables('keyVaultName'), variables('storageConnectionStringName'))]",
                "[resourceId('Microsoft.KeyVault/vaults/secrets',
variables('keyVaultName'), variables('appInsightsKeyName'))]"
            ],
            "properties": {
                "AzureWebJobsStorage": "[concat('@Microsoft.KeyVault(SecretUri=',
reference(variables('storageConnectionStringName')).secretUriWithVersion,
'))]]",
                "WEBSITE_CONTENTAZUREFILECONNECTIONSTRING": "[concat('@Microsoft.KeyVault(SecretUri=',
reference(variables('storageConnectionStringName')).secretUriWithVersion,
'))]]",
                "APPINSIGHTS_INSTRUMENTATIONKEY": "[concat('@Microsoft.KeyVault(SecretUri=',
reference(variables('appInsightsKeyName')).secretUriWithVersion, '))]]",
                "WEBSITE_ENABLE_SYNC_UPDATE_SITE": "true"
            }
        },
        {
            "type": "sourcecontrols",
            "name": "web",
            //...
            "dependsOn": [
                "[resourceId('Microsoft.Web/sites',
variables('functionAppName'))]",
                "[resourceId('Microsoft.Web/sites/config',
variables('functionAppName'), 'appsettings')]"
            ],
        }
    ]
}

```

```

},
{
  "type": "Microsoft.KeyVault/vaults",
  "name": "[variables('keyVaultName')]",
  //...
  "dependsOn": [
    "[resourceId('Microsoft.Web/sites',
variables('functionAppName'))]"
  ],
  "properties": {
    //...
    "accessPolicies": [
      {
        "tenantId": "
[reference(resourceId('Microsoft.Web/sites/'), variables('functionAppName')),
'2020-12-01', 'Full').identity.tenantId]",
        "objectId": "
[reference(resourceId('Microsoft.Web/sites/'), variables('functionAppName')),
'2020-12-01', 'Full').identity.principalId]",
        "permissions": {
          "secrets": [ "get" ]
        }
      }
    ]
  },
  "resources": [
    {
      "type": "secrets",
      "name": "[variables('storageConnectionStringName')]",
      //...
      "dependsOn": [
        "[resourceId('Microsoft.KeyVault/vaults/',
variables('keyVaultName'))]",
        "[resourceId('Microsoft.Storage/storageAccounts',
variables('storageAccountName'))]"
      ],
      "properties": {
        "value": "
[concat('DefaultEndpointsProtocol=https;AccountName=',
variables('storageAccountName'), ';AccountKey=',
listKeys(variables('storageAccountResourceId'),'2019-09-01').key1)]"
      }
    },
    {
      "type": "secrets",
      "name": "[variables('appInsightsKeyName')]",
      //...
      "dependsOn": [
        "[resourceId('Microsoft.KeyVault/vaults/',
variables('keyVaultName'))]",
        "[resourceId('Microsoft.Insights/components',
variables('appInsightsName'))]"
      ],
      "properties": {
        "value": "

```

```
[reference(resourceId('microsoft.insights/components/'),
variables('appInsightsName')), '2019-09-01').InstrumentationKey]"
    }
]
}
]
```

ⓘ Note

In this example, the source control deployment depends on the application settings. This is normally unsafe behavior, as the app setting update behaves asynchronously. However, because we have included the `WEBSITE_ENABLE_SYNC_UPDATE_SITE` application setting, the update is synchronous. This means that the source control deployment will only begin once the application settings have been fully updated. For more app settings, see [Environment variables and app settings in Azure App Service](#).

Troubleshooting key vault references

If a reference isn't resolved properly, the reference string is used instead (for example, `@Microsoft.KeyVault(...)`). It may cause the application to throw errors, because it's expecting a secret of a different value.

Failure to resolve is commonly due to a misconfiguration of the [Key Vault access policy](#). However, it could also be due to a secret no longer existing or a syntax error in the reference itself.

If the syntax is correct, you can view other causes for error by checking the current resolution status in the portal. Navigate to Application Settings and select "Edit" for the reference in question. The edit dialog shows status information, including any errors. If you don't see the status message, it means that the syntax is invalid and not recognized as a key vault reference.

You can also use one of the built-in detectors to get additional information.

Using the detector for App Service

1. In the portal, navigate to your app.
2. Select **Diagnose and solve problems**.
3. Choose **Availability and Performance** and select **Web app down**.

4. In the search box, search for and select **Key Vault Application Settings Diagnostics**.

Using the detector for Azure Functions

1. In the portal, navigate to your app.
 2. Navigate to **Platform features**.
 3. Select **Diagnose and solve problems**.
 4. Choose **Availability and Performance** and select **Function app down or reporting errors**.
 5. Select **Key Vault Application Settings Diagnostics**.
-

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

Tutorial: Send email and invoke other business processes from App Service

Article • 03/16/2023

In this tutorial, you learn how to integrate your App Service app with your business processes. This is common to web app scenarios, such as:

- Send confirmation email for a transaction
- Add user to Facebook group
- Connect to third-party systems like SAP, Salesforce, etc.
- Exchange standard B2B messages

In this tutorial, you send emails with Gmail from your App Service app by using [Azure Logic Apps](#). There are other ways to send emails from a web app, such as SMTP configuration provided by your language framework. However, Azure Logic Apps brings a lot more power to your App Service app without adding complexity to your code. Azure Logic Apps provides a simple configuration interface for the most popular business integrations, and your app can call them anytime with an HTTP request.

Prerequisite

Deploy an app with the language framework of your choice to App Service. To follow a tutorial to deploy a sample app, see below:

ASP.NET

[Tutorial: Build an ASP.NET app in Azure with SQL Database](#)

Create the logic app

1. In the [Azure portal](#), create a Consumption logic app resource by following the instructions in [Create an example Consumption workflow](#). When page with the common triggers and templates gallery opens, return to this tutorial.
2. Under the **Start with a common trigger** section, select the trigger named **When an HTTP request is received**.

The screenshot shows the Azure Logic Apps landing page. At the top, there's a video thumbnail titled "Introducing Azure Logic Apps". To the right of the video, there's a section titled "Building integration solutions is easier than ever" with a bulleted list of features:

- Create business processes and workflows visually
- Integrate with SaaS and enterprise applications
- Unlock value from on-premises and cloud applications

Below this, there's a section titled "Start with a common trigger" with a subtitle "Pick from one of the most commonly used triggers, then orchestrate any number of actions using the rich collection of connectors". A grid of trigger icons is shown, with the "When a HTTP request is received" icon highlighted by a red box.

3. In the trigger information box, select Use sample payload to generate schema.

The screenshot shows the configuration page for the "When a HTTP request is received" trigger. It includes fields for "HTTP POST URL" (with a note "URL will be generated after save") and "Request Body JSON Schema". At the bottom, there's a button labeled "Use sample payload to generate schema" which is highlighted with a red box.

4. Copy the following sample JSON into the textbox and select Done.

The screenshot shows a JSON schema editor with a single object definition:

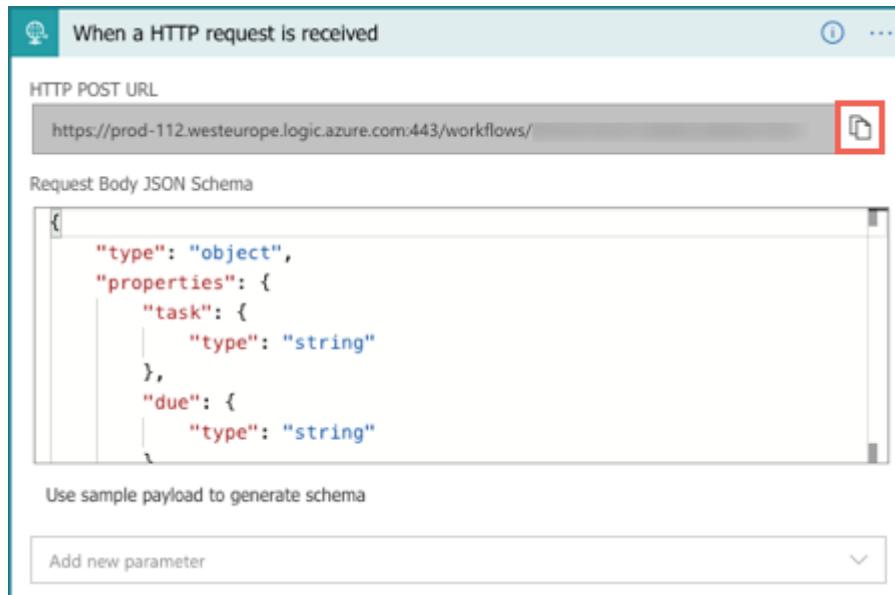
```
{
  "task": "<description>",
  "due": "<date>",
  "email": "<email-address>"
}
```

The schema is now generated for the request data you want. In practice, you can just capture the actual request data your application code generates and let Azure

generate the JSON schema for you.

5. On workflow designer toolbar, select **Save**.

You can now see the URL of your HTTP request trigger. Select the copy icon to copy it for later use.



This HTTP request definition is a trigger to anything you want to do in this logic app workflow, be it Gmail or anything else. Later you will invoke this URL in your App Service app. For more information on the request trigger, see the [HTTP request/response reference](#).

6. Under the trigger, select **New step**. Under the **Choose an operation** search box, select **All**.

7. In the search box, enter **Gmail**. Find and select the action named **Send email (V2)**.

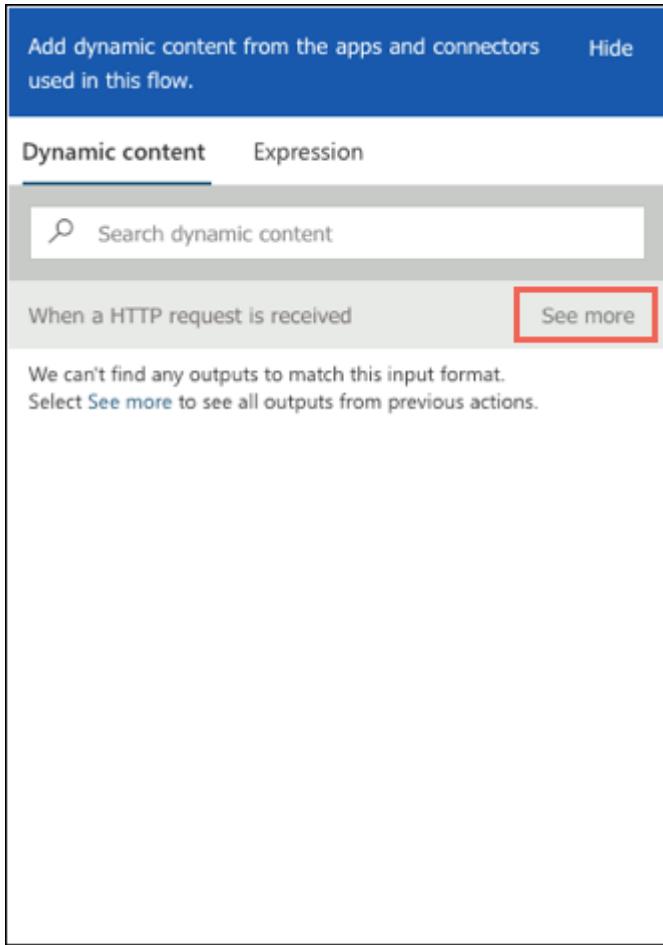
 **Tip**

You can search for other types of integrations, such as SendGrid, MailChimp, Microsoft 365, and SalesForce. For more information, see [Managed connectors for Azure Logic Apps](#).

8. In the **Gmail** action, select **Sign in** to authenticate access to the Gmail account from where you want to send the email.

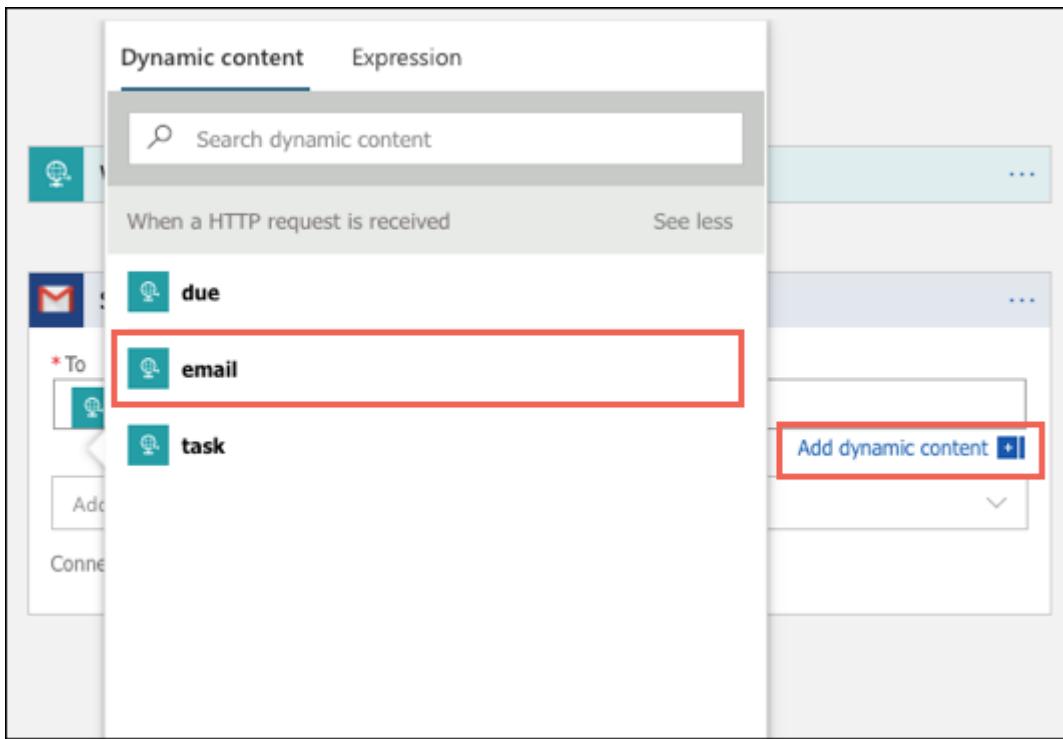


9. After you sign in, select inside the **To** box, which automatically opens the dynamic content list. In this list, next to the **When an HTTP request is received** action, select **See more**.



You should now see the three properties from your sample JSON data you used earlier. In this step, you use these properties from the HTTP request to construct an email.

10. For the **To** field value, select **email**. If you want, close the dynamic content list by selecting **Add dynamic content**.



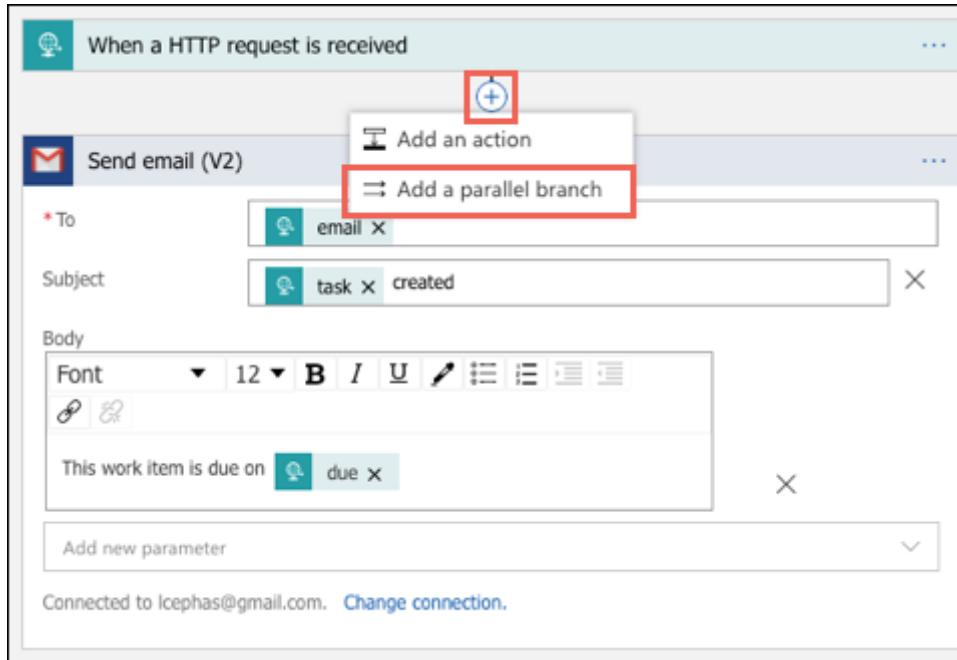
11. From the **Add new parameter list**, select **Subject** and **Body**.
12. Select inside the **Subject** box, and in the same way, select **task**. With the cursor still in the **Subject** box, type **created**.
13. Select inside in the **Body** box, and in the same way, select **due**. Move the cursor to the left of **due**, and type **This work item is due on**.

Tip

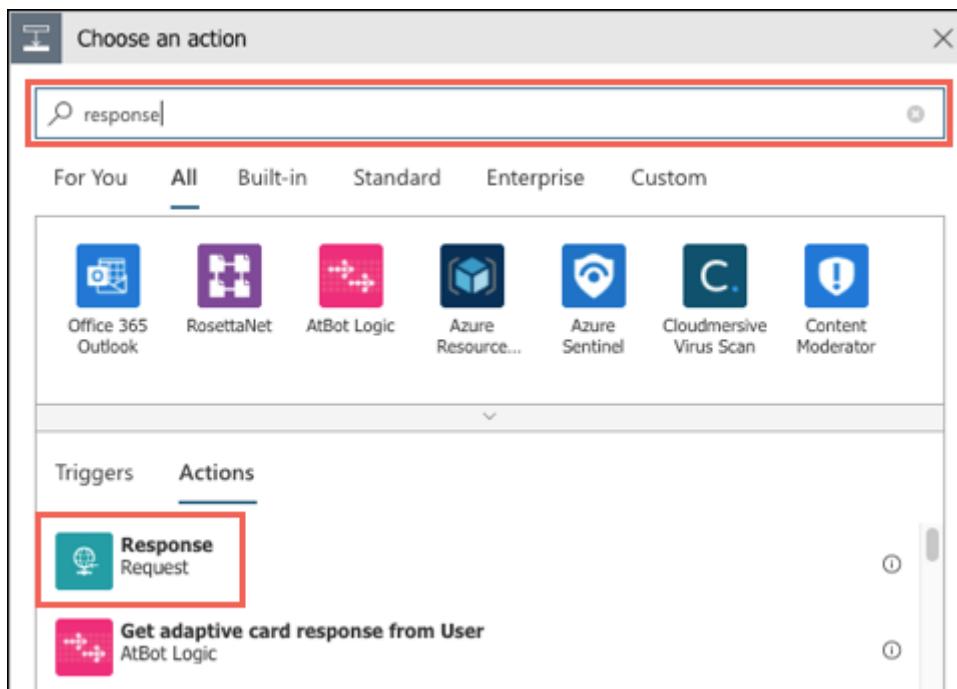
If you want to edit HTML content directly in the email body, on the designer toolbar, select **Code view**. Just make sure that you preserve the dynamic content code (for example, `@{triggerBody()?'due']}`)

```
{
  "definition": {
    "$schema": "https://schema.management.azure.com/providers/Microsoft.Logic/schemas/2016-06-01/workflowdefinition.json#",
    "actions": {
      "Send_email_(V2)": {
        "inputs": {
          "body": {
            "Body": "<p>This work item is due on @{triggerBody()?'due'}</p>",
            "Subject": "@{triggerBody()?'task'} created",
            "To": "@{triggerBody()?'email'}"
          },
          "host": {
            "connection": {
              "name": "@parameters('$connections')['gmail']['connectionId']"
            }
          },
          "method": "post",
          "path": "/v2/Mail"
        },
        "runAfter": {},
        "type": "ApiConnection"
      }
    },
    "contentVersion": "1.0.0.0"
  }
}
```

14. Next, add an asynchronous HTTP response to the HTTP trigger. Between the HTTP trigger and the Gmail action, select the + sign, and select Add a parallel branch.



15. In the search box, search for **response**, then select the **Response** action.



By default, the response action sends an HTTP 200. That's good enough for this tutorial. For more information, see the [HTTP request/response reference](#).

16. On the designer toolbar, select **Save** again.

Add HTTP request code to app

Make sure you have copied the URL of the HTTP request trigger from earlier. Because it contains sensitive information, it's best practice that you don't put it into the code directly. With App Service, you can reference it as an environment variable instead, using app settings.

In the [Cloud Shell](#), create the app setting with the following command (replace <app-name>, <resource-group-name>, and <logic-app-url>):

Azure CLI

```
az webapp config appsettings set --name <app-name> --resource-group <resource-group-name> --settings LOGIC_APP_URL=<your-logic-app-url>"
```

In your code, make a standard HTTP post to the URL using any HTTP client language that's available to your language framework, with the following configuration:

- The request body contains the same JSON format that you supplied to your logic app:

JSON

```
{  
    "task": "<description>",  
    "due": "<date>",  
    "email": "<email-address>"  
}
```

- The request contains the heading `Content-Type: application/json`.
- To optimize performance, send the request asynchronously if possible.

Click on the preferred language/framework tab below to see an example.

ASP.NET

In ASP.NET, you can send the HTTP post with the [System.Net.Http.HttpClient](#) class. For example:

C#

```
// requires using System.Net.Http;  
var client = new HttpClient();  
// requires using System.Text.Json;  
var jsonData = JsonSerializer.Serialize(new  
{  
    email = "a-valid@emailaddress.com",
```

```
    due = "4/1/2020",
    task = "My new task!"
});

HttpResponseMessage result = await client.PostAsync(
    // requires using System.Configuration;
    ConfigurationManager.AppSettings["LOGIC_APP_URL"],
    new StringContent(jsonData, Encoding.UTF8, "application/json"));

var statusCode = result.StatusCode.ToString();
```

If you're testing this code on the sample app for [Tutorial: Build an ASP.NET app in Azure with SQL Database](#), you could use it to send an email confirmation in the [Create action](#), after the `Todo` item is added. To use the asynchronous code above, convert the Create action to asynchronous.

Next steps

- [Tutorial: Host a RESTful API with CORS in Azure App Service](#)
- [HTTP request/response reference for Logic Apps](#)
- [Quickstart: Create an example Consumption workflow in multi-tenant Azure Logic Apps - Azure portal](#)
- [Environment variables and app settings reference](#)

Tutorial: Secure Cognitive Service connection from .NET App Service using Key Vault

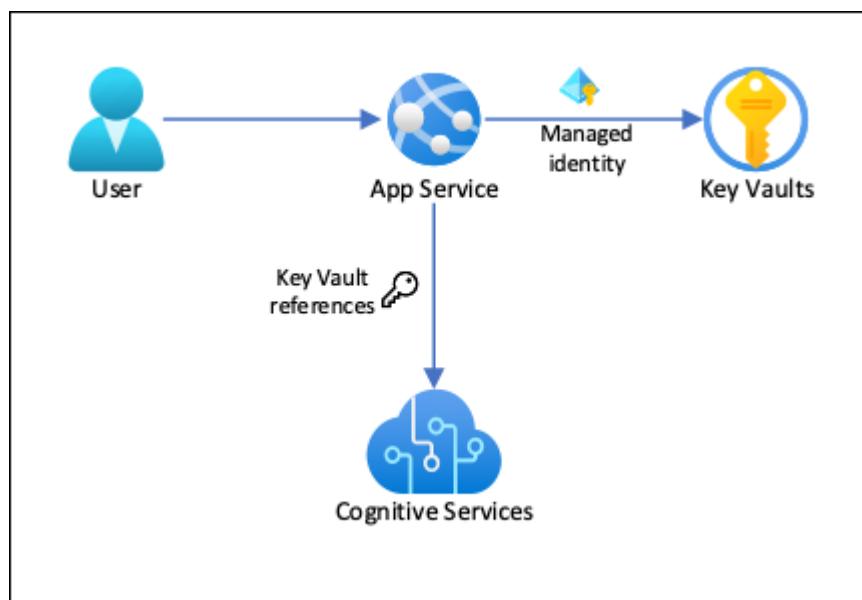
Article • 02/28/2022

Azure App Service can use [managed identities](#) to connect to back-end services without a connection string, which eliminates connection secrets to manage and keeps your back-end connectivity secure in a production environment. For back-end services that don't support managed identities and still requires connection secrets, you can use Key Vault to manage connection secrets. This tutorial uses Cognitive Services as an example to show you how it's done in practice. When you're finished, you have an app that makes programmatic calls to Cognitive Services, without storing any connection secrets inside App Service.

- [Sample application ↗](#)

💡 Tip

Azure Cognitive Services do [support authentication through managed identities](#), but this tutorial uses the [subscription key authentication](#) to demonstrate how you could connect to an Azure service that doesn't support managed identities from App Services.



With this architecture:

- Connectivity to Key Vault is secured by managed identities

- App Service accesses the secrets using [Key Vault references](#) as app settings.
- Access to the key vault is restricted to the app. App contributors, such as administrators, may have complete control of the App Service resources, and at the same time have no access to the Key Vault secrets.
- If your application code already accesses connection secrets with app settings, no change is required.

What you will learn:

- ✓ Enable managed identities
- ✓ Use managed identities to connect to Key Vault
- ✓ Use Key Vault references
- ✓ Access Cognitive Services

Prerequisites

Prepare your environment for the Azure CLI.

- Use the Bash environment in [Azure Cloud Shell](#). For more information, see [Quickstart for Bash in Azure Cloud Shell](#).

 [Launch Cloud Shell](#) 

- If you prefer to run CLI reference commands locally, [install](#) the Azure CLI. If you're running on Windows or macOS, consider running Azure CLI in a Docker container. For more information, see [How to run the Azure CLI in a Docker container](#).
 - 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 other sign-in options, see [Sign in with the Azure CLI](#).
 - When you're prompted, install the Azure CLI extension on first use. For more information about extensions, see [Use extensions with the Azure CLI](#).
 - Run [az version](#) to find the version and dependent libraries that are installed. To upgrade to the latest version, run [az upgrade](#).

Create app with connectivity to Cognitive Services

1. Create a resource group to contain all of your resources:

Azure CLI

```
# Save resource group name as variable for convenience  
groupName=myKVResourceGroup  
region=westeurope  
  
az group create --name $groupName --location $region
```

2. Create a Cognitive Services resource. Replace <cs-resource-name> with a unique name of your choice.

Azure CLI

```
# Save resource name as variable for convenience.  
cs resourceName=<cs-resource-name>  
  
az cognitiveservices account create --resource-group $groupName --name  
$cs resourceName --location $region --kind TextAnalytics --sku F0 --  
custom-domain $cs resourceName
```

! Note

--sku F0 creates a free tier Cognitive Services resource. Each subscription is limited to a quota of one free-tier TextAnalytics resource. If you're already over the quota, use --sku S instead.

Configure .NET app

Clone the sample repository locally and deploy the sample application to App Service. Replace <app-name> with a unique name.

Azure CLI

```
# Save app name as variable for convenience  
appName=<app-name>  
  
# Clone sample application  
git clone https://github.com/Azure-Samples/app-service-language-detector.git  
cd app-service-language-detector/dotnet  
  
az webapp up --sku F1 --resource-group $groupName --name $appName --plan  
$appName --location $region
```

Configure secrets as app settings

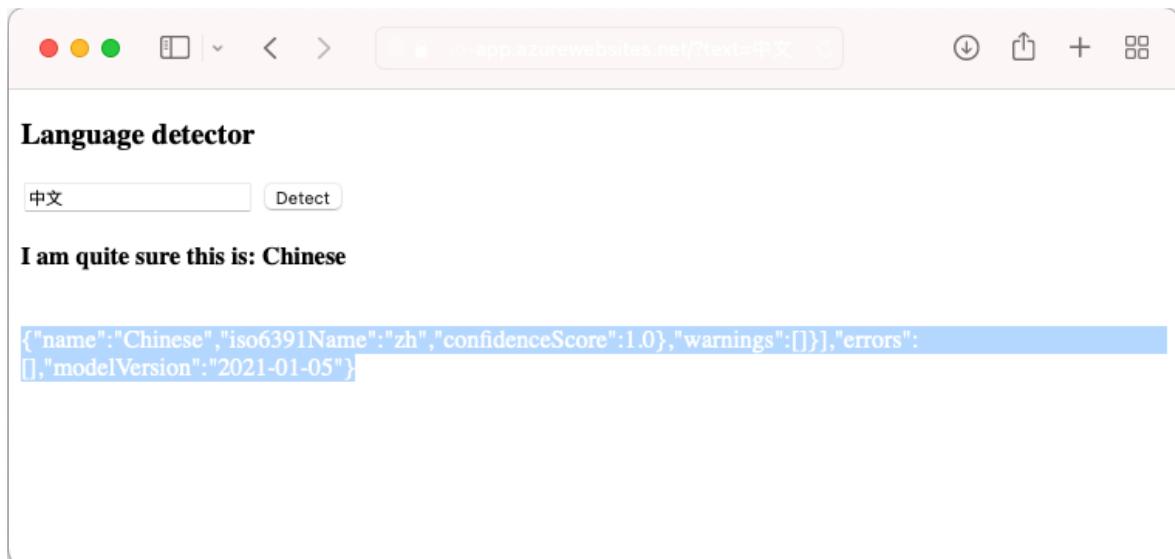
1. Configure the Cognitive Services secrets as app settings `CS_ACCOUNT_NAME` and `CS_ACCOUNT_KEY`.

Azure CLI

```
# Get subscription key for Cognitive Services resource
csKey1=$(az cognitiveservices account keys list --resource-group
$groupName --name $csResourceName --query key1 --output tsv)

az webapp config appsettings set --resource-group $groupName --name
$appName --settings CS_ACCOUNT_NAME="$csResourceName"
CS_ACCOUNT_KEY="$csKey1"
```

2. In the browser, navigate to your deploy app at `<app-name>.azurewebsites.net` and try out the language detector with strings in various languages.



If you look at the application code, you may notice the debug output for the detection results in the same font color as the background. You can see it by trying to highlight the white space directly below the result.

Secure back-end connectivity

At the moment, connection secrets are stored as app settings in your App Service app. This approach is already securing connection secrets from your application codebase. However, any contributor who can manage your app can also see the app settings. In this step, you move the connection secrets to a key vault, and lock down access so that only you can manage it and only the App Service app can read it using its managed identity.

1. Create a key vault. Replace <vault-name> with a unique name.

```
Azure CLI
```

```
# Save app name as variable for convenience
vaultName=<vault-name>

az keyvault create --resource-group $groupName --name $vaultName --
location $region --sku standard --enable-rbac-authorization
```

The `--enable-rbac-authorization` parameter sets Azure role-based access control (RBAC) as the permission model. This setting by default invalidates all access policies permissions.

2. Give yourself the *Key Vault Secrets Officer* RBAC role for the vault.

```
Azure CLI
```

```
vaultResourceId=$(az keyvault show --name $vaultName --query id --
output tsv)
myId=$(az ad signed-in-user show --query id --output tsv)
az role assignment create --role "Key Vault Secrets Officer" --
assignee-object-id $myId --assignee-principal-type User --scope
$vaultResourceId
```

3. Enable the system-assigned managed identity for your app, and give it the *Key Vault Secrets User* RBAC role for the vault.

```
Azure CLI
```

```
az webapp identity assign --resource-group $groupName --name $appName -
-scope $vaultResourceId --role "Key Vault Secrets User"
```

4. Add the Cognitive Services resource name and subscription key as secrets to the vault, and save their IDs as environment variables for the next step.

```
Azure CLI
```

```
csResourceKVUri=$(az keyvault secret set --vault-name $vaultName --name
csresource --value $cs resourceName --query id --output tsv)
csKeyKVUri=$(az keyvault secret set --vault-name $vaultName --name
cskey --value $csKey1 --query id --output tsv)
```

5. Previously, you set the secrets as app settings `CS_ACCOUNT_NAME` and `CS_ACCOUNT_KEY` in your app. Now, set them as `key vault references` instead.

Azure CLI

```
az webapp config appsettings set --resource-group $groupName --name $appName --settings CS_ACCOUNT_NAME="@Microsoft.KeyVault(SecretUri=$csResourceKVUri)" CS_ACCOUNT_KEY="@Microsoft.KeyVault(SecretUri=$csKeyKVUri)"
```

6. In the browser, navigate to <app-name>.azurewebsites.net again. If you get detection results back, then you're connecting to the Cognitive Services endpoint with key vault references.

Congratulations, your app is now connecting to Cognitive Services using secrets kept in your key vault, without any changes to your application code.

Clean up resources

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

Azure CLI

```
az group delete --name $groupName
```

This command may take a minute to run.

Next steps

- [Tutorial: Isolate back-end communication with Virtual Network integration](#)
- [Integrate your app with an Azure virtual network](#)
- [App Service networking features](#)

Tutorial: Secure Cognitive Service connection from JavaScript App Service using Key Vault

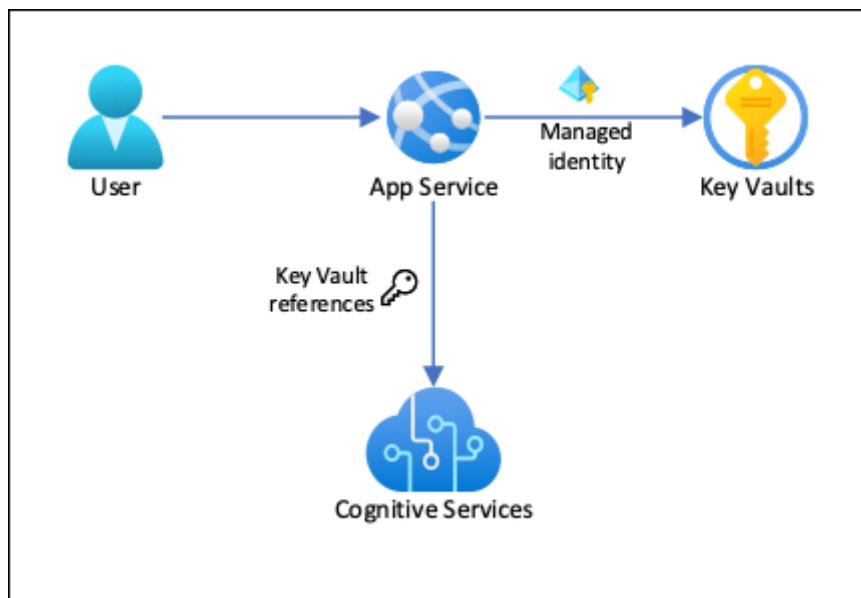
Article • 08/02/2024

Azure App Service can use [managed identities](#) to connect to back-end services without a connection string, which eliminates connection secrets to manage and keeps your back-end connectivity secure in a production environment. For back-end services that don't support managed identities and still requires connection secrets, you can use Key Vault to manage connection secrets. This tutorial uses Azure AI services as an example to show you how it's done in practice. When you're finished, you have an app that makes programmatic calls to Azure AI services, without storing any connection secrets inside App Service.

- [Sample application ↗](#)

💡 Tip

Azure AI services do [support authentication through managed identities](#), but this tutorial uses the [subscription key authentication](#) to demonstrate how you could connect to an Azure service that doesn't support managed identities from App Services.



With this architecture:

- Connectivity to Key Vault is secured by managed identities

- App Service accesses the secrets using [Key Vault references](#) as app settings.
- Access to the key vault is restricted to the app. App contributors, such as administrators, may have complete control of the App Service resources, and at the same time have no access to the Key Vault secrets.
- If your application code already accesses connection secrets with app settings, no change is required.

What you will learn:

- ✓ Enable managed identities
- ✓ Use managed identities to connect to Key Vault
- ✓ Use Key Vault references
- ✓ Access Azure AI services

Prerequisites

Prepare your environment for the Azure CLI.

- Use the Bash environment in [Azure Cloud Shell](#). For more information, see [Quickstart for Bash in Azure Cloud Shell](#).

 [Launch Cloud Shell](#) 

- If you prefer to run CLI reference commands locally, [install](#) the Azure CLI. If you're running on Windows or macOS, consider running Azure CLI in a Docker container. For more information, see [How to run the Azure CLI in a Docker container](#).
 - 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 other sign-in options, see [Sign in with the Azure CLI](#).
 - When you're prompted, install the Azure CLI extension on first use. For more information about extensions, see [Use extensions with the Azure CLI](#).
 - Run `az version` to find the version and dependent libraries that are installed. To upgrade to the latest version, run `az upgrade`.

Create app with connectivity to Azure AI services

1. Create a resource group to contain all of your resources:

Azure CLI

```
# Save resource group name as variable for convenience
groupName=myKVResourceGroup
region=westeurope

az group create --name $groupName --location $region
```

2. Create an Azure AI services resource. Replace <cs-resource-name> with a unique name of your choice.

Azure CLI

```
# Save resource name as variable for convenience.
cs resourceName=<cs-resource-name>

az cognitiveservices account create --resource-group $groupName --name
$cs resourceName --location $region --kind TextAnalytics --sku F0 --
custom-domain $cs resourceName
```

! Note

--sku F0 creates a free tier Azure AI services resource. Each subscription is limited to a quota of one free-tier TextAnalytics resource. If you're already over the quota, use --sku S instead.

Configure JavaScript app

Clone the sample repository locally and deploy the sample application to App Service. Replace <app-name> with a unique name.

Azure CLI

```
# Clone and prepare sample application
git clone https://github.com/Azure-Samples/app-service-language-detector.git
cd app-service-language-detector/javascript
zip -r default.zip .

# Save app name as variable for convenience
appName=<app-name>

az appservice plan create --resource-group $groupName --name $appName --sku
FREE --location $region --is-linux
az webapp create --resource-group $groupName --plan $appName --name $appName
--runtime "node:18-lts"
az webapp config appsettings set --resource-group $groupName --name $appName
```

```
--settings SCM_DO_BUILD_DURING_DEPLOYMENT=true  
az webapp deploy --resource-group $groupName --name $appName --src-path  
.default.zip
```

The preceding commands:

- Create a linux app service plan
- Create a web app for Node.js 18 LTS
- Configure the web app to install the npm packages on deployment
- Upload the zip file, and install the npm packages

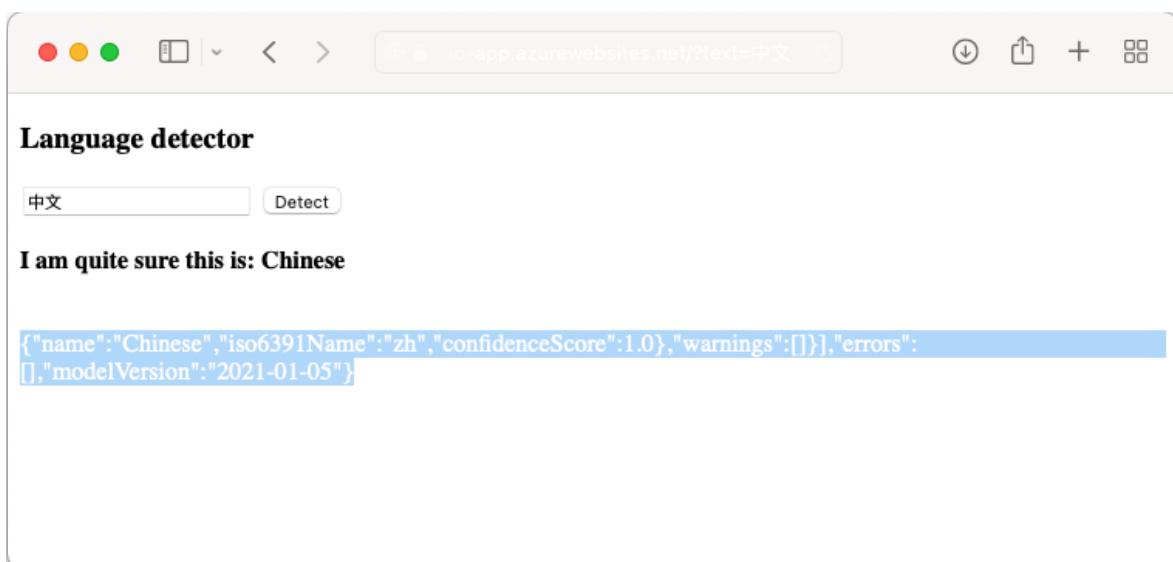
Configure secrets as app settings

1. Configure the Azure AI services secrets as app settings `CS_ACCOUNT_NAME` and `CS_ACCOUNT_KEY`.

Azure CLI

```
# Get subscription key for Cognitive Services resource  
csKey1=$(az cognitiveservices account keys list --resource-group  
$groupName --name $csResourceName --query key1 --output tsv)  
  
az webapp config appsettings set --resource-group $groupName --name  
$appName --settings CS_ACCOUNT_NAME="$csResourceName"  
CS_ACCOUNT_KEY="$csKey1"
```

2. In the browser, navigate to your deploy app at `<app-name>.azurewebsites.net` and try out the language detector with strings in various languages.



If you look at the application code, you may notice the debug output for the detection results in the same font color as the background. You can see it by trying

to highlight the white space directly below the result.

Secure back-end connectivity

At the moment, connection secrets are stored as app settings in your App Service app. This approach is already securing connection secrets from your application codebase. However, any contributor who can manage your app can also see the app settings. In this step, you move the connection secrets to a key vault, and lock down access so that only you can manage it and only the App Service app can read it using its managed identity.

1. Create a key vault. Replace `<vault-name>` with a unique name.

Azure CLI

```
# Save app name as variable for convenience
vaultName=<vault-name>

az keyvault create --resource-group $groupName --name $vaultName --
location $region --sku standard --enable-rbac-authorization
```

The `--enable-rbac-authorization` parameter sets Azure role-based access control (RBAC) as the permission model. This setting by default invalidates all access policies permissions.

2. Give yourself the *Key Vault Secrets Officer* RBAC role for the vault.

Azure CLI

```
vaultResourceId=$(az keyvault show --name $vaultName --query id --
output tsv)
myId=$(az ad signed-in-user show --query id --output tsv)
az role assignment create --role "Key Vault Secrets Officer" --
assignee-object-id $myId --assignee-principal-type User --scope
$vaultResourceId
```

3. Enable the system-assigned managed identity for your app, and give it the *Key Vault Secrets User* RBAC role for the vault.

Azure CLI

```
az webapp identity assign --resource-group $groupName --name $appName --
-scope $vaultResourceId --role "Key Vault Secrets User"
```

4. Add the Azure AI services resource name and subscription key as secrets to the vault, and save their IDs as environment variables for the next step.

```
Azure CLI
```

```
csResourceKVUri=$(az keyvault secret set --vault-name $vaultName --name csresource --value $csResourceName --query id --output tsv)
csKeyKVUri=$(az keyvault secret set --vault-name $vaultName --name cskey --value $csKey1 --query id --output tsv)
```

5. Previously, you set the secrets as app settings `CS_ACCOUNT_NAME` and `CS_ACCOUNT_KEY` in your app. Now, set them as [key vault references](#) instead.

```
Azure CLI
```

```
az webapp config appsettings set --resource-group $groupName --name $appName --settings
CS_ACCOUNT_NAME="@Microsoft.KeyVault(SecretUri=$csResourceKVUri)"
CS_ACCOUNT_KEY="@Microsoft.KeyVault(SecretUri=$csKeyKVUri)"
```

6. In the browser, navigate to `<app-name>.azurewebsites.net` again. If you get detection results back, then you're connecting to the Azure AI services endpoint with key vault references.

Congratulations, your app is now connecting to Azure AI services using secrets kept in your key vault, without any changes to your application code.

Clean up resources

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

```
Azure CLI
```

```
az group delete --name $groupName
```

This command may take a minute to run.

Next steps

- [Tutorial: Isolate back-end communication with Virtual Network integration](#)
- [Integrate your app with an Azure virtual network](#)

- App Service networking features
-

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

Tutorial: Secure Cognitive Service connection from PHP App Service using Key Vault

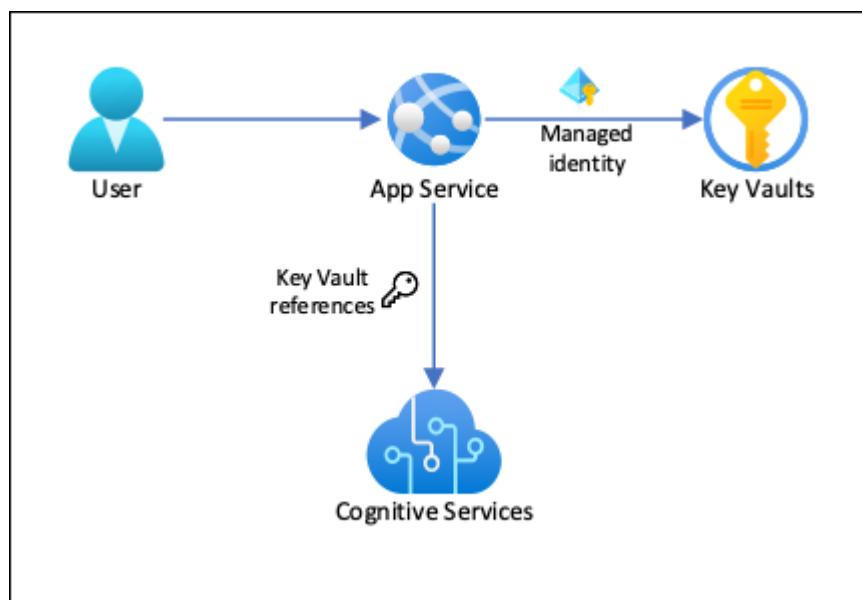
Article • 02/28/2022

Azure App Service can use [managed identities](#) to connect to back-end services without a connection string, which eliminates connection secrets to manage and keeps your back-end connectivity secure in a production environment. For back-end services that don't support managed identities and still requires connection secrets, you can use Key Vault to manage connection secrets. This tutorial uses Cognitive Services as an example to show you how it's done in practice. When you're finished, you have an app that makes programmatic calls to Cognitive Services, without storing any connection secrets inside App Service.

- [Sample application ↗](#)

💡 Tip

Azure Cognitive Services do [support authentication through managed identities](#), but this tutorial uses the [subscription key authentication](#) to demonstrate how you could connect to an Azure service that doesn't support managed identities from App Services.



With this architecture:

- Connectivity to Key Vault is secured by managed identities

- App Service accesses the secrets using [Key Vault references](#) as app settings.
- Access to the key vault is restricted to the app. App contributors, such as administrators, may have complete control of the App Service resources, and at the same time have no access to the Key Vault secrets.
- If your application code already accesses connection secrets with app settings, no change is required.

What you will learn:

- ✓ Enable managed identities
- ✓ Use managed identities to connect to Key Vault
- ✓ Use Key Vault references
- ✓ Access Cognitive Services

Prerequisites

Prepare your environment for the Azure CLI.

- Use the Bash environment in [Azure Cloud Shell](#). For more information, see [Quickstart for Bash in Azure Cloud Shell](#).

 [Launch Cloud Shell](#) 

- If you prefer to run CLI reference commands locally, [install](#) the Azure CLI. If you're running on Windows or macOS, consider running Azure CLI in a Docker container. For more information, see [How to run the Azure CLI in a Docker container](#).
 - 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 other sign-in options, see [Sign in with the Azure CLI](#).
 - When you're prompted, install the Azure CLI extension on first use. For more information about extensions, see [Use extensions with the Azure CLI](#).
 - Run [az version](#) to find the version and dependent libraries that are installed. To upgrade to the latest version, run [az upgrade](#).

Create app with connectivity to Cognitive Services

1. Create a resource group to contain all of your resources:

Azure CLI

```
# Save resource group name as variable for convenience  
groupName=myKVResourceGroup  
region=westeurope  
  
az group create --name $groupName --location $region
```

2. Create a Cognitive Services resource. Replace <cs-resource-name> with a unique name of your choice.

Azure CLI

```
# Save resource name as variable for convenience.  
cs resourceName=<cs-resource-name>  
  
az cognitiveservices account create --resource-group $groupName --name  
$cs resourceName --location $region --kind TextAnalytics --sku F0 --  
custom-domain $cs resourceName
```

! Note

--sku F0 creates a free tier Cognitive Services resource. Each subscription is limited to a quota of one free-tier TextAnalytics resource. If you're already over the quota, use --sku S instead.

Configure PHP app

Clone the sample repository locally and deploy the sample application to App Service. Replace <app-name> with a unique name.

Azure CLI

```
# Clone and prepare sample application  
git clone https://github.com/Azure-Samples/app-service-language-detector.git  
cd app-service-language-detector/php  
zip default.zip index.php  
  
# Save app name as variable for convenience  
appName=<app-name>  
  
az appservice plan create --resource-group $groupName --name $appName --sku  
FREE --location $region  
az webapp create --resource-group $groupName --plan $appName --name $appName  
az webapp deployment source config-zip --resource-group $groupName --name  
$appName --src ./default.zip
```

Configure secrets as app settings

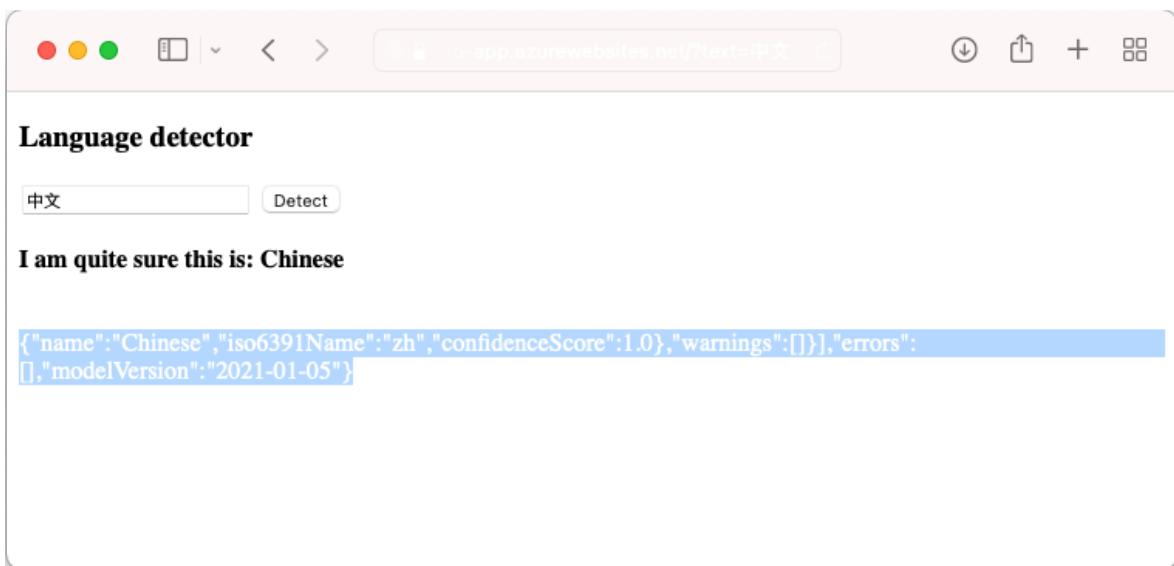
1. Configure the Cognitive Services secrets as app settings `CS_ACCOUNT_NAME` and `CS_ACCOUNT_KEY`.

Azure CLI

```
# Get subscription key for Cognitive Services resource
csKey1=$(az cognitiveservices account keys list --resource-group
$groupName --name $csResourceName --query key1 --output tsv)

az webapp config appsettings set --resource-group $groupName --name
$appName --settings CS_ACCOUNT_NAME="$csResourceName"
CS_ACCOUNT_KEY="$csKey1"
```

2. In the browser, navigate to your deploy app at `<app-name>.azurewebsites.net` and try out the language detector with strings in various languages.



If you look at the application code, you may notice the debug output for the detection results in the same font color as the background. You can see it by trying to highlight the white space directly below the result.

Secure back-end connectivity

At the moment, connection secrets are stored as app settings in your App Service app. This approach is already securing connection secrets from your application codebase. However, any contributor who can manage your app can also see the app settings. In this step, you move the connection secrets to a key vault, and lock down access so that

only you can manage it and only the App Service app can read it using its managed identity.

1. Create a key vault. Replace <vault-name> with a unique name.

Azure CLI

```
# Save app name as variable for convenience
vaultName=<vault-name>

az keyvault create --resource-group $groupName --name $vaultName --
location $region --sku standard --enable-rbac-authorization
```

The `--enable-rbac-authorization` parameter [sets Azure role-based access control \(RBAC\) as the permission model](#). This setting by default invalidates all access policies permissions.

2. Give yourself the *Key Vault Secrets Officer* RBAC role for the vault.

Azure CLI

```
vaultResourceId=$(az keyvault show --name $vaultName --query id --
output tsv)
myId=$(az ad signed-in-user show --query id --output tsv)
az role assignment create --role "Key Vault Secrets Officer" --
assignee-object-id $myId --assignee-principal-type User --scope
$vaultResourceId
```

3. Enable the system-assigned managed identity for your app, and give it the *Key Vault Secrets User* RBAC role for the vault.

Azure CLI

```
az webapp identity assign --resource-group $groupName --name $appName -
-scope $vaultResourceId --role "Key Vault Secrets User"
```

4. Add the Cognitive Services resource name and subscription key as secrets to the vault, and save their IDs as environment variables for the next step.

Azure CLI

```
csResourceKVUri=$(az keyvault secret set --vault-name $vaultName --name
csresource --value $cs resourceName --query id --output tsv)
csKeyKVUri=$(az keyvault secret set --vault-name $vaultName --name
cskey --value $csKey1 --query id --output tsv)
```

5. Previously, you set the secrets as app settings `CS_ACCOUNT_NAME` and `CS_ACCOUNT_KEY` in your app. Now, set them as [key vault references](#) instead.

Azure CLI

```
az webapp config appsettings set --resource-group $groupName --name $appName --settings CS_ACCOUNT_NAME="@Microsoft.KeyVault(SecretUri=$csResourceKVUri)" CS_ACCOUNT_KEY="@Microsoft.KeyVault(SecretUri=$csKeyKVUri)"
```

6. In the browser, navigate to `<app-name>.azurewebsites.net` again. If you get detection results back, then you're connecting to the Cognitive Services endpoint with key vault references.

Congratulations, your app is now connecting to Cognitive Services using secrets kept in your key vault, without any changes to your application code.

Clean up resources

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

Azure CLI

```
az group delete --name $groupName
```

This command may take a minute to run.

Next steps

- [Tutorial: Isolate back-end communication with Virtual Network integration](#)
- [Integrate your app with an Azure virtual network](#)
- [App Service networking features](#)

Tutorial: Secure Cognitive Service connection from Python App Service using Key Vault

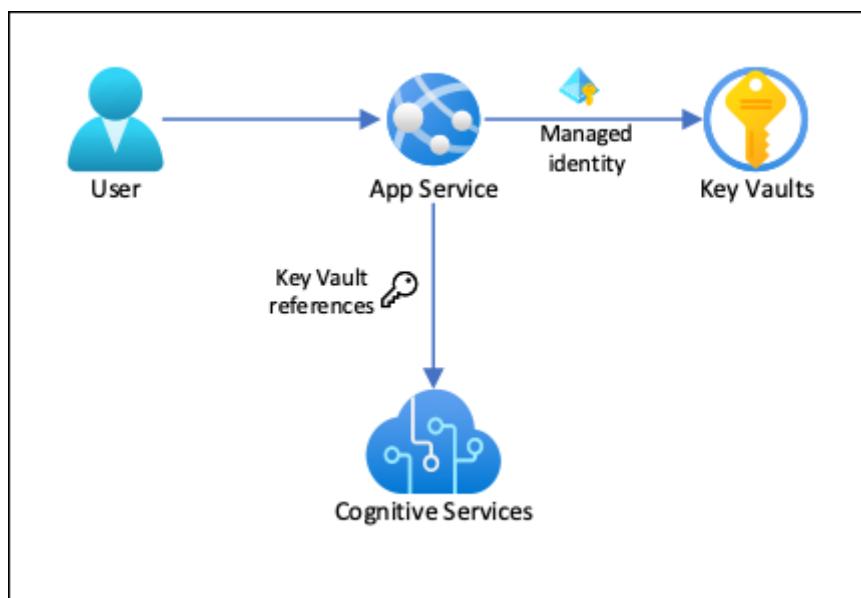
Article • 08/23/2024

Azure App Service can use [managed identities](#) to connect to back-end services without a connection string, which eliminates connection secrets to manage and keeps your back-end connectivity secure in a production environment. For back-end services that don't support managed identities and still requires connection secrets, you can use Key Vault to manage connection secrets. This tutorial uses Azure AI services as an example to show you how it's done in practice. When you're finished, you have an app that makes programmatic calls to Azure AI services, without storing any connection secrets inside App Service.

- [Sample application ↗](#)

💡 Tip

Azure AI services do [support authentication through managed identities](#), but this tutorial uses the [subscription key authentication](#) to demonstrate how you could connect to an Azure service that doesn't support managed identities from App Services.



With this architecture:

- Connectivity to Key Vault is secured by managed identities

- App Service accesses the secrets using [Key Vault references](#) as app settings.
- Access to the key vault is restricted to the app. App contributors, such as administrators, may have complete control of the App Service resources, and at the same time have no access to the Key Vault secrets.
- If your application code already accesses connection secrets with app settings, no change is required.

What you will learn:

- ✓ Enable managed identities
- ✓ Use managed identities to connect to Key Vault
- ✓ Use Key Vault references
- ✓ Access Azure AI services

Prerequisites

Prepare your environment for the Azure CLI.

- Use the Bash environment in [Azure Cloud Shell](#). For more information, see [Quickstart for Bash in Azure Cloud Shell](#).

 [Launch Cloud Shell](#) 

- If you prefer to run CLI reference commands locally, [install](#) the Azure CLI. If you're running on Windows or macOS, consider running Azure CLI in a Docker container. For more information, see [How to run the Azure CLI in a Docker container](#).
 - 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 other sign-in options, see [Sign in with the Azure CLI](#).
 - When you're prompted, install the Azure CLI extension on first use. For more information about extensions, see [Use extensions with the Azure CLI](#).
 - Run `az version` to find the version and dependent libraries that are installed. To upgrade to the latest version, run `az upgrade`.

Create app with connectivity to Azure AI services

1. Create a resource group to contain all of your resources:

Azure CLI

```
# Save resource group name as variable for convenience
groupName=myKVRSGroup
region=westeurope

az group create --name $groupName --location $region
```

2. Create an Azure AI services resource. Replace <cs-resource-name> with a unique name of your choice.

Azure CLI

```
# Save resource name as variable for convenience.
csResourceName=<cs-resource-name>

az cognitiveservices account create --resource-group $groupName --name
$csResourceName --location $region --kind TextAnalytics --sku F0 --
custom-domain $csResourceName
```

! Note

--sku F0 creates a free tier Azure AI services resource. Each subscription is limited to a quota of one free-tier TextAnalytics resource. If you're already over the quota, use --sku S instead.

Configure Python app

Clone the sample repository locally and deploy the sample application to App Service. Replace <app-name> with a unique name.

Azure CLI

```
# Clone and prepare sample application
git clone https://github.com/Azure-Samples/app-service-language-detector.git
cd app-service-language-detector/python
zip -r default.zip .

# Save app name as variable for convenience
appName=<app-name>

az appservice plan create --resource-group $groupName --name $appName --sku
FREE --location $region --is-linux
az webapp create --resource-group $groupName --plan $appName --name $appName
--runtime "python:3.11"
az webapp config appsettings set --resource-group $groupName --name $appName
```

```
--settings SCM_DO_BUILD_DURING_DEPLOYMENT=true  
az webapp deploy --resource-group $groupName --name $appName --src-path  
.default.zip
```

The preceding commands:

- Create a linux app service plan
- Create a web app for Python 3.11
- Configure the web app to install the python packages on deployment
- Upload the zip file, and install the python packages

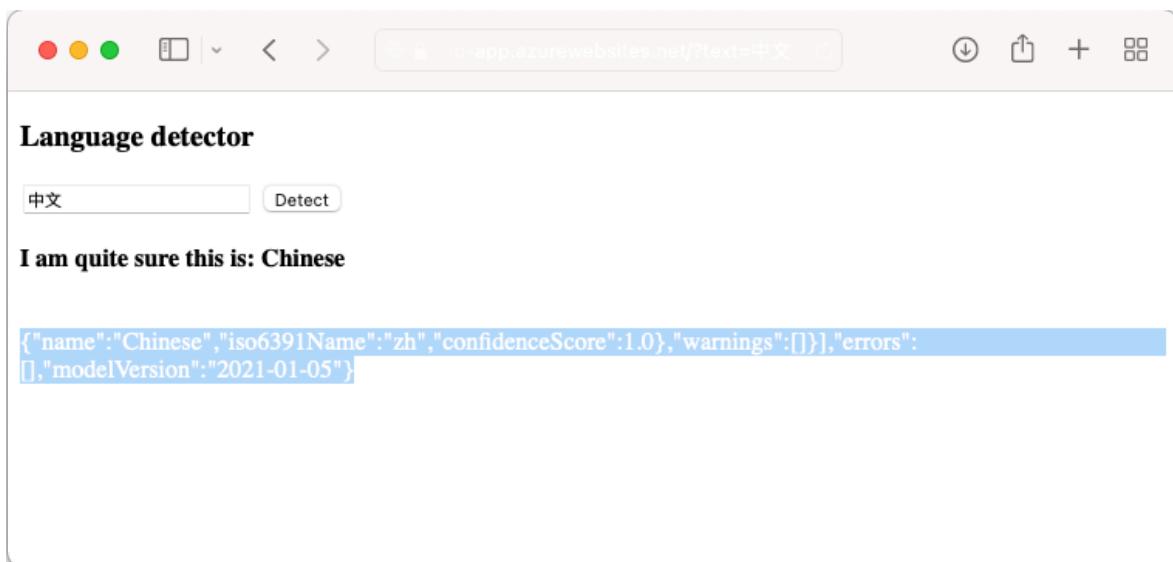
Configure secrets as app settings

1. Configure the Azure AI services secrets as app settings `CS_ACCOUNT_NAME` and `CS_ACCOUNT_KEY`.

Azure CLI

```
# Get subscription key for Cognitive Services resource  
csKey1=$(az cognitiveservices account keys list --resource-group  
$groupName --name $csResourceName --query key1 --output tsv)  
  
az webapp config appsettings set --resource-group $groupName --name  
$appName --settings CS_ACCOUNT_NAME="$csResourceName"  
CS_ACCOUNT_KEY="$csKey1"
```

2. In the browser, navigate to your deploy app at `<app-name>.azurewebsites.net` and try out the language detector with strings in various languages.



If you look at the application code, you may notice the debug output for the detection results in the same font color as the background. You can see it by trying

to highlight the white space directly below the result.

Secure back-end connectivity

At the moment, connection secrets are stored as app settings in your App Service app. This approach is already securing connection secrets from your application codebase. However, any contributor who can manage your app can also see the app settings. In this step, you move the connection secrets to a key vault, and lock down access so that only you can manage it and only the App Service app can read it using its managed identity.

1. Create a key vault. Replace `<vault-name>` with a unique name.

Azure CLI

```
# Save app name as variable for convenience
vaultName=<vault-name>

az keyvault create --resource-group $groupName --name $vaultName --
location $region --sku standard --enable-rbac-authorization
```

The `--enable-rbac-authorization` parameter sets Azure role-based access control (RBAC) as the permission model. This setting by default invalidates all access policies permissions.

2. Give yourself the *Key Vault Secrets Officer* RBAC role for the vault.

Azure CLI

```
vaultResourceId=$(az keyvault show --name $vaultName --query id --
output tsv)
myId=$(az ad signed-in-user show --query id --output tsv)
az role assignment create --role "Key Vault Secrets Officer" --
assignee-object-id $myId --assignee-principal-type User --scope
$vaultResourceId
```

3. Enable the system-assigned managed identity for your app, and give it the *Key Vault Secrets User* RBAC role for the vault.

Azure CLI

```
az webapp identity assign --resource-group $groupName --name $appName -
-scope $vaultResourceId --role "Key Vault Secrets User"
```

4. Add the Azure AI services resource name and subscription key as secrets to the vault, and save their IDs as environment variables for the next step.

```
Azure CLI
```

```
csResourceKVUri=$(az keyvault secret set --vault-name $vaultName --name csresource --value $csResourceName --query id --output tsv)
csKeyKVUri=$(az keyvault secret set --vault-name $vaultName --name cskey --value $csKey1 --query id --output tsv)
```

5. Previously, you set the secrets as app settings `CS_ACCOUNT_NAME` and `CS_ACCOUNT_KEY` in your app. Now, set them as [key vault references](#) instead.

```
Azure CLI
```

```
az webapp config appsettings set --resource-group $groupName --name $appName --settings
CS_ACCOUNT_NAME="@Microsoft.KeyVault(SecretUri=$csResourceKVUri)"
CS_ACCOUNT_KEY="@Microsoft.KeyVault(SecretUri=$csKeyKVUri)"
```

6. In the browser, navigate to `<app-name>.azurewebsites.net` again. If you get detection results back, then you're connecting to the Azure AI services endpoint with key vault references.

Congratulations, your app is now connecting to Azure AI services using secrets kept in your key vault, without any changes to your application code.

Clean up resources

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

```
Azure CLI
```

```
az group delete --name $groupName
```

This command may take a minute to run.

Next steps

- [Tutorial: Isolate back-end communication with Virtual Network integration](#)
- [Integrate your app with an Azure virtual network](#)

- App Service networking features
-

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

How to use managed identities for App Service and Azure Functions

Article • 09/30/2024

ⓘ Note

Starting June 1, 2024, all newly created App Service apps will have the option to generate a unique default hostname using the naming convention `<app-name>-<random-hash>. <region>.azurewebsites.net`. Existing app names will remain unchanged.

Example: `myapp-ds27dh7271ah175.westus-01.azurewebsites.net`

For further details, refer to [Unique Default Hostname for App Service Resource ↗](#).

This article shows you how to create a managed identity for App Service and Azure Functions applications and how to use it to access other resources.

ⓘ Important

Because [managed identities don't support cross-directory scenarios](#), they won't behave as expected if your app is migrated across subscriptions or tenants. To recreate the managed identities after such a move, see [Will managed identities be recreated automatically if I move a subscription to another directory?](#).

Downstream resources also need to have access policies updated to use the new identity.

ⓘ Note

Managed identities are not available for [apps deployed in Azure Arc](#).

A managed identity from Microsoft Entra ID allows your app to easily access other Microsoft Entra protected resources such as Azure Key Vault. The identity is managed by the Azure platform and does not require you to provision or rotate any secrets. For more about managed identities in Microsoft Entra ID, see [Managed identities for Azure resources](#).

Your application can be granted two types of identities:

- A **system-assigned identity** is tied to the app and is deleted if the app is deleted.
An app can only have one system-assigned identity.
- A **user-assigned identity** is a standalone Azure resource that can be assigned to your app. An app can have multiple user-assigned identities, and one user-assigned identity can be assigned to multiple Azure resources, such as two App Service apps.

The managed identity configuration is specific to the slot. To configure a managed identity for a deployment slot in the portal, navigate to the slot first. To find the managed identity for your web app or deployment slot in your Microsoft Entra tenant from the Azure portal, search for it directly from the **Overview** page of your tenant. Usually, the slot name is similar to <app-name>/slots/<slot-name>.

This video shows you how to use managed identities for App Service.

[https://learn-video.azurefd.net/vod/player?id=4fdf7a78-b3ce-48df-b3ce-cd7796d0ad5a&locale=en-us&embedUrl=%2Fazure%2Fapp-service%2Foverview-managed-identity ↗](https://learn-video.azurefd.net/vod/player?id=4fdf7a78-b3ce-48df-b3ce-cd7796d0ad5a&locale=en-us&embedUrl=%2Fazure%2Fapp-service%2Foverview-managed-identity)

The steps in the video are also described in the following sections.

Add a system-assigned identity

Azure portal

1. Access your app's settings in the [Azure portal](#) ↗ under the **Settings** group in the left navigation pane.
2. Select **Identity**.
3. Within the **System assigned** tab, switch **Status** to **On**. Click **Save**.

The screenshot shows the Azure portal interface for managing a resource's identity. On the left, there's a navigation menu with options like Deployment, Settings, and Identity. The 'Identity' option under Settings is highlighted with a red box. The main content area shows two tabs: 'System assigned' (which is selected) and 'User assigned'. A note explains that system-assigned managed identities are restricted to one per resource and are tied to the resource's lifecycle. It mentions Azure RBAC for granting permissions. Below this is a 'Status' switch, which is currently set to 'On'. At the bottom, there are 'Save', 'Discard', 'Refresh', and 'Got feedback?' buttons, with 'Save' also highlighted with a red box.

Add a user-assigned identity

Creating an app with a user-assigned identity requires that you create the identity and then add its resource identifier to your app config.

The screenshot shows the Azure portal for a 'Web App' named 'my-demo-app'. In the 'Identity' section, the 'User assigned' tab is selected, indicated by a red box. A modal window titled 'Add user assigned managed i...' is open, showing a list of user-assigned managed identities. One identity, 'test', is selected and highlighted with a red box. Below the list, there's a 'Selected identities:' section showing the chosen identity. At the bottom of the modal, there's a large blue 'Add' button, which is also highlighted with a red box.

First, you'll need to create a user-assigned identity resource.

1. Create a user-assigned managed identity resource according to [these instructions](#).
2. In the left navigation for your app's page, scroll down to the **Settings** group.
3. Select **Identity**.
4. Select **User assigned > Add**.
5. Search for the identity you created earlier, select it, and select **Add**.

Once you select **Add**, the app restarts.

Configure target resource

You may need to configure the target resource to allow access from your app or function. For example, if you [request a token](#) to access Key Vault, you must also add an access policy that includes the managed identity of your app or function. Otherwise, your calls to Key Vault will be rejected, even if you use a valid token. The same is true for Azure SQL Database. To learn more about which resources support Microsoft Entra tokens, see [Azure services that support Microsoft Entra authentication](#).

Important

The back-end services for managed identities maintain a cache per resource URI for around 24 hours. If you update the access policy of a particular target resource and immediately retrieve a token for that resource, you may continue to get a cached token with outdated permissions until that token expires. There's currently no way to force a token refresh.

Connect to Azure services in app code

With its managed identity, an app can obtain tokens for Azure resources that are protected by Microsoft Entra ID, such as Azure SQL Database, Azure Key Vault, and Azure Storage. These tokens represent the application accessing the resource, and not any specific user of the application.

App Service and Azure Functions provide an internally accessible [REST endpoint](#) for token retrieval. The REST endpoint can be accessed from within the app with a standard HTTP GET, which can be implemented with a generic HTTP client in every language. For .NET, JavaScript, Java, and Python, the Azure Identity client library provides an abstraction over this REST endpoint and simplifies the development experience. Connecting to other Azure services is as simple as adding a credential object to the service-specific client.

HTTP GET

A raw HTTP GET request looks like the following example:

HTTP

```
GET /MSI/token?resource=https://vault.azure.net&api-version=2019-08-01
HTTP/1.1
Host: localhost:4141
X-IDENTITY-HEADER: 853b9a84-5bfa-4b22-a3f3-0b9a43d9ad8a
```

And a sample response might look like the following:

HTTP

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "access_token": "eyJ0eXAi...",
    "expires_on": "1586984735",
    "resource": "https://vault.azure.net",
    "token_type": "Bearer",
    "client_id": "00001111-aaaa-2222-bbbb-3333cccc4444"
}
```

This response is the same as the [response for the Microsoft Entra service-to-service access token request](#). To access Key Vault, you will then add the value of `access_token` to a client connection with the vault.

For more information on the REST endpoint, see [REST endpoint reference](#).

Remove an identity

When you remove a system-assigned identity, it's deleted from Microsoft Entra ID. System-assigned identities are also automatically removed from Microsoft Entra ID when you delete the app resource itself.

Azure portal

1. In the left navigation of your app's page, scroll down to the **Settings** group.

2. Select **Identity**. Then follow the steps based on the identity type:

- **System-assigned identity:** Within the **System assigned** tab, switch **Status** to **Off**. Click **Save**.
- **User-assigned identity:** Select the **User assigned** tab, select the checkbox for the identity, and select **Remove**. Select **Yes** to confirm.

Note

There is also an application setting that can be set, WEBSITE_DISABLE_MSI, which just disables the local token service. However, it leaves the identity in place, and tooling will still show the managed identity as "on" or "enabled." As a result, use of this setting is not recommended.

REST endpoint reference

An app with a managed identity makes this endpoint available by defining two environment variables:

- **IDENTITY_ENDPOINT** - the URL to the local token service.
- **IDENTITY_HEADER** - a header used to help mitigate server-side request forgery (SSRF) attacks. The value is rotated by the platform.

The **IDENTITY_ENDPOINT** is a local URL from which your app can request tokens. To get a token for a resource, make an HTTP GET request to this endpoint, including the following parameters:

 Expand table

Parameter	In	Description
name		
resource	Query	The Microsoft Entra resource URI of the resource for which a token should be obtained. This could be one of the Azure services that support Microsoft Entra authentication or any other resource URI.
api-version	Query	The version of the token API to be used. Use <code>2019-08-01</code> .
X-IDENTITY-HEADER	Header	The value of the IDENTITY_HEADER environment variable. This header is used to help mitigate server-side request forgery (SSRF) attacks.
client_id	Query	(Optional) The client ID of the user-assigned identity to be used. Cannot be used on a request that includes <code>principal_id</code> , <code>mi_res_id</code> , or <code>object_id</code> . If all ID parameters (<code>client_id</code> , <code>principal_id</code> , <code>object_id</code> , and <code>mi_res_id</code>) are omitted, the system-assigned identity is used.
principal_id	Query	(Optional) The principal ID of the user-assigned identity to be used. <code>object_id</code> is an alias that may be used instead. Cannot be used on a request that includes <code>client_id</code> , <code>mi_res_id</code> , or <code>object_id</code> . If all ID

Parameter	In	Description
name		parameters (<code>client_id</code> , <code>principal_id</code> , <code>object_id</code> , and <code>mi_res_id</code>) are omitted, the system-assigned identity is used.
mi_res_id	Query	(Optional) The Azure resource ID of the user-assigned identity to be used. Cannot be used on a request that includes <code>principal_id</code> , <code>client_id</code> , or <code>object_id</code> . If all ID parameters (<code>client_id</code> , <code>principal_id</code> , <code>object_id</code> , and <code>mi_res_id</code>) are omitted, the system-assigned identity is used.

ⓘ Important

If you are attempting to obtain tokens for user-assigned identities, you must include one of the optional properties. Otherwise the token service will attempt to obtain a token for a system-assigned identity, which may or may not exist.

Next steps

- [Tutorial: Connect to SQL Database from App Service without secrets using a managed identity](#)
- [Access Azure Storage securely using a managed identity](#)
- [Call Microsoft Graph securely using a managed identity](#)
- [Connect securely to services with Key Vault secrets](#)

ⓘ Note: The author created this article with assistance from AI. [Learn more](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

Tutorial: Connect to Azure databases from App Service without secrets using a managed identity

Article • 09/30/2024

[App Service](#) provides a highly scalable, self-patching web hosting service in Azure. It also provides a [managed identity](#) for your app, which is a turn-key solution for securing access to Azure databases, including:

- [Azure SQL Database](#)
- [Azure Database for MySQL](#)
- [Azure Database for PostgreSQL](#)

Note

This tutorial doesn't include guidance for [Azure Cosmos DB](#), which supports Microsoft Entra authentication differently. For more information, see the Azure Cosmos DB documentation, such as [Use system-assigned managed identities to access Azure Cosmos DB data](#).

Managed identities in App Service make your app more secure by eliminating secrets from your app, such as credentials in the connection strings. This tutorial shows you how to connect to the above-mentioned databases from App Service using managed identities.

What you will learn:

- ✓ Configure a Microsoft Entra user as an administrator for your Azure database.
- ✓ Connect to your database as the Microsoft Entra user.
- ✓ Configure a system-assigned or user-assigned managed identity for an App Service app.
- ✓ Grant database access to the managed identity.
- ✓ Connect to the Azure database from your code (.NET Framework 4.8, .NET 6, Node.js, Python, Java) using a managed identity.
- ✓ Connect to the Azure database from your development environment using the Microsoft Entra user.

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

Prerequisites

- Create an app in App Service based on .NET, Node.js, Python, or Java.
- Create a database server with Azure SQL Database, Azure Database for MySQL, or Azure Database for PostgreSQL.
- You should be familiar with the standard connectivity pattern (with username and password) and be able to connect successfully from your App Service app to your database of choice.

Prepare your environment for the Azure CLI.

- Use the Bash environment in [Azure Cloud Shell](#). For more information, see [Quickstart for Bash in Azure Cloud Shell](#).

 [Launch Cloud Shell](#) 

- If you prefer to run CLI reference commands locally, [install](#) the Azure CLI. If you're running on Windows or macOS, consider running Azure CLI in a Docker container. For more information, see [How to run the Azure CLI in a Docker container](#).
 - 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 other sign-in options, see [Sign in with the Azure CLI](#).
 - When you're prompted, install the Azure CLI extension 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](#).

1. Install the Service Connector passwordless extension

Install the latest Service Connector passwordless extension for the Azure CLI:

Azure CLI

```
az extension add --name serviceconnector-passwordless --upgrade
```

 Note

Please check the extension "serviceconnector-passwordless" version is "2.0.2" or higher by running `az version`. You may need to upgrade Azure CLI first to upgrade the extension version.

2. Create a passwordless connection

Next, create a passwordless connection with Service Connector.

💡 Tip

The Azure portal can help you compose the commands below. In the Azure portal, go to your [Azure App Service](#) resource, select **Service Connector** from the left menu and select **Create**. Fill out the form with all required parameters. Azure automatically generates the connection creation command, which you can copy to use in the CLI or execute in Azure Cloud Shell.

Azure SQL Database

The following Azure CLI command uses a `--client-type` parameter.

1. Optionally run the `az webapp connection create sql -h` to get the supported client types.
2. Choose a client type and run the corresponding command. Replace the placeholders below with your own information.

User-assigned managed identity

Azure CLI

```
az webapp connection create sql \
--resource-group <group-name> \
--name <server-name> \
--target-resource-group <sql-group-name> \
--server <sql-name> \
--database <database-name> \
--user-identity client-id=<client-id> subs-id=
<subscription-id> \
--client-type <client-type>
```

This Service Connector command completes the following tasks in the background:

- Enable system-assigned managed identity, or assign a user identity for the app `<server-name>` hosted by Azure App Service.
- Set the Microsoft Entra admin to the current signed-in user.
- Add a database user for the system-assigned managed identity or user-assigned managed identity. Grant all privileges of the database `<database-name>` to this user. The username can be found in the connection string in preceding command output.
- Set configurations named `AZURE_MYSQL_CONNECTIONSTRING`, `AZURE_POSTGRESQL_CONNECTIONSTRING`, or `AZURE_SQL_CONNECTIONSTRING` to the Azure resource based on the database type.
- For App Service, the configurations are set in the **App Settings** blade.

If you encounter any problem when creating a connection, refer to [Troubleshooting](#) for help.

3. Modify your code

The screenshot shows the Azure portal interface for modifying code. At the top, there's a navigation bar with 'Azure SQL Database' and a '.NET' tab selected. Below this, a large text area contains two numbered steps:

1. Install dependencies.
A 'Bash' terminal window shows the command: `dotnet add package Microsoft.Data.SqlClient`.
2. Get the Azure SQL Database connection string from the environment variable added by Service Connector.
A 'C#' code editor window shows the following code:

```
C#
using Microsoft.Data.SqlClient;

// AZURE_SQL_CONNECTIONSTRING should be one of the following:
// For system-assigned managed identity: "Server=tcp:<server-
// name>.database.windows.net;Database=<database-
// name>;Authentication=Active Directory
Default;TrustServerCertificate=True"
// For user-assigned managed identity: "Server=tcp:<server-
// name>.database.windows.net;Database=<database-
```

```
name>;Authentication=Active Directory Default;User Id=<client-
id-of-user-assigned-identity>;TrustServerCertificate=True"

string connectionString =
Environment.GetEnvironmentVariable("AZURE_SQL_CONNECTIONSTRING"
)!;

using var connection = new SqlConnection(connectionString);
connection.Open();
```

For more information, see [Using Active Directory Managed Identity authentication](#).

For more information, see [Homepage for client programming to Microsoft SQL Server](#). For more code samples, see [Create a passwordless connection to a database service via Service Connector](#).

4. Set up your dev environment

This sample code uses `DefaultAzureCredential` to get a useable token for your Azure database from Microsoft Entra ID and then adds it to the database connection. While you can customize `DefaultAzureCredential`, it's already versatile by default. It gets a token from the signed-in Microsoft Entra user or from a managed identity, depending on whether you run it locally in your development environment or in App Service.

Without any further changes, your code is ready to be run in Azure. To debug your code locally, however, your develop environment needs a signed-in Microsoft Entra user. In this step, you configure your environment of choice by signing in with your Microsoft Entra user.

Visual Studio Windows

1. Visual Studio for Windows is integrated with Microsoft Entra authentication. To enable development and debugging in Visual Studio, add your Microsoft Entra user in Visual Studio by selecting **File > Account Settings** from the menu, and select **Sign in** or **Add**.

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

For more information about setting up your dev environment for Microsoft Entra authentication, see [Azure Identity client library for .NET](#).

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

5. Test and publish

1. Run your code in your dev environment. Your code uses the signed-in Microsoft Entra user in your environment to connect to the back-end database. The user can access the database because it's configured as a Microsoft Entra administrator for the database.
2. Publish your code to Azure using the preferred publishing method. In App Service, your code uses the app's managed identity to connect to the back-end database.

Frequently asked questions

- [Does managed identity support SQL Server?](#)
- [I get the error Login failed for user '<token-identified principal>'.](#)
- [I made changes to App Service authentication or the associated app registration. Why do I still get the old token?](#)
- [How do I add the managed identity to a Microsoft Entra group?](#)
- [I get the error SSL connection is required. Please specify SSL options and retry.](#)
- [I created my app with the Web App + Database template, and now I can't configure a managed identity connection with the Service Connector commands.](#)

Does managed identity support SQL Server?

Yes. For more information, see:

- [Microsoft Entra authentication for SQL Server](#)
- [Enable Microsoft Entra authentication for SQL Server on Azure VMs](#)

I get the error `Login failed for user '<token-identified principal>'`.

The managed identity you're attempting to request a token for is not authorized to access the Azure database.

I made changes to App Service authentication or the associated app registration. Why do I still get the old token?

The back-end services of managed identities also [maintain a token cache](#) that updates the token for a target resource only when it expires. If you modify the configuration *after* trying to get a token with your app, you don't actually get a new token with the updated permissions until the cached token expires. The best way to work around this is to test your changes with a new InPrivate (Edge)/private (Safari)/Incognito (Chrome) window. That way, you're sure to start from a new authenticated session.

How do I add the managed identity to a Microsoft Entra group?

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

Azure CLI

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

To grant database permissions for a Microsoft Entra group, see documentation for the respective database type.

I get the error **SSL connection is required. Please specify SSL options and retry.**

Connecting to the Azure database requires additional settings and is beyond the scope of this tutorial. For more information, see one of the following links:

[Configure TLS connectivity in Azure Database for PostgreSQL - Single Server](#) [Configure SSL connectivity in your application to securely connect to Azure Database for MySQL](#)

I created my app with the Web App + Database template, and now I can't configure a managed identity connection with the Service Connector commands.

Service Connector needs network access to the database in order to grant access for the app identity. When you create a secure-by-default app and database architecture in the Azure portal with the Web App + Database template, the architecture locks down network access to the database and only allows connections from within the virtual network. It's also true for Azure Cloud Shell. However, you can [deploy Cloud Shell in the virtual network](#), then run the Service Connector command in that Cloud Shell.

Next steps

What you learned:

- ✓ Configure a Microsoft Entra user as an administrator for your Azure database.
- ✓ Connect to your database as the Microsoft Entra user.
- ✓ Configure a system-assigned or user-assigned managed identity for an App Service app.
- ✓ Grant database access to the managed identity.
- ✓ Connect to the Azure database from your code (.NET Framework 4.8, .NET 6, Node.js, Python, Java) using a managed identity.
- ✓ Connect to the Azure database from your development environment using the Microsoft Entra user.

[How to use managed identities for App Service and Azure Functions](#)

[Tutorial: Connect to SQL Database from .NET App Service without secrets using a managed identity](#)

[Tutorial: Connect to Azure services that don't support managed identities \(using Key Vault\)](#)

[Tutorial: Isolate back-end communication with Virtual Network integration](#)

Feedback

Was this page helpful?

 Yes

 No

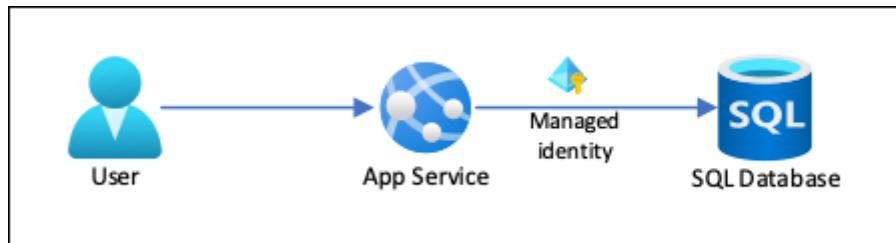
Tutorial: Connect to SQL Database from .NET App Service without secrets using a managed identity

Article • 04/17/2024

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

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

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



ⓘ Note

The steps covered in this tutorial support the following versions:

- .NET Framework 4.8 and above
- .NET 6.0 and above

For guidance for Azure Database for MySQL or Azure Database for PostgreSQL in other language frameworks (Node.js, Python, and Java), see [Tutorial: Connect to Azure databases from App Service without secrets using a managed identity](#).

What you will learn:

- ✓ Enable managed identities
- ✓ Grant SQL Database access to the managed identity

- ✓ Configure Entity Framework to use Microsoft Entra authentication with SQL Database
- ✓ Connect to SQL Database from Visual Studio using Microsoft Entra authentication

Note

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

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

Prerequisites

This article continues where you left off in either one of the following tutorials:

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

If you haven't already, follow one of the two tutorials first. Alternatively, you can adapt the steps for your own .NET app with SQL Database.

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

Prepare your environment for the Azure CLI.

- Use the Bash environment in [Azure Cloud Shell](#). For more information, see [Quickstart for Bash in Azure Cloud Shell](#).

A blue rectangular button with a white 'A' icon and the text 'Launch Cloud Shell'. To the right of the button is a small blue square with a white arrow pointing outwards, indicating a link.
- If you prefer to run CLI reference commands locally, [install](#) the Azure CLI. If you're running on Windows or macOS, consider running Azure CLI in a Docker container. For more information, see [How to run the Azure CLI in a Docker container](#).
 - 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 other sign-in options, see [Sign in with the Azure CLI](#).

- When you're prompted, install the Azure CLI extension 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`.

1. Grant database access to Microsoft Entra user

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

1. If your Microsoft Entra tenant doesn't have a user yet, create one by following the steps at [Add or delete users using Microsoft Entra ID](#).
2. Find the object ID of the Microsoft Entra user using the `az ad user list` and replace `<user-principal-name>`. The result is saved to a variable.

Azure CLI

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

💡 Tip

To see the list of all user principal names in Microsoft Entra ID, run `az ad user list --query '[].userPrincipalName'`.

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

Azure CLI

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

For more information on adding an Active Directory admin, see [Provision a Microsoft Entra administrator for your server](#)

2. Set up your dev environment

Visual Studio Windows

1. Visual Studio for Windows is integrated with Microsoft Entra authentication. To enable development and debugging in Visual Studio, add your Microsoft Entra user in Visual Studio by selecting **File > Account Settings** from the menu, and select **Sign in or Add**.
2. To set the Microsoft Entra user for Azure service authentication, select **Tools > Options** from the menu, then select **Azure Service Authentication > Account Selection**. Select the Microsoft Entra user you added and select **OK**.

For more information about setting up your dev environment for Microsoft Entra authentication, see [Azure Identity client library for .NET](#).

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

3. Modify your project

ⓘ Note

Microsoft.Azure.Services.AppAuthentication is no longer recommended to use with new Azure SDK. It is replaced with new **Azure Identity client library** available for .NET, Java, TypeScript and Python and should be used for all new development. Information about how to migrate to **Azure Identity** can be found here: [AppAuthentication to Azure.Identity Migration Guidance](#).

The steps you follow for your project depends on whether you're using [Entity Framework Core](#) (default for ASP.NET Core) or [Entity Framework](#) (default for ASP.NET).

Entity Framework Core

1. In Visual Studio, open the Package Manager Console and add the NuGet package [Microsoft.Data.SqlClient](#):

PowerShell

```
Install-Package Microsoft.Data.SqlClient -Version 5.1.0
```

2. In the [ASP.NET Core and SQL Database tutorial](#), the `MySqlConnection` connection string in `appsettings.json` isn't used at all yet. The local environment and the Azure environment both get connection strings from their respective environment variables in order to keep connection secrets out of the source file. But now with Active Directory authentication, there are no more secrets. In `appsettings.json`, replace the value of the `MySqlConnection` connection string with:

JSON

```
"Server=tcp:<server-name>.database.windows.net;Authentication=Active Directory Default;Database=<database-name>;"
```

⚠ Note

The [Active Directory Default](#) authentication type can be used both on your local machine and in Azure App Service. The driver attempts to acquire a token from Microsoft Entra ID using various means. If the app is deployed, it gets a token from the app's system-assigned managed identity. It can also authenticate with a user-assigned managed identity if you include: `User Id=<client-id-of-user-assigned-managed-identity>`; in your connection string. If the app is running locally, it tries to get a token from Visual Studio, Visual Studio Code, and Azure CLI.

That's everything you need to connect to SQL Database. When you debug in Visual Studio, your code uses the Microsoft Entra user you configured in [2. Set up your dev environment](#). You'll set up SQL Database later to allow connection from the managed identity of your App Service app. The `DefaultAzureCredential` class caches the token in memory and retrieves it from Microsoft Entra ID just before expiration. You don't need any custom code to refresh the token.

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

4. Use managed identity connectivity

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

ⓘ Note

While the instructions in this section are for a system-assigned identity, a user-assigned identity can just as easily be used. To do this, you would need to change the `az webapp identity assign` command to assign the desired user-assigned identity. Then, when creating the SQL user, make sure to use the name of the user-assigned identity resource rather than the site name.

Enable managed identity on app

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

Azure CLI

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

ⓘ Note

To enable managed identity for a [deployment slot](#), add `--slot <slot-name>` and use the name of the slot in `<slot-name>`.

Here's an example of the output:

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

Grant permissions to managed identity

ⓘ Note

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

Azure CLI

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

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

Bash

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

2. In the SQL prompt for the database you want, run the following commands to grant the minimum permissions your app needs. For example,

SQL

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

<identity-name> is the name of the managed identity in Microsoft Entra ID. If the identity is system-assigned, the name is always the same as the name of your App Service app. For a [deployment slot](#), the name of its system-assigned identity is *<app-name>/slots/<slot-name>*. To grant permissions for a Microsoft Entra group, use the group's display name instead (for example, *myAzureSQLDBAccessGroup*).

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

(!) Note

The back-end services of managed identities also [maintains a token cache](#) that updates the token for a target resource only when it expires. If you make a mistake configuring your SQL Database permissions and try to modify the permissions *after* trying to get a token with your app, you don't actually get a new token with the updated permissions until the cached token expires.

ⓘ Note

Microsoft Entra ID and managed identities are not supported for on-premises SQL Server.

Modify connection string

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

Azure CLI

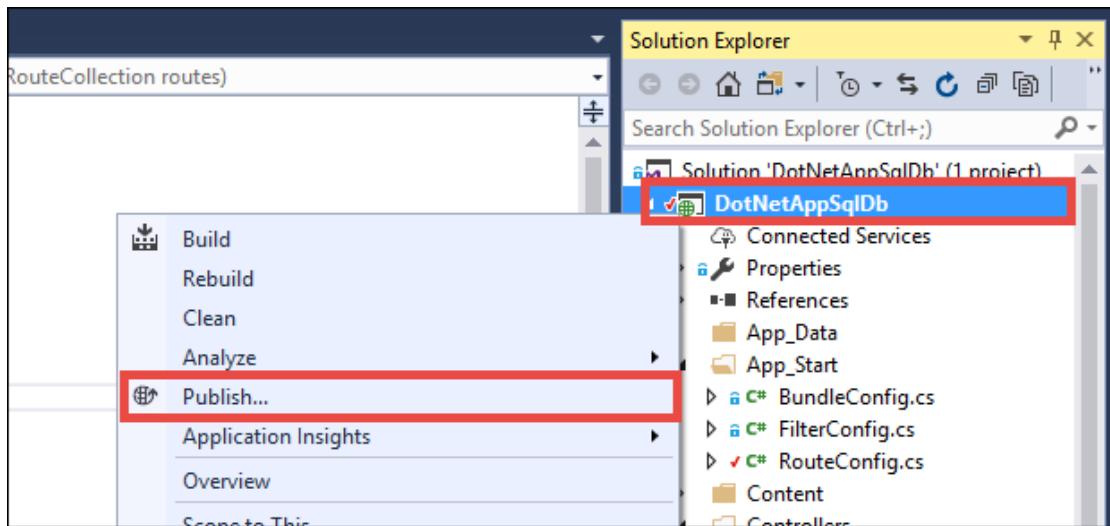
```
az webapp config connection-string delete --resource-group myResourceGroup -  
-name <app-name> --setting-names MyDbConnection
```

5. Publish your changes

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

ASP.NET

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



2. In the publish page, select Publish.

(i) Important

Ensure that your app service name doesn't match with any existing [App Registrations](#). This will lead to Principal ID conflicts.

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

A screenshot of a web browser window showing the 'Index - My ASP.NET App' page. The URL in the address bar is 'dotnetappsql1234.azurewebsites.net'. The page title is 'My TodoList App'. Below the title, the word 'Todos' is displayed in large font. A 'Create New' button is visible. A table lists three items: 'Deploy app to Azure' (Created Date: 2017-06-01, Done: checked, links to Edit | Details | Delete), 'Walk dog' (Created Date: 2017-06-03, Done: unchecked, links to Edit | Details | Delete), and 'Feed cat' (Created Date: 2017-06-04, Done: unchecked, links to Edit | Details | Delete). At the bottom of the page, a copyright notice reads '© 2017 - My ASP.NET Application'.

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

Clean up resources

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

Azure CLI

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

This command may take a minute to run.

Next steps

What you learned:

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

[Secure with custom domain and certificate](#)

[Tutorial: Connect an App Service app to SQL Database on behalf of the signed-in user](#)

[Tutorial: Connect to Azure databases from App Service without secrets using a managed identity](#)

[Tutorial: Connect to Azure services that don't support managed identities \(using Key Vault\)](#)

[Tutorial: Isolate back-end communication with Virtual Network integration](#)

Tutorial: Connect to a PostgreSQL Database from Java Tomcat App Service without secrets using a managed identity

Article • 06/05/2024

Azure App Service provides a highly scalable, self-patching web hosting service in Azure. It also provides a [managed identity](#) for your app, which is a turn-key solution for securing access to [Azure Database for PostgreSQL](#) and other Azure services. Managed identities in App Service make your app more secure by eliminating secrets from your app, such as credentials in the environment variables. In this tutorial, you learn how to:

- ✓ Create a PostgreSQL database.
- ✓ Deploy the sample app to Azure App Service on Tomcat using WAR packaging.
- ✓ Configure a Tomcat web application to use Microsoft Entra authentication with PostgreSQL Database.
- ✓ Connect to PostgreSQL Database with Managed Identity using Service Connector.

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

Prerequisites

- [Git](#)
- [Java JDK](#)
- [Maven](#)
- [Azure CLI](#) version 2.45.0 or higher.

Clone the sample app and prepare the repo

Run the following commands in your terminal to clone the sample repo and set up the sample app environment.

Bash

```
git clone https://github.com/Azure-Samples>Passwordless-Connections-for-Java-Apps  
cd Passwordless-Connections-for-Java-Apps/Tomcat/
```

Create an Azure Database for PostgreSQL

Follow these steps to create an Azure Database for Postgres in your subscription. The Tomcat app connects to this database and store its data when running, persisting the application state no matter where you run the application.

1. Sign into the Azure CLI, and optionally set your subscription if you have more than one connected to your login credentials.

```
Azure CLI
```

```
az login  
az account set --subscription <subscription-ID>
```

2. Create an Azure Resource Group, noting the resource group name.

```
Azure CLI
```

```
export RESOURCE_GROUP=<resource-group-name>  
export LOCATION=eastus  
  
az group create --name $RESOURCE_GROUP --location $LOCATION
```

3. Create an Azure Database for PostgreSQL server. The server is created with an administrator account, but it isn't used because we're going to use the Microsoft Entra admin account to perform administrative tasks.

```
Azure CLI
```

```
export POSTGRESQL_ADMIN_USER=azureuser  
# PostgreSQL admin access rights won't be used because Azure AD  
authentication is leveraged to administer the database.  
export POSTGRESQL_ADMIN_PASSWORD=<admin-password>  
export POSTGRESQL_HOST=<postgresql-host-name>  
  
# Create a PostgreSQL server.  
az postgres flexible-server create \  
    --resource-group $RESOURCE_GROUP \  
    --name $POSTGRESQL_HOST \  
    --location $LOCATION \  
    --admin-user $POSTGRESQL_ADMIN_USER \  
    --admin-password $POSTGRESQL_ADMIN_PASSWORD \  
    --public-access 0.0.0.0 \  
    --sku-name Standard_D2s_v3
```

4. Create a database for the application.

Azure CLI

```
export DATABASE_NAME=checklist

az postgres flexible-server db create \
--resource-group $RESOURCE_GROUP \
--server-name $POSTGRESQL_HOST \
--database-name $DATABASE_NAME
```

Deploy the application to App Service

Follow these steps to build a WAR file and deploy to Azure App Service on Tomcat using a WAR packaging.

1. The sample app contains a *pom.xml* file that can generate the WAR file. Run the following command to build the app.

Bash

```
mvn clean package -f pom.xml
```

2. Create an Azure App Service resource on Linux using Tomcat 9.0.

Azure CLI

```
export APPSERVICE_PLAN=<app-service-plan>
export APPSERVICE_NAME=<app-service-name>
# Create an App Service plan
az appservice plan create \
--resource-group $RESOURCE_GROUP \
--name $APPSERVICE_PLAN \
--location $LOCATION \
--sku B1 \
--is-linux

# Create an App Service resource.
az webapp create \
--resource-group $RESOURCE_GROUP \
--name $APPSERVICE_NAME \
--plan $APPSERVICE_PLAN \
--runtime "TOMCAT:10.0-java11"
```

3. Deploy the WAR package to App Service.

Azure CLI

```
az webapp deploy \
--resource-group $RESOURCE_GROUP \
--name $APPSCERVICE_NAME \
--src-path target/app.war \
--type war
```

Connect the Postgres database with identity connectivity

Next, connect the database using [Service Connector](#).

Install the Service Connector passwordless extension for the Azure CLI:

Azure CLI

```
az extension add --name serviceconnector-passwordless --upgrade
```

Then, connect your app to a Postgres database with a system-assigned managed identity using Service Connector.

To make this connection, run the [az webapp connection create](#) command.

Azure CLI

```
az webapp connection create postgres-flexible \
--resource-group $RESOURCE_GROUP \
--name $APPSCERVICE_NAME \
--target-resource-group $RESOURCE_GROUP \
--server $POSTGRESQL_HOST \
--database $DATABASE_NAME \
--system-identity \
--client-type java
```

This command creates a connection between your web app and your PostgreSQL server, and manages authentication through a system-assigned managed identity.

Next, update App Settings and add plugin in connection string

Azure CLI

```
export AZURE_POSTGRESQL_CONNECTIONSTRING=$(\
az webapp config appsettings list \
--resource-group $RESOURCE_GROUP \
--name $APPSCERVICE_NAME \
| jq -c -r '.[.] \
```

```
| select ( .name == "AZURE_POSTGRESQL_CONNECTIONSTRING" ) \ 
| .value')

az webapp config appsettings set \
--resource-group $RESOURCE_GROUP \
--name $APPSERVICE_NAME \
--settings 'CATALINA_OPTS=-
DdbUrl=""${AZURE_POSTGRESQL_CONNECTIONSTRING}"&authenticationPluginClassName=com.azure.identity.extensions.jdbc.postgresql.AzurePostgresqlAuthenticationPlugin'"
```

Test the sample web app

Run the following command to test the application.

Bash

```
export WEBAPP_URL=$(az webapp show \
--resource-group $RESOURCE_GROUP \
--name $APPSERVICE_NAME \
--query defaultHostName \
--output tsv)

# Create a list
curl -X POST -H "Content-Type: application/json" -d '{"name": "list1", "date": "2022-03-21T00:00:00", "description": "Sample checklist"}' https://$WEBAPP_URL/checklist

# Create few items on the list 1
curl -X POST -H "Content-Type: application/json" -d '{"description": "item 1"}' https://$WEBAPP_URL/checklist/1/item
curl -X POST -H "Content-Type: application/json" -d '{"description": "item 2"}' https://$WEBAPP_URL/checklist/1/item
curl -X POST -H "Content-Type: application/json" -d '{"description": "item 3"}' https://$WEBAPP_URL/checklist/1/item

# Get all lists
curl https://$WEBAPP_URL/checklist

# Get list 1
curl https://$WEBAPP_URL/checklist/1
```

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:

Azure CLI

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

This command may take a minute to run.

Next steps

Learn more about running Java apps on App Service on Linux in the developer guide.

[Java in App Service Linux dev guide](#)

Learn how to secure your app with a custom domain and certificate.

[Secure with custom domain and certificate](#)

Create and deploy a Flask Python web app to Azure with system-assigned managed identity

Article • 09/23/2024

In this tutorial, you deploy Python [Flask](#) code to create and deploy a web app running in Azure App Service. The web app uses its system-assigned [managed identity](#) (passwordless connections) with Azure role-based access control to access [Azure Storage](#) and [Azure Database for PostgreSQL - Flexible Server](#) resources. The code uses the `DefaultAzureCredential` class of the [Azure Identity client library for Python](#). The `DefaultAzureCredential` class automatically detects that a managed identity exists for the App Service and uses it to access other Azure resources.

You can configure passwordless connections to Azure services using Service Connector or you can configure them manually. This tutorial shows how to use Service Connector. For more information about passwordless connections, see [Passwordless connections for Azure services](#). For information about Service Connector, see the [Service Connector documentation](#).

This tutorial shows you how to create and deploy a Python web app using the Azure CLI. The commands in this tutorial are written to be run in a Bash shell. You can run the tutorial commands in any Bash environment with the CLI installed, such as your local environment or the [Azure Cloud Shell](#). With some modification -- for example, setting and using environment variables -- you can run these commands in other environments like Windows command shell. For examples of using a user-assigned managed identity, see [Create and deploy a Django web app to Azure with a user-assigned managed identity](#).

Get the sample app

A sample Python application using the Flask framework are available to help you follow along with this tutorial. Download or clone one of the sample applications to your local workstation.

1. Clone the sample in an Azure Cloud Shell session.

Console

```
git clone https://github.com/Azure-Samples/msdocs-flask-web-app-
```

```
managed-identity.git
```

2. Navigate to the application folder.

```
Console
```

```
cd msdocs-flask-web-app-managed-identity
```

Examine authentication code

The sample web app needs to authenticate to two different data stores:

- Azure blob storage server where it stores and retrieves photos submitted by reviewers.
- An Azure Database for PostgreSQL - Flexible Server database where it stores restaurants and reviews.

It uses `DefaultAzureCredential` to authenticate to both data stores. With `DefaultAzureCredential`, the app can be configured to run under the identity of different service principals, depending on the environment it's running in, without making changes to code. For example, in a local development environment, the app can run under the identity of the developer signed in to the Azure CLI, while in Azure, as in this tutorial, it can run under its system-assigned managed identity.

In either case, the security principal that the app runs under must have a role on each Azure resource the app uses that permits it to perform the actions on the resource that the app requires. In this tutorial, you use service connectors to automatically enable the system-assigned managed identity on your app in Azure and to assign that identity appropriate roles on your Azure storage account and Azure Database for PostgreSQL server.

After the system-assigned managed identity is enabled and is assigned appropriate roles on the data stores, you can use `DefaultAzureCredential` to authenticate with the required Azure resources.

The following code is used to create a blob storage client to upload photos in `app.py`. An instance of `DefaultAzureCredential` is supplied to the client, which it uses to acquire access tokens to perform operations against Azure storage.

```
Python
```

```
from azure.identity import DefaultAzureCredential
from azure.storage.blob import BlobServiceClient
```

```
azure_credential = DefaultAzureCredential()
blob_service_client = BlobServiceClient(
    account_url=account_url,
    credential=azure_credential)
```

An instance of `DefaultAzureCredential` is also used to get an access token for Azure Database for PostgreSQL in `./azureproject/get_conn.py`. In this case, the token is acquired directly by calling `get_token` on the credential instance and passing it the appropriate `scope` value. The token is then used in the place of the password in the PostgreSQL connection URI returned to the caller.

Python

```
azure_credential = DefaultAzureCredential()
token = azure_credential.get_token("https://osssrdbms-
aad.database.windows.net")
conn =
str(current_app.config.get('DATABASE_URI')).replace('PASSWORDORTOKEN',
token.token)
```

To learn more about authenticating your apps with Azure services, see [Authenticate Python apps to Azure services by using the Azure SDK for Python](#). To learn more about `DefaultAzureCredential`, including how to customize the credential chain it evaluates for your environment, see [DefaultAzureCredential overview](#).

Create an Azure PostgreSQL server

1. Set up the environment variables needed for the tutorial.

Bash

```
LOCATION="eastus"
RAND_ID=$RANDOM
RESOURCE_GROUP_NAME="msdocs-mi-web-app"
APP_SERVICE_NAME="msdocs-mi-web-$RAND_ID"
DB_SERVER_NAME="msdocs-mi-postgres-$RAND_ID"
ADMIN_USER="demoadmin"
ADMIN_PW="ChAnG33#ThsPssWD$RAND_ID"
```

Important

The `ADMIN_PW` must contain 8 to 128 characters from three of the following categories: English uppercase letters, English lowercase letters, numbers, and

nonalphanumeric characters. When creating usernames or passwords **do not** use the `$` character. Later you create environment variables with these values where the `$` character has special meaning within the Linux container used to run Python apps.

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

```
Azure CLI
```

```
az group create --location $LOCATION --name $RESOURCE_GROUP_NAME
```

3. Create a PostgreSQL server with the [az postgres flexible-server create](#) command.
(This and subsequent commands use the line continuation character for Bash Shell ('`\`). Change the line continuation character for your shell if needed.)

```
Azure CLI
```

```
az postgres flexible-server create \
--resource-group $RESOURCE_GROUP_NAME \
--name $DB_SERVER_NAME \
--location $LOCATION \
--admin-user $ADMIN_USER \
--admin-password $ADMIN_PW \
--sku-name Standard_D2ds_v4
```

The *sku-name* is the name of the pricing tier and compute configuration. For more information, see [Azure Database for PostgreSQL pricing](#). To list available SKUs, use `az postgres flexible-server list-skus --location $LOCATION`.

4. Create a database named `restaurant` using the [az postgres flexible-server execute](#) command.

```
Azure CLI
```

```
az postgres flexible-server execute \
--name $DB_SERVER_NAME \
--admin-user $ADMIN_USER \
--admin-password $ADMIN_PW \
--database-name postgres \
--querytext 'create database restaurant;'
```

Create an Azure App Service and deploy the code

1. Create an app service using the [az webapp up](#) command.

```
Azure CLI
```

```
az webapp up \
--resource-group $RESOURCE_GROUP_NAME \
--name $APP_SERVICE_NAME \
--runtime PYTHON:3.9 \
--sku B1
```

The *sku* defines the size (CPU, memory) and cost of the app service plan. The B1 (Basic) service plan incurs a small cost in your Azure subscription. For a full list of App Service plans, view the [App Service pricing](#) page.

2. Configure App Service to use the *start.sh* in the repo with the [az webapp config set](#) command.

```
Azure CLI
```

```
az webapp config set \
--resource-group $RESOURCE_GROUP_NAME \
--name $APP_SERVICE_NAME \
--startup-file "start.sh"
```

Create passwordless connectors to Azure resources

The Service Connector commands configure Azure Storage and Azure Database for PostgreSQL resources to use managed identity and Azure role-based access control. The commands create app settings in the App Service that connect your web app to these resources. The output from the commands lists the service connector actions taken to enable passwordless capability.

1. Add a PostgreSQL service connector with the [az webapp connection create postgres-flexible](#) command. The system-assigned managed identity is used to authenticate the web app to the target resource, PostgreSQL in this case.

```
Azure CLI
```

```
az webapp connection create postgres-flexible \
--resource-group $RESOURCE_GROUP_NAME \
--name $APP_SERVICE_NAME \
--target-resource-group $RESOURCE_GROUP_NAME \
--server $DB_SERVER_NAME \
--database restaurant \
```

```
--client-type python \
--system-identity
```

2. Add a storage service connector with the [az webapp connection create storage-blob](#) command.

This command also adds a storage account and adds the web app with role *Storage Blob Data Contributor* to the storage account.

Azure CLI

```
STORAGE_ACCOUNT_URL=$(az webapp connection create storage-blob \
    --new true \
    --resource-group $RESOURCE_GROUP_NAME \
    --name $APP_SERVICE_NAME \
    --target-resource-group $RESOURCE_GROUP_NAME \
    --client-type python \
    --system-identity \
    --query configurations[].value \
    --output tsv)
STORAGE_ACCOUNT_NAME=$(cut -d . -f1 <<< $(cut -d / -f3 <<<
$STORAGE_ACCOUNT_URL))
```

Create a container in the storage account

The sample Python app stores photos submitted by reviewers as blobs in a container in your storage account.

- When a user submits a photo with their review, the sample app writes the image to the container using its system-assigned managed identity for authentication and authorization. You configured this functionality in the last section.
- When a user views the reviews for a restaurant, the app returns a link to the photo in blob storage for each review that has one associated with it. For the browser to display the photo, it must be able to access it in your storage account. The blob data must be available for read publicly through anonymous (unauthenticated) access.

To enhance security, storage accounts are created with anonymous access to blob data disabled by default. In this section, you enable anonymous read access on your storage account and then create a container named *photos* that provides public (anonymous) access to its blobs.

1. Update the storage account to allow anonymous read access to blobs with the [az storage account update](#) command.

Azure CLI

```
az storage account update \
--name $STORAGE_ACCOUNT_NAME \
--resource-group $RESOURCE_GROUP_NAME \
--allow-blob-public-access true
```

Enabling anonymous access on the storage account doesn't affect access for individual blobs. You must explicitly enable public access to blobs at the container-level.

2. Create a container called *photos* in the storage account with the [az storage container create](#) command. Allow anonymous read (public) access to blobs in the newly created container.

Azure CLI

```
az storage container create \
--account-name $STORAGE_ACCOUNT_NAME \
--name photos \
--public-access blob \
--account-key $(az storage account keys list --account-name
$STORAGE_ACCOUNT_NAME \
--query [0].value --output tsv)
```

① Note

For brevity, this command uses the storage account key to authorize with the storage account. For most scenarios, Microsoft's recommended approach is to use Microsoft Entra ID and Azure (RBAC) roles. For a quick set of instructions, see [Quickstart: Create, download, and list blobs with Azure CLI](#). Note that several Azure roles permit you to create containers in a storage account, including "Owner", "Contributor", "Storage Blob Data Owner", and "Storage Blob Data Contributor".

To learn more about anonymous read access to blob data, see [Configure anonymous read access for containers and blobs](#).

Test the Python web app in Azure

The sample Python app uses the [azure.identity](#) package and its `DefaultAzureCredential` class. When the app is running in Azure, `DefaultAzureCredential` automatically detects if a managed identity exists for the App

Service and, if so, uses it to access other Azure resources (storage and PostgreSQL in this case). There's no need to provide storage keys, certificates, or credentials to the App Service to access these resources.

1. Browse to the deployed application at the URL

```
http://$APP_SERVICE_NAME.azurewebsites.net.
```

It can take a minute or two for the app to start. If you see a default app page that isn't the default sample app page, wait a minute and refresh the browser.

2. Test the functionality of the sample app by adding a restaurant and some reviews with photos for the restaurant.

The restaurant and review information is stored in Azure Database for PostgreSQL and the photos are stored in Azure Storage. Here's an example screenshot:

The screenshot shows a web application titled "Azure Restaurant Review". At the top, there is a logo and a link to "Azure Docs". Below the title, the name of the restaurant is "Contoso Café". Underneath the name, there are three details: "Street address: 1 Main Street", "Description: Nice coffee house.", and "Rating: ★★★★ 1 4.5 (2 reviews)". A "Reviews" section follows, featuring a green "Add new review" button. Below this, a table lists two reviews:

Date	User	Rating	Review	Photo
May 20, 2022, 10:12 a.m.	Davide Sagese	5	Friendly staff, good coffee.	
May 23, 2022, 5:24 p.m.	Francesca Lombo	4	Good breakfast choice.	

Clean up

In this tutorial, all the Azure resources were created in the same resource group. Removing the resource group removes with the `az group delete` command removes all resources in the resource group and is the fastest way to remove all Azure resources used for your app.

```
Azure CLI
```

```
az group delete --name $RESOURCE_GROUP_NAME
```

You can optionally add the `--no-wait` argument to allow the command to return before the operation is complete.

Next steps

- Create and deploy a Django web app to Azure with a user-assigned managed identity
 - Deploy a Python (Django or Flask) web app with PostgreSQL in Azure App Service
-

Feedback

Was this page helpful?

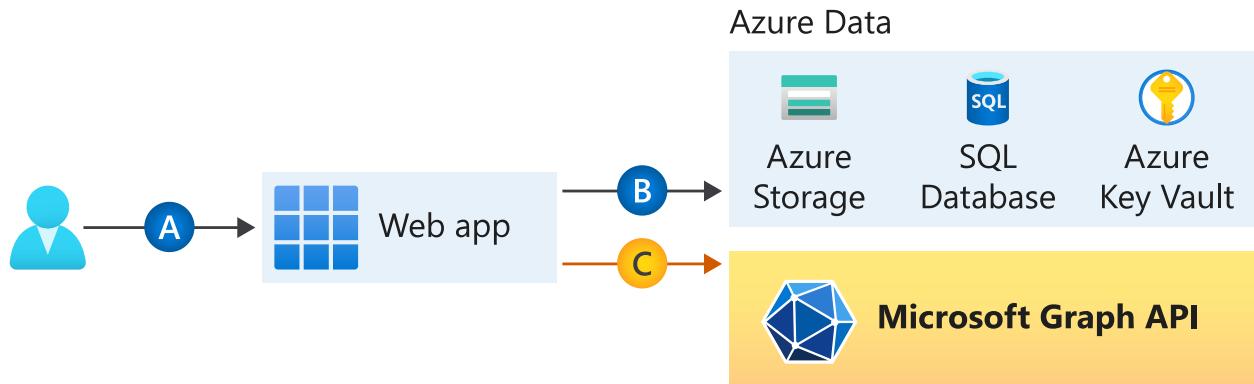


[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

Tutorial: Access Microsoft Graph from a secured .NET app as the app

Article • 08/18/2023

Learn how to access Microsoft Graph from a web app running on Azure App Service.



You want to call Microsoft Graph for the web app. A safe way to give your web app access to data is to use a [system-assigned managed identity](#). A managed identity from Azure Active Directory allows App Service to access resources through role-based access control (RBAC), without requiring app credentials. After assigning a managed identity to your web app, Azure takes care of the creation and distribution of a certificate. You don't have to worry about managing secrets or app credentials.

In this tutorial, you learn how to:

- ✓ Create a system-assigned managed identity on a web app.
- ✓ Add Microsoft Graph API permissions to a managed identity.
- ✓ Call Microsoft Graph from a web app by using managed identities.

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

Prerequisites

- A web application running on Azure App Service that has the [App Service authentication/authorization module enabled](#).

Enable managed identity on app

If you create and publish your web app through Visual Studio, the managed identity was enabled on your app for you.

1. In your app service, select **Identity** in the left pane and then select **System assigned**.
2. Verify that **Status** is set to **On**. If not, select **Save** and then select **Yes** to enable the system-assigned managed identity. When the managed identity is enabled, the status is set to **On** and the object ID is available.
3. Take note of the **Object ID** value, which you'll need in the next step.

SecureWebApp20200924113531 | Identity

System assigned User assigned

A system assigned managed identity enables Azure resources to authenticate. Necessary permissions can be granted via Azure role-based-access-control resource (e.g. Virtual Machine) can only have one system assigned managed identity.

Status **Off** **On**

Object ID **09382857-...-4138df10bfa2**

Permissions **Azure role assignments**

This resource is registered with Azure Active Directory. You can control it

Grant access to Microsoft Graph

When accessing the Microsoft Graph, the managed identity needs to have proper permissions for the operation it wants to perform. Currently, there's no option to assign such permissions through the Microsoft Entra admin center.

1. Run the following script to add the requested Microsoft Graph API permissions to the managed identity service principal object.

```
PowerShell
PowerShell
# Install the module.
# Install-Module Microsoft.Graph -Scope CurrentUser
```

```

# The tenant ID
$TenantId = "11111111-1111-1111-1111-111111111111"

# The name of your web app, which has a managed identity.
$webAppName = "SecureWebApp-20201106120003"
$resourceGroupName = "SecureWebApp-20201106120003ResourceGroup"

# The name of the app role that the managed identity should be
# assigned to.
$appRoleName = "User.Read.All"

# Get the web app's managed identity's object ID.
Connect-AzAccount -Tenant $TenantId
$managedIdentityObjectId = (Get-AzWebApp -ResourceGroupName
$resourceGroupName -Name $webAppName).identity.principalId

Connect-MgGraph -TenantId $TenantId -Scopes
'Application.Read.All', 'AppRoleAssignment.ReadWrite.All'

# Get Microsoft Graph app's service principal and app role.
$serverApplicationName = "Microsoft Graph"
$serverServicePrincipal = (Get-MgServicePrincipal -Filter
"DisplayName eq '$serverApplicationName'")
$serverServicePrincipalObjectId = $serverServicePrincipal.Id

$appRoleId = ($serverServicePrincipal.AppRoles | Where-Object
{$_ .Value -eq $appRoleName }).Id

# Assign the managed identity access to the app role.
New-MgServicePrincipalAppRoleAssignment `-
-ServicePrincipalId $managedIdentityObjectId `-
-PrincipalId $managedIdentityObjectId `-
-ResourceId $serverServicePrincipalObjectId `-
-AppRoleId $appRoleId

```

2. After executing the script, you can verify in the [Microsoft Entra admin center](#) that the requested API permissions are assigned to the managed identity.
3. Go to **Applications**, and then select **Enterprise applications**. This pane displays all the service principals in your tenant. **Add a filter** for "Application type==Managed identities" and select the service principal for the managed identity.

If you're following this tutorial, there are two service principals with the same display name (SecureWebApp2020094113531, for example). The service principal that has a **Homepage URL** represents the web app in your tenant. The service principal that appears in **Managed Identities** should *not* have a **Homepage URL** listed and the **Object ID** should match the object ID value of the managed identity in the [previous step](#).

4. Select the service principal for the managed identity.

The screenshot shows the 'Enterprise applications | All applications' page in the Azure portal. The left sidebar has sections for Overview, Manage, and Security. The main area has filters for Application type (set to 'Managed Identities'), Applications status (Any), and Application visibility (Any). A search bar is present above the table. The table lists two applications: 'SecureWebApp20200924113531' and 'SecureWebApp20200924113531'. The first row is selected, highlighting its name, object ID, and application ID.

Name	Homepage URL	Object ID	Application ID
SecureWebApp20200924113531	https://securewebapp20200924...	09382857-ed3f-4cbd-b524-4138df10bfa2	1e448214-461a-4efc-b031-454...
SecureWebApp20200924113531	https://securewebapp20200924...	3c52e043-be96-4a9f-b950-d4137eba91de	547082a3-7171-465b-a59d-9c6...

5. In Overview, select Permissions, and you'll see the added permissions for Microsoft Graph.

The screenshot shows the 'SecureWebApp20200924113531 | Permissions' page. The left sidebar includes sections for Overview, Deployment Plan, Diagnose and solve problems, Manage (Properties, Owners, Roles and administrators, Users and groups, Single sign-on, Provisioning, Self-service), Security (Permissions, Token encryption), Activity, and Sign-ins. The main area has a 'Permissions' section with a description of how admin consent works. It shows that admin consent is disabled for this app. Below this is a table of permissions granted to Microsoft Graph.

API Name	Permission	Type	Granted through	Granted by
Microsoft Graph	Read all users' full profil...	Application	Admin consent	An administrator

Call Microsoft Graph

The [ChainedTokenCredential](#), [ManagedIdentityCredential](#), and [EnvironmentCredential](#) classes are used to get a token credential for your code to authorize requests to Microsoft Graph. Create an instance of the [ChainedTokenCredential](#) class, which uses the managed identity in the App Service environment or the development environment variables to fetch tokens and attach them to the service client. The following code example gets the authenticated token credential and uses it to create a service client object, which gets the users in the group.

To see this code as part of a sample application, see the:

- Sample on GitHub [↗](#).

Install the Microsoft.Identity.Web.MicrosoftGraph client library package

Install the [Microsoft.Identity.Web.MicrosoftGraph NuGet package ↗](#) in your project by using the .NET Core command-line interface or the Package Manager Console in Visual Studio.

.NET Core command-line

Open a command line, and switch to the directory that contains your project file.

Run the install commands.

.NET CLI

```
dotnet add package Microsoft.Identity.Web.MicrosoftGraph  
dotnet add package Microsoft.Graph
```

Package Manager Console

Open the project/solution in Visual Studio, and open the console by using the **Tools > NuGet Package Manager > Package Manager Console** command.

Run the install commands.

PowerShell

```
Install-Package Microsoft.Identity.Web.MicrosoftGraph  
Install-Package Microsoft.Graph
```

.NET Example

C#

```
using System;  
using System.Collections.Generic;  
using System.Threading.Tasks;  
using Microsoft.AspNetCore.Mvc.RazorPages;  
using Microsoft.Extensions.Logging;  
using Microsoft.Graph;  
using Azure.Identity;
```

```

...
public IList<MSGraphUser> Users { get; set; }

public async Task OnGetAsync()
{
    // Create the Graph service client with a ChainedTokenCredential which
    gets an access
    // token using the available Managed Identity or environment variables
    if running
        // in development.
    var credential = new ChainedTokenCredential(
        new ManagedIdentityCredential(),
        new EnvironmentCredential());

    string[] scopes = new[] { "https://graph.microsoft.com/.default" };

    var graphServiceClient = new GraphServiceClient(
        credential, scopes);

    List<MSGraphUser> msGraphUsers = new List<MSGraphUser>();
    try
    {
        //var users = await graphServiceClient.Users.Request().GetAsync();
        var users = await graphServiceClient.Users.GetAsync();
        foreach (var u in users.Value)
        {
            MSGraphUser user = new MSGraphUser();
            user.userPrincipalName = u.UserPrincipalName;
            user.displayName = u.DisplayName;
            user.mail = u.Mail;
            user.jobTitle = u.JobTitle;

            msGraphUsers.Add(user);
        }
    }
    catch (Exception ex)
    {
        string msg = ex.Message;
    }

    Users = msGraphUsers;
}

```

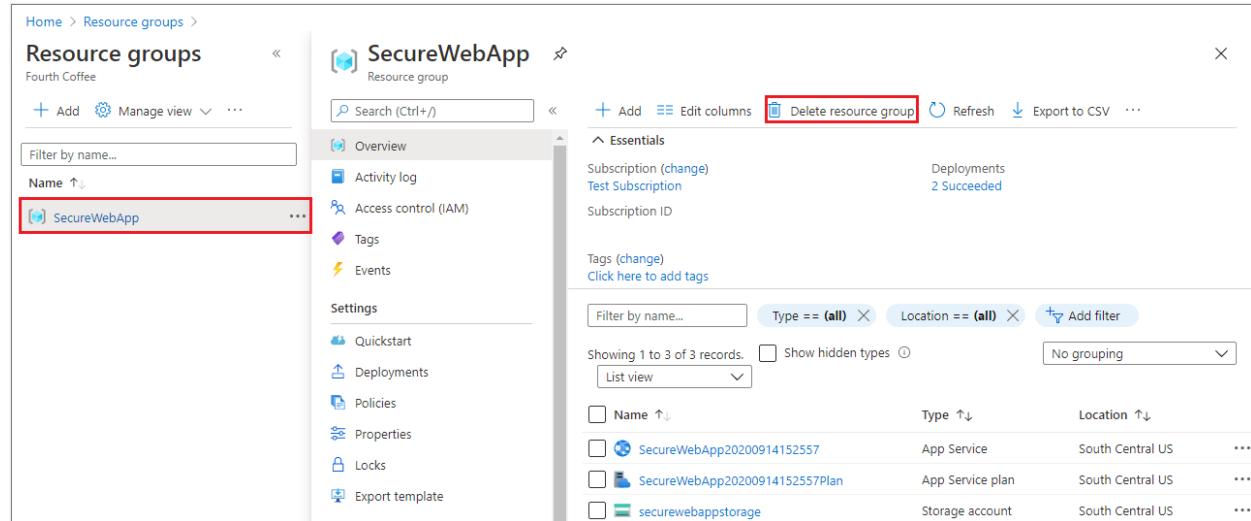
Clean up resources

If you're finished with this tutorial and no longer need the web app or associated resources, clean up the resources you created.

Delete the resource group

In the [Azure portal](#), select **Resource groups** from the portal menu and select the resource group that contains your app service and app service plan.

Select **Delete resource group** to delete the resource group and all the resources.



The screenshot shows the Azure Resource Groups blade. On the left, there's a list of resource groups under 'Fourth Coffee', with 'SecureWebApp' selected and highlighted by a red box. On the right, the details for 'SecureWebApp' are shown, including its subscription information ('Test Subscription'), deployment status ('2 Succeeded'), and resource list. The 'Delete resource group' button at the top of the main content area is also highlighted with a red box.

Name	Type	Location	Actions
SecureWebApp20200914152557	App Service	South Central US	...
SecureWebApp20200914152557Plan	App Service plan	South Central US	...
securewebappstorage	Storage account	South Central US	...

This command might take several minutes to run.

Next steps

In this tutorial, you learned how to:

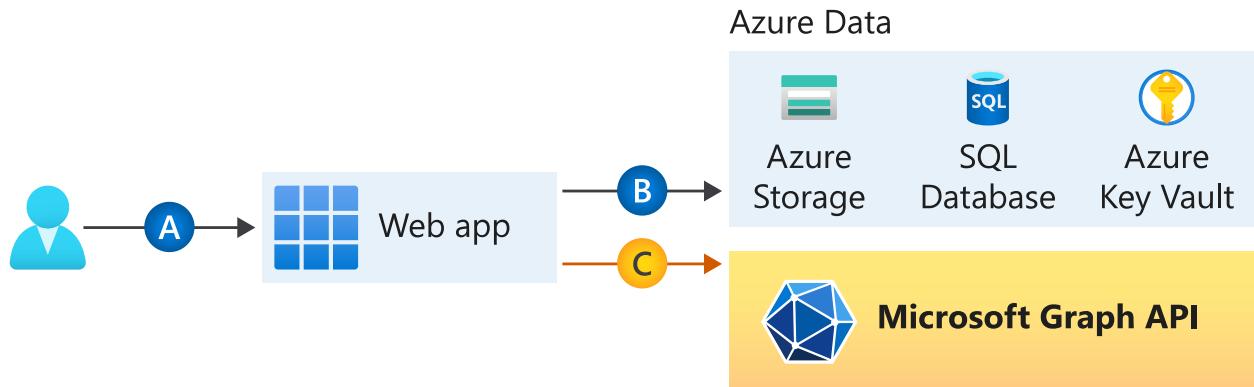
- ✓ Create a system-assigned managed identity on a web app.
- ✓ Add Microsoft Graph API permissions to a managed identity.
- ✓ Call Microsoft Graph from a web app by using managed identities.

Learn how to connect a [.NET Core app](#), [Python app](#), [Java app](#), or [Node.js app](#) to a database.

Tutorial: Access Microsoft Graph from a secured JavaScript app as the app

Article • 08/18/2023

Learn how to access Microsoft Graph from a web app running on Azure App Service.



You want to call Microsoft Graph for the web app. A safe way to give your web app access to data is to use a [system-assigned managed identity](#). A managed identity from Azure Active Directory allows App Service to access resources through role-based access control (RBAC), without requiring app credentials. After assigning a managed identity to your web app, Azure takes care of the creation and distribution of a certificate. You don't have to worry about managing secrets or app credentials.

In this tutorial, you learn how to:

- ✓ Create a system-assigned managed identity on a web app.
- ✓ Add Microsoft Graph API permissions to a managed identity.
- ✓ Call Microsoft Graph from a web app by using managed identities.

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

Prerequisites

- A web application running on Azure App Service that has the [App Service authentication/authorization module enabled](#).

Enable managed identity on app

If you create and publish your web app through Visual Studio, the managed identity was enabled on your app for you.

1. In your app service, select **Identity** in the left pane and then select **System assigned**.
2. Verify that **Status** is set to **On**. If not, select **Save** and then select **Yes** to enable the system-assigned managed identity. When the managed identity is enabled, the status is set to **On** and the object ID is available.
3. Take note of the **Object ID** value, which you'll need in the next step.

SecureWebApp20200924113531 | Identity

System assigned User assigned

A system assigned managed identity enables Azure resources to authenticate. Necessary permissions can be granted via Azure role-based-access-control resource (e.g. Virtual Machine) can only have one system assigned managed identity.

Status **Off** **On**

Object ID **09382857-...-4138df10bfa2**

Permissions **Azure role assignments**

This resource is registered with Azure Active Directory. You can control it

Grant access to Microsoft Graph

When accessing the Microsoft Graph, the managed identity needs to have proper permissions for the operation it wants to perform. Currently, there's no option to assign such permissions through the Microsoft Entra admin center.

1. Run the following script to add the requested Microsoft Graph API permissions to the managed identity service principal object.

```
PowerShell
PowerShell
# Install the module.
# Install-Module Microsoft.Graph -Scope CurrentUser
```

```

# The tenant ID
$TenantId = "11111111-1111-1111-1111-111111111111"

# The name of your web app, which has a managed identity.
$webAppName = "SecureWebApp-20201106120003"
$resourceGroupName = "SecureWebApp-20201106120003ResourceGroup"

# The name of the app role that the managed identity should be
# assigned to.
$appRoleName = "User.Read.All"

# Get the web app's managed identity's object ID.
Connect-AzAccount -Tenant $TenantId
$managedIdentityObjectId = (Get-AzWebApp -ResourceGroupName
$resourceGroupName -Name $webAppName).identity.principalId

Connect-MgGraph -TenantId $TenantId -Scopes
'Application.Read.All', 'AppRoleAssignment.ReadWrite.All'

# Get Microsoft Graph app's service principal and app role.
$serverApplicationName = "Microsoft Graph"
$serverServicePrincipal = (Get-MgServicePrincipal -Filter
"DisplayName eq '$serverApplicationName'")
$serverServicePrincipalObjectId = $serverServicePrincipal.Id

$appRoleId = ($serverServicePrincipal.AppRoles | Where-Object
{$_ .Value -eq $appRoleName }).Id

# Assign the managed identity access to the app role.
New-MgServicePrincipalAppRoleAssignment `-
-ServicePrincipalId $managedIdentityObjectId `-
-PrincipalId $managedIdentityObjectId `-
-ResourceId $serverServicePrincipalObjectId `-
-AppRoleId $appRoleId

```

2. After executing the script, you can verify in the [Microsoft Entra admin center](#) that the requested API permissions are assigned to the managed identity.
3. Go to **Applications**, and then select **Enterprise applications**. This pane displays all the service principals in your tenant. **Add a filter** for "Application type==Managed identities" and select the service principal for the managed identity.

If you're following this tutorial, there are two service principals with the same display name (SecureWebApp2020094113531, for example). The service principal that has a **Homepage URL** represents the web app in your tenant. The service principal that appears in **Managed Identities** should *not* have a **Homepage URL** listed and the **Object ID** should match the object ID value of the managed identity in the [previous step](#).

4. Select the service principal for the managed identity.

The screenshot shows the 'Enterprise applications | All applications' page in the Azure portal. The left sidebar includes 'Overview', 'Diagnose and solve problems', 'Manage' (with 'All applications' selected), 'User settings', 'Security', and 'Conditional Access'. The top navigation bar has links for 'New application', 'Columns', 'Preview features', and 'Got feedback?'. A search bar at the top says 'Try out the new Enterprise Apps search preview! Click to enable the preview.' Below the search bar are filters for 'Application type' (set to 'Managed Identities'), 'Applications status' (set to 'Any'), and 'Application visibility' (set to 'Any'). A large table lists applications, with the first two rows highlighted: 'SecureWebApp20200924113531' and 'SecureWebApp20200924113531'. The table columns include Name, Homepage URL, Object ID, and Application ID.

5. In Overview, select Permissions, and you'll see the added permissions for Microsoft Graph.

The screenshot shows the 'SecureWebApp20200924113531 | Permissions' page. The left sidebar includes 'Overview', 'Deployment Plan', 'Diagnose and solve problems', 'Manage' (with 'Properties' selected), 'Owners', 'Roles and administrators (Prev...)', 'Users and groups', 'Single sign-on', 'Provisioning', 'Self-service', 'Security' (with 'Permissions' selected), and 'Token encryption'. The main content area has a 'Permissions' heading with a note about granting admin consent. It shows a 'Grant admin consent for Fourth Coffee' button. Below this, there are tabs for 'Admin consent' (selected) and 'User consent'. A search bar for 'Search permissions' is followed by a table with columns: API Name, Permission, Type, Granted through, and Granted by. One row in the table is highlighted: 'Microsoft Graph' with 'Read all users' full profil...' permission, 'Application' type, 'Admin consent' granted through, and 'An administrator' granted by.

Call Microsoft Graph with Node.js

Your web app now has the required permissions and also adds Microsoft Graph's client ID to the login parameters.

The `DefaultAzureCredential` class from [@azure/identity](#) package is used to get a token credential for your code to authorize requests to Azure Storage. Create an instance of the `DefaultAzureCredential` class, which uses the managed identity to fetch tokens and attach them to the service client. The following code example gets the authenticated token credential and uses it to create a service client object, which gets the users in the group.

To see this code as part of a sample application, see the:

- [sample on GitHub ↗](#).

ⓘ Note

The [@azure/identity ↗](#) package isn't required in your web app for basic authentication/authorization or to authenticate requests with Microsoft Graph. It's possible to **securely call downstream APIs** with only the App Service authentication/authorization module enabled.

However, the App Service authentication/authorization is designed for more basic authentication scenarios. For more complex scenarios (handling custom claims, for example), you need the [@azure/identity ↗](#) package. There's a little more setup and configuration work in the beginning, but the [@azure/identity](#) package can run alongside the App Service authentication/authorization module. Later, when your web app needs to handle more complex scenarios, you can disable the App Service authentication/authorization module and [@azure/identity](#) will already be a part of your app.

Install client library packages

Install the [@azure/identity ↗](#) and the [@microsoft/microsoft-graph-client ↗](#) packages in your project with npm.

Bash

```
npm install @azure/identity @microsoft/microsoft-graph-client
```

Configure authentication information

Create an object to hold the [authentication settings ↗](#):

JavaScript

```
// partial code in app.js
const appSettings = {
    appCredentials: {
        clientId: process.env.WEBSITE_AUTH_CLIENT_ID, // Enter the client Id here,
        tenantId: "common", // Enter the tenant info here,
        clientSecret: process.env.MICROSOFT_PROVIDER_AUTHENTICATION_SECRET
    // Enter the client secret here,
```

```
        },
        authRoutes: {
            redirect: "/.auth/login/aad/callback", // Enter the redirect URI
            here
            unauthorized: "/unauthorized" // enter the relative path to
            unauthorized route
        },
    }
```

Call Microsoft Graph on behalf of the app

The following code shows how to call [Microsoft Graph controller](#) as the app and get some user information.

JavaScript

```
// graphController.js

const graphHelper = require('../utils/graphHelper');
const { DefaultAzureCredential } = require("@azure/identity");

exports.getUsersPage = async(req, res, next) => {

    const defaultAzureCredential = new DefaultAzureCredential();

    try {
        // get app's access token scoped to Microsoft Graph
        const tokenResponse = await
defaultAzureCredential.getToken("https://graph.microsoft.com/.default");

        // use token to create Graph client
        const graphClient =
graphHelper.getAuthenticatedClient(tokenResponse.token);

        // return profiles of users in Graph
        const users = await graphClient
            .api('/users')
            .get();

        res.render('users', { user: req.session.user, users: users });
    } catch (error) {
        next(error);
    }
}
```

The previous code relies on the following [getAuthenticatedClient](#) function to return Microsoft Graph client.

JavaScript

```
// utils/graphHelper.js

const graph = require('@microsoft/microsoft-graph-client');

getAuthenticatedClient = (accessToken) => {
    // Initialize Graph client
    const client = graph.Client.init({
        // Use the provided access token to authenticate requests
        authProvider: (done) => {
            done(null, accessToken);
        }
    });

    return client;
}
```

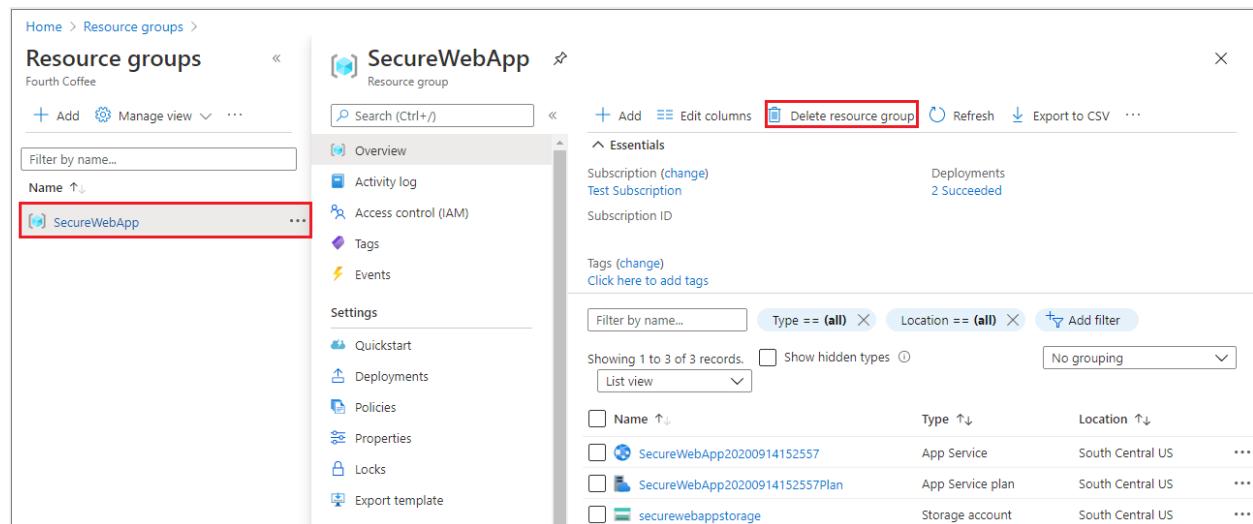
Clean up resources

If you're finished with this tutorial and no longer need the web app or associated resources, clean up the resources you created.

Delete the resource group

In the [Azure portal](#), select **Resource groups** from the portal menu and select the resource group that contains your app service and app service plan.

Select **Delete resource group** to delete the resource group and all the resources.



The screenshot shows the Azure Resource Groups blade. On the left, there's a list of resource groups, with "SecureWebApp" selected and highlighted by a red box. On the right, the details for "SecureWebApp" are shown, including its Overview, Access control (IAM), Tags, Events, and Settings sections. Under the "Essentials" tab, it shows the Subscription (change) to "Test Subscription", Deployment status (2 Succeeded), and a list of resources: "SecureWebApp20200914152557" (App Service, South Central US), "SecureWebApp20200914152557Plan" (App Service plan, South Central US), and "securewebappstorage" (Storage account, South Central US). At the top right, there's a "Delete resource group" button, which is also highlighted with a red box.

This command might take several minutes to run.

Next steps

In this tutorial, you learned how to:

- ✓ Create a system-assigned managed identity on a web app.
- ✓ Add Microsoft Graph API permissions to a managed identity.
- ✓ Call Microsoft Graph from a web app by using managed identities.

Learn how to connect a [.NET Core app](#), [Python app](#), [Java app](#), or [Node.js app](#) to a database.

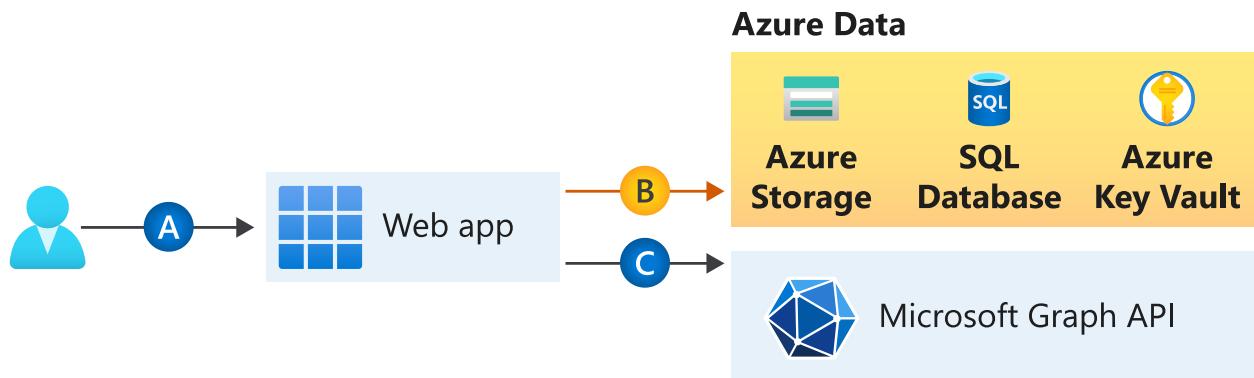
Tutorial: Access Azure services from a .NET web app

Article • 07/31/2023

Learn how to access Azure services, *such as Azure Storage*, from a web app (not a signed-in user) running on Azure App Service by using managed identities. This tutorial demonstrates connecting to Azure Storage as an example.

[Any service](#) that supports managed identity (*B* in the following image) can be securely accessed using this tutorial:

- Azure Storage
- Azure SQL Database
- Azure Key Vault



You want to add secure access to Azure services (Azure Storage, Azure SQL Database, Azure Key Vault, or other services) from your web app. You could use a shared key, but then you have to worry about operational security of who can create, deploy, and manage the secret. It's also possible that the key could be checked into GitHub, which hackers know how to scan for. A safer way to give your web app access to data is to use [managed identities](#).

A managed identity from Azure Active Directory (Azure AD) allows App Service to access resources through role-based access control (RBAC), without requiring app credentials. After assigning a managed identity to your web app, Azure takes care of the creation and distribution of a certificate. People don't have to worry about managing secrets or app credentials.

In this tutorial, you learn how to:

- ✓ Create a system-assigned managed identity on a web app.
- ✓ Create a storage account and an Azure Blob Storage container.
- ✓ Access storage from a web app by using managed identities.

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

Prerequisites

- A web application running on Azure App Service:
 - [.NET quickstart](#)
 - [JavaScript quickstart](#)

Enable managed identity on an app

If you create and publish your web app through Visual Studio, the managed identity was enabled on your app for you. In your app service, select **Identity** in the left pane, and then select **System assigned**. Verify that the **Status** is set to **On**. If not, select **On** and then **Save**. Select **Yes** in the confirmation dialog to enable the system-assigned managed identity. When the managed identity is enabled, the status is set to **On** and the object ID is available.

The screenshot shows the Azure portal interface for a web app named "SecureWebApp20200924113531". The left sidebar includes links for Security, Events (preview), Deployment (Quickstart, Deployment slots, Deployment Center, Deployment Center (Preview)), Settings (Configuration, Authentication / Authorization, Application Insights), and Identity. The "Identity" link is highlighted with a red box. The main content area shows the "Identity" settings. Under "System assigned", the status is set to "On". The "Object ID" field contains the value "09382857-ed3f-4cbd-b524-4138df10bfa2", which is also highlighted with a red box. A note at the bottom states: "This resource is registered with Azure Active Directory. You can control it".

This step creates a new object ID, different than the app ID created in the **Authentication/Authorization** pane. Copy the object ID of the system-assigned managed identity. You'll need it later.

Create a storage account and Blob Storage container

Now you're ready to create a storage account and Blob Storage container.

Every storage account must belong to an Azure resource group. A resource group is a logical container for grouping your Azure services. When you create a storage account, you have the option to either create a new resource group or use an existing resource group. This article shows how to create a new resource group.

A general-purpose v2 storage account provides access to all of the Azure Storage services: blobs, files, queues, tables, and disks. The steps outlined here create a general-purpose v2 storage account, but the steps to create any type of storage account are similar.

Blobs in Azure Storage are organized into containers. Before you can upload a blob later in this tutorial, you must first create a container.

Portal

To create a general-purpose v2 storage account in the Azure portal, follow these steps.

1. On the Azure portal menu, select **All services**. In the list of resources, enter **Storage Accounts**. As you begin typing, the list filters based on your input. Select **Storage Accounts**.
2. In the **Storage Accounts** window that appears, select **Create**.
3. Select the subscription in which to create the storage account.
4. Under the **Resource group** field, select the resource group that contains your web app from the drop-down menu.
5. Next, enter a name for your storage account. The name you choose must be unique across Azure. The name also must be between 3 and 24 characters in length and can include numbers and lowercase letters only.
6. Select a location (region) for your storage account, or use the default value.
7. Leave these fields set to their default values:

Field	Value
Deployment model	Resource Manager

Field	Value
Performance	Standard
Account kind	StorageV2 (general-purpose v2)
Replication	Read-access geo-redundant storage (RA-GRS)
Access tier	Hot

8. Select **Review + Create** to review your storage account settings and create the account.

9. Select **Create**.

To create a Blob Storage container in Azure Storage, follow these steps.

1. Go to your new storage account in the Azure portal.
2. In the left menu for the storage account, scroll to the **Data storage** section, and then select **Containers**.
3. Select the **+ Container** button.
4. Type a name for your new container. The container name must be lowercase, must start with a letter or number, and can include only letters, numbers, and the dash (-) character.
5. Set the level of public access to the container. The default level is **Private (no anonymous access)**.
6. Select **OK** to create the container.

Grant access to the storage account

You need to grant your web app access to the storage account before you can create, read, or delete blobs. In a previous step, you configured the web app running on App Service with a managed identity. Using Azure RBAC, you can give the managed identity access to another resource, just like any security principal. The Storage Blob Data Contributor role gives the web app (represented by the system-assigned managed identity) read, write, and delete access to the blob container and data.

 **Note**

Some operations on private blob containers are not supported by Azure RBAC, such as viewing blobs or copying blobs between accounts. A blob container with private access level requires a SAS token for any operation that is not authorized by Azure RBAC. For more information, see [When to use a shared access signature](#).

Portal

In the [Azure portal](#), go into your storage account to grant your web app access. Select **Access control (IAM)** in the left pane, and then select **Role assignments**. You'll see a list of who has access to the storage account. Now you want to add a role assignment to a robot, the app service that needs access to the storage account. Select **Add > Add role assignment** to open the **Add role assignment** page.

Assign the **Storage Blob Data Contributor** role to the **App Service** at subscription scope. For detailed steps, see [Assign Azure roles using the Azure portal](#).

Your web app now has access to your storage account.

Access Blob Storage

The [DefaultAzureCredential](#) class is used to get a token credential for your code to authorize requests to Azure Storage. Create an instance of the [DefaultAzureCredential](#) class, which uses the managed identity to fetch tokens and attach them to the service client. The following code example gets the authenticated token credential and uses it to create a service client object, which uploads a new blob.

To see this code as part of a sample application, see the [sample on GitHub](#).

Install client library packages

Install the [Blob Storage NuGet package](#) to work with Blob Storage and the [Azure Identity client library for .NET NuGet package](#) to authenticate with Azure AD credentials. Install the client libraries by using the .NET Core command-line interface or the Package Manager Console in Visual Studio.

.NET Core command-line

1. Open a command line, and switch to the directory that contains your project file.

2. Run the install commands.

```
.NET CLI
```

```
dotnet add package Azure.Storage.Blobs  
dotnet add package Azure.Identity
```

Package Manager Console

1. Open the project or solution in Visual Studio, and open the console by using the **Tools > NuGet Package Manager > Package Manager Console** command.
2. Run the install commands.

```
PowerShell
```

```
Install-Package Azure.Storage.Blobs  
Install-Package Azure.Identity
```

.NET example

```
C#
```

```
using System;  
using Azure.Storage.Blobs;  
using Azure.Storage.Blobs.Models;  
using System.Collections.Generic;  
using System.Threading.Tasks;  
using System.Text;  
using System.IO;  
using Azure.Identity;  
  
// Some code omitted for brevity.  
  
static public async Task UploadBlob(string accountName, string  
containerName, string blobName, string blobContents)  
{  
    // Construct the blob container endpoint from the arguments.  
    string containerEndpoint =  
        string.Format("https://'{0}'.blob.core.windows.net/{1}",  
                     accountName,  
                     containerName);  
  
    // Get a credential and create a client object for the blob container.  
    BlobContainerClient containerClient = new BlobContainerClient(new  
        Uri(containerEndpoint),
```

[new](#)

```
DefaultAzureCredential());  
  
try  
{  
    // Create the container if it does not exist.  
    await containerClient.CreateIfNotExistsAsync();  
  
    // Upload text to a new block blob.  
    byte[] byteArray = Encoding.ASCII.GetBytes(blobContents);  
  
    using (MemoryStream stream = new MemoryStream(byteArray))  
    {  
        await containerClient.UploadBlobAsync(blobName, stream);  
    }  
}  
catch (Exception e)  
{  
    throw e;  
}  
}
```

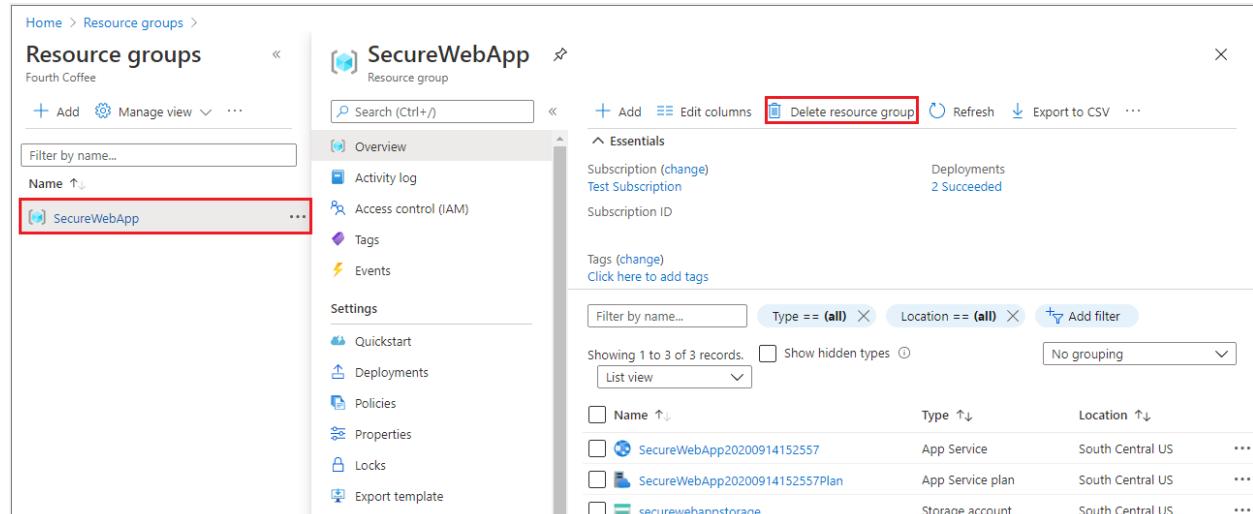
Clean up resources

If you're finished with this tutorial and no longer need the web app or associated resources, clean up the resources you created.

Delete the resource group

In the [Azure portal](#), select **Resource groups** from the portal menu and select the resource group that contains your app service and app service plan.

Select **Delete resource group** to delete the resource group and all the resources.



The screenshot shows the Azure Resource Groups blade. On the left, there's a list of resource groups under 'Fourth Coffee'. One resource group, 'SecureWebApp', is selected and highlighted with a red box. On the right, the details for 'SecureWebApp' are shown in a larger pane. At the top of this pane, there's a red box around the 'Delete resource group' button. Below the button, the 'Essentials' section displays the subscription information: 'Subscription (change)', 'Test Subscription', and 'Deployments 2 Succeeded'. The 'Properties' section shows the tags: 'Tags (change)' and 'Click here to add tags'. At the bottom, there's a table listing the resources within the group:

Name	Type	Location	...
SecureWebApp20200914152557	App Service	South Central US	...
SecureWebApp20200914152557Plan	App Service plan	South Central US	...
securewebappstorage	Storage account	South Central US	...

This command might take several minutes to run.

Next steps

In this tutorial, you learned how to:

- ✓ Create a system-assigned managed identity.
- ✓ Create a storage account and Blob Storage container.
- ✓ Access storage from a web app by using managed identities.

[Tutorial: Isolate back-end communication with Virtual Network integration](#)

[App Service accesses Microsoft Graph on behalf of the user](#)

[Secure with custom domain and certificate](#)

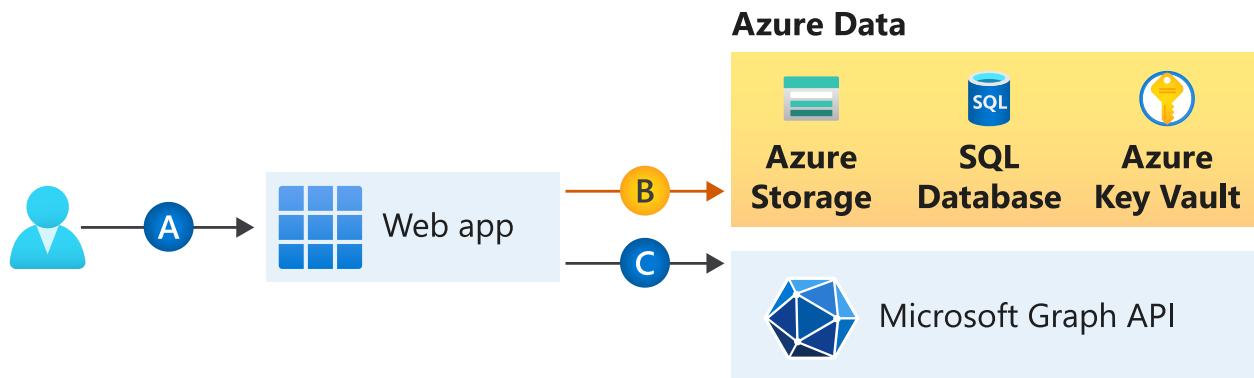
Tutorial: Access Azure services from a JavaScript web app

Article • 07/31/2023

Learn how to access Azure services, *such as Azure Storage*, from a web app (not a signed-in user) running on Azure App Service by using managed identities. This tutorial demonstrates connecting to Azure Storage as an example.

[Any service](#) that supports managed identity (*B* in the following image) can be securely accessed using this tutorial:

- Azure Storage
- Azure SQL Database
- Azure Key Vault



You want to add secure access to Azure services (Azure Storage, Azure SQL Database, Azure Key Vault, or other services) from your web app. You could use a shared key, but then you have to worry about operational security of who can create, deploy, and manage the secret. It's also possible that the key could be checked into GitHub, which hackers know how to scan for. A safer way to give your web app access to data is to use [managed identities](#).

A managed identity from Azure Active Directory (Azure AD) allows App Service to access resources through role-based access control (RBAC), without requiring app credentials. After assigning a managed identity to your web app, Azure takes care of the creation and distribution of a certificate. People don't have to worry about managing secrets or app credentials.

In this tutorial, you learn how to:

- ✓ Create a system-assigned managed identity on a web app.
- ✓ Create a storage account and an Azure Blob Storage container.
- ✓ Access storage from a web app by using managed identities.

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

Prerequisites

- A web application running on Azure App Service:
 - [.NET quickstart](#)
 - [JavaScript quickstart](#)

Enable managed identity on an app

If you create and publish your web app through Visual Studio, the managed identity was enabled on your app for you. In your app service, select **Identity** in the left pane, and then select **System assigned**. Verify that the **Status** is set to **On**. If not, select **On** and then **Save**. Select **Yes** in the confirmation dialog to enable the system-assigned managed identity. When the managed identity is enabled, the status is set to **On** and the object ID is available.

The screenshot shows the Azure portal interface for a web app named "SecureWebApp20200924113531". The left sidebar includes links for Security, Events (preview), Deployment (Quickstart, Deployment slots, Deployment Center, Deployment Center (Preview)), Settings (Configuration, Authentication / Authorization, Application Insights), and Identity. The "Identity" link is highlighted with a red box. The main content area shows the "Identity" settings. Under "System assigned", the status is set to "On". The "Object ID" field contains the value "09382857-ed3f-4cbd-b524-4138df10bfa2", which is also highlighted with a red box. A note at the bottom states: "This resource is registered with Azure Active Directory. You can control it".

This step creates a new object ID, different than the app ID created in the **Authentication/Authorization** pane. Copy the object ID of the system-assigned managed identity. You'll need it later.

Create a storage account and Blob Storage container

Now you're ready to create a storage account and Blob Storage container.

Every storage account must belong to an Azure resource group. A resource group is a logical container for grouping your Azure services. When you create a storage account, you have the option to either create a new resource group or use an existing resource group. This article shows how to create a new resource group.

A general-purpose v2 storage account provides access to all of the Azure Storage services: blobs, files, queues, tables, and disks. The steps outlined here create a general-purpose v2 storage account, but the steps to create any type of storage account are similar.

Blobs in Azure Storage are organized into containers. Before you can upload a blob later in this tutorial, you must first create a container.

Portal

To create a general-purpose v2 storage account in the Azure portal, follow these steps.

1. On the Azure portal menu, select **All services**. In the list of resources, enter **Storage Accounts**. As you begin typing, the list filters based on your input. Select **Storage Accounts**.
2. In the **Storage Accounts** window that appears, select **Create**.
3. Select the subscription in which to create the storage account.
4. Under the **Resource group** field, select the resource group that contains your web app from the drop-down menu.
5. Next, enter a name for your storage account. The name you choose must be unique across Azure. The name also must be between 3 and 24 characters in length and can include numbers and lowercase letters only.
6. Select a location (region) for your storage account, or use the default value.
7. Leave these fields set to their default values:

Field	Value
Deployment model	Resource Manager

Field	Value
Performance	Standard
Account kind	StorageV2 (general-purpose v2)
Replication	Read-access geo-redundant storage (RA-GRS)
Access tier	Hot

8. Select **Review + Create** to review your storage account settings and create the account.

9. Select **Create**.

To create a Blob Storage container in Azure Storage, follow these steps.

1. Go to your new storage account in the Azure portal.
2. In the left menu for the storage account, scroll to the **Data storage** section, and then select **Containers**.
3. Select the **+ Container** button.
4. Type a name for your new container. The container name must be lowercase, must start with a letter or number, and can include only letters, numbers, and the dash (-) character.
5. Set the level of public access to the container. The default level is **Private (no anonymous access)**.
6. Select **OK** to create the container.

Grant access to the storage account

You need to grant your web app access to the storage account before you can create, read, or delete blobs. In a previous step, you configured the web app running on App Service with a managed identity. Using Azure RBAC, you can give the managed identity access to another resource, just like any security principal. The Storage Blob Data Contributor role gives the web app (represented by the system-assigned managed identity) read, write, and delete access to the blob container and data.

 **Note**

Some operations on private blob containers are not supported by Azure RBAC, such as viewing blobs or copying blobs between accounts. A blob container with private access level requires a SAS token for any operation that is not authorized by Azure RBAC. For more information, see [When to use a shared access signature](#).

Portal

In the [Azure portal](#), go into your storage account to grant your web app access. Select **Access control (IAM)** in the left pane, and then select **Role assignments**. You'll see a list of who has access to the storage account. Now you want to add a role assignment to a robot, the app service that needs access to the storage account. Select **Add > Add role assignment** to open the **Add role assignment** page.

Assign the **Storage Blob Data Contributor** role to the **App Service** at subscription scope. For detailed steps, see [Assign Azure roles using the Azure portal](#).

Your web app now has access to your storage account.

Access Blob Storage

The `DefaultAzureCredential` class from [@azure/identity](#) package is used to get a token credential for your code to authorize requests to Azure Storage. The `BlobServiceClient` class from [@azure/storage-blob](#) package is used to upload a new blob to storage. Create an instance of the `DefaultAzureCredential` class, which uses the managed identity to fetch tokens and attach them to the blob service client. The following code example gets the authenticated token credential and uses it to create a service client object, which uploads a new blob.

To see this code as part of a sample application, see *StorageHelper.js* in the:

- [Sample on GitHub](#).

JavaScript example

Node.js

```
const { DefaultAzureCredential } = require("@azure/identity");
const { BlobServiceClient } = require("@azure/storage-blob");
const defaultAzureCredential = new DefaultAzureCredential();
```

```
// Some code omitted for brevity.

async function uploadBlob(accountName, containerName, blobName,
blobContents) {
    const blobServiceClient = new BlobServiceClient(
        `https://${accountName}.blob.core.windows.net`,
        defaultAzureCredential
    );

    const containerClient =
blobServiceClient.getContainerClient(containerName);

    try {
        await containerClient.createIfNotExists();
        const blockBlobClient =
containerClient.getBlockBlobClient(blobName);
        const uploadBlobResponse = await
blockBlobClient.upload(blobContents, blobContents.length);
        console.log(`Upload block blob ${blobName} successfully`,
uploadBlobResponse.requestId);
    } catch (error) {
        console.log(error);
    }
}
```

Clean up resources

If you're finished with this tutorial and no longer need the web app or associated resources, clean up the resources you created.

Delete the resource group

In the [Azure portal](#), select **Resource groups** from the portal menu and select the resource group that contains your app service and app service plan.

Select **Delete resource group** to delete the resource group and all the resources.

The screenshot shows the Azure Resource Groups blade. On the left, there's a list of resource groups with a search bar and filter options. A specific resource group, "SecureWebApp", is selected and highlighted with a red box. On the right, the details for "SecureWebApp" are shown under the "Essentials" section. It includes information about the subscription ("Test Subscription"), deployment status ("2 Succeeded"), and resource properties like "Subscription ID" and "Tags". Below this is a table listing resources within the group, such as "SecureWebApp20200914152557" (App Service, South Central US), "SecureWebApp20200914152557Plan" (App Service plan, South Central US), and "securewebappstorage" (Storage account, South Central US). The "Delete resource group" button is located at the top of the main content area.

This command might take several minutes to run.

Next steps

In this tutorial, you learned how to:

- ✓ Create a system-assigned managed identity.
- ✓ Create a storage account and Blob Storage container.
- ✓ Access storage from a web app by using managed identities.

[Tutorial: Isolate back-end communication with Virtual Network integration](#)

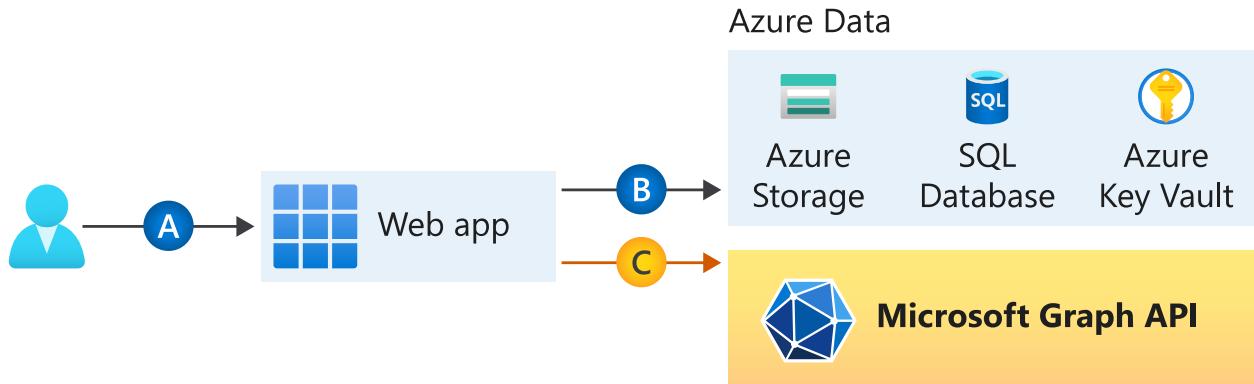
[App Service accesses Microsoft Graph on behalf of the user](#)

[Secure with custom domain and certificate](#)

Tutorial: Access Microsoft Graph from a secured .NET app as the user

Article • 02/08/2024

Learn how to access Microsoft Graph from a web app running on Azure App Service.



You want to add access to Microsoft Graph from your web app and perform some action as the signed-in user. This section describes how to grant delegated permissions to the web app and get the signed-in user's profile information from Microsoft Entra ID.

In this tutorial, you learn how to:

- ✓ Grant delegated permissions to a web app.
- ✓ Call Microsoft Graph from a web app for a signed-in user.

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

Prerequisites

- A web application running on Azure App Service that has the [App Service authentication/authorization module enabled](#).

Grant front-end access to call Microsoft Graph

Now that you've enabled authentication and authorization on your web app, the web app is registered with the Microsoft identity platform and is backed by a Microsoft Entra application. In this step, you give the web app permissions to access Microsoft Graph for the user. (Technically, you give the web app's Microsoft Entra application the permissions to access the Microsoft Graph Microsoft Entra application for the user.)

1. In the [Microsoft Entra admin center](#), select **Applications**.

2. Select App registrations > Owned applications > View all applications in this directory. Select your web app name, and then select API permissions.
3. Select Add a permission, and then select Microsoft APIs and Microsoft Graph.
4. Select Delegated permissions, and then select User.Read from the list. Select Add permissions.

Configure App Service to return a usable access token

The web app now has the required permissions to access Microsoft Graph as the signed-in user. In this step, you configure App Service authentication and authorization to give you a usable access token for accessing Microsoft Graph. For this step, you need to add the User.Read scope for the downstream service (Microsoft Graph):

`https://graph.microsoft.com/User.Read`.

Important

If you don't configure App Service to return a usable access token, you receive a `CompactToken parsing failed with error code: 80049217` error when you call Microsoft Graph APIs in your code.

Azure Resource Explorer

Go to [Azure Resource Explorer](#) and using the resource tree, locate your web app. The resource URL should be similar to

`https://resources.azure.com/subscriptions/subscriptionId/resourceGroups/SecureWebApp/providers/Microsoft.Web/sites/SecureWebApp20200915115914`.

The Azure Resource Explorer is now opened with your web app selected in the resource tree.

1. At the top of the page, select **Read/Write** to enable editing of your Azure resources.
2. In the left browser, drill down to **config > authsettingsV2**.
3. In the **authsettingsV2** view, select **Edit**.

4. Find the **login** section of **identityProviders** -> **azureActiveDirectory** and add the following **loginParameters** settings: "loginParameters": ["response_type=code id_token", "scope=openid offline_access profile https://graph.microsoft.com/User.Read"].

JSON

```
"identityProviders": {  
    "azureActiveDirectory": {  
        "enabled": true,  
        "login": {  
            "loginParameters": [  
                "response_type=code id_token",  
                "scope=openid offline_access profile  
https://graph.microsoft.com/User.Read"  
            ]  
        }  
    }  
},
```

5. Save your settings by selecting **PUT**. This setting can take several minutes to take effect. Your web app is now configured to access Microsoft Graph with a proper access token. If you don't, Microsoft Graph returns an error saying that the format of the compact token is incorrect.

Call Microsoft Graph with .NET

Your web app now has the required permissions and also adds Microsoft Graph's client ID to the login parameters.

Using the [Microsoft.Identity.Web library](#), the web app gets an access token for authentication with Microsoft Graph. In version 1.2.0 and later, the Microsoft.Identity.Web library integrates with and can run alongside the App Service authentication/authorization module. Microsoft.Identity.Web detects that the web app is hosted in App Service and gets the access token from the App Service authentication/authorization module. The access token is then passed along to authenticated requests with the Microsoft Graph API.

To see this code as part of a sample application, see the:

- [Sample on GitHub](#).

Note

The Microsoft.Identity.Web library isn't required in your web app for basic authentication/authorization or to authenticate requests with Microsoft Graph. It's possible to [securely call downstream APIs](#) with only the App Service authentication/authorization module enabled.

However, the App Service authentication/authorization is designed for more basic authentication scenarios. For more complex scenarios (handling custom claims, for example), you need the Microsoft.Identity.Web library or [Microsoft Authentication Library](#). There's a little more setup and configuration work in the beginning, but the Microsoft.Identity.Web library can run alongside the App Service authentication/authorization module. Later, when your web app needs to handle more complex scenarios, you can disable the App Service authentication/authorization module and Microsoft.Identity.Web will already be a part of your app.

Install client library packages

Install the [Microsoft.Identity.Web](#) and [Microsoft.Identity.Web.MicrosoftGraph](#) NuGet packages in your project by using the .NET Core command-line interface or the Package Manager Console in Visual Studio.

.NET Core command line

Open a command line, and switch to the directory that contains your project file.

Run the install commands.

.NET CLI

```
dotnet add package Microsoft.Identity.Web.MicrosoftGraph  
dotnet add package Microsoft.Identity.Web
```

Package Manager Console

Open the project/solution in Visual Studio, and open the console by using the **Tools > NuGet Package Manager > Package Manager Console** command.

Run the install commands.

PowerShell

```
Install-Package Microsoft.Identity.Web.GraphServiceClient
```

```
Install-Package Microsoft.Identity.Web
```

Startup.cs

In the `Startup.cs` file, the `AddMicrosoftIdentityWebApp` method adds `Microsoft.Identity.Web` to your web app. The `AddMicrosoftGraph` method adds Microsoft Graph support. For info on managing incremental consent and conditional access, [read this ↗](#).

C#

```
using Microsoft.AspNetCore.Builder;
using Microsoft.AspNetCore.Hosting;
using Microsoft.Extensions.Configuration;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Hosting;
using Microsoft.Identity.Web;
using Microsoft.AspNetCore.Authentication.OpenIdConnect;

// Some code omitted for brevity.
public class Startup
{
    // This method gets called by the runtime. Use this method to add
    // services to the container.
    public void ConfigureServices(IServiceCollection services)
    {
        services.AddOptions();
        string[] initialScopes = Configuration.GetValue<string>
("DownstreamApi:Scopes")?.Split(' ');

        services.AddAuthentication(OpenIdConnectDefaults.AuthenticationScheme)

        .AddMicrosoftIdentityWebApp(Configuration.GetSection("AzureAd"))
            .EnableTokenAcquisitionToCallDownstreamApi(initialScopes)

        .AddMicrosoftGraph(Configuration.GetSection("DownstreamApi"))
            .AddInMemoryTokenCaches();

        services.AddAuthorization(options =>
        {
            // By default, all incoming requests will be authorized according
            // to the default policy
            options.FallbackPolicy = options.DefaultPolicy;
        });
        services.AddRazorPages()
            .AddMvcOptions(options => {})
```

```

        .AddMicrosoftIdentityUI();

    services.AddControllersWithViews()
        .AddMicrosoftIdentityUI();
    }
}

```

appsettings.json

`AzureAd` specifies the configuration for the Microsoft.Identity.Web library. In the [Microsoft Entra admin center](#), select **Applications** from the portal menu and then select **App registrations**. Select the app registration created when you enabled the App Service authentication/authorization module. (The app registration should have the same name as your web app.) You can find the tenant ID and client ID in the app registration overview page. The domain name can be found in the Microsoft Entra overview page for your tenant.

`Graph` specifies the Microsoft Graph endpoint and the initial scopes needed by the app.

JSON

```
{
  "AzureAd": {
    "Instance": "https://login.microsoftonline.com/",
    "Domain": "[Enter the domain of your tenant, e.g. contoso.onmicrosoft.com]",
    "TenantId": "[Enter 'common', or 'organizations' or the Tenant Id (Obtained from the Microsoft Entra admin center. Select 'Endpoints' from the 'App registrations' blade and use the GUID in any of the URLs), e.g. da41245a5-11b3-996c-00a8-4d99re19f292]",
    "ClientId": "[Enter the Client Id (Application ID obtained from the Microsoft Entra admin center), e.g. ba74781c2-53c2-442a-97c2-3d60re42f403]",
    "ClientSecret": "[Copy the client secret added to the app from the Microsoft Entra admin center]",
    "ClientCertificates": [
    ],
    // the following is required to handle Continuous Access Evaluation challenges
    "ClientCapabilities": [ "cp1" ],
    "CallbackPath": "/signin-oidc"
  },
  "DownstreamApis": {
    "MicrosoftGraph": {
      // Specify BaseUrl if you want to use Microsoft graph in a national cloud.
      // See https://learn.microsoft.com/graph/deployments#microsoft-graph-and-graph-explorer-service-root-endpoints
      // "BaseUrl": "https://graph.microsoft.com/v1.0",
    }
  }
}
```

```

    // Set RequestAppToken this to "true" if you want to request an
    // application token (to call graph on
    // behalf of the application). The scopes will then automatically
    // be ['https://graph.microsoft.com/.default'].
    // "RequestAppToken": false

    // Set Scopes to request (unless you request an app token).
    "Scopes": [ "User.Read" ]

    // See https://aka.ms/ms-id-web/downstreamApiOptions for all the
    properties you can set.
}

},
"Logging": {
    "LogLevel": {
        "Default": "Information",
        "Microsoft": "Warning",
        "Microsoft.Hosting.Lifetime": "Information"
    }
},
"AllowedHosts": "*"
}

```

Call Microsoft Graph on behalf of the user

The following example shows how to call Microsoft Graph as the signed-in user and get some user information. The `GraphServiceClient` object is injected into the controller, and authentication has been configured for you by the `Microsoft.Identity.Web` library.

C#

```

// Index.cshtml.cs
using System.Threading.Tasks;
using Microsoft.AspNetCore.Mvc.RazorPages;
using Microsoft.Graph;
using System.IO;
using Microsoft.Identity.Web;
using Microsoft.Extensions.Logging;

// Some code omitted for brevity.

[AuthorizeForScopes(Scopes = new[] { "User.Read" })]
public class IndexModel : PageModel
{
    private readonly ILogger<IndexModel> _logger;
    private readonly GraphServiceClient _graphServiceClient;

    public IndexModel(ILogger<IndexModel> logger, GraphServiceClient graphServiceClient)
    {
        _logger = logger;
    }
}

```

```
_graphServiceClient = graphServiceClient;  
}  
  
public async Task OnGetAsync()  
{  
    try  
    {  
        var user = await _graphServiceClient.Me.GetAsync();  
        ViewData["Me"] = user;  
        ViewData["name"] = user.DisplayName;  
  
        using (var photoStream = await  
_graphServiceClient.Me.Photo.Content.GetAsync())  
        {  
            byte[] photoByte = ((MemoryStream)photoStream).ToArray();  
            ViewData["photo"] = Convert.ToBase64String(photoByte);  
        }  
    }  
    catch (Exception ex)  
    {  
        ViewData["photo"] = null;  
    }  
}  
}
```

Clean up resources

If you completed all the steps in this multipart tutorial, you created an App Service, App Service hosting plan, and a storage account in a resource group. You also created an app registration in Microsoft Entra ID. If you chose external configuration, you may have created a new external tenant. When no longer needed, delete these resources and app registration so that you don't continue to accrue charges.

In this tutorial, you learn how to:

- ✓ Delete the Azure resources created while following the tutorial.

Delete the resource group

In the [Azure portal](#), select **Resource groups** from the portal menu and select the resource group that contains your App Service and App Service plan.

Select **Delete resource group** to delete the resource group and all the resources.

The screenshot shows the Azure Resource Groups blade. On the left, there's a list of resources under the 'SecureWebApp' resource group, with 'SecureWebApp' itself highlighted by a red box. On the right, there's an 'Essentials' summary section and a detailed list of resources with their types and locations. The 'Delete resource group' button at the top right of the blade is also highlighted with a red box.

This command might take several minutes to run.

Delete the app registration

In the [Microsoft Entra admin center](#), select **Applications > App registrations**. Then select the application you created.

The screenshot shows the Microsoft Entra admin center's App registrations blade. On the left sidebar, 'App registrations' is selected and highlighted with a red box. The main area displays a list of applications, with 'GetStartedWebApp' selected and highlighted with a red box. The list includes columns for Display name, Application (client) ID, Created on, and Certificates & secrets.

In the app registration overview, select **Delete**.

The screenshot shows the 'GetStartedWebApp' app registration overview page. The 'Delete' button at the top right of the blade is highlighted with a red box. The page also includes sections for Overview, Quickstart, Integration assistant, Manage, and Essentials.

Delete the external tenant

If you created a new external tenant, you can [delete it](#). In to the [Microsoft Entra admin center](#), browse to **Identity > Overview > Manage tenants**.

Select the tenant you want to delete, and then select **Delete**.

You might need to complete required actions before you can delete the tenant. For example, you might need to delete all user flows and app registrations in the tenant.

If you're ready to delete the tenant, select **Delete**.

Next steps

In this tutorial, you learned how to:

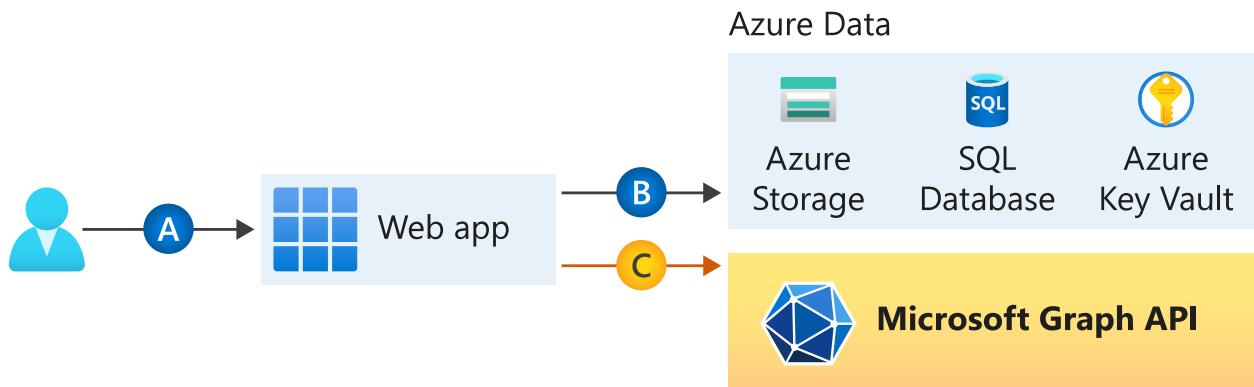
- ✓ Grant delegated permissions to a web app.
- ✓ Call Microsoft Graph from a web app for a signed-in user.

[App service accesses Microsoft Graph as the app](#)

Tutorial: Access Microsoft Graph from a secured JavaScript app as the user

Article • 02/08/2024

Learn how to access Microsoft Graph from a web app running on Azure App Service.



You want to add access to Microsoft Graph from your web app and perform some action as the signed-in user. This section describes how to grant delegated permissions to the web app and get the signed-in user's profile information from Microsoft Entra ID.

In this tutorial, you learn how to:

- ✓ Grant delegated permissions to a web app.
- ✓ Call Microsoft Graph from a web app for a signed-in user.

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

Prerequisites

- A web application running on Azure App Service that has the [App Service authentication/authorization module enabled](#).

Grant front-end access to call Microsoft Graph

Now that you've enabled authentication and authorization on your web app, the web app is registered with the Microsoft identity platform and is backed by a Microsoft Entra application. In this step, you give the web app permissions to access Microsoft Graph for the user. (Technically, you give the web app's Microsoft Entra application the permissions to access the Microsoft Graph Microsoft Entra application for the user.)

1. In the [Microsoft Entra admin center](#), select **Applications**.

2. Select App registrations > Owned applications > View all applications in this directory. Select your web app name, and then select API permissions.
3. Select Add a permission, and then select Microsoft APIs and Microsoft Graph.
4. Select Delegated permissions, and then select User.Read from the list. Select Add permissions.

Configure App Service to return a usable access token

The web app now has the required permissions to access Microsoft Graph as the signed-in user. In this step, you configure App Service authentication and authorization to give you a usable access token for accessing Microsoft Graph. For this step, you need to add the User.Read scope for the downstream service (Microsoft Graph):

`https://graph.microsoft.com/User.Read`.

Important

If you don't configure App Service to return a usable access token, you receive a `CompactToken parsing failed with error code: 80049217` error when you call Microsoft Graph APIs in your code.

Azure Resource Explorer

Go to [Azure Resource Explorer](#) and using the resource tree, locate your web app. The resource URL should be similar to

`https://resources.azure.com/subscriptions/subscriptionId/resourceGroups/SecureWebApp/providers/Microsoft.Web/sites/SecureWebApp20200915115914`.

The Azure Resource Explorer is now opened with your web app selected in the resource tree.

1. At the top of the page, select **Read/Write** to enable editing of your Azure resources.
2. In the left browser, drill down to **config > authsettingsV2**.
3. In the **authsettingsV2** view, select **Edit**.

4. Find the **login** section of **identityProviders** -> **azureActiveDirectory** and add the following **loginParameters** settings: "loginParameters": [
"response_type=code id_token", "scope=openid offline_access profile
[https://graph.microsoft.com/User.Read"](https://graph.microsoft.com/User.Read)].

JSON

```
"identityProviders": {  
    "azureActiveDirectory": {  
        "enabled": true,  
        "login": {  
            "loginParameters": [  
                "response_type=code id_token",  
                "scope=openid offline_access profile  
https://graph.microsoft.com/User.Read"  
            ]  
        }  
    }  
},
```

5. Save your settings by selecting **PUT**. This setting can take several minutes to take effect. Your web app is now configured to access Microsoft Graph with a proper access token. If you don't, Microsoft Graph returns an error saying that the format of the compact token is incorrect.

Call Microsoft Graph from Node.js

Your web app now has the required permissions and also adds Microsoft Graph's client ID to the login parameters.

To see this code as part of a sample application, see the:

- [Sample on GitHub ↗](#).

Install client library packages

Install the [@azure/identity ↗](#) and the [@microsoft/microsoft-graph-client ↗](#) packages in your project with npm.

Bash

```
npm install @microsoft/microsoft-graph-client
```

Configure authentication information

Create an object to hold the [authentication settings](#):

JavaScript

```
// partial code in app.js
const appSettings = {
    appCredentials: {
        clientId: process.env.WEBSITE_AUTH_CLIENT_ID, // Enter the client Id here,
        tenantId: "common", // Enter the tenant info here,
        clientSecret: process.env.MICROSOFT_PROVIDER_AUTHENTICATION_SECRET
    // Enter the client secret here,
    },
    authRoutes: {
        redirect: "./.auth/login/aad/callback", // Enter the redirect URI here
        error: "/error", // enter the relative path to error handling route
        unauthorized: "/unauthorized" // enter the relative path to unauthorized route
    },
    protectedResources: {
        graphAPI: {
            endpoint: "https://graph.microsoft.com/v1.0/me", // resource endpoint
            scopes: ["User.Read"] // resource scopes
        },
    },
}
```

Call Microsoft Graph on behalf of the user

The following code shows how to call [Microsoft Graph controller](#) as the app and get some user information.

JavaScript

```
// controllers/graphController.js

// get the name of the app service instance from environment variables
const appServiceName = process.env.WEBSITE_SITE_NAME;

const graphHelper = require('../utils/graphHelper');

exports.getProfilePage = async(req, res, next) => {

    try {
        // get user's access token scoped to Microsoft Graph from session
        // use token to create Graph client
    }
```

```

        const graphClient =
graphHelper.getAuthenticatedClient(req.session.protectedResources[ "graphAPI"
].accessToken);

        // return user's profile
        const profile = await graphClient
            .api('/me')
            .get();

        res.render('profile', { isAuthenticated:
req.session.isAuthenticated, profile: profile, appServiceName:
appServiceName });
    } catch (error) {
        next(error);
    }
}

```

The previous code relies on the following [getAuthenticatedClient](#) function to return Microsoft Graph client.

JavaScript

```

// utils/graphHelper.js

const graph = require('@microsoft/microsoft-graph-client');

getAuthenticatedClient = (accessToken) => {
    // Initialize Graph client
    const client = graph.Client.init({
        // Use the provided access token to authenticate requests
        authProvider: (done) => {
            done(null, accessToken);
        }
    });

    return client;
}

```

Clean up resources

If you completed all the steps in this multipart tutorial, you created an App Service, App Service hosting plan, and a storage account in a resource group. You also created an app registration in Microsoft Entra ID. If you chose external configuration, you may have created a new external tenant. When no longer needed, delete these resources and app registration so that you don't continue to accrue charges.

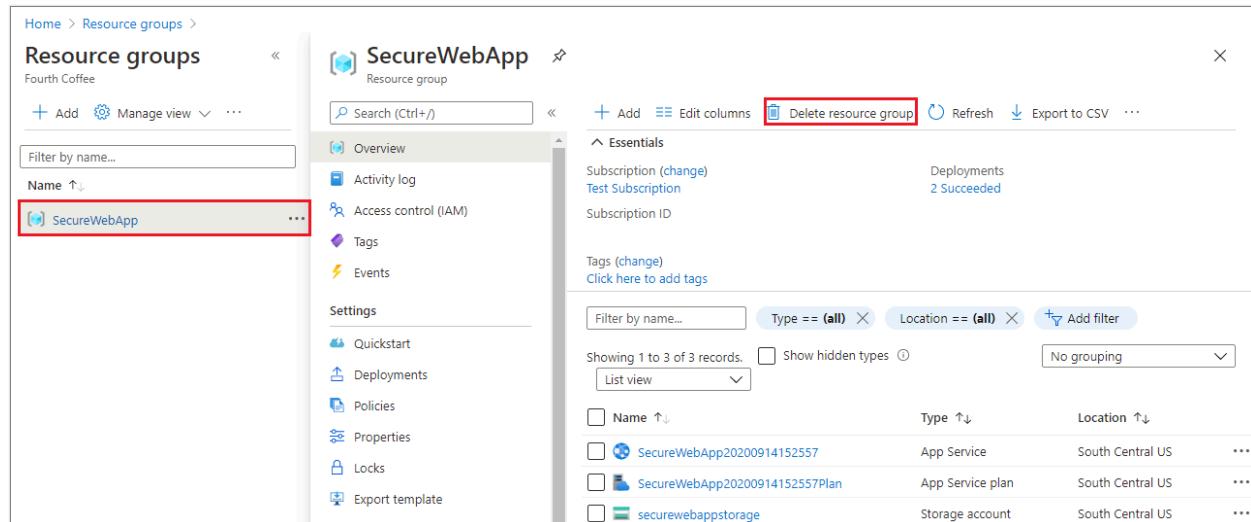
In this tutorial, you learn how to:

- ✓ Delete the Azure resources created while following the tutorial.

Delete the resource group

In the [Azure portal](#), select **Resource groups** from the portal menu and select the resource group that contains your App Service and App Service plan.

Select **Delete resource group** to delete the resource group and all the resources.

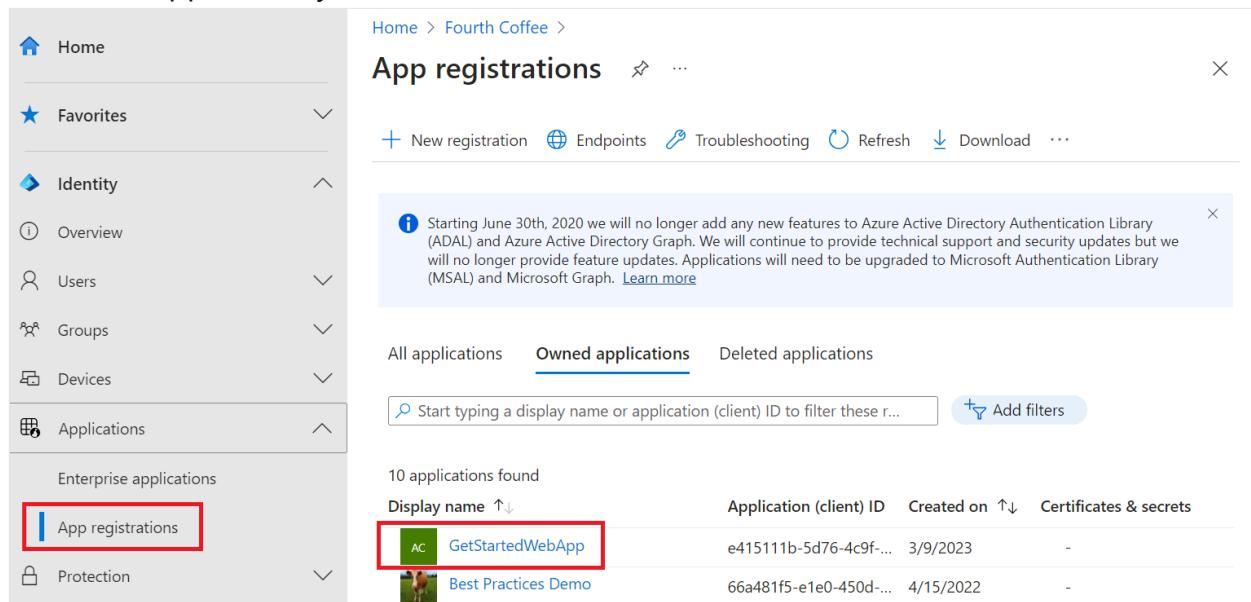


The screenshot shows the Azure Resource Groups blade. On the left, there's a list of resource groups, with 'SecureWebApp' selected and highlighted by a red box. In the center, there's an 'Overview' section with details like 'Subscription (change)', 'Test Subscription', 'Deployment ID', and 'Tags'. On the right, there's a table listing resources within the group, including 'SecureWebApp20200914152557' (App Service), 'SecureWebApp20200914152557Plan' (App Service plan), and 'securewebappstorage' (Storage account). The 'Delete resource group' button at the top right of the overview section is also highlighted with a red box.

This command might take several minutes to run.

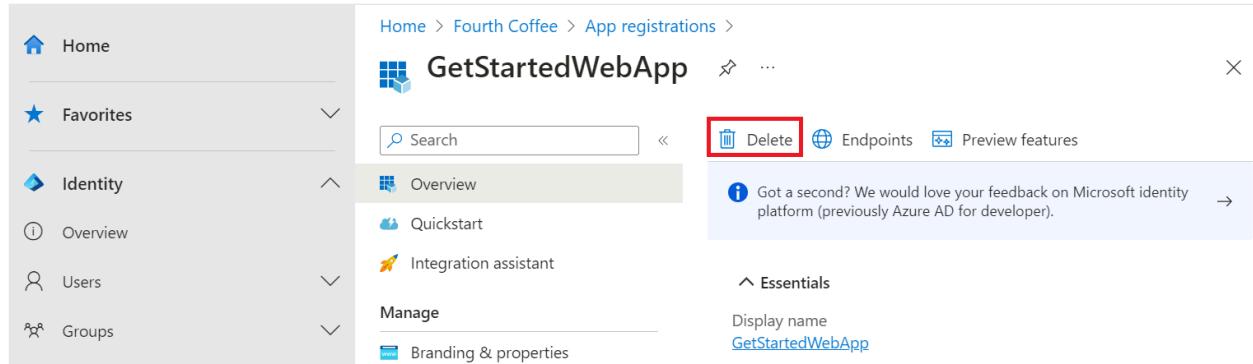
Delete the app registration

In the [Microsoft Entra admin center](#), select **Applications > App registrations**. Then select the application you created.



The screenshot shows the Microsoft Entra admin center Applications > App registrations blade. The 'App registrations' tab is selected. A message box at the top states: 'Starting June 30th, 2020 we will no longer add any new features to Azure Active Directory Authentication Library (ADAL) and Azure Active Directory Graph. We will continue to provide technical support and security updates but we will no longer provide feature updates. Applications will need to be upgraded to Microsoft Authentication Library (MSAL) and Microsoft Graph.' Below this, there are tabs for 'All applications', 'Owned applications' (which is selected and highlighted with a red box), and 'Deleted applications'. A search bar allows filtering by display name or application ID. The results table shows two applications: 'GetStartedWebApp' (client ID e415111b-5d76-4c9f...) and 'Best Practices Demo' (client ID 66a481f5-e1e0-450d...).

In the app registration overview, select **Delete**.



The screenshot shows the Microsoft Entra admin center interface. On the left, there's a sidebar with navigation links: Home, Favorites, Identity, Overview, Users, and Groups. The main area is titled 'GetStartedWebApp' and shows the 'Overview' tab selected. At the top right, there are buttons for 'Delete', 'Endpoints', and 'Preview features'. A feedback message is displayed: 'Got a second? We would love your feedback on Microsoft identity platform (previously Azure AD for developer.)'. Below the tabs, there's a section for 'Manage' with 'Display name' set to 'GetStartedWebApp'.

Delete the external tenant

If you created a new external tenant, you can [delete it](#). In to the [Microsoft Entra admin center](#), browse to **Identity > Overview > Manage tenants**.

Select the tenant you want to delete, and then select **Delete**.

You might need to complete required actions before you can delete the tenant. For example, you might need to delete all user flows and app registrations in the tenant.

If you're ready to delete the tenant, select **Delete**.

Next steps

In this tutorial, you learned how to:

- ✓ Grant delegated permissions to a web app.
- ✓ Call Microsoft Graph from a web app for a signed-in user.

App service accesses Microsoft Graph as the app

Tutorial: Connect an App Service app to SQL Database on behalf of the signed-in user

Article • 10/12/2023

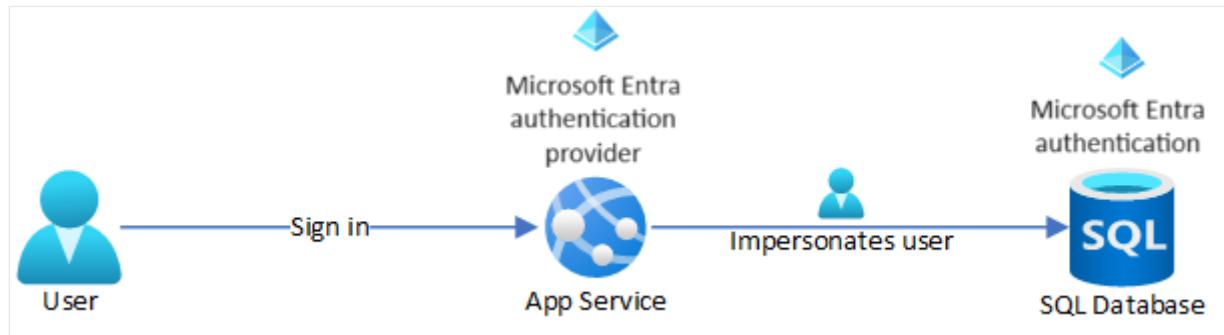
This tutorial shows you how to enable [built-in authentication](#) in an [App Service](#) app using the Microsoft Entra authentication provider, then extend it by connecting it to a back-end Azure SQL Database by impersonating the signed-in user (also known as the [on-behalf-of flow](#)). This is a more advanced connectivity approach to [Tutorial: Access data with managed identity](#) and has the following advantages in enterprise scenarios:

- Eliminates connection secrets to back-end services, just like the managed identity approach.
- Gives the back-end database (or any other Azure service) more control over who or how much to grant access to its data and functionality.
- Lets the app tailor its data presentation to the signed-in user.

In this tutorial, you add Microsoft Entra authentication to the sample web app you deployed in one of the following tutorials:

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

When you're finished, your sample app will authenticate users connect to SQL Database securely on behalf of the signed-in user.



ⓘ Note

The steps covered in this tutorial support the following versions:

- .NET Framework 4.8 and higher
- .NET 6.0 and higher

What you will learn:

- ✓ Enable built-in authentication for Azure SQL Database
- ✓ Disable other authentication options in Azure SQL Database
- ✓ Enable App Service authentication
- ✓ Use Microsoft Entra ID as the identity provider
- ✓ Access Azure SQL Database on behalf of the signed-in Microsoft Entra user

ⓘ Note

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

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

Prerequisites

This article continues where you left off in either one of the following tutorials:

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

If you haven't already, follow one of the two tutorials first. Alternatively, you can adapt the steps for your own .NET app with SQL Database.

Prepare your environment for the Azure CLI.

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:

[Expand table](#)

Option	Example/Link
Select Try It in the upper-right corner of a code or command block. Selecting Try It doesn't automatically copy the code or command	

Option	Example/Link
to Cloud Shell.	
Go to https://shell.azure.com , or select the Launch Cloud Shell button to open Cloud Shell in your browser.	Launch Cloud Shell 
Select the Cloud Shell button on the menu bar at the upper right in the Azure portal .	

To use Azure Cloud Shell:

1. Start Cloud Shell.
2. Select the **Copy** button on a code block (or command block) to copy the code or command.
3. Paste the code or command 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 or command.

1. Configure database server with Microsoft Entra authentication

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

1. If your Microsoft Entra tenant doesn't have a user yet, create one by following the steps at [Add or delete users using Microsoft Entra ID](#).
2. Find the object ID of the Microsoft Entra user using the `az ad user list` and replace `<user-principal-name>`. The result is saved to a variable.

Azure CLI

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

 Tip

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

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

Azure CLI

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

4. Restrict the database server authentication to Active Directory authentication. This step effectively disables SQL authentication.

Azure CLI

```
az sql server ad-only-auth enable --resource-group <group-name> --name <server-name>
```

For more information on adding an Active Directory admin, see [Provision Microsoft Entra admin \(SQL Database\)](#).

2. Enable user authentication for your app

You enable authentication with Microsoft Entra ID as the identity provider. For more information, see [Configure Microsoft Entra authentication for your App Services application](#).

1. In the [Azure portal](#) menu, select **Resource groups** or search for and select **Resource groups** from any page.
2. In **Resource groups**, find and select your resource group, then select your app.
3. In your app's left menu, select **Authentication**, and then select **Add identity provider**.
4. In the **Add an identity provider** page, select **Microsoft** as the **Identity provider** to sign in Microsoft and Microsoft Entra identities.
5. Accept the default settings and select **Add**.

Add an identity provider

Basics Permissions

Identity provider *

Microsoft

Choose a tenant for your application and its users

A tenant contains applications and a directory for user accounts. Choose a tenant based on whether it's configured for workforce users (employees and business guests) or external users. [Learn more ↗](#)

Workforce configuration (current tenant)
Manage employees and business guests

External configuration
Manage external users

App registration

An app registration associates your identity provider with your app. Enter the app registration information here, or go to your provider to create a new one. [Learn more ↗](#)

App registration type *

Create new app registration

Pick an existing app registration in this directory

Provide the details of an existing app registration

Name * ⓘ

my-demo-app

Supported account types *

Current tenant - Single tenant

Any Microsoft Entra directory - Multi-tenant

Any Microsoft Entra directory & personal Microsoft accounts

Personal Microsoft accounts only

[Help me choose...](#)

Additional checks

You can configure additional checks that will further control access, but your app may still need to make additional authorization decisions in code. [Learn more ↗](#)

Client application requirement *

Allow requests only from this application itself

Allow requests from specific client applications

Allow requests from any application (Not recommended)

Identity requirement *

Allow requests from any identity

Allow requests from specific identities

Tenant requirement *

Allow requests only from the issuer tenant

Allow requests from specific tenants

Use default restrictions based on issuer

App Service authentication settings

Requiring authentication ensures that requests to your app include information about the caller, but your app may still need to make additional authorization decisions to control access. If unauthenticated requests are allowed, any client can call the app and your code will need to handle both authentication and authorization. [Learn more ↗](#)

Restrict access *

Require authentication

Allow unauthenticated access

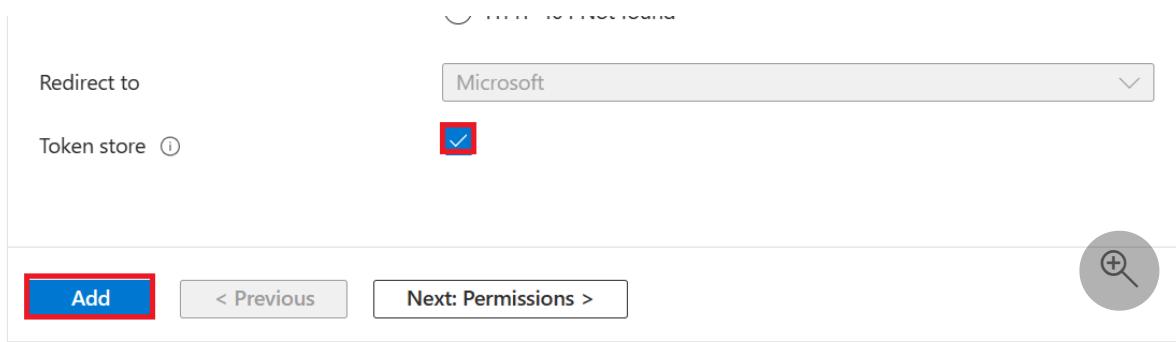
Unauthenticated requests *

HTTP 302 Found redirect: recommended for websites

HTTP 401 Unauthorized: recommended for APIs

HTTP 403 Forbidden

HTTP 404 Not found



💡 Tip

If you run into errors and reconfigure your app's authentication settings, the tokens in the token store may not be regenerated from the new settings. To make sure your tokens are regenerated, you need to sign out and sign back in to your app. An easy way to do it is to use your browser in private mode, and close and reopen the browser in private mode after changing the settings in your apps.

3. Configure user impersonation to SQL Database

Currently, your Azure app connects to SQL Database uses SQL authentication (username and password) managed as app settings. In this step, you give the app permissions to access SQL Database on behalf of the signed-in Microsoft Entra user.

1. In the **Authentication** page for the app, select your app name under **Identity provider**. This app registration was automatically generated for you. Select **API permissions** in the left menu.
2. Select **Add a permission**, then select **APIs my organization uses**.
3. Type *Azure SQL Database* in the search box and select the result.
4. In the **Request API permissions** page for Azure SQL Database, select **Delegated permissions** and **user_impersonation**, then select **Add permissions**.

Request API permissions

X

< All APIs

AS Azure SQL Database
https://sql.azure-synapse-dogfood.net

What type of permissions does your application require?

Delegated permissions

Your application needs to access the API as the signed-in user.

Application permissions

Your application runs as a background service or daemon without a signed-in user.

Select permissions

expand all

 The "Admin consent required" column shows the default value for an organization. However, user consent can be customized per permission, user, or app. This column may not reflect the value in your organization, or in organizations where this app will be used. [Learn more](#) X

Permission

Admin consent required

Permissions (1)



user_impersonation ⓘ

Access Azure SQL DB and Data Warehouse

No

Add permissions

Discard



4. Configure App Service to return a usable access token

The app registration in Microsoft Entra ID now has the required permissions to connect to SQL Database by impersonating the signed-in user. Next, you configure your App Service app to give you a usable access token.

In the Cloud Shell, run the following commands on the app to add the `scope` parameter to the authentication setting

`identityProviders.azureActiveDirectory.login.loginParameters`. It uses `[jq]` for JSON processing, which is installed already in the Cloud Shell.

Azure CLI

```
authSettings=$(az webapp auth show --resource-group <group-name> --name <app-name>)
authSettings=$(echo "$authSettings" | jq '.properties' | jq
'.identityProviders.azureActiveDirectory.login += {"loginParameters":'
['"scope=openid profile email offline_access'
'https://database.windows.net/user_impersonation"]}'')
```

```
az webapp auth set --resource-group <group-name> --name <app-name> --body  
"$authSettings"
```

The commands effectively add a `loginParameters` property with extra custom scopes. Here's an explanation of the requested scopes:

- `openid`, `profile`, and `email` are requested by App Service by default already. For information, see [OpenID Connect Scopes](#).
- `https://database.windows.net/user_impersonation` refers to Azure SQL Database. It's the scope that gives you a JWT token that includes SQL Database as a `token audience` ↗.
- `offline_access` is included here for convenience (in case you want to [refresh tokens](#)).

💡 Tip

To configure the required scopes using a web interface instead, see the Microsoft steps at [Refresh auth tokens](#).

Your apps are now configured. The app can now generate a token that SQL Database accepts.

5. Use the access token in your application code

The steps you follow for your project depends on whether you're using [Entity Framework](#) (default for ASP.NET) or [Entity Framework Core](#) (default for ASP.NET Core).

Entity Framework

1. In Visual Studio, open the Package Manager Console and update Entity Framework:

PowerShell

```
Update-Package EntityFramework
```

2. In your `DbContext` object (in `Models/MyDbContext.cs`), add the following code to the default constructor.

C#

```
var conn =  
(System.Data.SqlClient.SqlConnection)Database.Connection;  
conn.AccessToken =  
System.Web.HttpContext.Current.Request.Headers["X-MS-TOKEN-AAD-  
ACCESS-TOKEN"];
```

ⓘ Note

The code adds the access token supplied by App Service authentication to the connection object.

This code change doesn't work locally. For more information, see [How do I debug locally when using App Service authentication?](#).

6. Publish your changes

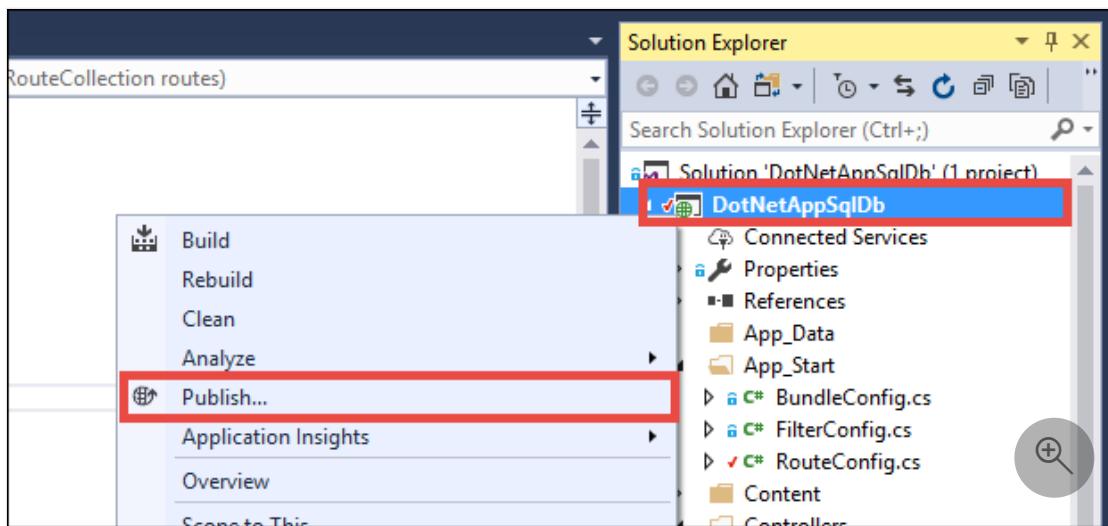
ASP.NET

1. If you came from [Tutorial: Build an ASP.NET app in Azure with SQL Database](#), you set a connection string in App Service using SQL authentication, with a username and password. Use the following command to remove the connection secrets, but replace <group-name>, <app-name>, <db-server-name>, and <db-name> with yours.

Azure CLI

```
az webapp config connection-string set --resource-group <group-name> --name <app-name> --connection-string-type SQLAzure --settings MySqlConnectionString="server=tcp:<db-server-name>.database.windows.net;database=<db-name>;"
```

2. Publish your changes in Visual Studio. In the **Solution Explorer**, right-click your **DotNetAppSqlDb** project and select **Publish**.



3. In the publish page, select Publish.

When the new webpage shows your to-do list, your app is connecting to the database on behalf of the signed-in Microsoft Entra user.

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

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

7. 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:

Azure CLI

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

This command may take a minute to run.

Frequently asked questions

- Why do I get a Login failed for user '<token-identified principal>'. error?
- How do I add other Microsoft Entra users or groups in Azure SQL Database?
- How do I debug locally when using App Service authentication?
- What happens when access tokens expire?

Why do I get a `Login failed for user '<token-identified principal>'.` error?

The most common causes of this error are:

- You're running the code locally, and there's no valid token in the `X-MS-TOKEN-AAD-ACCESS-TOKEN` request header. See [How do I debug locally when using App Service authentication?](#).
- Microsoft Entra authentication isn't configured on your SQL Database.
- The signed-in user isn't permitted to connect to the database. See [How do I add other Microsoft Entra users or groups in Azure SQL Database?](#).

How do I add other Microsoft Entra users or groups in Azure SQL Database?

1. Connect to your database server, such as with `sqlcmd` or [SSMS](#).
2. [Create contained users mapped to Microsoft Entra identities](#) in SQL Database documentation.

The following Transact-SQL example adds a Microsoft Entra identity to SQL Server and gives it some database roles:

SQL

```
CREATE USER [<user-or-group-name>] FROM EXTERNAL PROVIDER;
ALTER ROLE db_datareader ADD MEMBER [<user-or-group-name>];
ALTER ROLE db_datawriter ADD MEMBER [<user-or-group-name>];
```

```
ALTER ROLE db_ddladmin ADD MEMBER [<user-or-group-name>];  
GO
```

How do I debug locally when using App Service authentication?

Because App Service authentication is a feature in Azure, it's not possible for the same code to work in your local environment. Unlike the app running in Azure, your local code doesn't benefit from the authentication middleware from App Service. You have a few alternatives:

- Connect to SQL Database from your local environment with [Active Directory Interactive](#). The authentication flow doesn't sign in the user to the app itself, but it does connect to the back-end database with the signed-in user, and allows you to test database authorization locally.
- Manually copy the access token from `https://<app-name>.azurewebsites.net/.auth/me` into your code, in place of the `X-MS-TOKEN-AAD-ACCESS-TOKEN` request header.
- If you deploy from Visual Studio, use remote debugging of your App Service app.

What happens when access tokens expire?

Your access token expires after some time. For information on how to refresh your access tokens without requiring users to reauthenticate with your app, see [Refresh identity provider tokens](#).

Next steps

What you learned:

- ✓ Enable built-in authentication for Azure SQL Database
- ✓ Disable other authentication options in Azure SQL Database
- ✓ Enable App Service authentication
- ✓ Use Microsoft Entra ID as the identity provider
- ✓ Access Azure SQL Database on behalf of the signed-in Microsoft Entra user

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

[Tutorial: Access Microsoft Graph from a secured .NET app as the app](#)

[Tutorial: Isolate back-end communication with Virtual Network integration](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

Tutorial: Authenticate and authorize users end-to-end in Azure App Service

Article • 12/31/2023

Azure App Service provides a highly scalable, self-patching web hosting service using the Linux operating system. In addition, App Service has built-in support for [user authentication and authorization](#). This tutorial shows how to secure your apps with App Service authentication and authorization. It uses an Express.js with views. App Service authentication and authorization support all language runtimes, and you can learn how to apply it to your preferred language by following the tutorial.

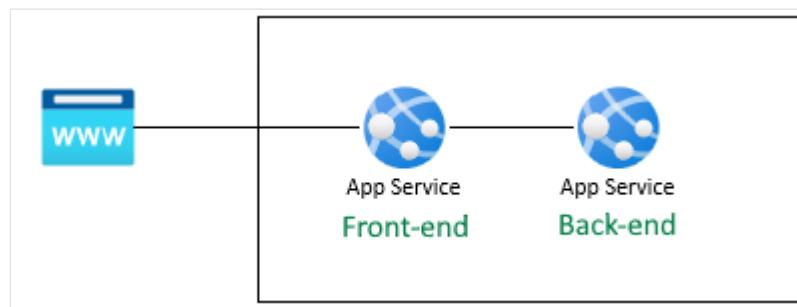
In the tutorial, you learn:

- ✓ Enable built-in authentication and authorization
- ✓ Secure apps against unauthenticated requests
- ✓ Use Microsoft Entra ID as the identity provider
- ✓ Access a remote app on behalf of the signed-in user
- ✓ Secure service-to-service calls with token authentication
- ✓ Use access tokens from server code
- ✓ Use access tokens from client (browser) code

Tip

After completing this scenario, continue to the next procedure to learn how to connect to Azure services as an authenticated user. Common scenarios include accessing Azure Storage or a database as the user who has specific abilities or access to specific tables or files.

The authentication in this procedure is provided at the hosting platform layer by Azure App Service. You must deploy the frontend and backend app and configure authentication for this web app to be used successfully.



Get the user profile

The frontend app is configured to securely use the backend API. The frontend application provides a Microsoft sign-in for the user, then allows the user to get their *fake* profile from the backend. This tutorial uses a fake profile to simplify the steps to complete the scenario.

Before your source code is executed on the frontend, the App Service injects the authenticated `accessToken` from the App Service `x-ms-token-aad-access-token` header. The frontend source code then accesses and sends the `accessToken` to the backend server as the `bearerToken` to securely access the backend API. The backend server validates the `bearerToken` before it's passed into your backend source code. Once your backend source code receives the `bearerToken`, it can be used.

In [the next article in this series](#), the `bearerToken` is exchanged for a token with a scope to access the Microsoft Graph API. The Microsoft Graph API returns the user's profile information.

Prerequisites

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

- [Node.js \(LTS\)](#)
 - Use the Bash environment in [Azure Cloud Shell](#). For more information, see [Quickstart for Bash in Azure Cloud Shell](#).
-  [Launch Cloud Shell](#)
- If you prefer to run CLI reference commands locally, [install](#) the Azure CLI. If you're running on Windows or macOS, consider running Azure CLI in a Docker container. For more information, see [How to run the Azure CLI in a Docker container](#).
 - 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 other sign-in options, see [Sign in with the Azure CLI](#).
 - When you're prompted, install the Azure CLI extension 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](#).

1. Clone the sample application

1. In the [Azure Cloud Shell](#), run the following command to clone the sample repository.

```
Azure CLI
```

```
git clone https://github.com/Azure-Samples/js-e2e-web-app-easy-auth-app-to-app
```

2. Create and deploy apps

Create the resource group, web app plan, the web app and deploy in a single step.

1. Change into the frontend web app directory.

```
Azure CLI
```

```
cd frontend
```

2. Create and deploy the frontend web app with `az webapp up`. Because web app name has to be globally unique, replace `<front-end-app-name>` with a unique set of initials or numbers.

```
Azure CLI
```

```
az webapp up --resource-group myAuthResourceGroup --name <front-end-app-name> --plan myPlan --sku FREE --location "West Europe" --os-type Linux --runtime "NODE:16-lts"
```

3. Change into the backend web app directory.

```
Azure CLI
```

```
cd ../backend
```

4. Deploy the backend web app to same resource group and app plan. Because web app name has to be globally unique, replace `<back-end-app-name>` with a unique set of initials or numbers.

```
Azure CLI
```

```
az webapp up --resource-group myAuthResourceGroup --name <back-end-app-name> --plan myPlan --sku FREE --location "West Europe" --runtime "NODE:16-lts"
```

3. Configure app setting

The frontend application needs to know the URL of the backend application for API requests. Use the following Azure CLI command to configure the app setting. The URL should be in the format of `https://<back-end-app-name>.azurewebsites.net`.

Azure CLI

```
az webapp config appsettings set --resource-group myAuthResourceGroup --name <front-end-app-name> --settings BACKEND_URL="https://<back-end-app-name>.azurewebsites.net"
```

4. Frontend calls the backend

Browse to the frontend app and return the *fake* profile from the backend. This action validates that the frontend is successfully requesting the profile from the backend, and the backend is returning the profile.

1. Open the frontend web app in a browser, `https://<front-end-app-name>.azurewebsites.net`.



2. Select the `Get user's profile` link.
3. View the *fake* profile returned from the backend web app.

Success

```
{  
  "displayName": "John Doe",  
  "withAuthentication": false  
}
```

The `withAuthentication` value of `false` indicates the authentication *isn't* set up yet.

5. Configure authentication

In this step, you enable authentication and authorization for the two web apps. This tutorial uses Microsoft Entra ID as the identity provider.

You also configure the frontend app to:

- Grant the frontend app access to the backend app
- Configure App Service to return a usable token
- Use the token in your code.

For more information, see [Configure Microsoft Entra authentication for your App Services application](#).

Enable authentication and authorization for backend app

1. In the [Azure portal](#) menu, select **Resource groups** or search for and select *Resource groups* from any page.
2. In **Resource groups**, find and select your resource group. In **Overview**, select your backend app.
3. In your backend app's left menu, select **Authentication**, and then select **Add identity provider**.
4. In the **Add an identity provider** page, select **Microsoft** as the **Identity provider** to sign in Microsoft and Microsoft Entra identities.
5. Accept the default settings and select **Add**.

Add an identity provider

Basics Permissions

Identity provider *

Microsoft



App registration

An app registration associates your identity provider with your app. Enter the app registration information here, or go to your provider to create a new one. [Learn more](#)

App registration type *

Create new app registration

Pick an existing app registration in this directory

Provide the details of an existing app registration

Name * ⓘ

dotnet-core-back-end-auth

Supported account types *

Current tenant - Single tenant

Any Azure AD directory - Multi-tenant

Any Azure AD directory & personal Microsoft accounts

Personal Microsoft accounts only

[Help me choose...](#)

App Service authentication settings

Requiring authentication ensures all users of your app will need to authenticate. If you allow unauthenticated requests, you'll need your own code for specific authentication requirements. [Learn more](#)

Authentication *

Require authentication

Allow unauthenticated access

Unauthenticated requests *

HTTP 302 Found redirect: recommended for websites

HTTP 401 Unauthorized: recommended for APIs

HTTP 403 Forbidden

Redirect to *

Microsoft



Token store ⓘ



Add

< Previous

Next: Permissions >

6. The **Authentication** page opens. Copy the **Client ID** of the Microsoft Entra application to a notepad. You need this value later.

The screenshot shows the Azure portal's 'Authentication' settings for an App Service. The left sidebar has a red box around the 'Authentication' link. The main content area shows the 'Authentication settings' section with 'Require authentication' set to 'Enabled'. It also lists 'Unauthenticated requests' (Return HTTP 302 Found (Redirect to identity provider)), 'Redirect to' (Microsoft), and 'Token store' (Enabled). Below this is the 'Identity provider' section, which lists 'Microsoft (dotnet-core-back-end-auth)' with an 'App (client) ID' of '462263c1-2c67-4538-ab6a-1514442a82ed'. There are 'Learn more', 'Edit', and 'Delete' buttons next to it.

If you stop here, you have a self-contained app that's already secured by the App Service authentication and authorization. The remaining sections show you how to secure a multi-app solution by "flowing" the authenticated user from the frontend to the backend.

Enable authentication and authorization for frontend app

1. In the [Azure portal](#) menu, select **Resource groups** or search for and select **Resource groups** from any page.
2. In **Resource groups**, find and select your resource group. In **Overview**, select your frontend app's management page.
3. In your frontend app's left menu, select **Authentication**, and then select **Add identity provider**.
4. In the **Add an identity provider** page, select **Microsoft** as the **Identity provider** to sign in Microsoft and Microsoft Entra identities.
5. Accept the default settings and select **Add**.
6. The **Authentication** page opens. Copy the **Client ID** of the Microsoft Entra application to a notepad. You need this value later.

Grant frontend app access to backend

Now that you've enabled authentication and authorization to both of your apps, each of them is backed by an AD application. To complete the authentication, you need to do three things:

- Grant the frontend app access to the backend app
- Configure App Service to return a usable token

- Use the token in your code.

💡 Tip

If you run into errors and reconfigure your app's authentication/authorization settings, the tokens in the token store may not be regenerated from the new settings. To make sure your tokens are regenerated, you need to sign out and sign back in to your app. An easy way to do it is to use your browser in private mode, and close and reopen the browser in private mode after changing the settings in your apps.

In this step, you **grant the frontend app access to the backend app** on the user's behalf. (Technically, you give the frontend's *AD application* the permissions to access the backend's *AD application* on the user's behalf.)

1. In the **Authentication** page for the frontend app, select your frontend app name under **Identity provider**. This app registration was automatically generated for you. Select **API permissions** in the left menu.
2. Select **Add a permission**, then select **My APIs > <back-end-app-name>**.
3. In the **Request API permissions** page for the backend app, select **Delegated permissions** and **user_impersonation**, then select **Add permissions**.

Request API permissions

X

< All APIs

dotnet-core-back-end-auth

api://XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX

What type of permissions does your application require?

Delegated permissions

Your application needs to access the API as the signed-in user.

Application permissions

Your application runs as a background service or daemon without a signed-in user.

Select permissions

expand all

Start typing a permission to filter these results

i The "Admin consent required" column shows the default value for an organization. However, user consent can be customized per permission, user, or app. This column may not reflect the value in your organization, or in organizations where this app will be used. [Learn more](#)

Permission

Admin consent required

Permissions (1)



user_impersonation

Access dotnet-core-back-end-auth

No

Add permissions

Discard

Configure App Service to return a usable access token

The frontend app now has the required permissions to access the backend app as the signed-in user. In this step, you configure App Service authentication and authorization to give you a usable access token for accessing the backend. For this step, you need the backend's client ID, which you copied from [Enable authentication and authorization for backend app](#).

In the Cloud Shell, run the following commands on the frontend app to add the `scope` parameter to the authentication setting

`identityProviders.azureActiveDirectory.login.loginParameters`. Replace `<front-end-app-name>` and `<back-end-client-id>`.

Azure CLI

```
az extension add --name authV2
authSettings=$(az webapp auth show -g myAuthResourceGroup -n <front-end-app-name>)
authSettings=$(echo "$authSettings" | jq '.properties' | jq
'.identityProviders.azureActiveDirectory.login += {"loginParameters":'
['scope=openid offline_access api://<back-end-client-
```

```
id>/user_impersonation"]})')
az webapp auth set --resource-group myAuthResourceGroup --name <front-end-
app-name> --body "$authSettings"
```

The commands effectively add a `loginParameters` property with additional custom scopes. Here's an explanation of the requested scopes:

- `openid` is requested by App Service by default already. For information, see [OpenID Connect Scopes](#).
- `offline_access` is included here for convenience (in case you want to [refresh tokens](#)).
- `api://<back-end-client-id>/user_impersonation` is an exposed API in your backend app registration. It's the scope that gives you a JWT token that includes the backend app as a [token audience](#).

💡 Tip

- To view the `api://<back-end-client-id>/user_impersonation` scope in the Azure portal, go to the **Authentication** page for the backend app, click the link under **Identity provider**, then click **Expose an API** in the left menu.
- To configure the required scopes using a web interface instead, see the Microsoft steps at [Refresh auth tokens](#).
- Some scopes require admin or user consent. This requirement causes the consent request page to be displayed when a user signs into the frontend app in the browser. To avoid this consent page, add the frontend's app registration as an authorized client application in the **Expose an API** page by clicking **Add a client application** and supplying the client ID of the frontend's app registration.

Your apps are now configured. The frontend is now ready to access the backend with a proper access token.

For information on how to configure the access token for other providers, see [Refresh identity provider tokens](#).

6. Frontend calls the authenticated backend

The frontend app needs to pass the user's authentication with the correct `user_impersonation` scope to the backend. The following steps review the code

provided in the sample for this functionality.

View the frontend app's source code:

1. Use the frontend App Service injected `x-ms-token-aad-access-token` header to programmatically get the user's accessToken.

JavaScript

```
// ./src/server.js
const accessToken = req.headers['x-ms-token-aad-access-token'];
```

2. Use the accessToken in the `Authentication` header as the `bearerToken` value.

JavaScript

```
// ./src/remoteProfile.js
// Get profile from backend
const response = await fetch(remoteUrl, {
    cache: "no-store", // no caching -- for demo purposes only
    method: 'GET',
    headers: {
        'Authorization': `Bearer ${accessToken}`
    }
});
if (response.ok) {
    const { profile } = await response.json();
    console.log(`profile: ${profile}`);
} else {
    // error handling
}
```

This tutorial returns a *fake* profile to simplify the scenario. The [next tutorial](#) in this series demonstrates how to exchange the backend bearerToken for a new token with the scope of a downstream Azure service, such as Microsoft Graph.

7. Backend returns profile to frontend

If the request from the frontend isn't authorized, the backend App service rejects the request with a 401 HTTP error code *before* the request reaches your application code. When the backend code is reached (because it including an authorized token), extract the bearerToken to get the accessToken.

View the backend app's source code:

JavaScript

```
// ./src/server.js
const bearerToken = req.headers['Authorization'] ||
req.headers['authorization'];

if (bearerToken) {
    const accessToken = bearerToken.split(' ')[1];
    console.log(`backend server.js accessToken: ${!!accessToken ? 'found' :
'not found'}`);
}

// TODO: get profile from Graph API
// provided in next article in this series
// return await getProfileFromMicrosoftGraph(accessToken)

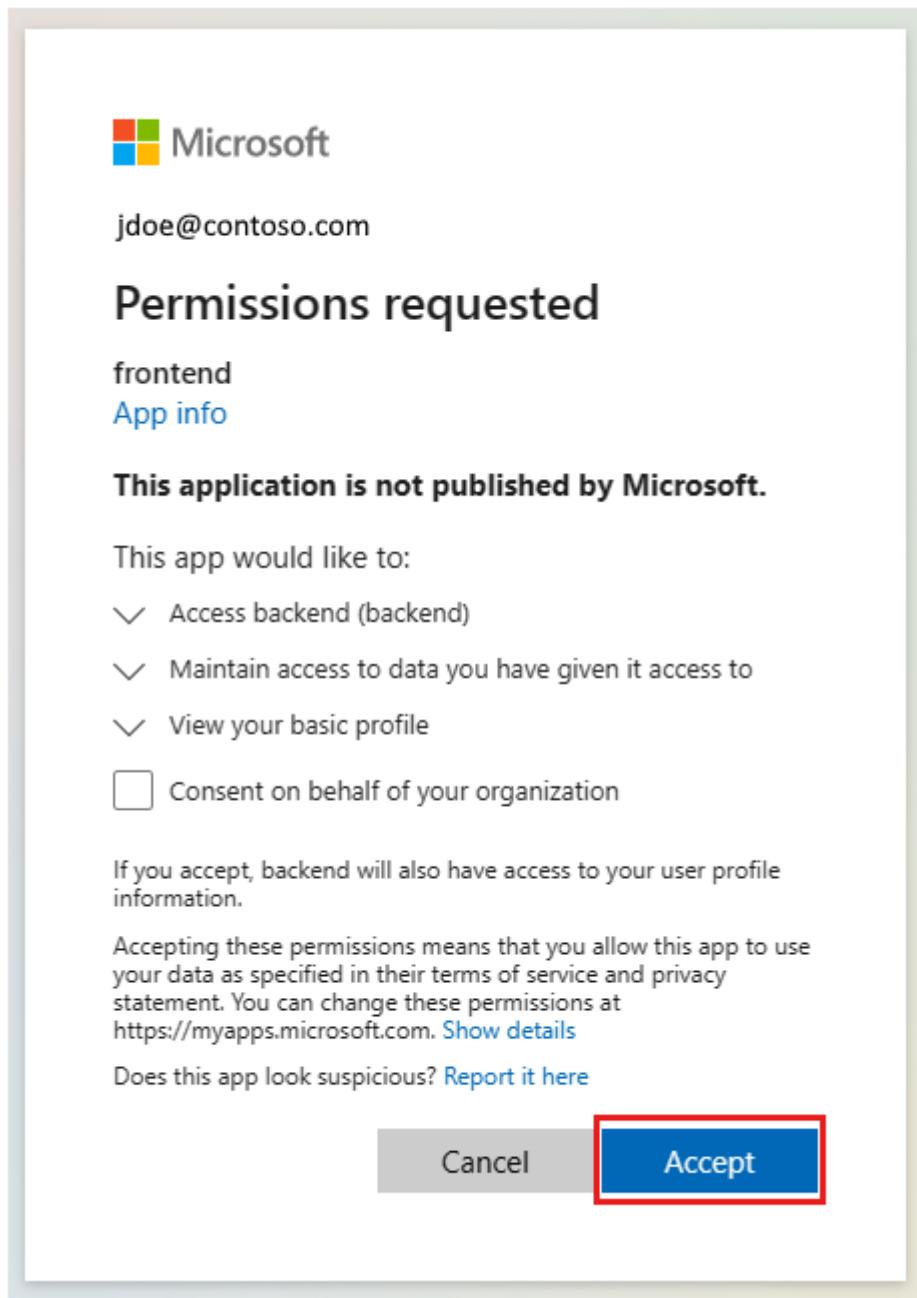
// return fake profile for this tutorial
return {
    "displayName": "John Doe",
    "withAuthentication": !!accessToken ? true : false
}
}
```

8. Browse to the apps

1. Use the frontend web site in a browser. The URL is in the format of

`https://<front-end-app-name>.azurewebsites.net/`.

2. The browser requests your authentication to the web app. Complete the authentication.

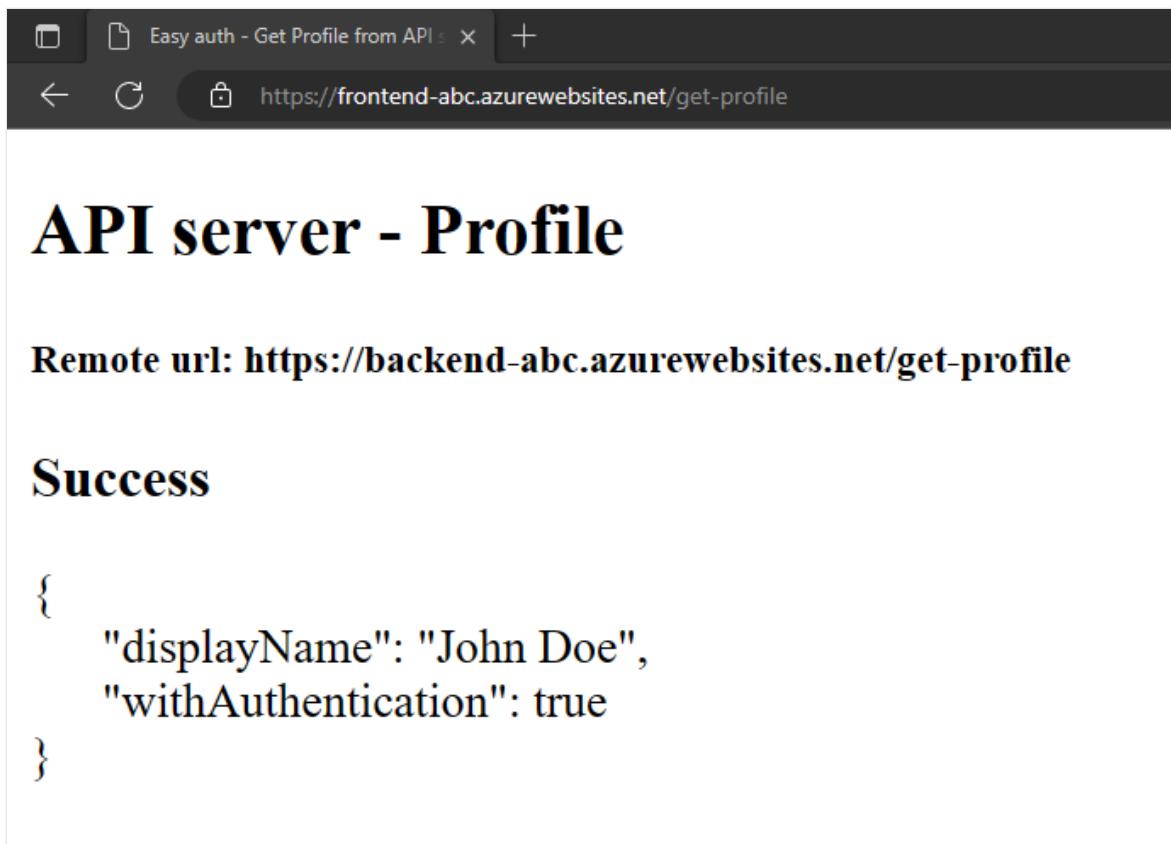


3. After authentication completes, the frontend application returns the home page of the app.

The screenshot shows the home page of the frontend application. The URL in the browser bar is https://frontend-abc.azurewebsites.net. The main content area has a large heading "Easy auth - Get Profile from API server - 1" and a link "Get user's profile" under the heading "from server API".

4. Select `Get user's profile`. This passes your authentication in the bearer token to the backend.

5. The backend end responds with the *fake* hard-coded profile name: `John Doe`.



A screenshot of a Microsoft Edge browser window. The title bar says "Easy auth - Get Profile from API". The address bar shows the URL "https://frontend-abc.azurewebsites.net/get-profile". The main content area displays the JSON response from the API:

```
{  
  "displayName": "John Doe",  
  "withAuthentication": true  
}
```

The `withAuthentication` value of `true` indicates the authentication *is* set up yet.

9. Clean up resources

In the preceding steps, you created Azure resources in a resource group.

1. Delete the resource group by running the following command in the Cloud Shell.
This command may take a minute to run.

```
Azure CLI  
  
az group delete --name myAuthResourceGroup
```

2. Use the authentication apps' **Client ID**, you previously found and made note of in the `Enable authentication and authorization` sections for the backend and frontend apps.
3. Delete app registrations for both frontend and backend apps.

```
Azure CLI  
  
# delete app - do this for both frontend and backend client ids  
az ad app delete <client-id>
```

Frequently asked questions

How do I test this authentication on my local development machine?

The authentication in this procedure is provided at the hosting platform layer by Azure App Service. There's no equivalent emulator. You must deploy the frontend and backend app and configuration authentication for each in order to use the authentication.

The app isn't displaying *fake* profile, how do I debug it?

The frontend and backend apps both have `/debug` routes to help debug the authentication when this application doesn't return the *fake* profile. The frontend debug route provides the critical pieces to validate:

- Environment variables:
 - The `BACKEND_URL` is configured correctly as `https://<back-end-app-name>.azurewebsites.net`. Don't include that trailing forward slash or the route.
- HTTP headers:
 - The `x-ms-token-*` headers are injected.
- Microsoft Graph profile name for signed in user is displayed.
- Frontend app's `scope` for the token has `user_impersonation`. If your scope doesn't include this, it could be an issue of timing. Verify your frontend app's `login` parameters in [Azure resources](#). Wait a few minutes for the replication of the authentication.

Did the application source code deploy correctly to each web app?

1. In the Azure portal for the web app, select **Development Tools -> Advanced Tools**, then select **Go ->**. This opens a new browser tab or window.
2. In the new browser tab, select **Browse Directory -> Site wwwroot**.
3. Verify the following are in the directory:
 - `package.json`
 - `node_modules.tar.gz`
 - `/src/index.js`

4. Verify the package.json's `name` property is the same as the web name, either `frontend` or `backend`.

5. If you changed the source code, and need to redeploy, use `az webapp up` from the directory that has the package.json file for that app.

Did the application start correctly

Both the web apps should return something when the home page is requested. If you can't reach `/debug` on a web app, the app didn't start correctly. Review the error logs for that web app.

1. In the Azure portal for the web app, select **Development Tools -> Advanced Tools**, then select **Go ->**. This opens a new browser tab or window.
2. In the new browser tab, select **Browse Directory -> Deployment Logs**.
3. Review each log to find any reported issues.

Is the frontend app able to talk to the backend app?

Because the frontend app calls the backend app from server source code, this isn't something you can see in the browser network traffic. Use the following list to determine the backend profile request success:

- The backend web app returns any errors to the frontend app if it was reached. If it wasn't reached, the frontend app reports the status code and message.
 - 401: The user didn't pass authentication correctly. This can indicate the scope isn't set correctly.
 - 404: The URL to the server doesn't match a route the server has
- Use the backend app's streaming logs to watch as you make the frontend request for the user's profile. There's debug information in the source code with `console.log` which helps determine where the failure happened.

What happens when the frontend token expires?

Your access token expires after some time. For information on how to refresh your access tokens without requiring users to reauthenticate with your app, see [Refresh identity provider tokens](#).

Next steps

What you learned:

- ✓ Enable built-in authentication and authorization
- ✓ Secure apps against unauthenticated requests
- ✓ Use Microsoft Entra ID as the identity provider
- ✓ Access a remote app on behalf of the signed-in user
- ✓ Secure service-to-service calls with token authentication
- ✓ Use access tokens from server code
- ✓ Use access tokens from client (browser) code

Advance to the next tutorial to learn how to use this user's identity to access an Azure service.

[Create a secure n-tier app in Azure App Service](#)

Tutorial: Flow authentication from App Service through back-end API to Microsoft Graph

Article • 03/22/2023

Learn how to create and configure a backend App service to accept a frontend app's user credential, then exchange that credential for a downstream Azure service. This allows a user to sign in to a frontend App service, pass their credential to a backend App service, then access an Azure service with the same identity.

In this tutorial, you learn how to:

- ✓ Configure the backend authentication app to provide a token scoped for the downstream Azure service
- ✓ Use JavaScript code to exchange the [signed-in user's access token](#) for a new token for downstream service.
- ✓ Use JavaScript code to access downstream service.

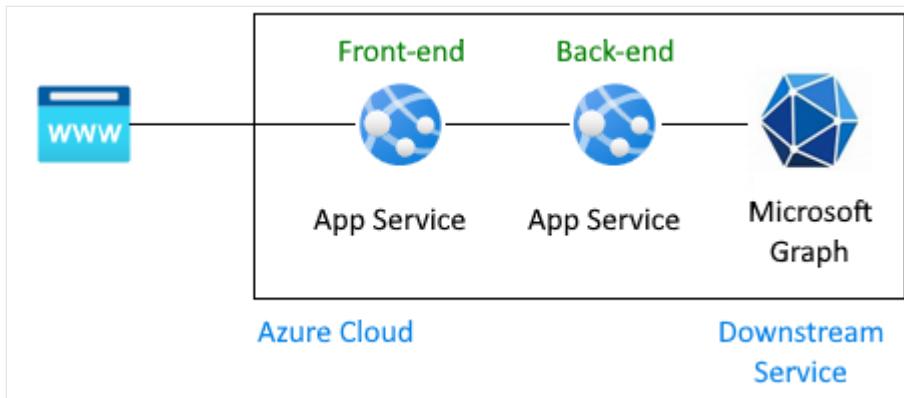
Prerequisites

Complete the previous tutorial, [Access Microsoft Graph from a secured JavaScript app as the user](#), before starting this tutorial but don't remove the resources at the end of the tutorial. This tutorial assumes you have the two App services and their corresponding authentication apps.

The previous tutorial used the Azure Cloud Shell as the shell for the Azure CLI. This tutorial continues that usage.

Architecture

The tutorial shows how to pass the user credential provided by the frontend app to the backend app then on to an Azure service. In this tutorial, the downstream service is Microsoft Graph. The user's credential is used to get their profile from Microsoft Graph.



Authentication flow for a user to get Microsoft Graph information in this architecture:

[Previous tutorial](#) covered:

1. Sign in user to a frontend App service configured to use Active Directory as the identity provider.
2. The frontend App service passes user's token to backend App service.
3. The backend App is secured to allow the frontend to make an API request. The user's access token has an audience for the backend API and scope of `user_impersonation`.
4. The backend app registration already has the Microsoft Graph with the scope `User.Read`. This is added by default to all app registrations.
5. At the end of the previous tutorial, a *fake* profile was returned to the frontend app because Graph wasn't connected.

This tutorial extends the architecture:

1. Grant admin consent to bypass the user consent screen for the back-end app.
2. Change the application code to convert the access token sent from the front-end app to an access token with the required permission for Microsoft Graph.
3. Provide code to have backend app **exchange token** for new token with scope of downstream Azure service such as Microsoft Graph.
4. Provide code to have backend app **use new token** to access downstream service as the current authenticate user.
5. **Redeploy** backend app with `az webapp up`.
6. At the end of this tutorial, a *real* profile is returned to the frontend app because Graph is connected.

This tutorial doesn't:

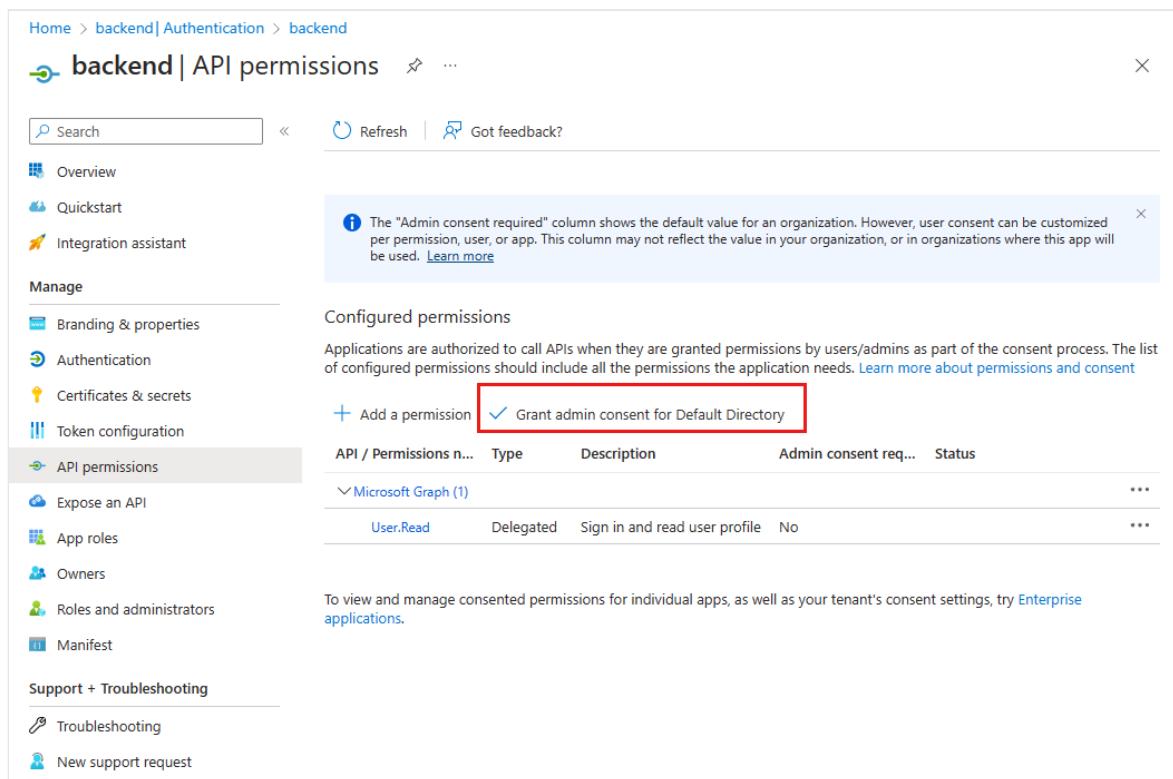
- Change the frontend app from the previous tutorial.
- Change the backend authentication app's scope permission because `User.Read` is added by default to all authentication apps.

1. Configure admin consent for the backend app

In the previous tutorial, when the user signed in to the frontend app, a pop-up displayed asking for user consent.

In this tutorial, in order to read user profile from Microsoft Graph, the back-end app needs to exchange the signed-in user's [access token](#) for a new access token with the required permissions for Microsoft Graph. Because the user isn't directly connected to the backend app, they can't access the consent screen interactively. You must work around this by configuring the back-end app's app registration in Microsoft Entra ID to [grant admin consent](#). This is a setting change typically done by an Active Directory administrator.

1. Open the Azure portal and search for your research for the backend App Service.
2. Find the **Settings** -> **Authentication** section.
3. Select the identity provider to go to the authentication app.
4. In the authentication app, select **Manage** -> **API permissions**.
5. Select **Grant admin consent for Default Directory**.



The screenshot shows the Azure portal interface for managing API permissions. The URL in the address bar is `Home > backend | Authentication > backend`. The main title is `backend | API permissions`. On the left, there's a sidebar with links like Overview, Quickstart, Integration assistant, Manage (which is expanded), Branding & properties, Authentication, Certificates & secrets, Token configuration, API permissions (which is selected and highlighted in grey), Expose an API, App roles, Owners, Roles and administrators, and Manifest. Under the Manage section, there's a link to Support + Troubleshooting with options for Troubleshooting and New support request. The main content area has a heading 'Configured permissions'. A note says: 'The "Admin consent required" column shows the default value for an organization. However, user consent can be customized per permission, user, or app. This column may not reflect the value in your organization, or in organizations where this app will be used.' Below this, there's a button labeled '+ Add a permission' with a checkmark and the text 'Grant admin consent for Default Directory'. A tooltip for this button says: 'Grant admin consent for Default Directory'. A table below lists permissions: API / Permissions n..., Type, Description, Admin consent req..., Status. One row is shown: Microsoft Graph (1) - User.Read, Delegated, Sign in and read user profile, No. At the bottom, a note says: 'To view and manage consented permissions for individual apps, as well as your tenant's consent settings, try Enterprise applications.'

6. In the pop-up window, select **Yes** to confirm the consent.

7. Verify the **Status** column says **Granted for Default Directory**. With this setting, the back-end app is no longer required to show a consent screen to the signed-in user and can directly request an access token. The signed-in user has access to the `User.Read` scope setting because that is the default scope with which the app registration is created.

The screenshot shows the Azure portal's 'API permissions' section for an application named 'backend'. On the left, a sidebar lists various management options like 'Overview', 'Quickstart', and 'Integration assistant'. The 'API permissions' section is currently selected. A message at the top right explains that the 'Admin consent required' column shows the default value for an organization, but user consent can be customized per permission, user, or app. Below this, a table lists configured permissions. The first row, 'Microsoft Graph (1)', contains the 'User.Read' permission, which is highlighted with a red box around its status column. The status column for 'User.Read' shows a green checkmark and the text 'Granted for Default Dire...', also with a red box around it. Other columns in the table include 'API / Permissions n...', 'Type', 'Description', 'Admin consent req...', and 'Status'.

API / Permissions n...	Type	Description	Admin consent req...	Status
Microsoft Graph (1)				Grant admin consent for Default Directory
User.Read	Delegated	Sign in and read user profile	No	Granted for Default Dire...

2. Install npm packages

In the previous tutorial, the backend app didn't need any npm packages for authentication because the only authentication was provided by configuring the identity provider in the Azure portal. In this tutorial, the signed-in user's access token for the back-end API must be exchanged for an access token with Microsoft Graph in its scope. This exchange is completed with two libraries because this exchange doesn't use App Service authentication anymore, but Microsoft Entra ID and MSAL.js directly.

- [@azure/MSAL-node](#) - exchange token
- [@microsoft/microsoft-graph-client](#) - connect to Microsoft Graph

1. Open the Azure Cloud Shell and change into the sample directory's backend app:

```
Azure CLI
cd js-e2e-web-app-easy-auth-app-to-app/backend
```

2. Install the Azure MSAL npm package:

Azure CLI

```
npm install @azure/msal-node
```

3. Install the Microsoft Graph npm package:

Azure CLI

```
npm install @microsoft/microsoft-graph-client
```

3. Add code to exchange current token for Microsoft Graph token

The source code to complete this step is provided for you. Use the following steps to include it.

1. Open the `./src/server.js` file.
2. Uncomment the following dependency at the top of the file:

JavaScript

```
import { getGraphProfile } from './with-graph/graph';
```

3. In the same file, uncomment the `graphProfile` variable:

JavaScript

```
let graphProfile={};
```

4. In the same file, uncomment the following `getGraphProfile` lines in the `get-profile` route to get the profile from Microsoft Graph:

JavaScript

```
// where did the profile come from
profileFromGraph=true;

// get the profile from Microsoft Graph
graphProfile = await getGraphProfile(accessToken);

// log the profile for debugging
console.log(`profile: ${JSON.stringify(graphProfile)}`);
```

5. Save the changes: `Ctrl` + `S`.

6. Redeploy the backend app:

Azure CLI

```
az webapp up --resource-group myAuthResourceGroup --name <back-end-app-name>
```

4. Inspect backend code to exchange backend API token for the Microsoft Graph token

In order to change the backend API audience token for a Microsoft Graph token, the backend app needs to find the Tenant ID and use that as part of the MSAL.js configuration object. Because the backend app was configured with Microsoft as the identity provider, the Tenant ID and several other required values are already in the App service app settings.

The following code is already provided for you in the sample app. You need to understand why it's there and how it works so that you can apply this work to other apps you build that need this same functionality.

Inspect code for getting the Tenant ID

1. Open the `./backend/src/with-graph/auth.js` file.

2. Review the `getTenantId()` function.

JavaScript

```
export function getTenantId() {

    const openIdIssuer = process.env.WEBSITE_AUTH_OPENID_ISSUER;
    const backendAppTenantId =
        openIdIssuer.replace(/https:\/\/sts\.windows\.net\/(.{1,36})\/v2\.0/gm,
        '$1');

    return backendAppTenantId;
}
```

3. This function gets the current tenant ID from the `WEBSITE_AUTH_OPENID_ISSUER` environment variable. The ID is parsed out of the variable with a regular

expression.

Inspect code to get Graph token using MSAL.js

1. Still in the `./backend/src/with-graph/auth.js` file, review the `getGraphToken()` function.
2. Build the MSAL.js configuration object, use the MSAL configuration to create the clientCredentialAuthority. Configure the on-behalf-of request. Then use the acquireTokenOnBehalfOf to exchange the backend API access token for a Graph access token.

JavaScript

```
// ./backend/src/auth.js
// Exchange current bearerToken for Graph API token
// Env vars were set by App Service
export async function getGraphToken(backEndAccessToken) {

    const config = {
        // MSAL configuration
        auth: {
            // the backend's authentication CLIENT ID
            clientId: process.env.WEBSITE_AUTH_CLIENT_ID,
            // the backend's authentication CLIENT SECRET
            clientSecret:
                process.env.MICROSOFT_PROVIDER_AUTHENTICATION_SECRET,
                // OAuth 2.0 authorization endpoint (v2)
                // should be: https://login.microsoftonline.com/BACKEND-
TENANT-ID
            authority:
                `https://login.microsoftonline.com/${getTenantId()}`
        },
        // used for debugging
        system: {
            loggerOptions: {
                loggerCallback(loglevel, message, containsPii) {
                    console.log(message);
                },
                piiLoggingEnabled: true,
                logLevel: MSAL.LogLevel.Verbose,
            }
        }
    };

    const clientCredentialAuthority = new
MSAL.ConfidentialClientApplication(config);

    const oboRequest = {
        oboAssertion: backEndAccessToken,
        // this scope must already exist on the backend authentication
```

```

app registration
    // and visible in resources.azure.com backend app auth config
    scopes: ["https://graph.microsoft.com/.default"]
}

// This example has App service validate token in runtime
// from headers that can't be set externally

// If you aren't using App service's authentication,
// you must validate your access token yourself
// before calling this code
try {
    const { accessToken } = await
clientCredentialAuthority.acquireTokenOnBehalfOf(oboRequest);
    return accessToken;
} catch (error) {
    console.log(`getGraphToken:error.type = ${error.type}
${error.message}`);
}
}

```

5. Inspect backend code to access Microsoft Graph with the new token

To access Microsoft Graph as a user signed in to the frontend application, the changes include:

- Configuration of the Active Directory app registration with an API permission to the downstream service, Microsoft Graph, with the necessary scope of `User.Read`.
- Grant admin consent to bypass the user consent screen for the back-end app.
- Change the application code to convert the access token sent from the front-end app to an access token with the required permission for the downstream service, Microsoft Graph.

Now that the code has the correct token for Microsoft Graph, use it to create a client to Microsoft Graph then get the user's profile.

1. Open the `./backend/src/graph.js`
2. In the `getGraphProfile()` function, get the token, then the authenticated client from the token then get the profile.

JavaScript

```

//
import graph from "@microsoft/microsoft-graph-client";
import { getGraphToken } from "./auth.js";

```

```

// Create client from token with Graph API scope
export function getAuthenticatedClient(accessToken) {
    const client = graph.Client.init({
        authProvider: (done) => {
            done(null, accessToken);
        }
    });

    return client;
}

export async function getGraphProfile(accessToken) {
    // exchange current backend token for token with
    // graph api scope
    const graphToken = await getGraphToken(accessToken);

    // use graph token to get Graph client
    const graphClient = getAuthenticatedClient(graphToken);

    // get profile of user
    const profile = await graphClient
        .api('/me')
        .get();

    return profile;
}

```

6. Test your changes

1. Use the frontend web site in a browser. The URL is in the format of <https://<front-end-app-name>.azurewebsites.net/>. You may need to refresh your token if it's expired.
2. Select `Get user's profile`. This passes your authentication in the bearer token to the backend.
3. The backend end responds with the *real* Microsoft Graph profile for your account.

API server - Profile

Remote url: <https://backend.azurewebsites.net/get-profile>

Success

```
{  
    "@odata.context": "https://graph.microsoft.com/v1.0/$metadata#users/$entity",  
    "authentication": true,  
    "businessPhones": [],  
    "displayName": "John Doe",  
    "givenName": "John",  
    "id": "70e50f99-AAAA-BBBB-CCCC-DDDD",  
    "jobTitle": null,  
    "mail": null,  
    "mobilePhone": null,  
    "officeLocation": null,  
    "preferredLanguage": null,  
    "surname": "Doe",  
    "userPrincipalName": "jdoe@contoso.com#EXT#@jdoe@contoso.com"  
}
```

7. Clean up

In the preceding steps, you created Azure resources in a resource group.

1. Delete the resource group by running the following command in the Cloud Shell.
This command may take a minute to run.

Azure CLI

```
az group delete --name myAuthResourceGroup
```

2. Use the authentication apps' Client ID, you previously found and made note of in the `Enable authentication and authorization` sections for the backend and frontend apps.
3. Delete app registrations for both frontend and backend apps.

Azure CLI

```
# delete app - do this for both frontend and backend client ids  
az ad app delete <client-id>
```

Frequently asked questions

I got an error `80049217`, what does it mean?

This error, `CompactToken parsing failed with error code: 80049217`, means the backend App service isn't authorized to return the Microsoft Graph token. This error is caused because the app registration is missing the `User.Read` permission.

I got an error `AADSTS65001`, what does it mean?

This error, `AADSTS65001: The user or administrator has not consented to use the application with ID \<backend-authentication-id>. Send an interactive authorization request for this user and resource`, means the backend authentication app hasn't been configured for Admin consent. Because the error shows up in the log for the backend app, the frontend application can't tell the user why they didn't see their profile in the frontend app.

How do I connect to a different downstream Azure service as user?

This tutorial demonstrates an API app authenticated to **Microsoft Graph**, however, the same general steps can be applied to access any Azure service on behalf of the user.

1. No change to the frontend application. Only changes to the backend's authentication app registration and backend app source code.
2. Exchange the user's token scoped for backend API for a token to the downstream service you want to access.
3. Use token in downstream service's SDK to create the client.
4. Use downstream client to access service functionality.

Next steps

- [Tutorial: Create a secure n-tier app in Azure App Service](#)
- [Deploy a Node.js + MongoDB web app to Azure](#)

Scale up an app in Azure App Service

Article • 09/12/2024

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, or disk space, or 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. Basic, Standard, and Premium service plans scale out to as many as 3, 10, and 30 instances, respectively. [App Service Environments](#) in the Isolated tier further increase 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.

Important

[App Service offers an automatic scale-out option to handle varying incoming HTTP requests.](#)

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

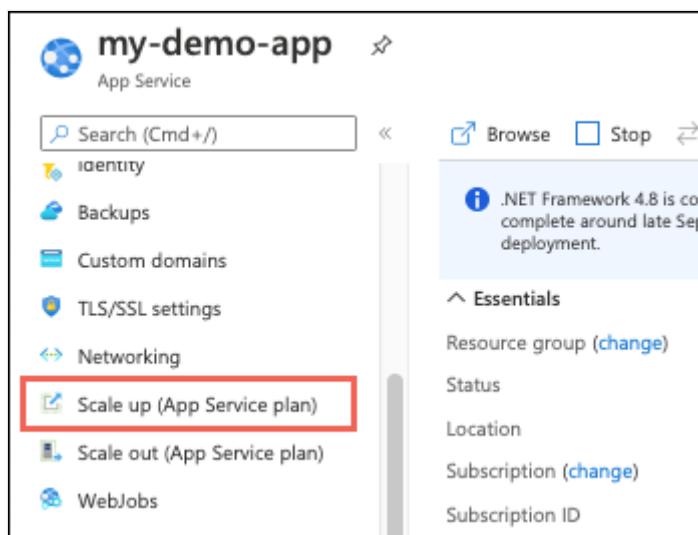
Before you switch an App Service plan from the Free tier, you must first remove the [spending limits](#) in place for your Azure subscription. To view or change options for your App Service subscription, see [Cost Management + Billing](#) in the Azure portal.

Scale up your pricing tier

 **Note**

To scale up to Premium V3 tier, see [Configure Premium V3 tier for App Service](#).

1. In your browser, open the [Azure portal](#).
2. In the left navigation of your App Service app page, select **Scale up (App Service plan)**.



3. Select one of the pricing tiers and select **Select**.

Hardware Features

Name	ACU/vCPU	vCPU	Memory (GB)	Remote Storage (GB)	Scale (instances)
Popular options					
<input type="checkbox"/> Free F1	60 minutes/day...	N/A	1	1	N/A
<input type="checkbox"/> Basic B1	100	1	1.75	10	3
<input type="checkbox"/> Premium v3 P0V3	195*	1	4	250	30
<input type="checkbox"/> Premium v3 P1V3	195	2	8	250	30
<input type="checkbox"/> Premium v3 P2V3	195	4	16	250	30
<input type="checkbox"/> Premium v3 P3V3	195	8	32	250	30
<input checked="" type="checkbox"/> Premium v3 P1mv3	195*	2	16	250	30
<input type="checkbox"/> Premium v3 P2mv3	195*	4	32	250	30
<input type="checkbox"/> Premium v3 P3mv3	195*	8	64	250	30
<input type="checkbox"/> Premium v3 P4mv3	195*	16	128	250	30
<input type="checkbox"/> Premium v3 P5mv3	195*	32	256	250	30
<input type="checkbox"/> Isolated v2 I1V2	195	2	8	1000	N/A
<input type="checkbox"/> Isolated v2 I2V2	195	4	16	1000	N/A
<input type="checkbox"/> Isolated v2 I3V2	195	8	32	1000	N/A
Dev/Test (For less demanding workloads)					
Select	*ACU/vCPU is an approximation of the SKU's relative performance.				Learn more about pricing ↗

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.

my-demo-app
App Service

Search (Ctrl+/
Overview

Browse Stop Swap Restart Delete Get publish profile

Click here to access our Quickstart guide for deploying code to your app →

Resource group (change)
myResourceGroup

Status
Running

Location
West Europe

Subscription (change)
mySubscription

Subscription ID
00000000-0000-0000-0000-000000000000

URL
<https://my-demo-a>

App Service Plan
myAppServicePlan

FTP/deployment user
my-demo-app\user

FTP hostname
ftp://waws-prod-an

FTPS hostname
ftps://waws-prod-a

2. On the **Overview** page for the resource group, select a resource that you want to scale. The following screenshot shows a SQL Database resource.

myResourceGroup
Resource group

Search (Ctrl+/
Overview

Add Edit columns Delete resource group Refresh Move

Subscription (change)
mySubscription

Subscription ID
00000000-0000-0000-0000-000000000000

Tags (change)
Click here to add tags

Deployments
5 Succeeded

Filter by name... 172 types All locations

1 of 4 items selected Show hidden types

NAME	TYPE
dotnetcoredb	SQL server
coreDB (dotnetcoredb/coreDB)	SQL database
myAppServicePlan	App Service plan
my-demo-app	App Service

To scale up the related resource, see the documentation for the specific resource type. For example, to scale up a single SQL database, see [Scale single database resources in Azure SQL Database](#). To scale up an Azure Database for MySQL resource, see [Scale Azure Database for 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](#).

Related content

- [Get started with autoscale in Azure](#)
 - [Configure Premium V3 tier for App Service](#)
 - [Tutorial: Run a load test to identify performance bottlenecks in a web app](#)
-

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

Automatic scaling in Azure App Service

Article • 06/26/2024

ⓘ Note

Automatic scaling is available for all app types: Windows and Linux (deploy as code and container). Automatic scaling is not supported for deployment slot traffic.

Automatic scaling is a new scale-out option that automatically handles scaling decisions for your web apps and App Service Plans. It's different from the pre-existing [Azure autoscale](#), which lets you define scaling rules based on schedules and resources. With automatic scaling, you can adjust scaling settings to improve your app's performance and avoid cold start issues. The platform prewarms instances to act as a buffer when scaling out, ensuring smooth performance transitions. You're charged per second for every instance, including prewarmed instances.

A comparison of scale-out and scale in options available on App Service:

[\[+\] Expand table](#)

	Manual	Autoscale	Automatic scaling
Available pricing tiers	Basic and Up	Standard and Up	Premium V2 (P1V2, P2V2, P3V2) and Premium V3 (P0V3, P1V3, P2V3, P3V3, P1MV3, P2MV3, P3MV3, P4MV3, P5MV3) pricing tiers
Rule-based scaling	No	Yes	No, the platform manages the scale-out and in based on HTTP traffic.
Schedule-based scaling	No	Yes	No
Always ready instances	No, your web app runs on the number of manually scaled instances.	No, your web app runs on other instances available during the scale-out operation, based on threshold defined for autoscale rules.	Yes (minimum 1)
Prewarmed	No	No	Yes (default 1)

	Manual	Autoscale	Automatic scaling
instances			
Per-app maximum	No	No	Yes

How automatic scaling works

You enable automatic scaling for an App Service Plan and configure a range of instances for each of the web apps. As your web app starts receiving HTTP traffic, App Service monitors the load and adds instances. Resources may be shared when multiple web apps within an App Service Plan are required to scale-out simultaneously.

Here are a few scenarios where you should scale-out automatically:

- You don't want to set up autoscale rules based on resource metrics.
- You want your web apps within the same App Service Plan to scale differently and independently of each other.
- Your web app is connected to a databases or legacy system, which may not scale as fast as the web app. Scaling automatically allows you to set the maximum number of instances your App Service Plan can scale to. This setting helps the web app to not overwhelm the backend.

Enable automatic scaling

Maximum burst is the highest number of instances that your App Service Plan can increase to based on incoming HTTP requests. For Premium v2 & v3 plans, you can set a maximum burst of up to 30 instances. The maximum burst must be equal to or greater than the number of workers specified for the App Service Plan.

Azure portal

To enable automatic scaling, navigate to the web app's left menu and select **scale-out (App Service Plan)**. Select **Automatic**, update the **Maximum burst** value, and select the **Save** button.

Scaling

App service provides multiple features that help applications perform their best when scaling demand changes. You can choose to scale your resource manually to a specific instance count, or via a custom Autoscale rule based policy that scales based on metric(s) thresholds, or schedule instance count which scales during designated time windows. You can also use Automatic Scaling features which enables platform managed scale in and scale out for your apps based on incoming HTTP traffic. [Learn more about Azure Autoscale, Automatic Scaling or view the how-to video.](#)

Scale out method

Manual

Maintain a constant instance count for your application

Automatic

Platform managed scale up and down based on traffic

Rules Based

User defined rules to scale on a schedule or based on any app metric

Maximum burst ⓘ

1

Always ready instances ⓘ

1

Enforce scale out limit ⓘ

Save

Discard

Set minimum number of web app instances

Always ready instances is an app-level setting to specify the minimum number of instances. If load exceeds what the always ready instances can handle, additional instances are added (up to the specified **maximum burst** for the App Service Plan).

Azure portal

To set the minimum number of web app instances, navigate to the web app's left menu and select **scale-out (App Service Plan)**. Update the **Always ready instances** value, and select the **Save** button.

Scaling

App service provides multiple features that help applications perform their best when scaling demand changes. You can choose to scale your resource manually to a specific instance count, or via a custom Autoscale rule based policy that scales based on metric(s) thresholds, or schedule instance count which scales during designated time windows. You can also use Automatic Scaling features which enables platform managed scale in and scale out for your apps based on incoming HTTP traffic.

[Learn more about Azure Autoscale, Automatic Scaling or view the how-to video.](#)

Scale out method

Manual

Maintain a constant instance count for your application

Automatic

Platform managed scale up and down based on traffic

Rules Based

User defined rules to scale on a schedule or based on any app metric

Maximum burst ⓘ



3

Always ready instances ⓘ



1

Enforce scale out limit ⓘ



Maximum scale limit ⓘ



3

[Save](#)

[Discard](#)

Set maximum number of web app instances

The **maximum scale limit** sets the maximum number of instances a web app can scale to. The maximum scale limit helps when a downstream component like a database has limited throughput. The per-app maximum can be between 1 and the **maximum burst**.

Azure portal

To set the maximum number of web app instances, navigate to the web app's left menu and select **scale-out (App Service Plan)**. Select **Enforce scale-out limit**, update the **Maximum scale limit**, and select the **Save** button.

Scaling

App service provides multiple features that help applications perform their best when scaling demand changes. You can choose to scale your resource manually to a specific instance count, or via a custom Autoscale rule based policy that scales based on metric(s) thresholds, or schedule instance count which scales during designated time windows. You can also use Automatic Scaling features which enables platform managed scale in and scale out for your apps based on incoming HTTP traffic.

[Learn more about Azure Autoscale, Automatic Scaling or view the how-to video.](#)

Scale out method

Manual
Maintain a constant instance count for your application

Automatic
Platform managed scale up and down based on traffic

Rules Based
User defined rules to scale on a schedule or based on any app metric

Maximum burst ⓘ

5

Always ready instances ⓘ

1

Enforce scale out limit ⓘ

Maximum scale limit ⓘ

2

[Save](#)

[Discard](#)

Update prewarmed instances

The **prewarmed instance** setting provides warmed instances as a buffer during HTTP scale and activation events. Prewarmed instances continue to buffer until the maximum scale-out limit is reached. The default prewarmed instance count is 1 and, for most scenarios, this value should remain as 1.

Azure portal

You can't change the prewarmed instance setting in the portal, you must instead use the Azure CLI.

Disable automatic scaling

Azure portal

To disable automatic scaling, navigate to the web app's left menu and select **scale-out (App Service Plan)**. Select **Manual**, and select the **Save** button.

Scaling

App service provides multiple features that help applications perform their best when scaling demand changes. You can choose to scale your resource manually to a specific instance count, or via a custom Autoscale rule based policy that scales based on metric(s) thresholds, or schedule instance count which scales during designated time windows. You can also use Automatic Scaling features which enables platform managed scale in and scale out for your apps based on incoming HTTP traffic.

[Learn more about Azure Autoscale, Automatic Scaling or view the how-to video.](#)

Scale out method

Manual

Maintain a constant instance count for your application

Automatic

Platform managed scale up and down based on traffic

Rules Based

User defined rules to scale on a schedule or based on any app metric

Maximum burst (i)



3

Always ready instances (i)



1

Enforce scale out limit (i)



Maximum scale limit (i)



3

Save

Discard

Does automatic scaling support Azure Function apps?

⊗ Caution

Automatic Scaling is disabled when App Service web apps and Azure Function apps are in the same App Service Plan.

No, you can only have Azure App Service web apps in the App Service Plan where you wish to enable automatic scaling. For Functions, it's recommended to use the [Azure Functions Premium plan](#) instead.

How does automatic scaling work behind the scenes?

Applications set to automatically scale are continuously monitored, with worker health assessments occurring at least once every few seconds. If the system detects increased load on the application, health checks become more frequent. In the event of deteriorating worker health and slowing requests, additional instances are requested.

The speed at which instances are added varies based on the individual application's load pattern and startup time. Applications with brief startup times and intermittent bursts of load may see one virtual machine added every few seconds to a minute.

Once the load subsides, the platform initiates a review for potential scaling in. This process typically begins approximately 5-10 minutes after the load stops increasing. During scaling in, instances are removed at a maximum rate of one every few seconds to a minute.

Moreover, if multiple web applications are deployed within the same app service plan, the platform endeavors to allocate resources across available instances based on the load of each individual web application.

How do I get billed for prewarmed instances?

To understand how you're billed for prewarmed instances, consider this scenario: Let's say your web app has five instances that are always ready, along with one prewarmed instance set as the default.

When your web app is idle and not receiving any HTTP requests, it runs with the five always-ready instances. During this time, you aren't billed for a prewarmed instance because the always-ready instances aren't being used, and thus no prewarmed instance is allocated.

However, as soon as your web app starts receiving HTTP requests and the five always-ready instances become active, a prewarmed instance is allocated, and billing for it begins.

If the rate of HTTP requests keeps increasing and App Service decides to scale beyond the initial five instances, it will start utilizing the prewarmed instance. This means that when there are six active instances, a seventh instance is immediately provisioned to fill the prewarmed buffer.

This process of scaling and prewarming continues until the maximum instance count for the app is reached. It's important to note that no instances are prewarmed or activated beyond the maximum instance count.

Why does AppServiceHTTPLogs have log entries similar to "/admin/host/ping" with a 404 status?

App Service Automatic Scaling periodically checks the `/admin/host/ping` endpoint along with other health check mechanisms inherent to the platform. These checks are specifically implemented features. Occasionally, due to existing platform configurations, 404 errors may be returned by these pings. However, it's important to note that these 404 errors shouldn't affect your app's availability or scaling performance.

If your web app returns a 5xx status, these endpoint pings may result in intermittent restarts, though this is uncommon. We are currently implementing enhancements to address these intermittent restarts. Until then, please ensure that your web app doesn't return a 5xx status at this endpoint. Please be aware that these ping endpoints can't be customized.

How do I track the number of scaled-out instances during the Automatic Scaling event?

`AutomaticScalingInstanceCount` metric will report the number of virtual machines on which the app is running including the prewarmed instance if it is deployed. This metric can also be used to track the maximum number of instances your web app scaled out during an Automatic Scaling event. This metric is available only for the apps that have Automatic Scaling enabled.

How does ARR Affinity affect Automatic Scaling?

Azure App Service uses Application Request Routing cookies known as an ARR Affinity. ARR Affinity cookies restrict scaling because they send requests only to servers associated with the cookie, rather than any available instance. For apps that store state, it's better to scale up (increase resources on a single instance). For stateless apps, scaling out (adding more instances) offers more flexibility and scalability. ARR Affinity cookies are enabled by default on App Service. Depending on your application needs, you may choose to disable ARR affinity cookies when using Automatic scaling.

To disable ARR Affinity cookies: select your App Service app, and under **Settings**, select **Configuration**. Next select the **General settings** tab. Under **ARR affinity**, select **Off** and then select the **save** button.

More resources

- [Get started with autoscale in Azure](#)

- Configure PremiumV3 tier for App Service
 - Scale up server capacity
 - High-density hosting
-

Feedback

Was this page helpful?



Yes



No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

Get started with autoscale in Azure

Article • 11/01/2024

Autoscale allows you to automatically scale your applications or resources based on demand. Use Autoscale to provision enough resources to support the demand on your application without over provisioning and incurring unnecessary costs.

This article describes how to configure the autoscale settings for your resources in the Azure portal.

Azure autoscale supports many resource types. For more information about supported resources, see [autoscale supported resources](#).

Discover the autoscale settings in your subscription

[https://www.microsoft.com/en-us/videoplayer/embed/RE4u7ts?postJs||Msg=true ↗](https://www.microsoft.com/en-us/videoplayer/embed/RE4u7ts?postJs||Msg=true)

To discover the resources that you can autoscale, follow these steps.

1. Open the [Azure portal](#). ↗
2. Search for and select *Azure Monitor* using the search bar at the top of the page.
3. Select **Autoscale** to view all the resources for which autoscale is applicable, along with their current autoscale status.
4. Use the filter pane at the top to select resources a specific resource group, resource types, or a specific resource.

The screenshot shows the Microsoft Azure interface for monitoring autoscale settings. On the left, there's a navigation sidebar with sections like 'Key vaults', 'Managed Services' (including 'Managed Prometheus', 'Azure Managed Grafana', and 'Azure Monitor SCOM managed instance'), 'Settings' (with 'Autoscale' highlighted and a red box around it), and 'Support + Troubleshooting'. The main content area displays a table of resources:

Name	Resource type	Resource group	Location	Instance count	Autoscale status
appsvc_linux_centralus...	App Service plan	appsvc_linux_centralus_basic	Central US	1	Not configured
ScaleableAppServicePl...	App Service plan	rg-001	West Central US	3	Enabled
SlackLogicAppsPlan	App Service plan	LogicSlack	West US	0	Not configured
dtcluster	Azure Data Explorer Cluster	DefaultResourceGroup-CUS	East US 2	2	Enabled
VMSS1	Virtual machine scale set	rg-vmss2	East US	1	Enabled
azmonmedia001	Media service	rg-001	East US	0	Not configured
default	Endpoint	rg-001	East US	0	Not configured
azmon-sb-001	Service Bus Namespace	Azmon-001	East US	Resource capacity not found	Not configured

The page shows the instance count and the autoscale status for each resource.

Autoscale statuses are:

- **Not configured:** Autoscale isn't set up yet for this resource.
- **Enabled:** Autoscale is enabled for this resource.
- **Disabled:** Autoscale is disabled for this resource.

You can also reach the scaling page by selecting **Scaling** from the **Settings** menu for each resource.

The screenshot shows the Microsoft Azure interface for managing a virtual machine scale set named 'VMSS1'. On the left, there's a navigation sidebar with sections like 'Overview', 'Activity log', 'Access control (IAM)', 'Tags', 'Diagnose and solve problems', 'Settings' (with 'Scaling' highlighted and a red box around it), and 'Networking'. The main content area displays the scaling configuration:

Scaling

Virtual machine profile

Operating system	Linux
Publisher	-
Offer	-
Plan	-
Capacity reservation group	-

Networking

Public IP address	-
Public IP address (IPv6)	-
Virtual network/subnet	VMSS1VNET/VMSS1Subnet

Size

Size	Standard_DS1_v2
vCPUs	1
RAM	3.5 GiB

Create your first autoscale setting

 **Note**

In addition to the Autoscale instructions in this article, there's new, automatic scaling in Azure App Service. You'll find more on this capability in the [automatic scaling](#) article.

Follow the steps below to create your first autoscale setting.

1. Open the **Autoscale** pane in Azure Monitor and select a resource that you want to scale. The following steps use an App Service plan associated with a web app. You can [create your first ASP.NET web app in Azure in 5 minutes](#).
2. The current instance count is 1. Select **Custom autoscale**.
3. Enter a **Name** and **Resource group** or use the default.
4. Select **Scale based on a metric**.
5. Select **Add a rule**. to open a context pane on the right side.

Autoscale setting ...

VMSS-001 (Virtual machine scale set)

Save Discard Refresh Logs Feedback

Configure Scale-In Policy Predictive charts Run history JSON Notify Diagnostic settings

Autoscale is a built-in feature that helps applications perform their best when demand changes. You can choose to scale your resource manually to a specific instance count, or via a custom Autoscale policy that scales based on metric(s) thresholds, or schedule instance count which scales during designated time windows. Autoscale enables your resource to be performant and cost effective by adding and removing instances based on demand. [Learn more about Azure Autoscale](#) or [view the how-to video](#).

Choose how to scale your resource

- Manual scale**: Maintain a fixed instance count
- Custom autoscale**: Scale on any schedule, based on any metrics (selected)

Custom autoscale

Autoscale setting name * **VMSS-001-Autoscale-441**

Resource group **rg-001**

Predictive autoscale Mode **Disabled** Pre-launch setup of instances (minutes) **0**

Enable Forecast only or Predictive autoscale. [Learn more about Predictive autoscale](#).

Default * Auto created default scale condition [Edit](#) [Delete](#)

Delete warning **The very last or default recurrence rule cannot be deleted. Instead, you can disable autoscale to turn off autoscale.**

Scale mode **Scale based on a metric** Scale to a specific instance count

Rules **Scale is based on metric trigger rules but no rule(s) is defined; click [Add a rule](#) to create a rule. For example: Add a rule that increases instance count by 1 when CPU Percentage is above 70%. If no rules is defined, the resource will be set to default instance count.**

Instance limits Minimum * **1** Maximum * **1** Default * **1**

Schedule **This scale condition is executed when none of the other scale condition(s) match**

+ Add a scale condition

6. The default rule scales your resource by one instance if the **Percentage CPU** metric is greater than 70 percent.

Keep the default values and select **Add**.

7. You've created your first scale-out rule. Best practice is to have at least one scale-in rule. To add another rule, select **Add a rule**.

8. Set **Operator** to *Less than*.

9. Set **Metric threshold** to trigger scale action to **20**.

10. Set **Operation** to *Decrease count by*.

11. Select **Add**.

Scale rule

X

Metric source

Current resource (VMSS-001)

Resource type

Virtual machine scale sets

Resource

VMSS-001

Criteria

Metric namespace *

Virtual Machine Host

Metric name

Percentage CPU

1 minute time grain

Dimension Name

Operator

Dimension Values

Add

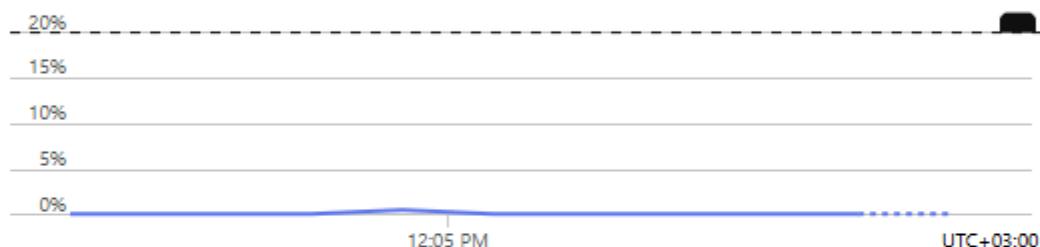
VMName

=

All values



If you select multiple values for a dimension, autoscale will aggregate the metric across the selected values, not evaluate the metric for each values individually.



Percentage CPU (Average)

0.96 %

Enable metric divide by instance count (i)

Operator *

Less than

Metric threshold to trigger scale action * (i)

20



%

Duration (minutes) * (i)

10

Time grain (minutes) (i)

1

Time grain statistic * (i)

Average

Time aggregation * (i)

Average

Action

Operation *

Decrease count by

Cool down (minutes) * (i)

5

instance count *

1



Add



You have configured a scale setting that scales out and scales in based on CPU usage, but you're still limited to a maximum of one instance. Change the instance limits to allow for more instances.

12. Under Instance limits set Maximum to 3

13. Select Save.

Autoscale setting ...
VMSS-001 (Virtual machine scale set)

Save Discard Refresh Logs Feedback

specific instance count, or via a custom Autoscale policy that scales based on metric(s) thresholds, or schedule instance count which scales during designated time windows. Autoscale enables your resource to be performant and cost effective by adding and removing instances based on demand. [Learn more about Azure Autoscale](#) or [view the how-to video](#).

Choose how to scale your resource

Manual scale Maintain a fixed instance count

Custom autoscale Scale on any schedule, based on any metrics

Custom autoscale

Autoscale setting name * VMSS-001-Autoscale-441

Resource group rg-001

Predictive autoscale Mode Disabled Pre-launch setup of instances (minutes) 0

Enable Forecast only or Predictive autoscale. [Learn more about Predictive autoscale](#).

Default* Auto created default scale condition

Delete warning The very last or default recurrence rule cannot be deleted. Instead, you can disable autoscale to turn off autoscale.

Scale mode Scale based on a metric

Rules

Scale out When VMSS-001 (Average) Percentage CPU > 70 Increase count by 1

Scale in When VMSS-001 (Average) Percentage CPU < 20 Decrease count by 1

+ Add a rule

Instance limits Minimum * 1 Maximum * 3 Default * 1

Schedule This scale condition is executed when none of the other scale condition(s) match

+ Add a scale condition

You have successfully created your first scale setting to autoscale your web app based on CPU usage. When CPU usage is greater than 70%, an additional instance is added, up to a maximum of 3 instances. When CPU usage is below 20%, an instance is removed up to a minimum of 1 instance. By default there will be 1 instance.

Scheduled scale conditions

The default scale condition defines the scale rules that are active when no other scale condition is in effect. You can add scale conditions that are active on a given date and time, or that recur on a weekly basis.

Scale based on a repeating schedule

Set your resource to scale to a single instance on a Sunday.

1. Select **Add a scale condition**.
2. Enter a description for the scale condition.
3. Select **Scale to a specific instance count**. You can also scale based on metrics and thresholds that are specific to this scale condition.
4. Enter **1** in the **Instance count** field.
5. Select **Repeat specific days**.
6. Select **Sunday**
7. Set the **Start time** and **End time** for when the scale condition should be applied.
Outside of this time range, the default scale condition applies.
8. Select **Save**

The screenshot shows the 'Add a scale condition' dialog box. The 'Scale down Sundays' condition is selected. The 'Scale mode' is set to 'Scale to a specific instance count' (1 instance). The 'Repeat specific days' option is selected, and 'Sunday' is checked. The 'Start time' is set to 00:00 and the 'End time' is set to 23:59. A note at the bottom states: 'Specify an end time, else this scale condition will apply for all days until it reaches the start time of another scale condition'. There is a magnifying glass icon in the bottom right corner.

You have now defined a scale condition that reduces the number of instances of your resource to 1 every Sunday.

Scale differently on specific dates

Set Autoscale to scale differently for specific dates, when you know that there will be an unusual level of demand for the service.

1. Select **Add a scale condition**.
2. Select **Scale based on a metric**.
3. Select **Add a rule** to define your scale-out and scale-in rules. Set the rules to be same as the default condition.
4. Set the **Maximum** instance limit to **10**
5. Set the **Default** instance limit to **3**
6. Select **Specify start/end dates**
7. Enter the **Start date** and **End date** for when the scale condition should be applied.

The screenshot shows the Azure portal interface for managing an Autoscale setting. The top navigation bar includes the Microsoft Azure logo, a search bar, and user information (pdavis@contoso.com). The main page title is "Autoscale setting" for "VMSS-001 (Virtual machine scale set)".

Instance limits: Minimum is 1, Maximum is 10 (highlighted with a red box), and Default is 3.

Schedule: A condition "Super sale day" is specified, and the note "This scale condition is executed when none of the other scale condition(s) match" is displayed.

Scale mode: "Scale based on a metric" is selected (highlighted with a red box).

Rules: Two rules are defined:

- Scale out:** When VMSS-001 (Average) Percentage CPU > 70, Increase count by 1.
- Scale in:** When VMSS-001 (Average) Percentage CPU < 20, Decrease count by 1.

Instance limits: Minimum is 1, Maximum is 10 (highlighted with a red box), and Default is 3 (highlighted with a red box).

Schedule: "Specify start/end dates" is selected (highlighted with a red box).

Timezone: (UTC+02:00) E. Europe.

Start date: 27/08/2024 at 12:00:00 AM (highlighted with a red box).

End date: 27/08/2024 at 11:59:00 PM (highlighted with a red box).

You have now defined a scale condition for a specific day. When CPU usage is greater than 70%, an additional instance is added, up to a maximum of 10 instances to handle anticipated load. When CPU usage is below 20%, an instance is removed up to a minimum of 1 instance. By default, autoscale scales to 3 instances when this scale condition becomes active.

Additional settings

View the history of your resource's scale events

Whenever your resource has any scaling event, it's logged in the activity log. You can view the history of the scale events in the **Run history** tab.

The screenshot shows the Microsoft Azure interface for monitoring an Autoscale setting named "VMSS-001". The "Run history" tab is active. A chart titled "Observed resource instance count" displays the number of instances over time, showing a step-down from 3 to 1 at approximately 2:15 PM. Below the chart is a table of "Autoscale events for this time range" with the following data:

Operation name	Status	Time	Time stamp
> Autoscale scale down completed	Succeeded	an hour ago	Mon Apr 03 2023 14:16:17 ...
> Autoscale scale down initiated	Succeeded	2 hours ago	Mon Apr 03 2023 14:10:19 ...
> Autoscale scale up completed	Succeeded	2 hours ago	Mon Apr 03 2023 14:04:17 ...
> Autoscale scale down completed	Succeeded	2 hours ago	Mon Apr 03 2023 14:01:53 ...

View the scale settings for your resource

Autoscale is an Azure Resource Manager resource. Like other resources, you can see the resource definition in JSON format. To view the autoscale settings in JSON, select the **JSON** tab.

```
1  {
2      "location": "israelcentral",
3      "tags": {},
4      "properties": {
5          "name": "VMSS-001-Autoscale-441",
6          "enabled": false,
7          "targetResourceUri": "/subscriptions/aaaa0a0a-bb1b-cc2c-dd3d-eeeeee4e4e/resourceGroups/rg-001/providers/Microsoft.Compute/virtualMachineScaleSets/VMSS-001",
8          "profiles": [
9              {
10                  "name": "Super sale day",
11                  "capacity": {
12                      "minimum": "1",
13                      "maximum": "10",
14                      "default": "3"
15                  },
16                  "rules": [
17                      {
18                          "scaleAction": {
19                              "direction": "Increase",
20                              "type": "ChangeCount",
21                              "value": "1",
22                              "cooldown": "PT5M"
23                          },
24                          "metricTrigger": {
25                              "metricName": "Percentage CPU",
26                              "metricNamespace": "microsoft.compute/virtualmachinescalesets",
27                              "metricResourceUri": "/subscriptions/aaaa0a0a-bb1b-cc2c-dd3d-eeeeee4e4e/resourceGroups/rg-001/providers/Microsoft.Compute/virtualMachineScaleSets/VMSS-001",
28                              "operator": "GreaterThan",
29                              "statistic": "Average",
30                              "threshold": 70,
31                              "timeAggregation": "Average",
32                              "timeGrain": "PT1M"
33                          }
34                      }
35                  ]
36              }
37          ]
38      }
39  }
```

You can make changes in JSON directly, if necessary. These changes will be reflected after you save them.

Predictive autoscale

Predictive autoscale uses machine learning to help manage and scale Azure Virtual Machine Scale Sets with cyclical workload patterns. It forecasts the overall CPU load to your virtual machine scale set, based on your historical CPU usage patterns. It predicts the overall CPU load by observing and learning from historical usage. This process ensures that scale-out occurs in time to meet the demand. For more information, see [Predictive autoscale](#).

Scale-in policy

When scaling a Virtual machine Scale Set, the scale-in policy determines which virtual machines are selected for removal when a scale-in event occurs. The scale-in policy can be set to either **Default**, **NewestVM**, or **OldestVM**. For more information, see [Use custom scale-in policies with Azure Virtual Machine Scale Sets](#).

Configure **Scale-In Policy** Predictive charts Run history JSON Notify Diagnostic settings

Configure the order in which virtual machines are selected for deletion during a scale-in operation. [Learn more about scale-in policies](#)

Scale-In Policy Default – Balance across availability zones, then delete virtual machine with highest instance ID

Apply force delete to scale-in operations

Notify

You can configure notifications to be sent when a scale event occurs. Notifications can be sent to an email address or to a webhook. For more information, see [Autoscale notifications](#).

Cool-down period effects

Autoscale uses a cool-down period. This period is the amount of time to wait after a scale operation before scaling again. The cool-down period allows the metrics to stabilize and avoids scaling more than once for the same condition. Cool-down applies to both scale-in and scale-out events. For example, if the cooldown is set to 10 minutes and Autoscale has just scaled-in, Autoscale won't attempt to scale again for another 10 minutes in either direction. For more information, see [Autoscale evaluation steps](#).

Flapping

Flapping refers to a loop condition that causes a series of opposing scale events. Flapping happens when one scale event triggers an opposite scale event. For example, scaling in reduces the number of instances causing the CPU to rise in the remaining instances. This in turn triggers a scale-out event, which causes CPU usage to drop, repeating the process. For more information, see [Flapping in Autoscale](#) and [Troubleshooting autoscale](#)

Move autoscale to a different region

This section describes how to move Azure autoscale to another region under the same subscription and resource group. You can use REST API to move autoscale settings.

Prerequisites

- Ensure that the subscription and resource group are available and the details in both the source and destination regions are identical.
- Ensure that Azure autoscale is available in the [Azure region you want to move to ↗](#).

Move

Use [REST API](#) to create an autoscale setting in the new environment. The autoscale setting created in the destination region is a copy of the autoscale setting in the source region.

[Diagnostic settings](#) that were created in association with the autoscale setting in the source region can't be moved. You'll need to re-create diagnostic settings in the destination region, after the creation of autoscale settings is completed.

Learn more about moving resources across Azure regions

To learn more about moving resources between regions and disaster recovery in Azure, see [Move resources to a new resource group or subscription](#).

Next steps

- [Create an activity log alert to monitor all autoscale engine operations on your subscription ↗](#)
- [Create an activity log alert to monitor all failed autoscale scale-in/scale-out operations on your subscription ↗](#)
- [Use autoscale actions to send email and webhook alert notifications in Azure Monitor](#)

Feedback

Was this page helpful?



Yes



No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

Tutorial: Add Azure Content Delivery Network to an Azure App Service web app

Article • 09/27/2024

ⓘ Important

Azure CDN Standard from Microsoft (classic) will be retired on **September 30, 2027**. To avoid any service disruption, it is important that you [migrate your Azure CDN Standard from Microsoft \(classic\) profiles](#) to Azure Front Door Standard or Premium tier by September 30, 2027. For more information, see [Azure CDN Standard from Microsoft \(classic\) retirement](#).

Azure CDN from Edgio will be retired on **November 4, 2025**. You must [migrate your workload](#) to Azure Front Door before this date to avoid service disruption. For more information, see [Azure CDN from Edgio retirement FAQ](#).

This tutorial shows how to add [Azure Content Delivery Network](#) to a [web app](#) in [Azure App Service](#). Web apps are services for hosting web applications, REST APIs, and mobile back ends.

Here's the home page of the sample static HTML site that you work with:

The screenshot shows a web browser window with the URL `s://contoso0501.azurewebsites.net/index.html` in the address bar. The main content of the page is the heading "Azure App Service - Sample Static HTML Site". Below this, there are two sections: "Azure App Service Web Apps" and "Azure Content Delivery Network (CDN)".

Azure App Service Web Apps: This section shows a screenshot of the Azure portal interface, specifically the "Web App" blade, displaying various configuration options and logs.

Azure Content Delivery Network (CDN): This section features a diagram illustrating the CDN architecture. It shows a blue cloud icon at the top left, with three dashed green arrows pointing from it to a laptop, a smartphone, and a tablet computer arranged in a triangle below it, representing how the CDN distributes content to multiple devices.

What you learn:

- ✓ Create a content delivery network endpoint.
- ✓ Refresh cached assets.
- ✓ Use query strings to control cached versions.

Prerequisites

To complete this tutorial:

- [Install Git ↗](#)
- [Install the Azure CLI](#)

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

Create the web app

To create the web app that you work with, follow the [static HTML quickstart](#) through the **Browse to the app** step.

Sign in to the Azure portal

Open a browser and sign in to the [Azure portal ↗](#).

Dynamic site acceleration optimization

If you want to optimize your content delivery network endpoint for dynamic site acceleration (DSA), you should use the [content delivery network portal](#) to create your profile and endpoint. With [DSA optimization](#), the performance of web pages with dynamic content is measurably improved. For instructions about how to optimize a content delivery network endpoint for DSA from the content delivery network portal, see [content delivery network endpoint configuration to accelerate delivery of dynamic files](#). Otherwise, if you don't want to optimize your new endpoint, you can use the web app portal to create it by following the steps in the next section. For [Azure CDN from Edgio](#) profiles, you can't change the optimization of a content delivery network endpoint after it has been created.

Create a content delivery network profile and endpoint

In the left navigation, select **App Services**, and then select the app that you created in the static HTML quickstart.

The screenshot shows the Microsoft Azure portal interface. The left sidebar has a 'FAVORITES' section with 'App Services' selected, indicated by a red box. The main content area is titled 'App Services' and shows a single record:

Name	Status	Location	Pricing Tier
contoso0501	Running	West Europe	Free

Filtering is applied: 'Subscription equals Contoso Subscription'. The table header includes sorting options for Name, Status, Location, and Pricing Tier.

In the **App Service** page, in the **Settings** section, select **Networking > Azure CDN**.

 contoso0501 | Networking

Web App

Search Refresh Send us your feedback

Check your web app's networking configuration. Select any of the listed features to change your network set up. [Learn more](#)

Inbound Traffic
Manage access and incoming services.

Features

 Access restriction	<input type="radio"/> Off
 App assigned address	<input checked="" type="radio"/> N/A ⓘ
 Private endpoints	<input checked="" type="radio"/> N/A ⓘ

Inbound address

20.105.232.22

Inbound networking features

-  Azure CDN
-  Azure Front Door

Web App
These custom domains direct traffic to your web app.

Domains

contoso0501.azurewebsites.net

The screenshot shows the Azure portal interface for a Web App named "contoso0501". The left sidebar contains a navigation menu with various options like Overview, Activity log, and Networking. The "Networking" option is highlighted with a red box. The main content area displays the "Inbound Traffic" settings, showing that "Access restriction" is off and "App assigned address" is N/A. It also lists "Private endpoints" as N/A. Below this is the "Inbound address" section with the IP "20.105.232.22". A separate box shows "Inbound networking features" with "Azure CDN" and "Azure Front Door" listed. An arrow points from this box down to the "Web App" section, which details custom domains. The "contoso0501.azurewebsites.net" domain is listed under "Domains".

In the **Azure Content Delivery Network** page, provide the **New endpoint** settings as specified in the table.



Azure Content Delivery Network

The Azure Content Delivery Network (CDN) is designed to send audio, video, images, and other files faster and more reliably to customers using servers that are closest to the users. This dramatically increases speed and availability, resulting in significant user experience improvements.

[Learn more](#)

Endpoints

Click on your endpoint below to manage CDN and configure different features.

Hostname	↑↓	Status	↑↓	Protocol	↑↓
Loading...					

[Migrate custom domains to CDN](#)

New endpoint

CDN profile ⓘ

 Create new Use existing

myCDNProfile



Pricing tier *

Microsoft CDN (classic)

[View full pricing details](#)

CDN endpoint name *

contoso0501



.azureedge.net

Origin hostname ⓘ

contoso0501.azurewebsites.net

[Create](#)[\[+\] Expand table](#)

Setting	Suggested value	Description
content delivery network profile	myCDNProfile	A content delivery network profile is a collection of content delivery network endpoints with the same pricing tier.
Pricing tier	Microsoft content delivery network (classic)	The pricing tier specifies the provider and available features.
content delivery network endpoint name	Any name that is unique in the azureedge.net domain	You access your cached resources at the domain <endpointname>.azureedge.net.

Select **Create** to create a content delivery network profile.

Azure creates the profile and endpoint. The new endpoint appears in the **Endpoints** list, and when it's provisioned, the status is **Running**.

Azure CDN ...



Azure Content Delivery Network

The Azure Content Delivery Network (CDN) is designed to send audio, video, images, and other files faster and more reliably to customers using servers that are closest to the users. This dramatically increases speed and availability, resulting in significant user experience improvements.

[Learn more ↗](#)

Endpoints

Click on your endpoint below to manage CDN and configure different features.

Hostname	Status	Protocol
contoso0501.azureedge.net	✓ Running	HTTP, HTTPS

[Migrate custom domains to CDN ↗](#)

Test the content delivery network endpoint

Because it takes time for the registration to propagate, the endpoint isn't immediately available for use:

- For **Azure CDN Standard from Microsoft (classic)** profiles, propagation usually completes in 10 minutes.
- For **Azure CDN Standard from Edgio** and **Azure CDN Premium from Edgio** profiles, propagation usually completes within 90 minutes.

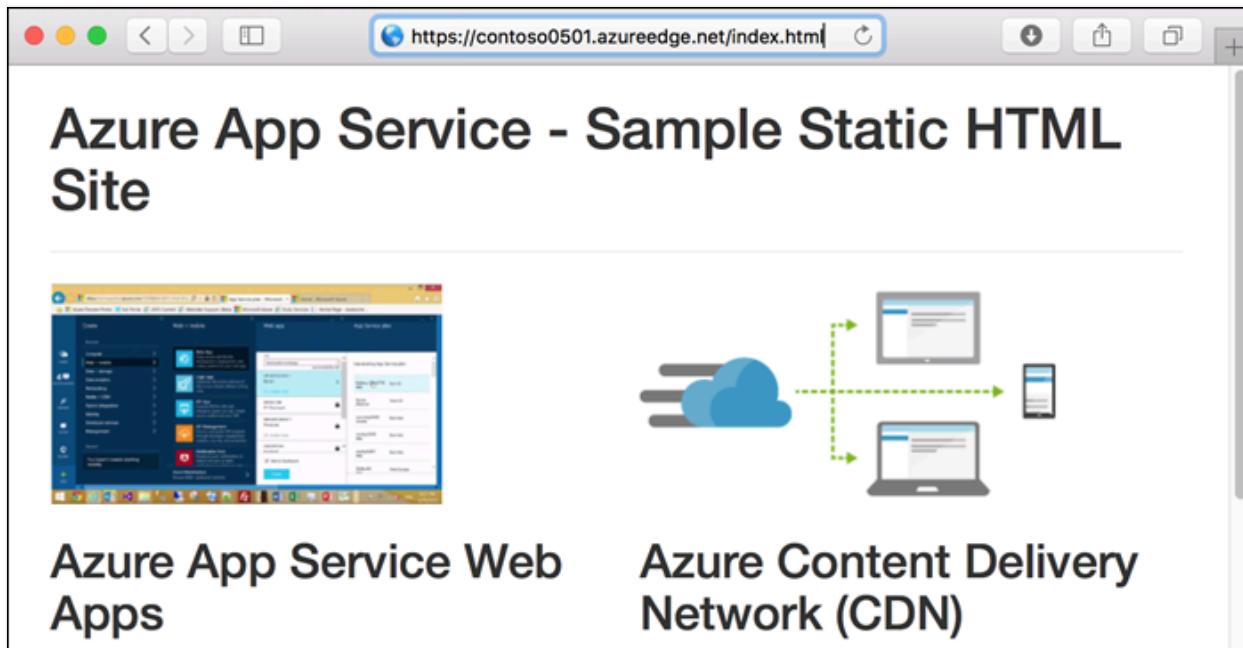
The sample app has an *index.html* file and *css*, *img*, and *js* folders that contain other static assets. The content paths for all of these files are the same at the content delivery network endpoint. For example, both of the following URLs access the *bootstrap.css* file in the *css* folder:

`http://<appname>.azurewebsites.net/css/bootstrap.css`

`http://<endpointname>.azureedge.net/css/bootstrap.css`

Navigate a browser to the following URL:

`http://<endpointname>.azureedge.net/index.html`



Azure App Service - Sample Static HTML Site

Azure App Service Web Apps

Azure Content Delivery Network (CDN)

You see the same page that you ran earlier in an Azure web app. Azure Content Delivery Network has retrieved the origin web app's assets and is serving them from the content delivery network endpoint.

To ensure that this page is cached in the content delivery network, refresh the page. Two requests for the same asset are sometimes required for the content delivery network to cache the requested content.

For more information about creating Azure Content Delivery Network profiles and endpoints, see [Getting started with Azure Content Delivery Network](#).

Purge the content delivery network

The content delivery network periodically refreshes its resources from the origin web app based on the time to live (TTL) configuration. The default TTL is seven days.

At times you might need to refresh the content delivery network before the TTL expiration; for example, when you deploy updated content to the web app. To trigger a refresh, manually purge the content delivery network resources.

In this section of the tutorial, you deploy a change to the web app and purge the content delivery network to trigger the content delivery network to refresh its cache.

Deploy a change to the web app

Open the *index.html* file and add - V2 to the H1 heading, as shown in the following example:

```
<h1>Azure App Service - Sample Static HTML Site - V2</h1>
```

Commit your change and deploy it to the web app.

Bash

```
git commit -am "version 2"  
git push azure main
```

Once deployment has completed, browse to the web app URL to see the change.

```
http://<appname>.azurewebsites.net/index.html
```



If you browse to the content delivery network endpoint URL for the home page, you don't see the changes because the cached version in the content delivery network hasn't expired yet.

```
http://<endpointname>.azureedge.net/index.html
```



Purge the content delivery network in the portal

To trigger the content delivery network to update its cached version, purge the content delivery network.

In the portal left navigation, select **Resource groups**, and then select the resource group that you created for your web app (myResourceGroup).

Microsoft Azure

Search resources, services, and docs (G+)

Create a resource

Home

Dashboard

All services

FAVORITES

All resources

Resource groups

App Services

Function App

SQL databases

Resource groups

Unsecure resources

Recommendations

List view

Name ↑↓

Subscription ↑↓

myResourceGroup

Contoso Subscription

The screenshot shows the Microsoft Azure Resource Groups page. On the left, there's a sidebar with links like 'Create a resource', 'Home', 'Dashboard', 'All services', and 'FAVORITES'. Under 'FAVORITES', 'Resource groups' is selected and highlighted with a red box. The main area shows a list of resources under 'Resource groups'. One item, 'myResourceGroup', is highlighted with a red box. At the top right, there are buttons for 'Create', 'Manage view', 'Refresh', 'Export to CSV', and 'Op'. Below the list, there are filters for 'Name ↑↓' and 'Subscription ↑↓', with 'Contoso Subscription' selected. There are also sections for 'Unsecure resources' and 'Recommendations'.

In the list of resources, select your content delivery network endpoint.

myResourceGroup

Resource group

Search

Overview

Activity log

Access control (IAM)

Tags

Resource visualizer

Events

Settings

Deployments

Security

Policies

Properties

Locks

Cost Management

Cost analysis

Cost alerts (preview)

Budgets

Essentials

Subscription (move)
Contoso Subscription

Subscription ID
abcdef01-2345-6789-0abc-def012345678

Tags (edit)
Click here to add tags

Resources

Filter for any field...

Type equals all

Location equals all

Add filter

Showing 1 to 4 of 4 records.

Name ↑↓

Type ↑↓

Location ↑↓

contoso0501

contoso0501 (myCDNProfile/contoso0501)

contoso0501

myCDNProfile

JSON View

The screenshot shows the 'myResourceGroup' details page. On the left, there's a sidebar with various settings like 'Deployments', 'Security', 'Policies', etc. The main area shows the 'Essentials' section with subscription information ('Contoso Subscription') and tags ('Click here to add tags'). Below that is the 'Resources' section, which lists four items: 'contoso0501' (App Service, West Europe), 'contoso0501 (myCDNProfile/contoso0501)' (Endpoint, Global), 'contoso0501' (App Service plan, West Europe), and 'myCDNProfile' (Front Door and CDN pr..., Global). The second item in the list is highlighted with a red box.

At the top of the Endpoint page, select Purge.

The screenshot shows the Azure portal's 'Endpoint' blade for 'contoso0501 (myCDNProfile/contoso0501)'. The 'Purge' button is highlighted with a red box. The blade includes sections for Overview, Activity log, Access control (IAM), Tags, Diagnose and solve problems, Settings (with Origin, Custom domains, Compression, Caching rules, and Geo-filtering options), and Essentials. The Essentials section displays details like Resource group (move myResourceGroup), Status (Running), Location (Global), Subscription (Contoso Subscription), Subscription ID (abcdef1-2345-6789-0abc-def012345678), Endpoint hostname (https://contoso0501.azureedge.net), Origin hostname (https://contoso0501.azurewebsites.net), Protocols (HTTP, HTTPS), and Optimization type (General web delivery). A 'Custom domains' table shows no entries.

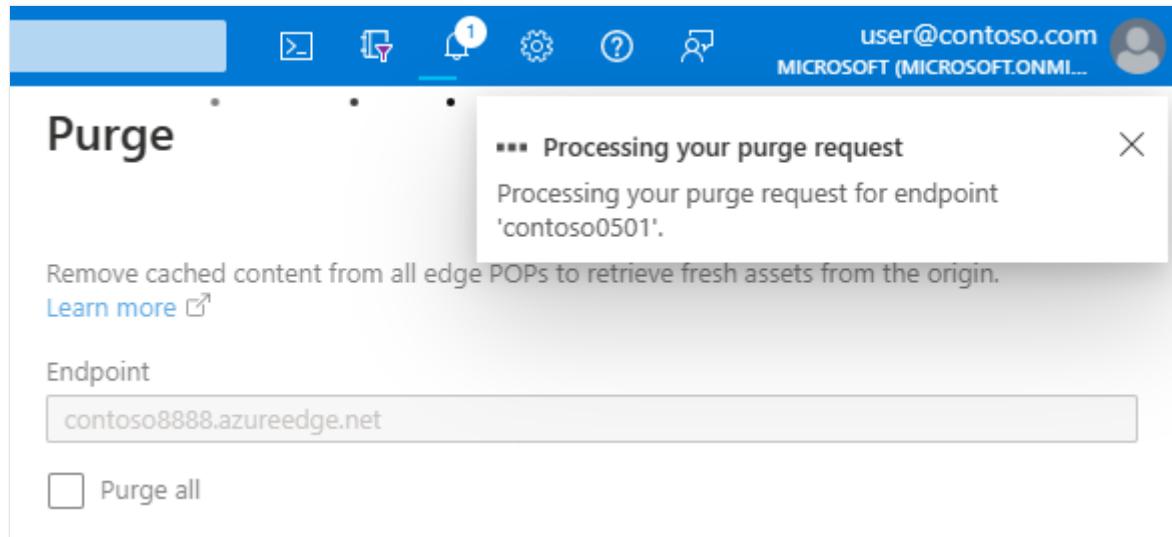
Enter the content paths you want to purge. You can pass a complete file path to purge an individual file, or a path segment to purge and refresh all content in a folder. Because you changed *index.html*, ensure that is in one of the paths.

At the bottom of the page, select **Purge**.

The screenshot shows the 'Purge' dialog box. It contains a message about removing cached content from edge POPs and a 'Learn more' link. The 'Endpoint' field is set to 'contoso0501.azureedge.net'. The 'Purge all' checkbox is checked. Below it, a table lists content paths: '/index.html', '/img/', '/css/', and '/js/'. The '/js/' row has a green checkmark next to its ellipsis button. At the bottom is a large blue 'Purge' button.

Verify that the content delivery network is updated

Wait until the purge request finishes processing, which is typically a couple of minutes. To see the current status, select the bell icon at the top of the page.



When you browse to the content delivery network endpoint URL for *index.html*, you see the V2 that you added to the title on the home page, which indicates that the content delivery network cache has been refreshed.

`http://<endpointname>.azureedge.net/index.html`



For more information, see [Purge an Azure Content Delivery Network endpoint](#).

Use query strings to version content

Azure Content Delivery Network offers the following caching behavior options:

- Ignore query strings
- Bypass caching for query strings
- Cache every unique URL

The first option is the default, which means there's only one cached version of an asset regardless of the query string in the URL.

In this section of the tutorial, you change the caching behavior to cache every unique URL.

Change the cache behavior

In the Azure portal CDN Endpoint page, select **Cache**.

Select **Cache every unique URL** from the **Query string caching behavior** dropdown list.

Select **Save**.

The screenshot shows the Azure portal interface for managing a CDN endpoint named 'contoso0501'. On the left, there's a sidebar with various navigation options like Overview, Activity log, Access control (IAM), Tags, Diagnose and solve problems, Settings, Origin, Custom domains, Compression, Caching rules (which is currently selected and highlighted with a red box), and Geo-filtering. At the top, there's a search bar, a 'Save' button (also highlighted with a red box), and a 'Discard' button. The main content area has a section titled 'About This Feature' which explains how CDN caches content and handles query strings. Below this, there's a dropdown menu labeled 'Query string caching behavior' with four options: 'Ignore query strings' (selected), 'Ignore query strings', 'Bypass caching for query strings', and 'Cache every unique URL' (highlighted with a red box).

Verify that unique URLs are cached separately

In a browser, navigate to the home page at the content delivery network endpoint, and include a query string:

```
http://<endpointname>.azureedge.net/index.html?q=1
```

Azure Content Delivery Network returns the current web app content, which includes V2 in the heading.

To ensure that this page is cached in the content delivery network, refresh the page.

Open *index.html*, change V2 to V3, then deploy the change.

```
Bash

git commit -am "version 3"
git push azure main
```

In a browser, go to the content delivery network endpoint URL with a new query string, such as `q=2`. Azure Content Delivery Network gets the current *index.html* file and displays V3. However, if you navigate to the content delivery network endpoint with the `q=1` query string, you see V2.

`http://<endpointname>.azureedge.net/index.html?q=2`



`http://<endpointname>.azureedge.net/index.html?q=1`



This output shows that each query string is treated differently:

- q=1 was used before, so cached contents are returned (V2).
- q=2 is new, so the latest web app contents are retrieved and returned (V3).

For more information, see [Control Azure Content Delivery Network caching behavior with query strings](#).

Clean up resources

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

```
Azure CLI
az group delete --name myResourceGroup
```

This command may take a minute to run.

Next steps

What you learned:

- ✓ Create a content delivery network endpoint.
- ✓ Refresh cached assets.
- ✓ Use query strings to control cached versions.

Learn how to optimize content delivery network performance in the following articles:

Tutorial: Optimize Azure Content Delivery Network for the type of content delivery

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

High-density hosting on Azure App Service using per-app scaling

Article • 02/06/2024

ⓘ Note

We recommend that you use the Azure Az PowerShell module to interact with Azure. To get started, see [Install Azure PowerShell](#). To learn how to migrate to the Az PowerShell module, see [Migrate Azure PowerShell from AzureRM to Az](#).

When using App Service, you can scale your apps by scaling the [App Service plan](#) they run on. When multiple apps are running in the same App Service plan, each scaled-out instance runs all the apps in the plan.

Per-app scaling can be enabled at the App Service plan level to allow for scaling an app independently from the App Service plan that hosts it. This way, an App Service plan can be scaled to 10 instances, but an app can be set to use only five.

ⓘ Note

Per-app scaling is available only for **Standard**, **Premium**, **Premium V2**, **Premium V3**, and **Isolated** pricing tiers.

Apps are allocated to available App Service plan using a best effort approach for an even distribution across instances. While an even distribution is not guaranteed, the platform will make sure that two instances of the same app will not be hosted on the same App Service plan instance.

The platform does not rely on metrics to decide on worker allocation. Applications are rebalanced only when instances are added or removed from the App Service plan.

Per app scaling using PowerShell

Create a plan with per-app scaling by passing in the `-PerSiteScaling $true` parameter to the `New-AzAppServicePlan` cmdlet.

PowerShell

```
New-AzAppServicePlan -ResourceGroupName $ResourceGroup -Name $AppServicePlan  
`  
    -Location $Location  
    -Tier Premium -WorkerSize Small  
    -NumberofWorkers 5 -PerSiteScaling $true
```

Enable per-app scaling with an existing App Service Plan by passing in the `-PerSiteScaling $true` parameter to the `Set-AzAppServicePlan` cmdlet.

PowerShell

```
# Enable per-app scaling for the App Service Plan using the "PerSiteScaling"  
# parameter.  
Set-AzAppServicePlan -ResourceGroupName $ResourceGroup  
`-Name $AppServicePlan -PerSiteScaling $true
```

At the app level, configure the number of instances the app can use in the App Service plan.

In the example below, the app is limited to two instances regardless of how many instances the underlying app service plan scales out to.

PowerShell

```
# Get the app we want to configure to use "PerSiteScaling"  
$newapp = Get-AzWebApp -ResourceGroupName $ResourceGroup -Name $webapp  
  
# Modify the NumberOfWorkers setting to the desired value.  
$newapp.SiteConfig.NumberOfWorkers = 2  
  
# Post updated app back to azure  
Set-AzWebApp $newapp
```

ⓘ Important

`$newapp.SiteConfig.NumberOfWorkers` is different from `$newapp.MaxNumberOfWorkers`. Per-app scaling uses `$newapp.SiteConfig.NumberOfWorkers` to determine the scale characteristics of the app.

Per-app scaling using Azure Resource Manager

The following Azure Resource Manager template creates:

- An App Service plan that's scaled out to 10 instances
- an app that's configured to scale to a max of five instances.

The App Service plan is setting the `PerSiteScaling` property to true `"perSiteScaling": true`. The app is setting the **number of workers** to use to 5 `"properties": { "numberOfWorkers": "5" }`.

JSON

```
{
  "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",
  "contentVersion": "1.0.0.0",
  "parameters": {
    "appServicePlanName": { "type": "string" },
    "appName": { "type": "string" }
  },
  "resources": [
    {
      "comments": "App Service Plan with per site perSiteScaling = true",
      "type": "Microsoft.Web/serverFarms",
      "sku": {
        "name": "P1",
        "tier": "Premium",
        "size": "P1",
        "family": "P",
        "capacity": 10
      },
      "name": "[parameters('appServicePlanName')]",
      "apiVersion": "2015-08-01",
      "location": "West US",
      "properties": {
        "name": "[parameters('appServicePlanName')]",
        "perSiteScaling": true
      }
    },
    {
      "type": "Microsoft.Web/sites",
      "name": "[parameters('appName')]",
      "apiVersion": "2015-08-01-preview",
      "location": "West US",
      "dependsOn": [ "[resourceId('Microsoft.Web/serverFarms',
parameters('appServicePlanName'))]" ],
      "properties": { "serverFarmId": "[resourceId('Microsoft.Web/serverFarms',
parameters('appServicePlanName'))]" }
    },
    "resources": [ {
      "comments": "",
      "type": "config",
      "name": "web",
      "apiVersion": "2015-08-01",
      "location": "West US",
      "properties": {
        "bindings": [
          {
            "name": "siteConfig",
            "type": "applicationSettings"
          }
        ]
      }
    } ]
  ]
}
```

```
        "dependsOn": [ "[resourceId('Microsoft.Web/Sites',  
parameters('appName'))]" ],  
        "properties": { "numberOfWorkers": "5" }  
    } ]  
}  
}
```

Recommended configuration for high-density hosting

Per app scaling is a feature that is enabled in both global Azure regions and [App Service Environments](#). However, the recommended strategy is to use App Service Environments to take advantage of their advanced features and the larger App Service plan capacity.

Follow these steps to configure high-density hosting for your apps:

1. Designate an App Service plan as the high-density plan and scale it out to the desired capacity.
2. Set the `PerSiteScaling` flag to true on the App Service plan.
3. New apps are created and assigned to that App Service plan with the `numberOfWorkers` property set to 1.
 - Using this configuration yields the highest density possible.
4. The number of workers can be configured independently per app to grant additional resources as needed. For example:
 - A high-use app can set `numberOfWorkers` to 3 to have more processing capacity for that app.
 - Low-use apps would set `numberOfWorkers` to 1.

Next steps

- [Azure App Service plans in-depth overview](#)
- [Introduction to App Service Environment](#)
- [Tutorial: Run a load test to identify performance bottlenecks in a web app](#)

Azure App Service Local Cache overview

Article • 06/29/2023

ⓘ Note

Local cache is not supported in function apps or containerized App Service apps, such as in [Windows Containers](#) or in [App Service on Linux](#). A version of local cache that is available for these app types is [App Cache](#).

Azure App Service content is stored on Azure Storage and is surfaced in a durable manner as a content share. This design is intended to work with a variety of apps and has the following attributes:

- The content is shared across multiple virtual machine (VM) instances of the app.
- The content is durable and can be modified by running apps.
- Log files and diagnostic data files are available under the same shared content folder.
- Publishing new content directly updates the content folder. You can immediately view the same content through the SCM website and the running app (typically some technologies such as ASP.NET do initiate an app restart on some file changes to get the latest content).

While many apps use one or all of these features, some apps just need a high-performance, read-only content store that they can run from with high availability. These apps can benefit from a VM instance of a specific local cache.

The Azure App Service Local Cache feature provides a web role view of your content. This content is a write-but-discard cache of your storage content that is created asynchronously on-site startup. When the cache is ready, the site is switched to run against the cached content. Apps that run on Local Cache have the following benefits:

- They are immune to latencies that occur when they access content on Azure Storage.
- They are not affected by connection issues to the storage, since the read-only copy is cached on the worker.
- They have fewer app restarts due to storage share changes.

ⓘ Note

If you are using Java (Java SE, Tomcat, or JBoss EAP), then by default the Java artifacts--.jar, .war, and .ear files--are copied locally to the worker. If your Java

application depends on read-only access to other files as well, set `JAVA_COPY_ALL` to `true` for those files to also be copied. If Local Cache is enabled, it takes precedence over this Java-specific enhancement.

How the local cache changes the behavior of App Service

- *D:\home* points to the local cache, which is created on the VM instance when the app starts up. *D:\local* continues to point to the temporary VM-specific storage.
- The local cache contains a one-time copy of the */site* and */siteextensions* folders of the shared content store, at *D:\home\site* and *D:\home\siteextensions*, respectively. The files are copied to the local cache when the app starts. The size of the two folders for each app is limited to 1 GB by default, but can be increased to 2 GB. Note that as the cache size increases, it will take longer to load the cache. If you've increased local cache limit to 2 GB and the copied files exceed the maximum size of 2 GB, App Service silently ignores local cache and reads from the remote file share. If there is no limit defined or the limit is set to anything lower than 2 GB and the copied files exceeds the limit, the deployment or swap may fail with an error.
- The local cache is read-write. However, any modification is discarded when the app moves virtual machines or gets restarted. Do not use the local cache for apps that store mission-critical data in the content store.
- *D:\home\LogFiles* and *D:\home\Data* contain log files and app data. The two subfolders are stored locally on the VM instance, and are copied to the shared content store periodically. Apps can persist log files and data by writing them to these folders. However, the copy to the shared content store is best-effort, so it is possible for log files and data to be lost due to a sudden crash of a VM instance.
- **Log streaming** is affected by the best-effort copy. You could observe up to a one-minute delay in the streamed logs.
- In the shared content store, there is a change in the folder structure of the *LogFiles* and *Data* folders for apps that use the local cache. There are now subfolders in them that follow the naming pattern of "unique identifier" + time stamp. Each of the subfolders corresponds to a VM instance where the app is running or has run.
- Other folders in *D:\home* remain in the local cache and are not copied to the shared content store.
- App deployment through any supported method publishes directly to the durable shared content store. To refresh the *D:\home\site* and *D:\home\siteextensions* folders in the local cache, the app needs to be restarted. To make the lifecycle seamless, see the information later in this article.

- The default content view of the SCM site continues to be that of the shared content store.

Enable Local Cache in App Service

ⓘ Note

Local Cache is not supported in the F1 or D1 tier.

You configure Local Cache by using a combination of reserved app settings. You can configure these app settings by using the following methods:

- [Azure portal](#)
- [Azure Resource Manager](#)

Configure Local Cache by using the Azure portal

You enable Local Cache on a per-web-app basis by using this app setting:

`WEBSITE_LOCAL_CACHE_OPTION = Always`

The screenshot shows the 'Web app settings' blade for an app named 'rmsrddemo'. On the left, a navigation menu lists 'Properties', 'Application settings' (which is selected and highlighted in blue), 'Scale (App Service Plan)', 'Troubleshoot', 'App Service Plan', 'Authentication / Authorization', and 'Custom domains and SSL'. On the right, under 'Application settings', there is a note: 'Auto swap destinations cannot be configured from production slot'. Below this are two sections: 'Auto Swap' (with 'Off' and 'On' buttons) and 'Auto Swap Slot' (a dropdown menu). Under 'Debugging', 'Remote debugging' is set to 'On'. A 'Remote Visual Studio version' dropdown shows options for 2012, 2013, and 2015. At the bottom, there is a table for 'App settings' with two entries: 'WEBSITE_NODE_DEFAULT_V...' and 'WEBSITE_LOCAL_CACHE_OP...'. The 'WEBSITE_LOCAL_CACHE_OP...' entry has its 'Value' field set to 'Always' and has a checked 'Slot setting' checkbox. There are also 'Key' and 'Value' input fields and another 'Slot setting' checkbox.

Configure Local Cache by using Azure Resource Manager

`jsonc`

```
...
{
    "apiVersion": "2015-08-01",
    "type": "config",
    "name": "appsettings",
    "dependsOn": [
        "[resourceId('Microsoft.Web/sites/', variables('siteName'))]"
    ],
    "properties": {
        "WEBSITE_LOCAL_CACHE_OPTION": "Always",
        "WEBSITE_LOCAL_CACHE_SIZEINMB": "1000"
    }
}
...
...
```

Change the size setting in Local Cache

By default, the local cache size is **1 GB**. This includes the /site and /siteextensions folders that are copied from the content store, as well as any locally created logs and data folders. To increase this limit, use the app setting `WEBSITE_LOCAL_CACHE_SIZEINMB`. You can increase the size up to **2 GB** (2000 MB) per app. Note that it will take longer to load local cache as the size increases.

Best practices for using App Service Local Cache

We recommend that you use Local Cache in conjunction with the [Staging Environments](#) feature.

- Add the *sticky* app setting `WEBSITE_LOCAL_CACHE_OPTION` with the value `Always` to your **Production** slot. If you're using `WEBSITE_LOCAL_CACHE_SIZEINMB`, also add it as a sticky setting to your Production slot.
- Create a **Staging** slot and publish to your Staging slot. You typically don't set the staging slot to use Local Cache to enable a seamless build-deploy-test lifecycle for staging if you get the benefits of Local Cache for the production slot.
- Test your site against your Staging slot.
- When you are ready, issue a [swap operation](#) between your Staging and Production slots.

- Sticky settings include name and sticky to a slot. So when the Staging slot gets swapped into Production, it inherits the Local Cache app settings. The newly swapped Production slot will run against the local cache after a few minutes and will be warmed up as part of slot warmup after swap. So when the slot swap is complete, your Production slot is running against the local cache.

Frequently asked questions (FAQ)

How can I tell if Local Cache applies to my app?

If your app needs a high-performance, reliable content store, does not use the content store to write critical data at runtime, and is less than 2 GB in total size, then the answer is "yes"! To get the total size of your /site and /siteextensions folders, you can use the site extension "Azure Web Apps Disk Usage."

How can I tell if my site has switched to using Local Cache?

If you're using the Local Cache feature with Staging Environments, the swap operation does not complete until Local Cache is warmed up. To check if your site is running against Local Cache, you can check the worker process environment variable `WEBSITE_LOCALCACHE_READY`. Use the instructions on the [worker process environment variable](#) page to access the worker process environment variable on multiple instances.

I just published new changes, but my app does not seem to have them. Why?

If your app uses Local Cache, then you need to restart your site to get the latest changes. Don't want to publish changes to a production site? See the slot options in the previous best practices section.

Note

The [run from package](#) deployment option is not compatible with local cache.

Where are my logs?

With Local Cache, your logs and data folders do look a little different. However, the structure of your subfolders remains the same, except that the subfolders are nestled under a subfolder with the format "unique VM identifier" + time stamp.

I have Local Cache enabled, but my app still gets restarted. Why is that? I thought Local Cache helped with frequent app restarts.

Local Cache does help prevent storage-related app restarts. However, your app could still undergo restarts during planned infrastructure upgrades of the VM. The overall app restarts that you experience with Local Cache enabled should be fewer.

Does Local Cache exclude any directories from being copied to the faster local drive?

As part of the step that copies the storage content, any folder that is named repository is excluded. This helps with scenarios where your site content may contain a source control repository that may not be needed in day to day operation of the app.

How to flush the local cache logs after a site management operation?

To flush the local cache logs, stop and restart the app. This action clears the old cache.

Why does App Service starts showing previously deployed files after a restart when Local Cache is enabled?

In case App Service starts showing previously deployed files on a restart, check for the presence of the App Setting - '[WEBSITE_DISABLE_SCM_SEPARATION=true](#)'. After adding this setting any deployments via KUDU start writing to the local VM instead of the persistent storage. Best practices mentioned above in this article should be leveraged, wherein the deployments should always be done to the staging slot which does not have Local Cache enabled.

More resources

[Environment variables and app settings reference](#)

Quickstart: Add app authentication to your web app running on Azure App Service

Article • 05/31/2024

ⓘ Note

Beginning June 1, 2024, all newly created App Service apps will have the option to create a unique default hostname with a naming convention of <app-name>-<random-hash>. <region>.azurewebsites.net. The names of existing apps will not change.

Example: myapp-ds27dh7271aah175.westus-01.azurewebsites.net

For more information, refer to [Unique Default Hostname for App Service Resource ↗](#).

Learn how to enable authentication for your web app running on Azure App Service and limit access to users in your organization.

In this tutorial, you learn how to:

- ✓ Configure authentication for the web app.
- ✓ Limit access to the web app to users in your organization by using Microsoft Entra as the identity provider.

Automatic authentication provided by App Service

App Service provides built-in authentication and authorization support, so you can sign in users with no code in your web app. Using the optional App Service authentication/authorization module simplifies authentication and authorization for your app. When you're ready for custom authentication and authorization, you build on this architecture.

App service authentication provides:

- Easily turn on and configure through the Azure portal and app settings.
- No SDKs, specific languages, or changes to application code are required.

- Several identity providers are supported:
 - Microsoft Entra
 - Microsoft Account
 - Facebook
 - Google
 - Twitter

When the authentication/authorization module is enabled, every incoming HTTP request passes through it before being handled by your app code. To learn more, see [Authentication and authorization in Azure App Service](#).

1. Prerequisites

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

2. Create and publish a web app on App Service

For this tutorial, you need a web app deployed to App Service. You can use an existing web app, or you can follow one of the quickstarts to create and publish a new web app to App Service:

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

Whether you use an existing web app or create a new one, take note of the following:

- **Web app name.**
- **Resource group** that the web app is deployed to.

You need these names throughout this tutorial.

3. Configure authentication and authorization

Now that you have a web app running on App Service, enable authentication and authorization. You use Microsoft Entra as the identity provider. For more information, see [Configure Microsoft Entra authentication for your App Service application](#).

1. In the [Azure portal](#) menu, select **Resource groups**, or search for and select **Resource groups** from any page.
2. In **Resource groups**, find and select your resource group. In **Overview**, select your app's management page.

The screenshot shows the Azure Resource Groups Overview page for the 'GetStartedWebApp' resource group. The left sidebar includes links for Overview, Activity log, Access control (IAM), Tags, Resource visualizer, Events, Deployments, Security, Policies, and Properties. The main area displays the 'Essentials' section with a 'Resources' tab selected. A table lists three resources:

	Name	Type	Location	...
<input type="checkbox"/>	ASP-GetStartedWebApp-a3f5	App Service plan	North Central US	...
<input type="checkbox"/>	GetStartedWebApp	App Service	North Central US	...
<input type="checkbox"/>	GetStartedWebApp	Application Insights	North Central US	...

3. On your app's left menu, select **Authentication**, and then select **Add identity provider**.
4. In the **Add an identity provider** page, select **Microsoft** as the **Identity provider** to sign in Microsoft and Microsoft Entra identities.
5. For **Tenant type**, select **Workforce configuration (current tenant)** for employees and business guests.
6. For **App registration > App registration type**, select **Create new app registration** to create a new app registration in Microsoft Entra.
7. Enter a display **Name** for your application. Users of your application might see the display name when they use the app, for example during sign-in.
8. For **App registration > Supported account types**, select **Current tenant-single tenant** so only users in your organization can sign in to the web app.
9. In the **Additional checks** section, select:
 - Allow requests only from this application itself for **Client application requirement**

- Allow requests from any identity for Identity requirement
- Allow requests only from the issuer tenant for Tenant requirement

10. In the **App Service authentication settings** section, set:

- Require authentication for Authentication
- HTTP 302 Found redirect: recommended for websites for Unauthenticated requests
- Token store box

11. At the bottom of the **Add an identity provider** page, select **Add** to enable authentication for your web app.

Add an identity provider

...

Basics Permissions

Identity provider *

Microsoft



Choose a tenant for your application and its users

A tenant contains applications and a directory for user accounts. Choose a tenant based on whether it's configured for workforce users (employees and business guests) or external users. [Learn more](#)

Workforce configuration (current tenant)
Manage employees and business guests

External configuration (Preview)
Manage external users

App registration

An app registration associates your identity provider with your app. Enter the app registration information here, or go to your provider to create a new one. [Learn more](#)

App registration type *

Create new app registration

- Pick an existing app registration in this directory
- Provide the details of an existing app registration

Name * ⓘ

GetStartedWebApp

Supported account types *

Current tenant - Single tenant

- Any Microsoft Entra directory - Multi-tenant
- Any Microsoft Entra directory & personal Microsoft accounts
- Personal Microsoft accounts only

[Help me choose...](#)

Additional checks

You can configure additional checks that will further control access, but your app may still need to make additional authorization decisions in code. [Learn more](#)

Client application requirement *

Allow requests only from this application itself

- Allow requests from specific client applications
- Allow requests from any application (Not recommended)

Identity requirement *

Allow requests from any identity

- Allow requests from specific identities

Tenant requirement *

Allow requests only from the issuer tenant

- Allow requests from specific tenants
- Use default restrictions based on issuer

App Service authentication settings

Requiring authentication ensures that requests to your app include information about the caller, but your app may still need to make additional authorization decisions to control access. If unauthenticated requests are allowed, any client can call the app and your code will need to handle both authentication and authorization. [Learn more](#)

Restrict access *

Require authentication

- Allow unauthenticated access

Unauthenticated requests *

HTTP 302 Found redirect: recommended for websites

- HTTP 401 Unauthorized: recommended for APIs
- HTTP 403 Forbidden
- HTTP 404 Not found

Redirect to

Microsoft



[Add](#)< PreviousNext: Permissions >

You now have an app that's secured by the App Service authentication and authorization.

ⓘ Note

To allow accounts from other tenants, change the 'Issuer URL' to '<https://login.microsoftonline.com/common/v2.0>' by editing your 'Identity Provider' from the 'Authentication' blade.

4. Verify limited access to the web app

When you enabled the App Service authentication/authorization module in the previous section, an app registration was created in your workforce or external tenant. The app registration has the display name you created in a previous step.

1. To check the settings, sign in to the [Microsoft Entra admin center](#) as at least an **Application Developer**. If you chose external configuration, use the **Settings** icon in the top menu to switch to the external tenant with your web app from the **Directories + subscriptions** menu. When you are in the correct tenant:
 2. Browse to **Identity > Applications > App registrations** and select **Applications > App registrations** from the menu.
 3. Select the app registration that was created.
 4. In the overview, verify that **Supported account types** is set to **My organization only**.
 5. To verify that access to your app is limited to users in your organization, go to your web app **Overview** and select the **Default domain** link. Or, start a browser in incognito or private mode and go to <https://<app-name>.azurewebsites.net> (see note at top).

Home > Resource groups > GetStartedWebApp >

GetStartedWebApp Web App

Search

Overview

Activity log

Access control (IAM)

Tags

Diagnose and solve problems

Microsoft Defender for Cloud

Events (preview)

Browse Stop Swap Restart Delete Refresh Download publish profile Reset publish profile ...

JSON View

Resource group : GetStartedWebApp

Status : Running

Location : North Central US

Subscription : Test Subscription

Subscription ID :

Tags (edit) : Click here to add tags

Default domain : [getstartedwebapp.azurewebsites.net](#)

App Service Plan : [ASP-GetStartedWebApp-a3f5 \(F1: 0\)](#)

Operating System : Windows

Health Check : Not Configured

6. You should be directed to a secured sign-in page, verifying that unauthenticated users aren't allowed access to the site.
7. Sign in as a user in your organization to gain access to the site. You can also start up a new browser and try to sign in by using a personal account to verify that users outside the organization don't have access.

5. Clean up resources

If you completed all the steps in this multipart tutorial, you created an App Service, App Service hosting plan, and a storage account in a resource group. You also created an app registration in Microsoft Entra ID. If you chose external configuration, you may have created a new external tenant. When no longer needed, delete these resources and app registration so that you don't continue to accrue charges.

In this tutorial, you learn how to:

- ✓ Delete the Azure resources created while following the tutorial.

Delete the resource group

In the [Azure portal](#), select **Resource groups** from the portal menu and select the resource group that contains your App Service and App Service plan.

Select **Delete resource group** to delete the resource group and all the resources.

The screenshot shows the Azure Resource Groups blade. On the left, there's a list of resources under the 'SecureWebApp' resource group, with 'SecureWebApp' itself highlighted by a red box. On the right, there's an 'Essentials' summary card and a detailed list of resources with their types and locations. The 'Delete resource group' button at the top right of the blade is also highlighted with a red box.

This command might take several minutes to run.

Delete the app registration

In the [Microsoft Entra admin center](#), select **Applications > App registrations**. Then select the application you created.

The screenshot shows the Microsoft Entra admin center's App registrations blade. On the left sidebar, 'App registrations' is selected and highlighted with a red box. The main area displays a list of applications, with 'GetStartedWebApp' selected and highlighted with a red box. The list includes columns for Display name, Application (client) ID, Created on, and Certificates & secrets.

In the app registration overview, select **Delete**.

The screenshot shows the 'GetStartedWebApp' app registration overview page. The 'Delete' button at the top right of the blade is highlighted with a red box. The page also includes a feedback survey banner and some basic details about the app.

Delete the external tenant

If you created a new external tenant, you can [delete it](#). In to the [Microsoft Entra admin center](#), browse to **Identity > Overview > Manage tenants**.

Select the tenant you want to delete, and then select **Delete**.

You might need to complete required actions before you can delete the tenant. For example, you might need to delete all user flows and app registrations in the tenant.

If you're ready to delete the tenant, select **Delete**.

Next steps

In this tutorial, you learned how to:

- ✓ Configure authentication for the web app.
- ✓ Limit access to the web app to users in your organization.

App Service accesses storage

Authentication and authorization in Azure App Service and Azure Functions

Article • 09/27/2024

ⓘ Note

Starting June 1, 2024, all newly created App Service apps will have the option to generate a unique default hostname using the naming convention `<app-name>-<random-hash>.<region>.azurewebsites.net`. Existing app names will remain unchanged.

Example: `myapp-ds27dh7271ah175.westus-01.azurewebsites.net`

For further details, refer to [Unique Default Hostname for App Service Resource ↗](#).

Azure App Service provides built-in authentication and authorization capabilities (sometimes referred to as "Easy Auth"), so you can sign in users and access data by writing minimal or no code in your web app, RESTful API, and mobile back end, and also [Azure Functions](#). This article describes how App Service helps simplify authentication and authorization for your app.

Why use the built-in authentication?

You're not required to use this feature for authentication and authorization. You can use the bundled security features in your web framework of choice, or you can write your own utilities. However, you will need to ensure that your solution stays up to date with the latest security, protocol, and browser updates.

Implementing a secure solution for authentication (signing-in users) and authorization (providing access to secure data) can take significant effort. You must make sure to follow industry best practices and standards, and keep your implementation up to date. The built-in authentication feature for App Service and Azure Functions can save you time and effort by providing out-of-the-box authentication with federated identity providers, allowing you to focus on the rest of your application.

- Azure App Service allows you to integrate a variety of auth capabilities into your web app or API without implementing them yourself.
- It's built directly into the platform and doesn't require any particular language, SDK, security expertise, or even any code to utilize.

- You can integrate with multiple login providers. For example, Microsoft Entra, Facebook, Google, X.

Your app might need to support more complex scenarios such as Visual Studio integration or incremental consent. There are several different authentication solutions available to support these scenarios. To learn more, read [Identity scenarios](#).

Identity providers

App Service uses [federated identity](#), in which a third-party identity provider manages the user identities and authentication flow for you. The following identity providers are available by default:

[] Expand table

Provider	Sign-in endpoint	How-To guidance
Microsoft Entra	<code>/auth/login/aad</code>	App Service Microsoft Entra platform login
Facebook	<code>/auth/login/facebook</code>	App Service Facebook login
Google	<code>/auth/login/google</code>	App Service Google login
X	<code>/auth/login/x</code>	App Service X login
GitHub	<code>/auth/login/github</code>	App Service GitHub login
Sign in with Apple	<code>/auth/login/apple</code>	App Service Sign in With Apple login (Preview)
Any OpenID Connect provider	<code>/auth/login/<providerName></code>	App Service OpenID Connect login

When you configure this feature with one of these providers, its sign-in endpoint is available for user authentication and for validation of authentication tokens from the provider. You can provide your users with any number of these sign-in options.

Considerations for using built-in authentication

Enabling this feature will cause all requests to your application to be automatically redirected to HTTPS, regardless of the App Service configuration setting to enforce HTTPS. You can disable this with the `requireHttps` setting in the V2 configuration. However, we do recommend sticking with HTTPS, and you should ensure no security tokens ever get transmitted over non-secure HTTP connections.

App Service can be used for authentication with or without restricting access to your site content and APIs. Access restrictions can be set in the **Authentication > Authentication settings** section of your web app. To restrict app access only to authenticated users, set **Action to take when request is not authenticated** to log in with one of the configured identity providers. To authenticate but not restrict access, set **Action to take when request is not authenticated** to "Allow anonymous requests (no action)."

 **Important**

You should give each app registration its own permission and consent. Avoid permission sharing between environments by using separate app registrations for separate deployment slots. When testing new code, this practice can help prevent issues from affecting the production app.

How it works

[Feature architecture](#)

[Authentication flow](#)

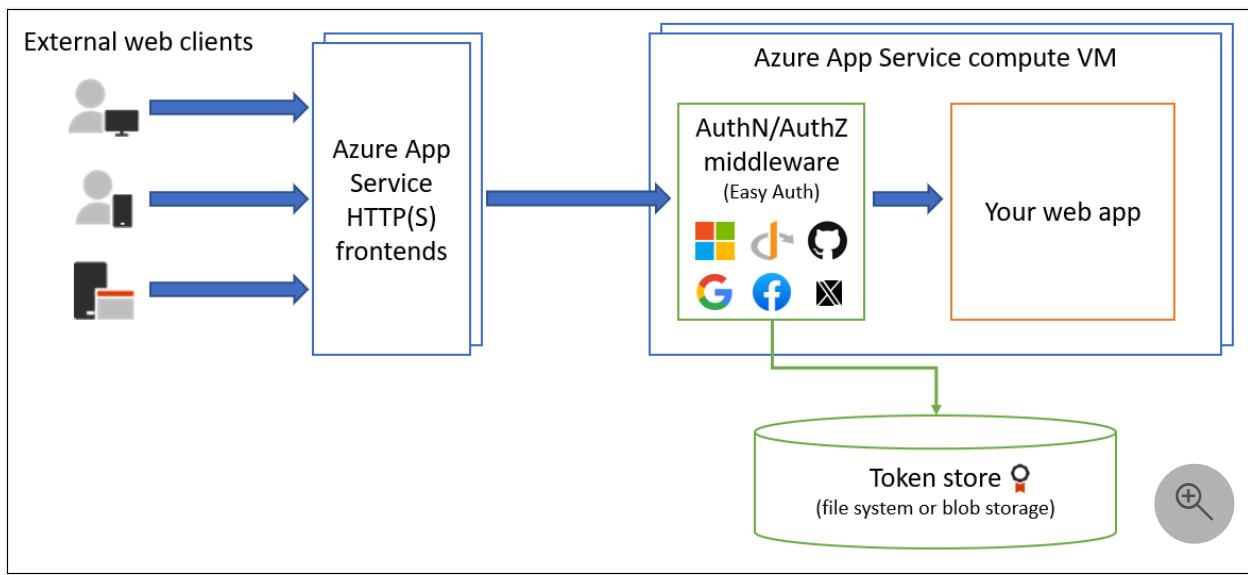
[Authorization behavior](#)

[Token store](#)

[Logging and tracing](#)

Feature architecture

The authentication and authorization middleware component is a feature of the platform that runs on the same VM as your application. When it's enabled, every incoming HTTP request passes through it before being handled by your application.



The platform middleware handles several things for your app:

- Authenticates users and clients with the specified identity provider(s)
- Validates, stores, and refreshes OAuth tokens issued by the configured identity provider(s)
- Manages the authenticated session
- Injects identity information into HTTP request headers

The module runs separately from your application code and can be configured using Azure Resource Manager settings or using [a configuration file](#). No SDKs, specific programming languages, or changes to your application code are required.

Feature architecture on Windows (non-container deployment)

The authentication and authorization module runs as a native [IIS module](#) in the same sandbox as your application. When it's enabled, every incoming HTTP request passes through it before being handled by your application.

Feature architecture on Linux and containers

The authentication and authorization module runs in a separate container, isolated from your application code. Using what's known as the [Ambassador pattern](#), it interacts with the incoming traffic to perform similar functionality as on Windows. Because it does not run in-process, no direct integration with specific language frameworks is possible; however, the relevant information that your app needs is passed through using request headers as explained below.

Authentication flow

The authentication flow is the same for all providers, but differs depending on whether you want to sign in with the provider's SDK:

- Without provider SDK: The application delegates federated sign-in to App Service. This is typically the case with browser apps, which can present the provider's login page to the user. The server code manages the sign-in process, so it is also called *server-directed flow* or *server flow*. This case applies to browser apps and mobile apps that use an embedded browser for authentication.
- With provider SDK: The application signs users in to the provider manually and then submits the authentication token to App Service for validation. This is typically the case with browser-less apps, which can't present the provider's sign-in page to the user. The application code manages the sign-in process, so it is also called *client-directed flow* or *client flow*. This case applies to REST APIs, [Azure Functions](#), and JavaScript browser clients, as well as browser apps that need more flexibility in the sign-in process. It also applies to native mobile apps that sign users in using the provider's SDK.

Calls from a trusted browser app in App Service to another REST API in App Service or [Azure Functions](#) can be authenticated using the server-directed flow. For more information, see [Customize sign-ins and sign-outs](#).

The table below shows the steps of the authentication flow.

[+] Expand table

Step	Without provider SDK	With provider SDK
1. Sign user in	Redirects client to <code>/ .auth/login/<provider></code> .	Client code signs user in directly with provider's SDK and receives an authentication token. For information, see the provider's documentation.
2. Post-authentication	Provider redirects client to <code>/ .auth/login/<provider>/callback</code> .	Client code posts token from provider to <code>/ .auth/login/<provider></code> for validation.
3. Establish authenticated session	App Service adds authenticated cookie to response.	App Service returns its own authentication token to client code.
4. Serve authenticated content	Client includes authentication cookie in subsequent requests (automatically handled by browser).	Client code presents authentication token in <code>X-ZUMO-AUTH</code> header.

For client browsers, App Service can automatically direct all unauthenticated users to `/auth/login/<provider>`. You can also present users with one or more `/auth/login/<provider>` links to sign in to your app using their provider of choice.

Authorization behavior

ⓘ Important

By default, this feature only provides authentication, not authorization. Your application may still need to make authorization decisions, in addition to any checks you configure here.

In the [Azure portal](#), you can configure App Service with a number of behaviors when incoming request is not authenticated. The following headings describe the options.

Restrict access

- **Allow unauthenticated requests** This option defers authorization of unauthenticated traffic to your application code. For authenticated requests, App Service also passes along authentication information in the HTTP headers.

This option provides more flexibility in handling anonymous requests. For example, it lets you [present multiple sign-in providers](#) to your users. However, you must write code.

- **Require authentication** This option will reject any unauthenticated traffic to your application. Specific action to take is specified in the **Unauthenticated requests** section.

With this option, you don't need to write any authentication code in your app. Finer authorization, such as role-specific authorization, can be handled by inspecting the user's claims (see [Access user claims](#)).

✖ Caution

Restricting access in this way applies to all calls to your app, which may not be desirable for apps wanting a publicly available home page, as in many single-page applications.

ⓘ Note

When using the Microsoft identity provider for users in your organization, the default behavior is that any user in your Microsoft Entra tenant can request a token for your application. You can [configure the application in Microsoft Entra](#) if you want to restrict access to your app to a defined set of users. App Service also offers some [basic built-in authorization checks](#) which can help with some validations. To learn more about authorization in Microsoft Entra, see [Microsoft Entra authorization basics](#).

Unauthenticated requests

- **HTTP 302 Found redirect: recommended for websites** Redirects action to one of the configured identity providers. In these cases, a browser client is redirected to `/auth/login/<provider>` for the provider you choose.
- **HTTP 401 Unauthorized: recommended for APIs** If the anonymous request comes from a native mobile app, the returned response is an `HTTP 401 Unauthorized`. You can also configure the rejection to be an `HTTP 401 Unauthorized` for all requests.
- **HTTP 403 Forbidden** Configures the rejection to be an `HTTP 403 Forbidden` for all requests.
- **HTTP 404 Not found** Configures the rejection to be an `HTTP 404 Not found` for all requests.

Token store

App Service provides a built-in token store, which is a repository of tokens that are associated with the users of your web apps, APIs, or native mobile apps. When you enable authentication with any provider, this token store is immediately available to your app. If your application code needs to access data from these providers on the user's behalf, such as:

- post to the authenticated user's Facebook timeline
- read the user's corporate data using the Microsoft Graph API

You typically must write code to collect, store, and refresh these tokens in your application. With the token store, you just [retrieve the tokens](#) when you need them and [tell App Service to refresh them](#) when they become invalid.

The ID tokens, access tokens, and refresh tokens are cached for the authenticated session, and they're accessible only by the associated user.

If you don't need to work with tokens in your app, you can disable the token store in your app's [Authentication / Authorization](#) page.

Logging and tracing

If you [enable application logging](#), you will see authentication and authorization traces directly in your log files. If you see an authentication error that you didn't expect, you can conveniently find all the details by looking in your existing application logs. If you enable [failed request tracing](#), you can see exactly what role the authentication and authorization module may have played in a failed request. In the trace logs, look for references to a module named `EasyAuthModule_32/64`.

Considerations when using Azure Front Door

When using Azure App Service with Easy Auth behind Azure Front Door or other reverse proxies, a few additional things have to be taken into consideration.

1. Disable Caching for the authentication workflow

See [Disable cache for auth workflow](#) to learn more on how to configure rules in Azure Front Door to disable caching for authentication and authorization-related pages.

2. Use the Front Door endpoint for redirects

App Service is usually not accessible directly when exposed via Azure Front Door. This can be prevented, for example, by exposing App Service via Private Link in Azure Front Door Premium. To prevent the authentication workflow to redirect traffic back to App Service directly, it is important to configure the application to redirect back to `https://<front-door-endpoint>/.auth/login/<provider>/callback`.

3. Ensure that App Service is using the right redirect URI

In some configurations, the App Service is using the App Service FQDN as the redirect URI instead of the Front Door FQDN. This will lead to an issue when the client is being redirected to App Service instead of Front Door. To change that, the `forwardProxy` setting needs to be set to `Standard` to make App Service respect the `X-Forwarded-Host` header set by Azure Front Door.

Other reverse proxies like Azure Application Gateway or 3rd-party products might use different headers and need a different forwardProxy setting.

This configuration cannot be done via the Azure portal today and needs to be done via `az rest`:

[Export settings](#)

```
az rest --uri /subscriptions/REPLACE-ME-SUBSCRIPTIONID/resourceGroups/REPLACE-ME-RESOURCEGROUP/providers/Microsoft.Web/sites/REPLACE-ME-APPNAME/config/authsettingsV2?api-version=2020-09-01 --method get > auth.json
```

Update settings

Search for

JSON

```
"httpSettings": {  
    "forwardProxy": {  
        "convention": "Standard"  
    }  
}
```

and ensure that `convention` is set to `Standard` to respect the `X-Forwarded-Host` header used by Azure Front Door.

Import settings

```
az rest --uri /subscriptions/REPLACE-ME-SUBSCRIPTIONID/resourceGroups/REPLACE-ME-RESOURCEGROUP/providers/Microsoft.Web/sites/REPLACE-ME-APPNAME/config/authsettingsV2?api-version=2020-09-01 --method put --body @auth.json
```

More resources

- [How-To: Configure your App Service or Azure Functions app to use Microsoft Entra login](#)
- [Customize sign-ins and sign-outs](#)
- [Work with OAuth tokens and sessions](#)
- [Access user and application claims](#)
- [File-based configuration](#)

Samples:

- [Tutorial: Add authentication to your web app running on Azure App Service](#)
- [Tutorial: Authenticate and authorize users end-to-end in Azure App Service \(Windows or Linux\)](#)
- [.NET Core integration of Azure AppService EasyAuth \(3rd party\) ↗](#)
- [Getting Azure App Service authentication working with .NET Core \(3rd party\) ↗](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

Authentication scenarios and recommendations

Article • 07/28/2024

If you have a web app or an API running in Azure App Service, you can restrict access to it based on the identity of the users or applications that request it. App Service offers several authentication solutions to help you achieve this goal. In this article, you will learn about the different authentication options, their benefits and drawbacks, and which authentication solution to use for specific scenarios.

Authentication solutions

- **Azure App Service built-in authentication** - Allows you to sign users in and access data by writing minimal or no code in your web app, RESTful API, or mobile back end. It's built directly into the platform and doesn't require any particular language, library, security expertise, or even any code to use.
- **Microsoft Authentication Library (MSAL)** - Enables developers to acquire security tokens from the Microsoft identity platform to authenticate users and access secured web APIs. Available for multiple supported platforms and frameworks, these are general purpose libraries that can be used in various hosted environments. Developers can also integrate with multiple sign-in providers, like Microsoft Entra, Facebook, Google, X.
- **Microsoft.Identity.Web** - A higher-level library wrapping MSAL.NET, it provides a set of ASP.NET Core abstractions that simplify adding authentication support to web apps and web APIs integrating with the Microsoft identity platform. It provides a single-surface API convenience layer that ties together ASP.NET Core, its authentication middleware, and MSAL.NET. This library can be used in apps in various hosted environments. You can integrate with multiple sign-in providers, like Microsoft Entra, Facebook, Google, X.

Scenario recommendations

The following table lists each authentication solution and some important factors for when you would use it.

[] [Expand table](#)

Authentication method	When to use
Built-in App Service authentication	<ul style="list-style-type: none"> * You want less code to own and manage. * Your app's language and SDKs don't provide user sign-in or authorization. * You don't have the ability to modify your app code (for example, when migrating legacy apps). * You need to handle authentication through configuration and not code. * You need to sign in external or social users.
Microsoft Authentication Library (MSAL)	<ul style="list-style-type: none"> * You need a code solution in one of several different languages * You need to add custom authorization logic. * You need to support incremental consent. * You need information about the signed-in user in your code. * You need to sign in external or social users. * Your app needs to handle the access token expiring without making the user sign in again.
Microsoft.Identity.Web	<ul style="list-style-type: none"> * You have an ASP.NET Core app. * You need single sign-on support in your IDE during local development. * You need to add custom authorization logic. * You need to support incremental consent. * You need conditional access in your web app. * You need information about the signed-in user in your code. * You need to sign in external or social users. * Your app needs to handle the access token expiring without making the user sign in again.

The following table lists authentication scenarios and the authentication solution(s) you would use.

[\[+\] Expand table](#)

Scenario	App Service	Microsoft Authentication	Microsoft.Identity.Web
	built-in auth	Library	
Need a fast and simple way to limit access to users in your organization?			
Unable to modify the application code (app migration scenario)?			

Scenario	App Service built-in auth	Microsoft Authentication Library	Microsoft.Identity.Web
Your app's language and libraries support user sign-in/authorization?	✗	✓	✓
Even if you can use a code solution, would you rather <i>not</i> use libraries? Don't want the maintenance burden?	✓	✗	✗
Does your web app need to provide incremental consent?	✗	✓	✓
Do you need conditional access in your web app?	✗	✗	✓
Your app need to handle the access token expiring without making the user sign in again (use a refresh token)?	✓	✓	✓
Need custom authorization logic or info about the signed-in user?	✗	✓	✓
Need to sign in users from external or social identity providers?	✓	✓	✓
You have an ASP.NET Core app?	✓	✗	✓
You have a single page app or static web app?	✓	✓	✓
Want Visual Studio integration?	✗	✗	✓
Need single sign-on support in your IDE during local development?	✗	✗	✓

Next steps

To get started with built-in App Service authentication, read:

- [Enable App Service built-in authentication](#)

To get started with Microsoft Authentication Library (MSAL), read:

- Add sign-in with Microsoft to a web app
- Only allow authenticated user to access a web API
- Sign in users to a single-page application (SPA)

To get started with [Microsoft.Identity.Web](#), read:

- Sign in users to a web app
- Protect a web API
- Sign in users to a Blazor Server app

Learn more about [App Service built-in authentication and authorization](#)

Feedback

Was this page helpful?



[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

Identity and access management (IAM) fundamental concepts

Article • 05/31/2024

This article provides fundamental concepts and terminology to help you understand identity and access management (IAM).

What is identity and access management (IAM)?

Identity and access management ensures that the right people, machines, and software components get access to the right resources at the right time. First, the person, machine, or software component proves they're who or what they claim to be. Then, the person, machine, or software component is allowed or denied access to or use of certain resources.

Here are some fundamental concepts to help you understand identity and access management:

Identity

A digital identity is a collection of unique identifiers or attributes that represent a human, software component, machine, asset, or resource in a computer system. An identifier can be:

- An email address
- Sign-in credentials (username/password)
- Bank account number
- Government issued ID
- MAC address or IP address

Identities are used to authenticate and authorize access to resources, communicate with other humans, conduct transactions, and other purposes.

At a high level, there are three types of identities:

- **Human identities** represent people such as employees (internal workers and frontline workers) and external users (customers, consultants, vendors, and partners).

- **Workload identities** represent software workloads such as an application, service, script, or container.
- **Device identities** represent devices such as desktop computers, mobile phones, IoT sensors, and IoT managed devices. Device identities are distinct from human identities.

Authentication

Authentication is the process of challenging a person, software component, or hardware device for credentials in order to verify their identity, or prove they're who or what they claim to be. Authentication typically requires the use of credentials (like username and password, fingerprints, certificates, or one-time passcodes). Authentication is sometimes shortened to *AuthN*.

Multifactor authentication (MFA) is a security measure that requires users to provide more than one piece of evidence to verify their identities, such as:

- Something they know, for example a password.
- Something they have, like a badge or **security token**.
- Something they are, like a biometric (fingerprint or face).

Single sign-on (SSO) allows users to authenticate their identity once and then later silently authenticate when accessing various resources that rely on the same identity. Once authenticated, the IAM system acts as the source of identity truth for the other resources available to the user. It removes the need for signing on to multiple, separate target systems.

Authorization

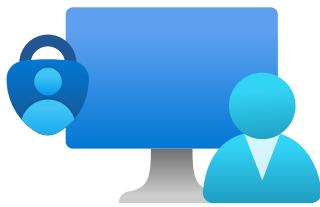
Authorization validates that the user, machine, or software component has been granted access to certain resources. Authorization is sometimes shortened to *AuthZ*.

Authentication vs. authorization

The terms authentication and authorization are sometimes used interchangeably, because they often seem like a single experience to users. They're actually two separate processes:

- Authentication proves the identity of a user, machine, or software component.
- Authorization grants or denies the user, machine, or software component access to certain resources.

Authentication



Confirms users are who they say they are

Authorization



Validates users have permission to complete the attempted action

Here's a quick overview of authentication and authorization:

[] [Expand table](#)

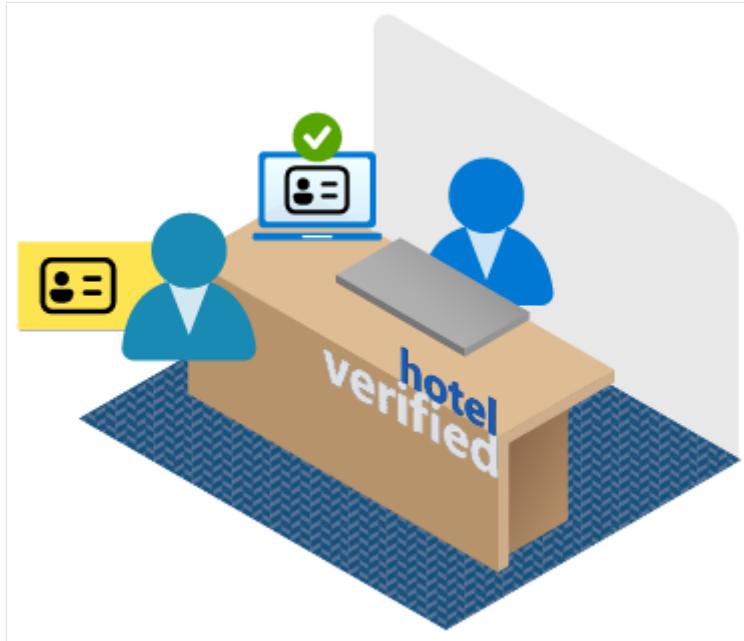
Authentication	Authorization
Can be thought of as a gatekeeper, allowing access only to those entities who provide valid credentials.	Can be thought of as a guard, ensuring that only those entities with the proper clearance can enter certain areas.
Verifies whether a user, machine, or software is who or what they claim to be.	Determines if the user, machine, or software is allowed to access a particular resource.
Challenges the user, machine, or software for verifiable credentials (for example, passwords, biometric identifiers, or certificates).	Determines what level of access a user, machine, or software has.
Done before authorization.	Done after successful authentication.
Information is transferred in an ID token.	Information is transferred in an access token.
Often uses the OpenID Connect (OIDC) (which is built on the OAuth 2.0 protocol) or SAML protocols.	Often uses the OAuth 2.0 protocol.

For more detailed information, read [Authentication vs. authorization](#).

Example

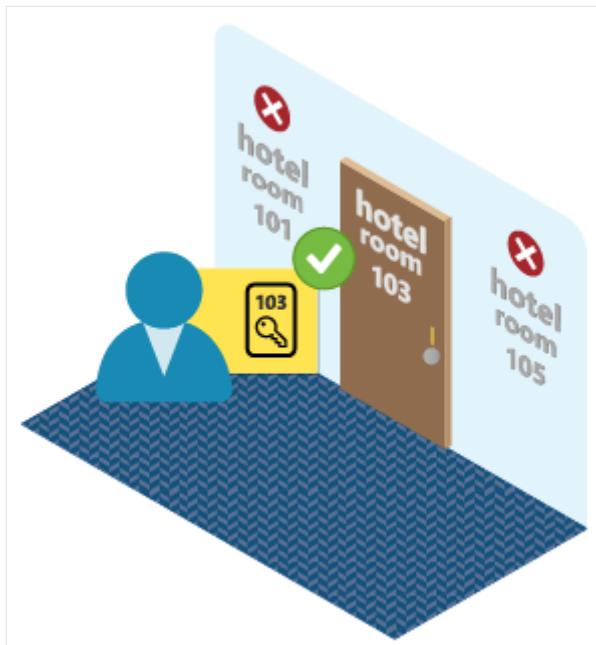
Suppose you want to spend the night in a hotel. You can think of authentication and authorization as the security system for the hotel building. Users are people who want to stay at the hotel, resources are the rooms or areas that people want to use. Hotel staff is another type of user.

If you're staying at the hotel, you first go to reception to start the "authentication process". You show an identification card and credit card and the receptionist matches your ID against the online reservation. After the receptionist has verified who you are, the receptionist grants you permission to access the room you've been assigned. You're given a keycard and can go now to your room.



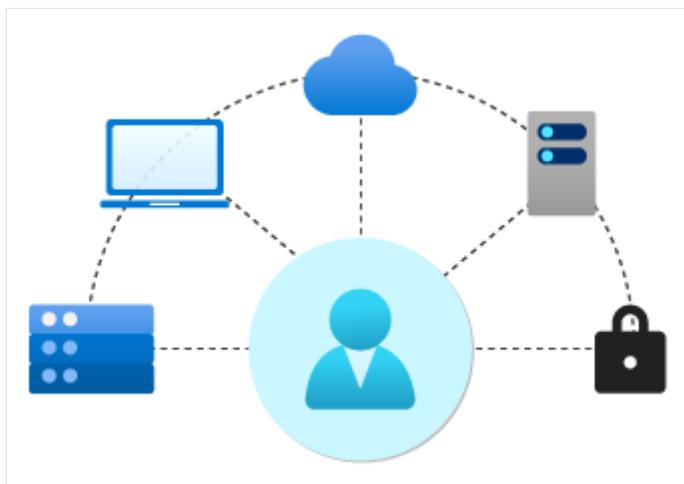
The doors to the hotel rooms and other areas have keycard sensors. Swiping the keycard in front of a sensor is the "authorization process". The keycard only lets you open the doors to rooms you're permitted to access, such as your hotel room and the hotel exercise room. If you swipe your keycard to enter any other hotel guest room, your access is denied.

Individual [permissions](#), such as accessing the exercise room and a specific guest room, are collected into [roles](#) which can be granted to individual users. When you're staying at the hotel, you're granted the Hotel Patron role. Hotel room service staff would be granted the Hotel Room Service role. This role permits access to all hotel guest rooms (but only between 11am and 4pm), the laundry room, and the supply closets on each floor.



Identity provider

An identity provider creates, maintains, and manages identity information while offering authentication, authorization, and auditing services.



With modern authentication, all services, including all authentication services, are supplied by a central identity provider. Information that's used to authenticate the user with the server is stored and managed centrally by the identity provider.

With a central identity provider, organizations can establish authentication and authorization policies, monitor user behavior, identify suspicious activities, and reduce malicious attacks.

[Microsoft Entra](#) is an example of a cloud-based identity provider. Other examples include X, Google, Amazon, LinkedIn, and GitHub.

Next steps

- Read [Introduction to identity and access management](#) to learn more.
 - Learn about [Single sign-on \(SSO\)](#).
 - Learn about [multifactor authentication \(MFA\)](#).
-

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#)

Configure your App Service or Azure Functions app to use Microsoft Entra sign-in

Article • 05/21/2024

ⓘ Note

Beginning June 1, 2024, all newly created App Service apps will have the option to create a unique default hostname with a naming convention of <app-name>-<random-hash>. <region>.azurewebsites.net. The names of existing apps will not change.

Example: myapp-ds27dh7271aah175.westus-01.azurewebsites.net

For more information, refer to [Unique Default Hostname for App Service Resource](#).

Select another authentication provider to jump to it.

This article shows you how to configure authentication for Azure App Service or Azure Functions so that your app signs in users with the [Microsoft identity platform](#) (Microsoft Entra) as the authentication provider.

Choose a tenant for your application and its users

Before your application can sign in users, you first need to register it in a workforce or external tenant. If you're making your app available to employee or business guests, register your app in a workforce tenant. If your app is for consumers and business customers, register it in an external tenant.

1. Sign in to the [Azure portal](#) and navigate to your app.
2. On your app's left menu, select **Authentication**, and then select **Add identity provider**.
3. In the **Add an identity provider** page, select **Microsoft** as the **Identity provider** to sign in Microsoft and Microsoft Entra identities.

4. For **Tenant type**, select **Workforce configuration (current tenant)** for employees and business guests or select **External configuration** for consumers and business customers.

Choose the app registration

The App Service Authentication feature can automatically create an app registration for you or you can use a registration that you or a directory admin created separately.

Create a new app registration automatically, unless you need to create an app registration separately. You can customize the app registration in the [Microsoft Entra admin center](#) later if you want.

The following situations are the most common cases to use an existing app registration:

- Your account doesn't have permissions to create app registrations in your Microsoft Entra tenant.
- You want to use an app registration from a different Microsoft Entra tenant than the one your app is in.
- The option to create a new registration isn't available for government clouds.

Workforce configuration

[Create and use a new app registration](#) or [use an existing registration created separately](#).

Option 1: Create and use a new app registration

Use this option unless you need to create an app registration separately. You can customize the app registration in Microsoft Entra once it's created.

Note

The option to create a new registration automatically isn't available for government clouds. Instead, [define a registration separately](#).

Enter the **Name** for the new app registration.

Select the **Supported account type**:

- **Current tenant - Single tenant.** Accounts in this organizational directory only. All user and guest accounts in your directory can use your application or API.

Use this option if your target audience is internal to your organization.

- **Any Microsoft Entra directory - Multitenant.** Accounts in any organizational directory. All users with a work or school account from Microsoft can use your application or API. This includes schools and businesses that use Office 365. Use this option if your target audience is business or educational customers and to enable multitenancy.
- **Any Microsoft Entra directory & personal Microsoft accounts.** Accounts in any organizational directory and personal Microsoft accounts (for example, Skype, Xbox). All users with a work or school, or personal Microsoft account can use your application or API. It includes schools and businesses that use Office 365 as well as personal accounts that are used to sign in to services like Xbox and Skype. Use this option to target the widest set of Microsoft identities and to enable multitenancy.
- **Personal Microsoft accounts only.** Personal accounts that are used to sign in to services like Xbox and Skype. Use this option to target the widest set of Microsoft identities.

You can change the name of the registration or the supported account types later if you want.

A client secret is created as a slot-sticky [application setting](#) named `MICROSOFT_PROVIDER_AUTHENTICATION_SECRET`. You can update that setting later to use [Key Vault references](#) if you wish to manage the secret in Azure Key Vault.

Option 2: Use an existing registration created separately

Either:

- Select **Pick an existing app registration in this directory** and select an app registration from the drop-down.
- Select **Provide the details of an existing app registration** and provide:
 - Application (client) ID.
 - Client secret (recommended). A secret value that the application uses to prove its identity when requesting a token. This value is saved in your app's configuration as a slot-sticky application setting named `MICROSOFT_PROVIDER_AUTHENTICATION_SECRET`. If the client secret isn't set, sign-in operations from the service use the OAuth 2.0 implicit grant flow, which isn't* recommended.
 - Issuer URL, which takes the form `<authentication-endpoint>/<tenant-id>/v2.0`. Replace `<authentication-endpoint>` with the authentication

endpoint value specific to the cloud environment. For example, a workforce tenant in global Azure would use "https://login.microsoftonline.com" as its authentication endpoint.

If you need to manually create an app registration in a workforce tenant, follow the [register an application](#) quickstart. As you go through the registration process, be sure to note the application (client) ID and client secret values.

During the registration process, in the **Redirect URIs** section, select **Web** for platform and type <app-url>/.auth/login/aad/callback. For example,

`https://contoso.azurewebsites.net/.auth/login/aad/callback`.

After creation, modify the app registration:

1. From the left navigation, select **Expose an API** > **Add** > **Save**. This value uniquely identifies the application when it's used as a resource, allowing tokens to be requested that grant access. It's used as a prefix for scopes you create.

For a single-tenant app, you can use the default value, which is in the form `api://<application-client-id>`. You can also specify a more readable URI like `https://contoso.com/api` based on one of the verified domains for your tenant. For a multitenant app, you must provide a custom URI. To learn more about accepted formats for App ID URIs, see the [app registrations best practices reference](#).

2. Select **Add a scope**.
 - a. In **Scope name**, enter *user_impersonation*.
 - b. In **Who can consent**, select **Admins and users** if you want to allow users to consent to this scope.
 - c. In the text boxes, enter the consent scope name and description you want users to see on the consent page. For example, enter *Access <application-name>*.
 - d. Select **Add scope**.
3. (Recommended) To create a client secret:
 - a. From the left navigation, select **Certificates & secrets** > **Client secrets** > **New client secret**.
 - b. Enter a description and expiration and select **Add**.
 - c. In the **Value** field, copy the client secret value. It won't be shown again once you navigate away from this page.

4. (Optional) To add multiple Reply URLs, select Authentication.

Configure additional checks

Configure Additional checks, which determines which requests are allowed to access your application. You can customize this behavior now or adjust these settings later from the main **Authentication** screen by choosing **Edit** next to **Authentication settings**.

For **Client application requirement**, choose whether to:

- Allow requests only from this application itself
- Allow requests from specific client applications
- Allow requests from any application (Not recommended)

For **Identity requirement**, choose whether to:

- Allow requests from any identity
- Allow requests from specific identities

For **Tenant requirement**, choose whether to:

- Allow requests only from the issuer tenant
- Allow requests from specific tenants
- Use default restrictions based on issuer

Your app may still need to make additional authorization decisions in code. For more information, see [Use a built-in authorization policy](#).

Configure authentication settings

These options determine how your application responds to unauthenticated requests, and the default selections will redirect all requests to sign in with this new provider. You can change customize this behavior now or adjust these settings later from the main **Authentication** screen by choosing **Edit** next to **Authentication settings**. To learn more about these options, see [Authentication flow](#).

For **Restrict access**, decide whether to:

- Require authentication
- Allow unauthenticated access

For **Unauthenticated requests**

- HTTP 302 Found redirect: recommended for websites
- HTTP 401 Unauthorized: recommended for APIs
- HTTP 403 Forbidden
- HTTP 404 Not found

Select **Token store** (recommended). The token store collects, stores, and refreshes tokens for your application. You can disable this later if your app doesn't need tokens or you need to optimize performance.

Add the identity provider

If you selected workforce configuration, you can select **Next: Permissions** and add any Microsoft Graph permissions needed by the application. These will be added to the app registration, but you can also change them later. If you selected external configuration, you can add Microsoft Graph permissions later.

Select **Add**.

You're now ready to use the Microsoft identity platform for authentication in your app. The provider will be listed on the **Authentication** screen. From there, you can edit or delete this provider configuration.

For an example of configuring Microsoft Entra sign-in for a web app that accesses Azure Storage and Microsoft Graph, see [this tutorial](#).

Authorize requests

By default, App Service Authentication only handles authentication, determining if the caller is who they say they are. Authorization, determining if that caller should have access to some resource, is an extra step beyond authentication. You can learn more about these concepts from [Microsoft identity platform authorization basics](#).

Your app can [make authorization decisions in code](#). App Service Authentication does provide some [built-in checks](#), which can help, but they may not alone be sufficient to cover the authorization needs of your app.

Tip

Multi-tenant applications should validate the issuer and tenant ID of the request as part of this process to make sure the values are allowed. When App Service Authentication is configured for a multi-tenant scenario, it doesn't validate which tenant the request comes from. An app may need to be limited to specific tenants,

based on if the organization has signed up for the service, for example. See the [Microsoft identity platform multi-tenant guidance](#).

Perform validations from application code

When you perform authorization checks in your app code, you can use the [claims information that App Service Authentication makes available](#). The injected `x-ms-client-principal` header contains a Base64-encoded JSON object with the claims asserted about the caller. By default, these claims go through a claims mapping, so the claim names may not always match what you would see in the token. For example, the `tid` claim is mapped to `http://schemas.microsoft.com/identity/claims/tenantid` instead.

You can also work directly with the underlying access token from the injected `x-ms-token-aad-access-token` header.

Use a built-in authorization policy

The created app registration authenticates incoming requests for your Microsoft Entra tenant. By default, it also lets anyone within the tenant to access the application, which is fine for many applications. However, some applications need to restrict access further by making authorization decisions. Your application code is often the best place to handle custom authorization logic. However, for common scenarios, the Microsoft identity platform provides built-in checks that you can use to limit access.

This section shows how to enable built-in checks using the [App Service authentication V2 API](#). Currently, the only way to configure these built-in checks is via [Azure Resource Manager templates](#) or the [REST API](#).

Within the API object, the Microsoft Entra identity provider configuration has a `validation` section that can include a `defaultAuthorizationPolicy` object as in the following structure:

JSON

```
{  
  "validation": {  
    "defaultAuthorizationPolicy": {  
      "allowedApplications": [],  
      "allowedPrincipals": {  
        "identities": []  
      }  
    }  
  }  
}
```

```
    }  
}
```

[+] Expand table

Property	Description
<code>defaultAuthorizationPolicy</code>	A grouping of requirements that must be met in order to access the app. Access is granted based on a logical <code>AND</code> over each of its configured properties. When <code>allowedApplications</code> and <code>allowedPrincipals</code> are both configured, the incoming request must satisfy both requirements in order to be accepted.
<code>allowedApplications</code>	An allowlist of string application client IDs representing the client resource that is calling into the app. When this property is configured as a nonempty array, only tokens obtained by an application specified in the list will be accepted. This policy evaluates the <code>appid</code> or <code>azp</code> claim of the incoming token, which must be an access token. See the Microsoft identity platform claims reference .
<code>allowedPrincipals</code>	A grouping of checks that determine if the principal represented by the incoming request may access the app. Satisfaction of <code>allowedPrincipals</code> is based on a logical <code>OR</code> over its configured properties.
<code>identities</code> (under <code>allowedPrincipals</code>)	An allowlist of string object IDs representing users or applications that have access. When this property is configured as a nonempty array, the <code>allowedPrincipals</code> requirement can be satisfied if the user or application represented by the request is specified in the list. There's a limit of 500 characters total across the list of identities. This policy evaluates the <code>oid</code> claim of the incoming token. See the Microsoft identity platform claims reference .

Additionally, some checks can be configured through an [application setting](#), regardless of the API version being used. The `WEBSITE_AUTH_AAD_ALLOWED_TENANTS` application setting can be configured with a comma-separated list of up to 10 tenant IDs (for example, "559a2f9c-c6f2-4d31-b8d6-5ad1a13f8330,5693f64a-3ad5-4be7-b846-e9d1141bcebc") to require that the incoming token is from one of the specified tenants, as specified by the `tid` claim. The `WEBSITE_AUTH_AAD_REQUIRE_CLIENT_SERVICE_PRINCIPAL` application setting can be configured to "true" or "1" to require the incoming token to include an `oid` claim. This setting is ignored and treated as true if

`allowedPrincipals.identities` has been configured (since the `oid` claim is checked against this provided list of identities).

Requests that fail these built-in checks are given an HTTP `403 Forbidden` response.

Configure client apps to access your App Service

In the prior sections, you registered your App Service or Azure Function to authenticate users. This section explains how to register native clients or daemon apps in Microsoft Entra so that they can request access to APIs exposed by your App Service on behalf of users or themselves, such as in an N-tier architecture. Completing the steps in this section isn't required if you only wish to authenticate users.

Native client application

You can register native clients to request access your App Service app's APIs on behalf of a signed in user.

1. From the portal menu, select **Microsoft Entra**.
2. From the left navigation, select **App registrations > New registration**.
3. In the **Register an application** page, enter a **Name** for your app registration.
4. In **Redirect URI**, select **Public client (mobile & desktop)** and type the URL `<app-url>/auth/login/aad/callback`. For example,
`https://contoso.azurewebsites.net/.auth/login/aad/callback`.
5. Select **Register**.
6. After the app registration is created, copy the value of **Application (client) ID**.

 **Note**

For a Microsoft Store application, use the [package SID](#) as the URI instead.

7. From the left navigation, select **API permissions > Add a permission > My APIs**.
8. Select the app registration you created earlier for your App Service app. If you don't see the app registration, make sure that you've added the `user_impersonation` scope in the app registration.

9. Under **Delegated permissions**, select **user_impersonation**, and then select **Add permissions**.

You have now configured a native client application that can request access your App Service app on behalf of a user.

Daemon client application (service-to-service calls)

In an N-tier architecture, your client application can acquire a token to call an App Service or Function app on behalf of the client app itself (not on behalf of a user). This scenario is useful for non-interactive daemon applications that perform tasks without a logged in user. It uses the standard OAuth 2.0 [client credentials](#) grant.

1. From the portal menu, select **Microsoft Entra**.
2. From the left navigation, select **App registrations > New registration**.
3. In the **Register an application** page, enter a **Name** for your app registration.
4. For a daemon application, you don't need a Redirect URI so you can keep that empty.
5. Select **Register**.
6. After the app registration is created, copy the value of **Application (client) ID**.
7. From the left navigation, select **Certificates & secrets > Client secrets > New client secret**.
8. Enter a description and expiration and select **Add**.
9. In the **Value** field, copy the client secret value. It won't be shown again once you navigate away from this page.

You can now [request an access token using the client ID and client secret](#) by setting the **resource** parameter to the **Application ID URI** of the target app. The resulting access token can then be presented to the target app using the standard [OAuth 2.0 Authorization header](#), and App Service authentication will validate and use the token as usual to now indicate that the caller (an application in this case, not a user) is authenticated.

At present, this allows *any* client application in your Microsoft Entra tenant to request an access token and authenticate to the target app. If you also want to enforce *authorization* to allow only certain client applications, you must perform some extra configuration.

1. [Define an App Role](#) in the manifest of the app registration representing the App Service or Function app you want to protect.
2. On the app registration representing the client that needs to be authorized, select **API permissions > Add a permission > My APIs**.

3. Select the app registration you created earlier. If you don't see the app registration, make sure that you've [added an App Role](#).
4. Under **Application permissions**, select the App Role you created earlier, and then select **Add permissions**.
5. Make sure to select **Grant admin consent** to authorize the client application to request the permission.
6. Similar to the previous scenario (before any roles were added), you can now [request an access token](#) for the same target `resource`, and the access token will include a `roles` claim containing the App Roles that were authorized for the client application.
7. Within the target App Service or Function app code, you can now validate that the expected roles are present in the token (this isn't performed by App Service authentication). For more information, see [Access user claims](#).

You have now configured a daemon client application that can access your App Service app using its own identity.

Best practices

Regardless of the configuration you use to set up authentication, the following best practices keep your tenant and applications more secure:

- Configure each App Service app with its own app registration in Microsoft Entra.
- Give each App Service app its own permissions and consent.
- Avoid permission sharing between environments by using separate app registrations for separate deployment slots. When you're testing new code, this practice can help prevent issues from affecting the production app.

Migrate to the Microsoft Graph

Some older apps may also have been set up with a dependency on the [deprecated Azure AD Graph](#), which is scheduled for full retirement. For example, your app code may have called Azure AD Graph to check group membership as part of an authorization filter in a middleware pipeline. Apps should move to the [Microsoft Graph](#) by following the [guidance provided by Microsoft Entra as part of the Azure AD Graph deprecation process](#). In following those instructions, you may need to make some changes to your configuration of App Service authentication. Once you have added Microsoft Graph permissions to your app registration, you can:

1. Update the **Issuer URL** to include the "/v2.0" suffix if it doesn't already.

2. Remove requests for Azure AD Graph permissions from your sign-in configuration.

The properties to change depend on [which version of the management API you're using](#):

- If you're using the V1 API (`/authsettings`), this would be in the `additionalLoginParams` array.
- If you're using the V2 API (`/authsettingsv2`), this would be in the `loginParameters` array.

You would need to remove any reference to "https://graph.windows.net", for example. This includes the `resource` parameter (which isn't supported by the "/v2.0" endpoint) or any scopes you're specifically requesting that are from the Azure AD Graph.

You would also need to update the configuration to request the new Microsoft Graph permissions you set up for the application registration. You can use the `.default scope` to simplify this setup in many cases. To do so, add a new sign-in parameter `scope=openid profile email https://graph.microsoft.com/.default`.

With these changes, when App Service Authentication attempts to sign in, it will no longer request permissions to the Azure AD Graph, and instead it will get a token for the Microsoft Graph. Any use of that token from your application code would also need to be updated, as per the [guidance provided by Microsoft Entra](#).

Next steps

- [App Service Authentication / Authorization overview](#).
- [Tutorial: Authenticate and authorize users end-to-end in Azure App Service](#)
- [Tutorial: Authenticate and authorize users in a web app that accesses Azure Storage and Microsoft Graph](#)
- [Tutorial: Authenticate and authorize users end-to-end in Azure App Service](#)

Configure your App Service or Azure Functions app to use Facebook login

Article • 03/31/2021

This article shows how to configure Azure App Service or Azure Functions to use Facebook as an authentication provider.

To complete the procedure in this article, you need a Facebook account that has a verified email address and a mobile phone number. To create a new Facebook account, go to facebook.com.

Register your application with Facebook

1. Go to the [Facebook Developers](#) website and sign in with your Facebook account credentials.

If you don't have a Facebook for Developers account, select **Get Started** and follow the registration steps.

2. Select **My Apps > Add New App**.

3. In **Display Name** field:

- a. Type a unique name for your app.
- b. Provide your **Contact Email**.
- c. Select **Create App ID**.
- d. Complete the security check.

The developer dashboard for your new Facebook app opens.

4. Select **Dashboard > Facebook Login > Set up > Web**.

5. In the left navigation under **Facebook Login**, select **Settings**.

6. In the **Valid OAuth redirect URIs** field, enter `https://<app-name>.azurewebsites.net/.auth/login/facebook/callback`. Remember to replace `<app-name>` with the name of your Azure App Service app.

7. Select **Save Changes**.

8. In the left pane, select **Settings > Basic**.

9. In the App Secret field, select Show. Copy the values of App ID and App Secret. You use them later to configure your App Service app in Azure.

ⓘ Important

The app secret is an important security credential. Do not share this secret with anyone or distribute it within a client application.

10. The Facebook account that you used to register the application is an administrator of the app. At this point, only administrators can sign in to this application.

To authenticate other Facebook accounts, select **App Review** and enable **Make <your-app-name> public** to enable the general public to access the app by using Facebook authentication.

Add Facebook information to your application

1. Sign in to the [Azure portal](#) and navigate to your app.
2. Select **Authentication** in the menu on the left. Click **Add identity provider**.
3. Select **Facebook** in the identity provider dropdown. Paste in the App ID and App Secret values that you obtained previously.

The secret will be stored as a slot-sticky [application setting](#) named `FACEBOOK_PROVIDER_AUTHENTICATION_SECRET`. You can update that setting later to use [Key Vault references](#) if you wish to manage the secret in Azure Key Vault.

4. If this is the first identity provider configured for the application, you will also be prompted with an **App Service authentication settings** section. Otherwise, you may move on to the next step.

These options determine how your application responds to unauthenticated requests, and the default selections will redirect all requests to log in with this new provider. You can change customize this behavior now or adjust these settings later from the main **Authentication** screen by choosing **Edit** next to **Authentication settings**. To learn more about these options, see [Authentication flow](#).

5. (Optional) Click **Next: Scopes** and add any scopes needed by the application. These will be requested at login time for browser-based flows.
6. Click **Add**.

You're now ready to use Facebook for authentication in your app. The provider will be listed on the **Authentication** screen. From there, you can edit or delete this provider configuration.

Next steps

- [App Service Authentication / Authorization overview.](#)
- [Tutorial: Authenticate and authorize users end-to-end in Azure App Service](#)

Configure your App Service or Azure Functions app to use GitHub login

Article • 03/01/2022

This article shows how to configure Azure App Service or Azure Functions to use GitHub as an authentication provider.

To complete the procedure in this article, you need a GitHub account. To create a new GitHub account, go to [GitHub](#).

Register your application with GitHub

1. Sign in to the [Azure portal](#) and go to your application. Copy your **URL**. You'll use it to configure your GitHub app.
2. Follow the instructions for [creating an OAuth app on GitHub](#). In the **Authorization callback URL** section, enter the HTTPS URL of your app and append the path `/.auth/login/github/callback`. For example,
`https://contoso.azurewebsites.net/.auth/login/github/callback`.
3. On the application page, make note of the **Client ID**, which you will need later.
4. Under **Client Secrets**, select **Generate a new client secret**.
5. Make note of the client secret value, which you will need later.

Important

The client secret is an important security credential. Do not share this secret with anyone or distribute it with your app.

Add GitHub information to your application

1. Sign in to the [Azure portal](#) and navigate to your app.
2. Select **Authentication** in the menu on the left. Click **Add identity provider**.
3. Select **GitHub** in the identity provider dropdown. Paste in the **Client ID** and **Client secret** values that you obtained previously.

The secret will be stored as a slot-sticky [application setting](#) named `GITHUB_PROVIDER_AUTHENTICATION_SECRET`. You can update that setting later to use [Key Vault references](#) if you wish to manage the secret in Azure Key Vault.

4. If this is the first identity provider configured for the application, you will also be prompted with an **App Service authentication settings** section. Otherwise, you may move on to the next step.

These options determine how your application responds to unauthenticated requests, and the default selections will redirect all requests to log in with this new provider. You can change customize this behavior now or adjust these settings later from the main **Authentication** screen by choosing **Edit** next to **Authentication settings**. To learn more about these options, see [Authentication flow](#).

5. Click **Add**.

You're now ready to use GitHub for authentication in your app. The provider will be listed on the **Authentication** screen. From there, you can edit or delete this provider configuration.

Configure your App Service or Azure Functions app to use Google login

Article • 03/25/2024

This topic shows you how to configure Azure App Service or Azure Functions to use Google as an authentication provider.

To complete the procedure in this topic, you must have a Google account that has a verified email address. To create a new Google account, go to accounts.google.com.

Register your application with Google

1. Follow the Google documentation at [Sign In with Google for Web - Setup](#) to create a client ID and client secret. There's no need to make any code changes. Just use the following information:
 - For **Authorized JavaScript Origins**, use `https://<app-name>.azurewebsites.net` with the name of your app in `<app-name>`.
 - For **Authorized Redirect URI**, use `https://<app-name>.azurewebsites.net/.auth/login/google/callback`.
2. Copy the App ID and the App secret values.

Important

The App secret is an important security credential. Do not share this secret with anyone or distribute it within a client application.

Add Google information to your application

1. Sign in to the [Azure portal](#) and navigate to your app.
2. Select **Authentication** in the menu on the left. Click **Add identity provider**.
3. Select **Google** in the identity provider dropdown. Paste in the App ID and App Secret values that you obtained previously.

The secret will be stored as a slot-sticky [application setting](#) named `GOOGLE_PROVIDER_AUTHENTICATION_SECRET`. You can update that setting later to use

[Key Vault references](#) if you wish to manage the secret in Azure Key Vault.

4. If this is the first identity provider configured for the application, you will also be prompted with an **App Service authentication settings** section. Otherwise, you may move on to the next step.

These options determine how your application responds to unauthenticated requests, and the default selections will redirect all requests to log in with this new provider. You can change customize this behavior now or adjust these settings later from the main **Authentication** screen by choosing **Edit** next to **Authentication settings**. To learn more about these options, see [Authentication flow](#).

5. Click **Add**.

 **Note**

For adding scope: You can define what permissions your application has in the provider's registration portal. The app can request scopes at login time which leverage these permissions.

You are now ready to use Google for authentication in your app. The provider will be listed on the **Authentication** screen. From there, you can edit or delete this provider configuration.

Next steps

- [App Service Authentication / Authorization overview](#).
- [Tutorial: Authenticate and authorize users end-to-end in Azure App Service](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | Get help at Microsoft Q&A

Configure your App Service or Azure Functions app to use X login

Article • 08/09/2024

This article shows how to configure Azure App Service or Azure Functions to use X as an authentication provider.

To complete the procedure in this article, you need an X account that has a verified email address and phone number. To create a new X account, go to [x.com](#).

Register your application with X

1. Sign in to the [Azure portal](#) and go to your application. Copy your **URL**. You'll use it to configure your X app.
2. Go to the [X Developers](#) website, sign in with your X account credentials, and select **Create an app**.
3. Enter the **App name** and the **Application description** for your new app. Paste your application's **URL** into the **Website URL** field. In the **Callback URLs** section, enter the HTTPS URL of your App Service app and append the path `/ .auth/login/x/callback`. For example,
`https://contoso.azurewebsites.net/.auth/login/x/callback`.
4. At the bottom of the page, type at least 100 characters in **Tell us how this app will be used**, then select **Create**. Click **Create** again in the pop-up. The application details are displayed.
5. Select the **Keys and Access Tokens** tab.

Make a note of these values:

- API key
- API secret key

Important

The API secret key is an important security credential. Do not share this secret with anyone or distribute it with your app.

Add X information to your application

1. Sign in to the [Azure portal](#) and navigate to your app.
2. Select **Authentication** in the menu on the left. Click **Add identity provider**.
3. Select **Twitter** in the identity provider dropdown. Paste in the `API key` and `API secret key` values that you obtained previously.

The secret will be stored as a slot-sticky [application setting](#) named `TWITTER_PROVIDER_AUTHENTICATION_SECRET`. You can update that setting later to use [Key Vault references](#) if you wish to manage the secret in Azure Key Vault.

4. If this is the first identity provider configured for the application, you will also be prompted with an **App Service authentication settings** section. Otherwise, you may move on to the next step.

These options determine how your application responds to unauthenticated requests, and the default selections will redirect all requests to log in with this new provider. You can change customize this behavior now or adjust these settings later from the main **Authentication** screen by choosing **Edit** next to **Authentication settings**. To learn more about these options, see [Authentication flow](#).

5. Click **Add**.

You're now ready to use X for authentication in your app. The provider will be listed on the **Authentication** screen. From there, you can edit or delete this provider configuration.

Next steps

- [App Service Authentication / Authorization overview](#).
- [Tutorial: Authenticate and authorize users end-to-end in Azure App Service](#)

Feedback

Was this page helpful?

 Yes

 No

Configure your App Service or Azure Functions app to sign in using an OpenID Connect provider

Article • 09/04/2023

This article shows you how to configure Azure App Service or Azure Functions to use a custom authentication provider that adheres to the [OpenID Connect specification](#). OpenID Connect (OIDC) is an industry standard used by many identity providers (IDPs). You don't need to understand the details of the specification in order to configure your app to use an adherent IDP.

You can configure your app to use one or more OIDC providers. Each must be given a unique alphanumeric name in the configuration, and only one can serve as the default redirect target.

Register your application with the identity provider

Your provider will require you to register the details of your application with it. One of these steps involves specifying a redirect URI. This redirect URI will be of the form `<app-url>/auth/login/<provider-name>/callback`. Each identity provider should provide more instructions on how to complete these steps. `<provider-name>` will refer to the friendly name you give to the OpenID provider name in Azure.

Note

Some providers may require additional steps for their configuration and how to use the values they provide. For example, Apple provides a private key which is not itself used as the OIDC client secret, and you instead must use it to craft a JWT which is treated as the secret you provide in your app config (see the "Creating the Client Secret" section of the [Sign in with Apple documentation](#))

You'll need to collect a **client ID** and **client secret** for your application.

Important

The client secret is an important security credential. Don't share this secret with anyone or distribute it within a client application.

Additionally, you'll need the OpenID Connect metadata for the provider. This is often exposed via a [configuration metadata document](#), which is the provider's Issuer URL suffixed with `/well-known/openid-configuration`. Gather this configuration URL.

If you're unable to use a configuration metadata document, you'll need to gather the following values separately:

- The issuer URL (sometimes shown as `issuer`)
- The [OAuth 2.0 Authorization endpoint](#) (sometimes shown as `authorization_endpoint`)
- The [OAuth 2.0 Token endpoint](#) (sometimes shown as `token_endpoint`)
- The URL of the [OAuth 2.0 JSON Web Key Set](#) document (sometimes shown as `jwks_uri`)

Add provider information to your application

1. Sign in to the [Azure portal](#) and navigate to your app.
2. Select **Authentication** in the menu on the left. Select **Add identity provider**.
3. Select **OpenID Connect** in the identity provider dropdown.
4. Provide the unique alphanumeric name selected earlier for **OpenID provider name**.
5. If you have the URL for the **metadata document** from the identity provider, provide that value for **Metadata URL**. Otherwise, select the **Provide endpoints separately** option and put each URL gathered from the identity provider in the appropriate field.
6. Provide the earlier collected **Client ID** and **Client Secret** in the appropriate fields.
7. Specify an application setting name for your client secret. Your client secret will be stored as an app setting to ensure secrets are stored in a secure fashion. You can update that setting later to use [Key Vault references](#) if you wish to manage the secret in Azure Key Vault.
8. Press the **Add** button to finish setting up the identity provider.

Note

The OpenID provider name can't contain symbols like `-` because an appsetting will be created based on this and it doesn't support it. Use `_` instead.

 **Note**

Azure requires "openid," "profile," and "email" scopes. Make sure you've configured your App Registration in your ID Provider with at least these scopes.

Next steps

- [App Service Authentication / Authorization overview](#).
- [Tutorial: Authenticate and authorize users end-to-end in Azure App Service](#)

Configure your App Service or Azure Functions app to sign in using a Sign in with Apple provider (Preview)

Article • 09/23/2021

This article shows you how to configure Azure App Service or Azure Functions to use Sign in with Apple as an authentication provider.

To complete the procedure in this article, you must have enrolled in the Apple developer program. To enroll in the Apple developer program, go to developer.apple.com/programs/enroll.

✖ Caution

Enabling Sign in with Apple will disable management of the App Service Authentication / Authorization feature for your application through some clients, such as the Azure portal, Azure CLI, and Azure PowerShell. The feature relies on a new API surface which, during preview, is not yet accounted for in all management experiences.

Create an application in the Apple Developer portal

You'll need to create an App ID and a service ID in the Apple Developer portal.

1. On the Apple Developer portal, go to **Certificates, Identifiers, & Profiles**.
2. On the **Identifiers** tab, select the (+) button.
3. On the **Register a New Identifier** page, choose **App IDs** and select **Continue**. (App IDs include one or more Service IDs.)

Register a New Identifier

[Continue](#)

App IDs
Register an App ID to enable your app to access available services and identify your app in a provisioning profile. You can enable app services when you create an App ID or modify these settings later.

Services IDs
For each website that uses Sign In with Apple, register a services identifier (Services ID), configure your domain and return URL, and create an associated private key.

4. On the **Register an App ID** page, provide a description and a bundle ID, and select **Sign in with Apple** from the capabilities list. Then select **Continue**. Take note of

your App ID Prefix (Team ID) from this step, you'll need it later.

Register an App ID

[Back](#) [Continue](#)

Platform <input checked="" type="radio"/> iOS, tvOS, watchOS <input type="radio"/> macOS	App ID Prefix E88K2FF6LU (Team ID)
Description easy auth test sign in with apple You cannot use special characters such as @, &, *, ;, "	Bundle ID <input checked="" type="radio"/> Explicit <input type="radio"/> Wildcard com.microsoft.easyauthtest.client We recommend using a reverse-domain name style string (i.e., com.domainname.appname). It cannot contain an asterisk (*).

Capabilities

ENABLED	NAME
<input type="checkbox"/>	Access WiFi Information

5. Review the app registration information and select **Register**.

6. Again, on the Identifiers tab, select the (+) button.

Certificates, Identifiers & Profiles

Certificates	Identifiers +
Identifiers	NAME IDENTIFIER
Devices	NE [REDACTED] com.mi[REDACTED]
Profiles	Xcode iOS Wildcard App ID *
Keys	
More	

7. On the Register a New Identifier page, choose Services IDs and select Continue.

Register a New Identifier

[Continue](#)

- App IDs
Register an App ID to enable your app to access available services and identify your app in a provisioning profile. You can enable app services when you create an App ID or modify these settings later.
- Services IDs
For each website that uses Sign In with Apple, register a services identifier (Services ID), configure your domain and return URL, and create an associated private key.

8. On the Register a Services ID page, provide a description and an identifier. The description is what will be shown to the user on the consent screen. The identifier will be your client ID used in configuring the Apple provider with your app service.

Then select **Configure**.

Register a Services ID

[Back](#) [Continue](#)

Description easy auth test sign in with apple You cannot use special characters such as @, &, *, ;, "	Identifier com.microsoft.easyauthtest.client We recommend using a reverse-domain name style string (i.e., com.domainname.appname). It cannot contain an asterisk (*).
ENABLED <input checked="" type="checkbox"/>	NAME Sign In with Apple Configure

9. On the pop-up window, set the Primary App ID to the App ID you created earlier. Specify your application's domain in the domain section. For the return URL, use the URL `<app-url>/auth/login/apple/callback`. For example,

`https://contoso.azurewebsites.net/.auth/login/apple/callback`. Then select Add

and Save.

The screenshot shows the 'Web Authentication Configuration' page. At the top, there is a brief description of using Sign In with Apple. Below it, a 'Primary App ID' dropdown is set to 'Easy Auth test ID (SGGM6D27TK.com.micros)'. The 'Domains' section contains a 'Web Domain' input field with 'easyauth.net' and an 'Add' button next to a 'Return URLs' input field containing 'https://easyauth.net/.auth/login/apple/callback'. At the bottom right are 'Cancel' and 'Save' buttons, with 'Save' being highlighted by an orange box.

Primary App ID
Easy Auth test ID (SGGM6D27TK.com.micros)

Domains

Provide your web domain and return URLs to redirect users after successfully signing in to your website. After registering your Services ID, you'll need to verify your web domain. Return to this page to continue the configuration process.

Web Domain
To verify ownership of your domain, click Download to get the verification file, upload it at the URL listed below, and click Verify.

https://example.com/well-known/apple-developer-domain-association.txt

easyauth.net

Return URLs
https://easyauth.net/.auth/login/apple/callback

Cancel

10. Review the service registration information and select **Save**.

Generate the client secret

Apple requires app developers to create and sign a JWT token as the client secret value. To generate this secret, first generate and download an elliptic curve private key from the Apple Developer portal. Then, use that key to [sign a JWT](#) with a [specific payload](#).

Create and download the private key

1. On the **Keys** tab in the Apple Developer portal, choose **Create a key** or select the (+) button.
2. On the **Register a New Key** page give the key a name, check the box next to **Sign in with Apple** and select **Configure**.
3. On the **Configure Key** page, link the key to the primary app ID you created previously and select **Save**.
4. Finish creating the key by confirming the information and selecting **Continue** and then reviewing the information and selecting **Register**.

5. On the [Download Your Key](#) page, download the key. It will download as a `.p8` (PKCS#8) file - you'll use the file contents to sign your client secret JWT.

Structure the client secret JWT

Apple requires the client secret be the base64-encoding of a JWT token. The decoded JWT token should have a payload structured like this example:

```
JSON

{
  "alg": "ES256",
  "kid": "URKEYID001",
}.{
  "sub": "com.yourcompany.app1",
  "nbf": 1560203207,
  "exp": 1560289607,
  "iss": "ABC123DEFG",
  "aud": "https://appleid.apple.com"
}.[Signature]
```

- **sub**: The Apple Client ID (also the service ID)
- **iss**: Your Apple Developer Team ID
- **aud**: Apple is receiving the token, so they're the audience
- **exp**: No more than six months after **nbf**

The base64-encoded version of the above payload looks like this:

```
eyJhbGciOiJFUzI1NiIsImtpZCI6IlVSS0VZSUQwMDEifQ.eyJzdWIiOiJjb20ueW91cmNvbXBhbnkuYXBw
MSIiIm5iZiI6MTU2MDIwMzIwNywiZXhwIjoxNTYwMjg5Nja3LCJpc3MiOiJBQkMxMjNERUZHIIiwiYXVkJ
iaHR0cHM6Ly9hcHBsZWlkLmFwcGx1LmNvbSJ9.ABSXELWuTbgqfrIUz7bLi6nXvkXAz508vt0jb2dSHTQTi
b1x1DSP4_4Ur1KI-pdzNg1sgeocolPNTmDKaz08-
BHAZCsdeeTNlgFEzBytIpMKFFVEQbEtGRkam5Iec1UK7S9oOva4EK4jV4VmDrr-
LGWW03TaAxAvy3_ZoKohvFFkVG
```

Note: Apple doesn't accept client secret JWTs with an expiration date more than six months after the creation (or nbf) date. That means you'll need to rotate your client secret, at minimum, every six months.

More information about generating and validating tokens can be found in [Apple's developer documentation](#).

Sign the client secret JWT

You'll use the `.p8` file you downloaded previously to sign the client secret JWT. This file is a [PCKS#8 file](#) that contains the private signing key in PEM format. There are many libraries that can create and sign the JWT for you.

There are different kinds of open-source libraries available online for creating and signing JWT tokens. For more information about generating JWT tokens, see [JSON Web Token \(JWT\)](#). For example, one way of generating the client secret is by importing the [Microsoft.IdentityModel.Tokens NuGet package](#) and running a small amount of C# code shown below.

C#

```
using Microsoft.IdentityModel.Tokens;

public static string GetAppleClientSecret(string teamId, string clientId,
string keyId, string p8key)
{
    string audience = "https://appleid.apple.com";

    string issuer = teamId;
    string subject = clientId;
    string kid = keyId;

    IList<Claim> claims = new List<Claim> {
        new Claim ("sub", subject)
    };

    CngKey cngKey = CngKey.Import(Convert.FromBase64String(p8key),
CngKeyBlobFormat.Pkcs8PrivateBlob);

    SigningCredentials signingCred = new SigningCredentials(
        new ECDsaSecurityKey(new ECDsaCng(cngKey)),
        SecurityAlgorithms.EcdsaSha256
    );

    JwtSecurityToken token = new JwtSecurityToken(
        issuer,
        audience,
        claims,
        DateTime.Now,
        DateTime.Now.AddDays(180),
        signingCred
    );
    token.Header.Add("kid", kid);
    token.Header.Remove("typ");

    JwtSecurityTokenHandler tokenHandler = new JwtSecurityTokenHandler();

    return tokenHandler.WriteToken(token);
}
```

- **teamId**: Your Apple Developer Team ID
- **clientId**: The Apple Client ID (also the service ID)
- **p8key**: The PEM format key - you can obtain the key by opening the `.p8` file in a text editor, and copying everything between `-----BEGIN PRIVATE KEY-----` and `-----END PRIVATE KEY-----` without line breaks
- **keyId**: The ID of the downloaded key

This token returned is the client secret value you'll use to configure the Apple provider.

Important

The client secret is an important security credential. Do not share this secret with anyone or distribute it within a client application.

Add the client secret as an [application setting](#) for the app, using a setting name of your choice. Make note of this name for later.

Add provider information to your application

Note

The required configuration is in a new API format, currently only supported by [file-based configuration \(preview\)](#). You will need to follow the below steps using such a file.

This section will walk you through updating the configuration to include your new IDP. An example configuration follows.

1. Within the `identityProviders` object, add an `apple` object if one doesn't already exist.
2. Assign an object to that key with a `registration` object within it, and optionally a `login` object:

JSON

```
"apple" : {  
    "registration" : {  
        "clientId": "<client ID>",  
        "clientSecretSettingName":  
        "APP_SETTING_CONTAINING_APPLE_CLIENT_SECRET"  
    },
```

```
    "login": {
        "scopes": []
    }
}
```

- a. Within the `registration` object, set the `clientId` to the client ID you collected.
- b. Within the `registration` object, set `clientSecretSettingName` to the name of the application setting where you stored the client secret.
- c. Within the `login` object, you may choose to set the `scopes` array to include a list of scopes used when authenticating with Apple, such as "name" and "email". If scopes are configured, they'll be explicitly requested on the consent screen when users sign in for the first time.

Once this configuration has been set, you're ready to use your Apple provider for authentication in your app.

A complete configuration might look like the following example (where the `APPLE_GENERATED_CLIENT_SECRET` setting points to an application setting containing a generated JWT):

JSON

```
{
    "platform": {
        "enabled": true
    },
    "globalValidation": {
        "redirectToProvider": "apple",
        "unauthenticatedClientAction": "RedirectLoginPage"
    },
    "identityProviders": {
        "apple": {
            "registration": {
                "clientId": "com.contoso.example.client",
                "clientSecretSettingName": "APPLE_GENERATED_CLIENT_SECRET"
            },
            "login": {
                "scopes": []
            }
        }
    },
    "login": {
        "tokenStore": {
            "enabled": true
        }
    }
}
```

Next steps

- [App Service Authentication / Authorization overview.](#)
- [Tutorial: Authenticate and authorize users end-to-end in Azure App Service](#)

Customize sign-in and sign-out in Azure App Service authentication

Article • 07/08/2024

This article shows you how to customize user sign-ins and sign-outs while using the built-in [authentication and authorization in App Service](#).

Use multiple sign-in providers

The portal configuration doesn't offer a turn-key way to present multiple sign-in providers to your users (such as both Facebook and Twitter). However, it isn't difficult to add the functionality to your app. The steps are outlined as follows:

First, in the **Authentication / Authorization** page in the Azure portal, configure each of the identity provider you want to enable.

In **Action to take when request is not authenticated**, select **Allow Anonymous requests (no action)**.

In the sign-in page, or the navigation bar, or any other location of your app, add a sign-in link to each of the providers you enabled (`/auth/login/<provider>`). For example:

HTML

```
<a href="/.auth/login/aad">Log in with Microsoft Entra</a>
<a href="/.auth/login/facebook">Log in with Facebook</a>
<a href="/.auth/login/google">Log in with Google</a>
<a href="/.auth/login/twitter">Log in with Twitter</a>
<a href="/.auth/login/apple">Log in with Apple</a>
```

When the user clicks on one of the links, the respective sign-in page opens to sign in the user.

To redirect the user post-sign-in to a custom URL, use the `post_login_redirect_uri` query string parameter (not to be confused with the Redirect URI in your identity provider configuration). For example, to navigate the user to `/Home/Index` after sign-in, use the following HTML code:

HTML

```
<a href="/.auth/login/<provider>?post_login_redirect_uri=/Home/Index">Log
```

in

Client-directed sign-in

In a client-directed sign-in, the application signs in the user to the identity provider using a provider-specific SDK. The application code then submits the resulting authentication token to App Service for validation (see [Authentication flow](#)) using an HTTP POST request. This validation itself doesn't actually grant you access to the desired app resources, but a successful validation will give you a session token that you can use to access app resources.

To validate the provider token, App Service app must first be configured with the desired provider. At runtime, after you retrieve the authentication token from your provider, post the token to `/.auth/login/<provider>` for validation. For example:

```
POST https://<appname>.azurewebsites.net/.auth/login/aad HTTP/1.1
Content-Type: application/json
```

```
{"id_token": "<token>", "access_token": "<token>"}
```

The token format varies slightly according to the provider. See the following table for details:

[\[+\] Expand table](#)

Provider value	Required in request body	Comments
aad	<pre>{"access_token": "<access_token>"}</pre>	The <code>id_token</code> , <code>refresh_token</code> , and <code>expires_in</code> properties are optional.
microsoftaccount	<pre>{"access_token": "<access_token>"}</pre> or <pre>{"authentication_token": "<token>"}</pre>	<code>authentication_token</code> is preferred over <code>access_token</code> . The <code>expires_in</code> property is optional. When requesting the token from Live services, always request the <code>wl.basic</code> scope.
google	<pre>{"id_token": "<id_token>"}</pre>	The <code>authorization_code</code> property is optional. Providing an <code>authorization_code</code> value will add an access token and a refresh token to the token store. When specified,

Provider value	Required in request body	Comments
		<code>authorization_code</code> can also optionally be accompanied by a <code>redirect_uri</code> property.
facebook	{"access_token": " <user_access_token>"}	Use a valid user access token from Facebook.
twitter	{"access_token": " <access_token>," "access_token_secret": " <access_token_secret>"}	

ⓘ Note

The GitHub provider for App Service authentication does not support customized sign-in and sign-out.

If the provider token is validated successfully, the API returns with an `authenticationToken` in the response body, which is your session token.

JSON

```
{
  "authenticationToken": "...",
  "user": {
    "userId": "sid:..."
  }
}
```

Once you have this session token, you can access protected app resources by adding the `X-ZUMO-AUTH` header to your HTTP requests. For example:

```
GET https://<appname>.azurewebsites.net/api/products/1
X-ZUMO-AUTH: <authenticationToken_value>
```

Sign out of a session

Users can initiate a sign-out by sending a `GET` request to the app's `/auth/logout` endpoint. The `GET` request does the following:

- Clears authentication cookies from the current session.
- Deletes the current user's tokens from the token store.
- For Microsoft Entra and Google, performs a server-side sign-out on the identity provider.

Here's a simple sign-out link in a webpage:

HTML

```
<a href="/.auth/logout">Sign out</a>
```

By default, a successful sign-out redirects the client to the URL `/auth/logout/complete`.

You can change the post-sign-out redirect page by adding the `post_logout_redirect_uri` query parameter. For example:

```
GET /.auth/logout?post_logout_redirect_uri=/index.html
```

It's recommended that you [encode ↗](#) the value of `post_logout_redirect_uri`.

When using fully qualified URLs, the URL must be either hosted in the same domain or configured as an allowed external redirect URL for your app. In the following example, to redirect to `https://myexternalurl.com` that's not hosted in the same domain:

```
GET /.auth/logout?post_logout_redirect_uri=https%3A%2F%2Fmyexternalurl.com
```

Run the following command in the [Azure Cloud Shell](#):

Azure CLI

```
az webapp auth update --name <app_name> --resource-group <group_name> --  
allowed-external-redirect-urls "https://myexternalurl.com"
```

Preserve URL fragments

After users sign in to your app, they usually want to be redirected to the same section of the same page, such as `/wiki/Main_Page#SectionZ`. However, because [URL fragments ↗](#) (for example, `#SectionZ`) are never sent to the server, they are not preserved by default after the OAuth sign-in completes and redirects back to your app. Users then get a

suboptimal experience when they need to navigate to the desired anchor again. This limitation applies to all server-side authentication solutions.

In App Service authentication, you can preserve URL fragments across the OAuth sign-in. To do this, set an app setting called `WEBSITE_AUTH_PRESERVE_URL_FRAGMENT` to `true`. You can do it in the [Azure portal](#), or simply run the following command in the [Azure Cloud Shell](#):

Azure CLI

```
az webapp config appsettings set --name <app_name> --resource-group <group_name> --settings WEBSITE_AUTH_PRESERVE_URL_FRAGMENT="true"
```

Setting the sign-in accounts domain hint

Both Microsoft Account and Microsoft Entra lets you sign in from multiple domains. For example, Microsoft Account allows `outlook.com`, `live.com`, and `hotmail.com` accounts. Microsoft Entra allows any number of custom domains for the sign-in accounts. However, you may want to accelerate your users straight to your own branded Microsoft Entra sign-in page (such as `contoso.com`). To suggest the domain name of the sign-in accounts, follow these steps.

1. In <https://resources.azure.com>, At the top of the page, select **Read/Write**.
2. In the left browser, navigate to **subscriptions** > `<subscription-name>` > **resourceGroups** > `<resource-group-name>` > **providers** > **Microsoft.Web** > **sites** > `<app-name>` > **config** > **authsettingsV2**.
3. Click **Edit**.
4. Add a `loginParameters` array with a `domain_hint` item.

JSON

```
"identityProviders": {  
    "azureActiveDirectory": {  
        "login": {  
            "loginParameters": ["domain_hint=<domain-name>"],  
        }  
    }  
}
```

5. Click **Put**.

This setting appends the `domain_hint` query string parameter to the login redirect URL.

ⓘ Important

It's possible for the client to remove the `domain_hint` parameter after receiving the redirect URL, and then login with a different domain. So while this function is convenient, it's not a security feature.

Authorize or deny users

While App Service takes care of the simplest authorization case (i.e. reject unauthenticated requests), your app may require more fine-grained authorization behavior, such as limiting access to only a specific group of users. In certain cases, you need to write custom application code to allow or deny access to the signed-in user. In other cases, App Service or your identity provider may be able to help without requiring code changes.

- [Server level](#)
- [Identity provider level](#)
- [Application level](#)

Server level (Windows apps only)

For any Windows app, you can define authorization behavior of the IIS web server, by editing the `Web.config` file. Linux apps don't use IIS and can't be configured through `Web.config`.

1. Navigate to `https://<app-name>.scm.azurewebsites.net/DebugConsole`
2. In the browser explorer of your App Service files, navigate to `site/wwwroot`. If a `Web.config` doesn't exist, create it by selecting `+ > New File`.
3. Select the pencil for `Web.config` to edit it. Add the following configuration code and click **Save**. If `Web.config` already exists, just add the `<authorization>` element with everything in it. Add the accounts you want to allow in the `<allow>` element.

XML

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
  <system.web>
    <authorization>
```

```
<allow users="user1@contoso.com,user2@contoso.com"/>
<deny users="*"/>
</authorization>
</system.web>
</configuration>
```

Identity provider level

The identity provider may provide certain turn-key authorization. For example:

- You can [manage enterprise-level access](#) directly in Microsoft Entra. For instructions, see [How to remove a user's access to an application](#).
- For [Google](#), Google API projects that belong to an [organization](#) can be configured to allow access only to users in your organization (see [Google's Setting up OAuth 2.0 support page](#)).

Application level

If either of the other levels don't provide the authorization you need, or if your platform or identity provider isn't supported, you must write custom code to authorize users based on the [user claims](#).

More resources

- [Tutorial: Authenticate and authorize users end-to-end](#)
- [Environment variables and app settings for authentication](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#)

Work with user identities in Azure App Service authentication

Article • 07/22/2024

This article shows you how to work with user identities when using the built-in [authentication and authorization in App Service](#).

Access user claims in app code

For all language frameworks, App Service makes the claims in the incoming token (whether from an authenticated end user or a client application) available to your code by injecting them into the request headers. External requests aren't allowed to set these headers, so they're present only if set by App Service. Some example headers include:

[+] [Expand table](#)

Header	Description
X-MS-CLIENT-PRINCIPAL	A Base64 encoded JSON representation of available claims. For more information, see Decoding the client principal header .
X-MS-CLIENT-PRINCIPAL-ID	An identifier for the caller set by the identity provider.
X-MS-CLIENT-PRINCIPAL-NAME	A human-readable name for the caller set by the identity provider, such as email address or user principal name.
X-MS-CLIENT-PRINCIPAL-IDP	The name of the identity provider used by App Service Authentication.

Provider tokens are also exposed through similar headers. For example, Microsoft Entra also sets `X-MS-TOKEN-AAD-ACCESS-TOKEN` and `X-MS-TOKEN-AAD-ID-TOKEN` as appropriate.

Note

Different language frameworks might present these headers to the app code in different formats, such as lowercase or title case.

Code that is written in any language or framework can get the information that it needs from these headers. [Decoding the client principal header](#) covers this process. For some frameworks, the platform also provides extra options that might be more convenient.

Decoding the client principal header

`X-MS-CLIENT-PRINCIPAL` contains the full set of available claims as Base64 encoded JSON. These claims go through a default claims-mapping process, so some might have different names than you would see if processing the token directly. The decoded payload is structured as follows:

JSON
{ "auth_typ": "", "claims": [{ "typ": "", "val": "" }], "name_typ": "", "role_typ": "" }

[+] [Expand table](#)

Property	Type	Description
<code>auth_typ</code>	string	The name of the identity provider used by App Service Authentication.
<code>claims</code>	array of objects	An array of objects representing the available claims. Each object contains <code>typ</code> and <code>val</code> properties.
<code>typ</code>	string	The name of the claim. It might be subject to default claims mapping and could be different from the corresponding claim contained in a token.
<code>val</code>	string	The value of the claim.
<code>name_typ</code>	string	The name claim type, which is typically a URI providing scheme information about the <code>name</code> claim if one is defined.
<code>role_typ</code>	string	The role claim type, which is typically a URI providing scheme information about the <code>role</code> claim if one is defined.

To process this header, your app needs to decode the payload and iterate through the `claims` array to find the claims of interest. It might be convenient to convert them into a representation used by the app's language framework. Here's an example of this process in C# that constructs a `ClaimsPrincipal` type for the app to use:

C#

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Security.Claims;
using System.Text;
using System.Text.Json;
using System.Text.Json.Serialization;
using Microsoft.AspNetCore.Http;

public static class ClaimsPrincipalParser
{
    private class ClientPrincipalClaim
    {
        [JsonPropertyName("typ")]
        public string Type { get; set; }
        [JsonPropertyName("val")]
        public string Value { get; set; }
    }

    private class ClientPrincipal
    {
        [JsonPropertyName("auth_typ")]
        public string IdentityProvider { get; set; }
        [JsonPropertyName("name_typ")]
        public string NameClaimType { get; set; }
        [JsonPropertyName("role_typ")]
        public string RoleClaimType { get; set; }
        [JsonPropertyName("claims")]
        public IEnumerable<ClientPrincipalClaim> Claims { get; set; }
    }

    public static ClaimsPrincipal Parse(HttpContext req)
    {
        var principal = new ClientPrincipal();

        if (req.Headers.TryGetValue("x-ms-client-principal", out var header))
        {
            var data = header[0];
            var decoded = Convert.FromBase64String(data);
            var json = Encoding.UTF8.GetString(decoded);
            principal = JsonSerializer.Deserialize<ClientPrincipal>(json,
new JsonSerializerOptions { PropertyNameCaseInsensitive = true });
        }

        /**
         * At this point, the code can iterate through `principal.Cclaims` to
         * check claims as part of validation. Alternatively, we can
         * convert
         * it into a standard object with which to perform those checks
         * later
    }
}
```

```

        * in the request pipeline. That object can also be leveraged for
        * associating user data, etc. The rest of this function performs
such
        * a conversion to create a `ClaimsPrincipal` as might be used in
        * other .NET code.
    */

    var identity = new ClaimsIdentity(principal.IdentityProvider,
principal.NameClaimType, principal.RoleClaimType);
    identity.AddClaims(principal.Claims.Select(c => new Claim(c.Type,
c.Value)));

    return new ClaimsPrincipal(identity);
}
}

```

Framework-specific alternatives

For ASP.NET 4.6 apps, App Service populates `ClaimsPrincipal.Current` with the authenticated user's claims, so you can follow the standard .NET code pattern, including the `[Authorize]` attribute. Similarly, for PHP apps, App Service populates the `_SERVER['REMOTE_USER']` variable. For Java apps, the claims are accessible from the [Tomcat servlet](#).

For [Azure Functions](#), `ClaimsPrincipal.Current` isn't populated for .NET code, but you can still find the user claims in the request headers, or get the `ClaimsPrincipal` object from the request context or even through a binding parameter. For more information, see [Working with client identities in Azure Functions](#).

For .NET Core, [Microsoft.Identity.Web](#) supports populating the current user with App Service authentication. To learn more, you can read about it on the [Microsoft.Identity.Web wiki](#), or see it demonstrated in [this tutorial for a web app accessing Microsoft Graph](#).

Note

For claims mapping to work, you must enable the [Token store](#).

Access user claims using the API

If the [token store](#) is enabled for your app, you can also obtain other details on the authenticated user by calling `/.auth/me`.

Next steps

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

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

Work with OAuth tokens in Azure App Service authentication

Article • 12/02/2022

This article shows you how to work with OAuth tokens while using the built-in [authentication and authorization in App Service](#).

Retrieve tokens in app code

From your server code, the provider-specific tokens are injected into the request header, so you can easily access them. The following table shows possible token header names:

Provider	Header names
Azure Active Directory	X-MS-TOKEN-AAD-ID-TOKEN X-MS-TOKEN-AAD-ACCESS-TOKEN X-MS-TOKEN-AAD-EXPIRES-ON X-MS-TOKEN-AAD-REFRESH-TOKEN
Facebook Token	X-MS-TOKEN-FACEBOOK-ACCESS-TOKEN X-MS-TOKEN-FACEBOOK-EXPIRES-ON
Google	X-MS-TOKEN-GOOGLE-ID-TOKEN X-MS-TOKEN-GOOGLE-ACCESS-TOKEN X-MS-TOKEN-GOOGLE-EXPIRES-ON X-MS-TOKEN-GOOGLE-REFRESH-TOKEN
Twitter	X-MS-TOKEN-TWITTER-ACCESS-TOKEN X-MS-TOKEN-TWITTER-ACCESS-TOKEN-SECRET

ⓘ Note

Different language frameworks may present these headers to the app code in different formats, such as lowercase or title case.

From your client code (such as a mobile app or in-browser JavaScript), send an HTTP `GET` request to `/auth/me` ([token store](#) must be enabled). The returned JSON has the provider-specific tokens.

ⓘ Note

Access tokens are for accessing provider resources, so they are present only if you configure your provider with a client secret. To see how to get refresh tokens, see Refresh access tokens.

Refresh auth tokens

When your provider's access token (not the [session token](#)) expires, you need to reauthenticate the user before you use that token again. You can avoid token expiration by making a `GET` call to the `/auth/refresh` endpoint of your application. When called, App Service automatically refreshes the access tokens in the [token store](#) for the authenticated user. Subsequent requests for tokens by your app code get the refreshed tokens. However, for token refresh to work, the token store must contain [refresh tokens](#) for your provider. The way to get refresh tokens are documented by each provider, but the following list is a brief summary:

- **Google:** Append an `access_type=offline` query string parameter to your `/auth/login/google` API call. For more information, see [Google Refresh Tokens](#).
- **Facebook:** Doesn't provide refresh tokens. Long-lived tokens expire in 60 days (see [Facebook Expiration and Extension of Access Tokens](#)).
- **Twitter:** Access tokens don't expire (see [Twitter OAuth FAQ](#)).
- **Microsoft:** In <https://resources.azure.com>, do the following steps:
 1. At the top of the page, select **Read/Write**.
 2. In the left browser, navigate to `subscriptions > <subscription_name> > resourceGroups > <resource_group_name> > providers > Microsoft.Web > sites > <app_name> > config > authsettingsV2`.
 3. Click **Edit**.
 4. Modify the following property.

JSON

```
"identityProviders": {  
    "azureActiveDirectory": {  
        "login": {  
            "loginParameters": ["scope=openid profile email  
offline_access"]  
        }  
    }  
}
```

5. Click Put.

ⓘ Note

The scope that gives you a refresh token is `offline_access`. See how it's used in [Tutorial: Authenticate and authorize users end-to-end in Azure App Service](#). The other scopes are requested by default by App Service already. For information on these default scopes, see [OpenID Connect Scopes](#).

Once your provider is configured, you can [find the refresh token and the expiration time for the access token](#) in the token store.

To refresh your access token at any time, just call `/.auth/refresh` in any language. The following snippet uses jQuery to refresh your access tokens from a JavaScript client.

JavaScript

```
function refreshTokens() {
  let refreshUrl = "/.auth/refresh";
  $.ajax(refreshUrl) .done(function() {
    console.log("Token refresh completed successfully.");
  }) .fail(function() {
    console.log("Token refresh failed. See application logs for details.");
  });
}
```

If a user revokes the permissions granted to your app, your call to `/.auth/me` may fail with a `403 Forbidden` response. To diagnose errors, check your application logs for details.

Extend session token expiration grace period

The authenticated session expires after 8 hours. After an authenticated session expires, there is a 72-hour grace period by default. Within this grace period, you're allowed to refresh the session token with App Service without reauthenticating the user. You can just call `/.auth/refresh` when your session token becomes invalid, and you don't need to track token expiration yourself. Once the 72-hour grace period is lapses, the user must sign in again to get a valid session token.

If 72 hours isn't enough time for you, you can extend this expiration window. Extending the expiration over a long period could have significant security implications (such as

when an authentication token is leaked or stolen). So you should leave it at the default 72 hours or set the extension period to the smallest value.

To extend the default expiration window, run the following command in the [Cloud Shell](#).

Azure CLI

```
az webapp auth update --resource-group <group_name> --name <app_name> --  
token-refresh-extension-hours <hours>
```

ⓘ Note

The grace period only applies to the App Service authenticated session, not the tokens from the identity providers. There is no grace period for the expired provider tokens.

Next steps

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

Manage the API and runtime versions of App Service authentication

Article • 10/12/2023

This article shows you how to customize the API and runtime versions of the built-in [authentication and authorization in App Service](#).

There are two versions of the management API for App Service authentication. The V2 version is required for the "Authentication" experience in the Azure portal. An app already using the V1 API can upgrade to the V2 version once a few changes have been made. Specifically, secret configuration must be moved to slot-sticky application settings. This can be done automatically from the "Authentication" section of the portal for your app.

Update the configuration version

Warning

Migration to V2 will disable management of the App Service Authentication/Authorization feature for your application through some clients, such as its existing experience in the Azure portal, Azure CLI, and Azure PowerShell. This cannot be reversed.

The V2 API doesn't support creation or editing of Microsoft Account as a distinct provider as was done in V1. Rather, it uses the converged [Microsoft identity platform](#) to sign-in users with both Microsoft Entra ID and personal Microsoft accounts. When switching to the V2 API, the V1 Microsoft Entra configuration is used to configure the Microsoft identity platform provider. The V1 Microsoft Account provider will be carried forward in the migration process and continue to operate as normal, but you should move to the newer Microsoft identity platform model. See [Support for Microsoft Account provider registrations](#) to learn more.

The automated migration process will move provider secrets into application settings and then convert the rest of the configuration into the new format. To use the automatic migration:

1. Navigate to your app in the portal and select the **Authentication** menu option.
2. If the app is configured using the V1 model, you'll see an **Upgrade** button.

3. Review the description in the confirmation prompt. If you're ready to perform the migration, select **Upgrade** in the prompt.

Manually managing the migration

The following steps will allow you to manually migrate the application to the V2 API if you don't wish to use the automatic version mentioned above.

Moving secrets to application settings

1. Get your existing configuration by using the V1 API:

Azure CLI

```
az webapp auth show -g <group_name> -n <site_name>
```

In the resulting JSON payload, make note of the secret value used for each provider you've configured:

- Microsoft Entra ID: `clientSecret`
- Google: `googleClientSecret`
- Facebook: `facebookAppSecret`
- Twitter: `twitterConsumerSecret`
- Microsoft Account: `microsoftAccountClientSecret`

 **Important**

The secret values are important security credentials and should be handled carefully. Do not share these values or persist them on a local machine.

2. Create slot-sticky application settings for each secret value. You may choose the name of each application setting. Its value should match what you obtained in the previous step or [reference a Key Vault secret](#) that you've created with that value.

To create the setting, you can use the Azure portal or run a variation of the following for each provider:

Azure CLI

```
# For Web Apps, Google example
az webapp config appsettings set -g <group_name> -n <site_name> --slot-
settings GOOGLE_PROVIDER_AUTHENTICATION_SECRET=
```

```
<value_from_previous_step>

# For Azure Functions, Twitter example
az functionapp config appsettings set -g <group_name> -n <site_name> --
slot-settings TWITTER_PROVIDER_AUTHENTICATION_SECRET=
<value_from_previous_step>
```

ⓘ Note

The application settings for this configuration should be marked as slot-sticky, meaning that they will not move between environments during a **slot swap** operation. This is because your authentication configuration itself is tied to the environment.

3. Create a new JSON file named `authsettings.json`. Take the output that you received previously and remove each secret value from it. Write the remaining output to the file, making sure that no secret is included. In some cases, the configuration may have arrays containing empty strings. Make sure that `microsoftAccountOAuthScopes` doesn't, and if it does, switch that value to `null`.
4. Add a property to `authsettings.json` that points to the application setting name you created earlier for each provider:

- Microsoft Entra ID: `clientSecretSettingName`
- Google: `googleClientSecretSettingName`
- Facebook: `facebookAppSecretSettingName`
- Twitter: `twitterConsumerSecretSettingName`
- Microsoft Account: `microsoftAccountClientSecretSettingName`

An example file after this operation might look similar to the following, in this case only configured for Microsoft Entra ID:

JSON

```
{
  "id": "/subscriptions/00d563f8-5b89-4c6a-bcec-
c1b9f6d607e0/resourceGroups/myresourcegroup/providers/Microsoft.Web/sites/mywebapp/config/authsettings",
  "name": "authsettings",
  "type": "Microsoft.Web/sites/config",
  "location": "Central US",
  "properties": {
    "enabled": true,
    "runtimeVersion": "~1",
    "unauthenticatedClientAction": "AllowAnonymous",
```

```

        "tokenStoreEnabled": true,
        "allowedExternalRedirectUrls": null,
        "defaultProvider": "AzureActiveDirectory",
        "clientId": "3197c8ed-2470-480a-8fae-58c25558ac9b",
        "clientSecret": "",
        "clientSecretSettingName": "MICROSOFT_IDENTITY_AUTHENTICATION_SECRET",
        "clientSecretCertificateThumbprint": null,
        "issuer": "https://sts.windows.net/0b2ef922-672a-4707-9643-9a5726eec524/",
        "allowedAudiences": [
            "https://mywebapp.azurewebsites.net"
        ],
        "additionalLoginParams": null,
        "isAadAutoProvisioned": true,
        "aadClaimsAuthorization": null,
        "googleClientId": null,
        "googleClientSecret": null,
        "googleClientSecretSettingName": null,
        "googleOAuthScopes": null,
        "facebookAppId": null,
        "facebookAppSecret": null,
        "facebookAppSecretSettingName": null,
        "facebookOAuthScopes": null,
        "gitHubClientId": null,
        "gitHubClientSecret": null,
        "gitHubClientSecretSettingName": null,
        "gitHubOAuthScopes": null,
        "twitterConsumerKey": null,
        "twitterConsumerSecret": null,
        "twitterConsumerSecretSettingName": null,
        "microsoftAccountClientId": null,
        "microsoftAccountClientSecret": null,
        "microsoftAccountClientSecretSettingName": null,
        "microsoftAccountOAuthScopes": null,
        "isAuthFromFile": "false"
    }
}

```

5. Submit this file as the new Authentication/Authorization configuration for your app:

Azure CLI

```

az rest --method PUT --url
"/subscriptions/<subscription_id>/resourceGroups/<group_name>/providers
/Microsoft.Web/sites/<site_name>/config/authsettings?api-version=2020-
06-01" --body @./authsettings.json

```

6. Validate that your app is still operating as expected after this gesture.

7. Delete the file used in the previous steps.

You've now migrated the app to store identity provider secrets as application settings.

Support for Microsoft Account provider registrations

If your existing configuration contains a Microsoft Account provider and doesn't contain a Microsoft Entra provider, you can switch the configuration over to the Microsoft Entra provider and then perform the migration. To do this:

1. Go to [App registrations](#) in the Azure portal and find the registration associated with your Microsoft Account provider. It may be under the "Applications from personal account" heading.
2. Navigate to the "Authentication" page for the registration. Under "Redirect URLs", you should see an entry ending in `/auth/login/microsoftaccount/callback`. Copy this URI.
3. Add a new URI that matches the one you just copied, except instead have it end in `/auth/login/aad/callback`. This will allow the registration to be used by the App Service Authentication / Authorization configuration.
4. Navigate to the App Service Authentication / Authorization configuration for your app.
5. Collect the configuration for the Microsoft Account provider.
6. Configure the Microsoft Entra provider using the "Advanced" management mode, supplying the client ID and client secret values you collected in the previous step. For the Issuer URL, use `<authentication-endpoint>/<tenant-id>/v2.0`, and replace `<authentication-endpoint>` with the [authentication endpoint for your cloud environment](#) (e.g., "https://login.microsoftonline.com" for global Azure), also replacing `<tenant-id>` with your **Directory (tenant) ID**.
7. Once you've saved the configuration, test the login flow by navigating in your browser to the `/auth/login/aad` endpoint on your site and complete the sign-in flow.
8. At this point, you've successfully copied the configuration over, but the existing Microsoft Account provider configuration remains. Before you remove it, make sure that all parts of your app reference the Microsoft Entra provider through login links, etc. Verify that all parts of your app work as expected.
9. Once you've validated that things work against the Microsoft Entra provider, you may remove the Microsoft Account provider configuration.

Warning

It is possible to converge the two registrations by modifying the **supported account types** for the Microsoft Entra app registration. However, this would force a new consent prompt for Microsoft Account users, and those users' identity claims

may be different in structure, notably changing values since a new App ID is being used. This approach is not recommended unless thoroughly understood. You should instead wait for support for the two registrations in the V2 API surface.

Switching to V2

Once the above steps have been performed, navigate to the app in the Azure portal. Select the "Authentication (preview)" section.

Alternatively, you may make a PUT request against the `config/authsettingsv2` resource under the site resource. The schema for the payload is the same as captured in [File-based configuration](#).

Pin your app to a specific authentication runtime version

When you enable authentication/authorization, platform middleware is injected into your HTTP request pipeline as described in the [feature overview](#). This platform middleware is periodically updated with new features and improvements as part of routine platform updates. By default, your web or function app will run on the latest version of this platform middleware. These automatic updates are always backwards compatible. However, in the rare event that this automatic update introduces a runtime issue for your web or function app, you can temporarily roll back to the previous middleware version. This article explains how to temporarily pin an app to a specific version of the authentication middleware.

Automatic and manual version updates

You can pin your app to a specific version of the platform middleware by setting a `runtimeVersion` setting for the app. Your app always runs on the latest version unless you choose to explicitly pin it back to a specific version. There will be a few versions supported at a time. If you pin to an invalid version that is no longer supported, your app will use the latest version instead. To always run the latest version, set `runtimeVersion` to `~1`.

View and update the current runtime version

You can change the runtime version used by your app. The new runtime version should take effect after restarting the app.

View the current runtime version

You can view the current version of the platform authentication middleware either using the Azure CLI or via one of the built-in version HTTP endpoints in your app.

From the Azure CLI

Using the Azure CLI, view the current middleware version with the [az webapp auth show](#) command.

Azure CLI

```
az webapp auth show --name <my_app_name> \
--resource-group <my_resource_group>
```

In this code, replace `<my_app_name>` with the name of your app. Also replace `<my_resource_group>` with the name of the resource group for your app.

You'll see the `runtimeVersion` field in the CLI output. It will resemble the following example output, which has been truncated for clarity:

Output

```
{
  "additionalLoginParams": null,
  "allowedAudiences": null,
  ...
  "runtimeVersion": "1.3.2",
  ...
}
```

From the version endpoint

You can also hit `/auth/version` endpoint on an app also to view the current middleware version that the app is running on. It will resemble the following example output:

Output

```
{
  "version": "1.3.2"
}
```

Update the current runtime version

Using the Azure CLI, you can update the `runtimeVersion` setting in the app with the [az webapp auth update](#) command.

Azure CLI

```
az webapp auth update --name <my_app_name> \
--resource-group <my_resource_group> \
--runtime-version <version>
```

Replace `<my_app_name>` with the name of your app. Also replace `<my_resource_group>` with the name of the resource group for your app. Also, replace `<version>` with a valid version of the 1.x runtime or `~1` for the latest version. See the [release notes on the different runtime versions](#) to help determine the version to pin to.

You can run this command from the [Azure Cloud Shell](#) by choosing **Try it** in the preceding code sample. You can also use the [Azure CLI locally](#) to execute this command after executing `az login` to sign in.

Next steps

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

File-based configuration in Azure App Service authentication

Article • 02/02/2022

With [App Service authentication](#), the authentication settings can be configured with a file. You may need to use file-based configuration to use certain preview capabilities of App Service authentication / authorization before they are exposed via [Azure Resource Manager APIs](#).

ⓘ Important

Remember that your app payload, and therefore this file, may move between environments, as with [slots](#). It is likely you would want a different app registration pinned to each slot, and in these cases, you should continue to use the standard configuration method instead of using the configuration file.

Enabling file-based configuration

1. Create a new JSON file for your configuration at the root of your project (deployed to D:\home\site\wwwroot in your web / function app). Fill in your desired configuration according to the [file-based configuration reference](#). If modifying an existing Azure Resource Manager configuration, make sure to translate the properties captured in the `authsettings` collection into your configuration file.
2. Modify the existing configuration, which is captured in the [Azure Resource Manager APIs](#) under `Microsoft.Web/sites/<siteName>/config/authsettingsV2`. To modify this, you can use an [Azure Resource Manager template](#) or a tool like [Azure Resource Explorer](#). Within the authsettingsV2 collection, you will need to set two properties (and may remove others):
 - a. Set `platform.enabled` to "true"
 - b. Set `platform.configFilePath` to the name of the file (for example, "auth.json")

ⓘ Note

The format for `platform.configFilePath` varies between platforms. On Windows, both relative and absolute paths are supported. Relative is recommended. For Linux, only absolute paths are supported currently, so the value of the setting should be "/home/site/wwwroot/auth.json" or similar.

Once you have made this configuration update, the contents of the file will be used to define the behavior of App Service Authentication / Authorization for that site. If you ever wish to return to Azure Resource Manager configuration, you can do so by removing changing the setting `platform.configFilePath` to null.

Configuration file reference

Any secrets that will be referenced from your configuration file must be stored as [application settings](#). You may name the settings anything you wish. Just make sure that the references from the configuration file uses the same keys.

The following exhausts possible configuration options within the file:

```
JSON

{
  "platform": {
    "enabled": <true|false>
  },
  "globalValidation": {
    "unauthenticatedClientAction": "RedirectLoginPage|AllowAnonymous|RejectWith401|RejectWith404",
    "redirectToProvider": "<default provider alias>",
    "excludedPaths": [
      "/path1",
      "/path2",
      "/path3/subpath/*"
    ]
  },
  "httpSettings": {
    "requireHttps": <true|false>,
    "routes": {
      "apiPrefix": "<api prefix>"
    },
    "forwardProxy": {
      "convention": "NoProxy|Standard|Custom",
      "customHostHeaderName": "<host header value>",
      "customProtoHeaderName": "<proto header value>"
    }
  },
  "login": {
    "routes": {
      "logoutEndpoint": "<logout endpoint>"
    },
    "tokenStore": {
      "enabled": <true|false>,
      "tokenRefreshExtensionHours": "<double>",
      "fileSystem": {
        "directory": "<directory to store the tokens in if using a"
      }
    }
  }
}
```

```
file system token store (default)>"  
    },  
    "azureBlobStorage": {  
        "sasUrlSettingName": "<app setting name containing the sas  
url for the Azure Blob Storage if opting to use that for a token store>"  
    }  
,  
    "preserveUrlFragmentsForLogins": <true|false>,  
    "allowedExternalRedirectUrls": [  
        "https://uri1.azurewebsites.net/",  
        "https://uri2.azurewebsites.net/",  
        "url_scheme_of_your_app://easyauth.callback"  
    ],  
    "cookieExpiration": {  
        "convention": "FixedTime|IdentityDerived",  
        "timeToExpiration": "<timespan>"  
    },  
    "nonce": {  
        "validateNonce": <true|false>,  
        "nonceExpirationInterval": "<timespan>"  
    }  
,  
    "identityProviders": {  
        "azureActiveDirectory": {  
            "enabled": <true|false>,  
            "registration": {  
                "openIdIssuer": "<issuer url>",  
                "clientId": "<app id>",  
                "clientSecretSettingName":  
"APP_SETTING_CONTAINING_AAD_SECRET",  
            },  
            "login": {  
                "loginParameters": [  
                    "paramName1=value1",  
                    "paramName2=value2"  
                ]  
            },  
            "validation": {  
                "allowedAudiences": [  
                    "audience1",  
                    "audience2"  
                ]  
            }  
        },  
        "facebook": {  
            "enabled": <true|false>,  
            "registration": {  
                "appId": "<app id>",  
                "appSecretSettingName":  
"APP_SETTING_CONTAINING_FACEBOOK_SECRET"  
            },  
            "graphApiVersion": "v3.3",  
            "login": {  
                "scopes": [  
                    "public_profile",  
                    "email"  
                ]  
            }  
        }  
    }  
},  
"logLevel": "Information",  
"logFile": "D:\inetpub\logs\LogFiles\EasyAuthLog",  
"logFormat": "W3C",  
"logMaxSize": 10000000,  
"logMaxDays": 30,  
"logFileRetentionCount": 30,  
"logFileBackupCount": 5,  
"logFileCompress": true
```

```
        "email"
    ]
},
},
"gitHub": {
    "enabled": <true|false>,
    "registration": {
        "clientId": "<client id>",
        "clientSecretSettingName": "APP_SETTING_CONTAINING_GITHUB_SECRET"
    },
    "login": {
        "scopes": [
            "profile",
            "email"
        ]
    }
},
"google": {
    "enabled": true,
    "registration": {
        "clientId": "<client id>",
        "clientSecretSettingName": "APP_SETTING_CONTAINING_GOOGLE_SECRET"
    },
    "login": {
        "scopes": [
            "profile",
            "email"
        ]
    },
    "validation": {
        "allowedAudiences": [
            "audience1",
            "audience2"
        ]
    }
},
"twitter": {
    "enabled": <true|false>,
    "registration": {
        "consumerKey": "<consumer key>",
        "consumerSecretSettingName": "APP_SETTING_CONTAINING_TWITTER_CONSUMER_SECRET"
    }
},
"apple": {
    "enabled": <true|false>,
    "registration": {
        "clientId": "<client id>",
        "clientSecretSettingName": "APP_SETTING_CONTAINING_APPLE_SECRET"
    },
    "login": {
        "scopes": [

```

```

        "profile",
        "email"
    ]
},
},
"openIdConnectProviders": {
    "<providerName>": {
        "enabled": <true|false>,
        "registration": {
            "clientId": "<client id>",
            "clientCredential": {
                "clientSecretSettingName": "<name of app setting
containing client secret>"
            },
            "openIdConnectConfiguration": {
                "authorizationEndpoint": "<url specifying
authorization endpoint>",
                "tokenEndpoint": "<url specifying token endpoint>",
                "issuer": "<url specifying issuer>",
                "certificationUri": "<url specifying jwks
endpoint>",
                "wellKnownOpenIdConfiguration": "<url specifying
.well-known/open-id-configuration endpoint - if this property is set, the
other properties of this object are ignored, and authorizationEndpoint,
tokenEndpoint, issuer, and certificationUri are set to the corresponding
values listed at this endpoint>"
            }
        },
        "login": {
            "nameClaimType": "<name of claim containing name>",
            "scopes": [
                "openid",
                "profile",
                "email"
            ],
            "loginParameterNames": [
                "paramName1=value1",
                "paramName2=value2"
            ],
            ...
        }
    },
    //...
}
}

```

More resources

- Tutorial: Authenticate and authorize users end-to-end
- Environment variables and app settings for authentication

Security in Azure App Service

Article • 06/15/2023

This article shows you how [Azure App Service](#) helps secure your web app, mobile app back end, API app, and [function app](#). It also shows how you can further secure your app with the built-in App Service features.

The platform components of App Service, including Azure VMs, storage, network connections, web frameworks, management and integration features, are actively secured and hardened. App Service goes through vigorous compliance checks on a continuous basis to make sure that:

- Your app resources are [secured](#) from the other customers' Azure resources.
- [VM instances and runtime software are regularly updated](#) to address newly discovered vulnerabilities.
- Communication of secrets (such as connection strings) between your app and other Azure resources (such as [SQL Database](#)) stays within Azure and doesn't cross any network boundaries. Secrets are always encrypted when stored.
- All communication over the App Service connectivity features, such as [hybrid connection](#), is encrypted.
- Connections with remote management tools like Azure PowerShell, Azure CLI, Azure SDKs, REST APIs, are all encrypted.
- 24-hour threat management protects the infrastructure and platform against malware, distributed denial-of-service (DDoS), man-in-the-middle (MITM), and other threats.

For more information on infrastructure and platform security in Azure, see [Azure Trust Center](#).

The following sections show you how to further protect your App Service app from threats.

HTTPS and Certificates

App Service lets you secure your apps with [HTTPS](#). When your app is created, its default domain name (<app_name>.azurewebsites.net) is already accessible using HTTPS. If you [configure a custom domain for your app](#), you should also [secure it with a TLS/SSL certificate](#) so that client browsers can make secured HTTPS connections to your custom domain. There are several types of certificates supported by App Service:

- Free App Service Managed Certificate

- App Service certificate
- Third-party certificate
- Certificate imported from Azure Key Vault

For more information, see [Add a TLS/SSL certificate in Azure App Service](#).

Insecure protocols (HTTP, TLS 1.0, FTP)

To secure your app against all unencrypted (HTTP) connections, App Service provides one-click configuration to enforce HTTPS. Unsecured requests are turned away before they even reach your application code. For more information, see [Enforce HTTPS](#).

[TLS ↗](#) 1.0 is no longer considered secure by industry standards, such as [PCI DSS ↗](#). App Service lets you disable outdated protocols by [enforcing TLS 1.1/1.2](#).

App Service supports both FTP and FTPS for deploying your files. However, FTPS should be used instead of FTP, if at all possible. When one or both of these protocols are not in use, you should [disable them](#).

Static IP restrictions

By default, your App Service app accepts requests from all IP addresses from the internet, but you can limit that access to a small subset of IP addresses. App Service on Windows lets you define a list of IP addresses that are allowed to access your app. The allowed list can include individual IP addresses or a range of IP addresses defined by a subnet mask. For more information, see [Azure App Service Static IP Restrictions](#).

For App Service on Windows, you can also restrict IP addresses dynamically by configuring the `web.config`. For more information, see [Dynamic IP Security <dynamicIpSecurity>](#).

Client authentication and authorization

Azure App Service provides turn-key authentication and authorization of users or client apps. When enabled, it can sign in users and client apps with little or no application code. You may implement your own authentication and authorization solution, or allow App Service to handle it for you instead. The authentication and authorization module handles web requests before handing them off to your application code, and it denies unauthorized requests before they reach your code.

App Service authentication and authorization support multiple authentication providers, including Azure Active Directory, Microsoft accounts, Facebook, Google, and Twitter. For more information, see [Authentication and authorization in Azure App Service](#).

Service-to-service authentication

When authenticating against a back-end service, App Service provides two different mechanisms depending on your need:

- **Service identity** - Sign in to the remote resource using the identity of the app itself. App Service lets you easily create a [managed identity](#), which you can use to authenticate with other services, such as [Azure SQL Database](#) or [Azure Key Vault](#). For an end-to-end tutorial of this approach, see [Secure Azure SQL Database connection from App Service using a managed identity](#).
- **On-behalf-of (OBO)** - Make delegated access to remote resources on behalf of the user. With Azure Active Directory as the authentication provider, your App Service app can perform delegated sign-in to a remote service, such as [Microsoft Graph](#) or a remote API app in App Service. For an end-to-end tutorial of this approach, see [Authenticate and authorize users end-to-end in Azure App Service](#).

Connectivity to remote resources

There are three types of remote resources your app may need to access:

- [Azure resources](#)
- [Resources inside an Azure Virtual Network](#)
- [On-premises resources](#)

In each of these cases, App Service provides a way for you to make secure connections, but you should still observe security best practices. For example, always use encrypted connections even if the back-end resource allows unencrypted connections.

Furthermore, make sure that your back-end Azure service allows the minimum set of IP addresses. You can find the outbound IP addresses for your app at [Inbound and outbound IP addresses in Azure App Service](#).

Azure resources

When your app connects to Azure resources, such as [SQL Database](#) and [Azure Storage](#), the connection stays within Azure and doesn't cross any network boundaries. However, the connection goes through the shared networking in Azure, so always make sure that your connection is encrypted.

If your app is hosted in an [App Service environment](#), you should [connect to supported Azure services using Virtual Network service endpoints](#).

Resources inside an Azure Virtual Network

Your app can access resources in an [Azure Virtual Network](#) through [Virtual Network integration](#). The integration is established with a Virtual Network using a point-to-site VPN. The app can then access the resources in the Virtual Network using their private IP addresses. The point-to-site connection, however, still traverses the shared networks in Azure.

To isolate your resource connectivity completely from the shared networks in Azure, create your app in an [App Service environment](#). Since an App Service environment is always deployed to a dedicated Virtual Network, connectivity between your app and resources within the Virtual Network is fully isolated. For other aspects of network security in an App Service environment, see [Network isolation](#).

On-premises resources

You can securely access on-premises resources, such as databases, in three ways:

- [Hybrid connections](#) - Establishes a point-to-point connection to your remote resource through a TCP tunnel. The TCP tunnel is established using TLS 1.2 with shared access signature (SAS) keys.
- [Virtual Network integration](#) with site-to-site VPN - As described in [Resources inside an Azure Virtual Network](#), but the Virtual Network can be connected to your on-premises network through a [site-to-site VPN](#). In this network topology, your app can connect to on-premises resources like other resources in the Virtual Network.
- [App Service environment](#) with site-to-site VPN - As described in [Resources inside an Azure Virtual Network](#), but the Virtual Network can be connected to your on-premises network through a [site-to-site VPN](#). In this network topology, your app can connect to on-premises resources like other resources in the Virtual Network.

Application secrets

Don't store application secrets, such as database credentials, API tokens, and private keys in your code or configuration files. The commonly accepted approach is to access them as [environment variables](#) ↗ using the standard pattern in your language of choice. In App Service, the way to define environment variables is through [app settings](#) (and, especially for .NET applications, [connection strings](#)). App settings and connection strings

are stored encrypted in Azure, and they're decrypted only before being injected into your app's process memory when the app starts. The encryption keys are rotated regularly.

Alternatively, you can integrate your App Service app with [Azure Key Vault](#) for advanced secrets management. By [accessing the Key Vault with a managed identity](#), your App Service app can securely access the secrets you need.

Network isolation

Except for the **Isolated** pricing tier, all tiers run your apps on the shared network infrastructure in App Service. For example, the public IP addresses and front-end load balancers are shared with other tenants. The **Isolated** tier gives you complete network isolation by running your apps inside a dedicated [App Service environment](#). An App Service environment runs in your own instance of [Azure Virtual Network](#). It lets you:

- Serve your apps through a dedicated public endpoint, with dedicated front ends.
- Serve internal application using an internal load balancer (ILB), which allows access only from inside your Azure Virtual Network. The ILB has an IP address from your private subnet, which provides total isolation of your apps from the internet.
- [Use an ILB behind a web application firewall \(WAF\)](#). The WAF offers enterprise-level protection to your public-facing applications, such as DDoS protection, URI filtering, and SQL injection prevention.

DDoS protection

For web workloads, we highly recommend utilizing [Azure DDoS protection](#) and a [web application firewall](#) to safeguard against emerging DDoS attacks. Another option is to deploy [Azure Front Door](#) along with a web application firewall. Azure Front Door offers platform-level [protection against network-level DDoS attacks](#).

For more information, see [Introduction to Azure App Service Environments](#).

Tutorial: Create a highly available multi-region app in Azure App Service

Article • 03/16/2023

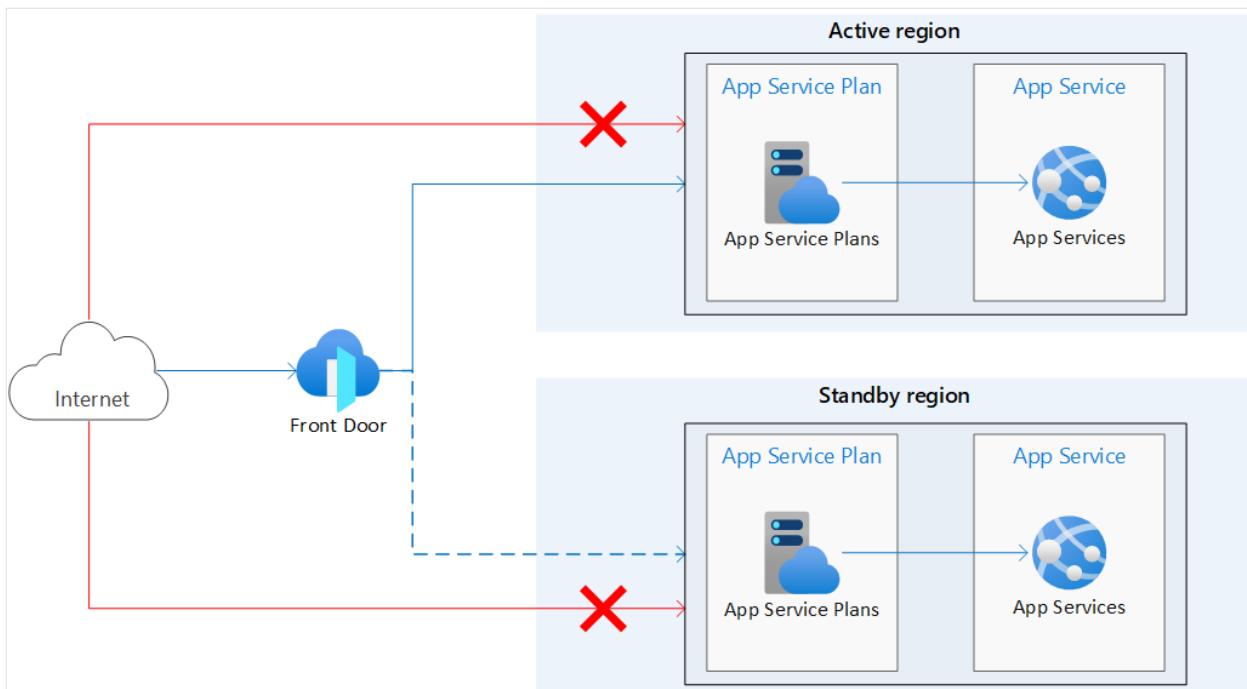
High availability and fault tolerance are key components of a well-architected solution. It's best to prepare for the unexpected by having an emergency plan that can shorten downtime and keep your systems up and running automatically when something fails.

When you deploy your application to the cloud, you choose a region in that cloud where your application infrastructure is based. If your application is deployed to a single region, and the region becomes unavailable, your application will also be unavailable. This lack of availability may be unacceptable under the terms of your application's SLA. If so, deploying your application and its services across multiple regions is a good solution.

In this tutorial, you'll learn how to deploy a highly available multi-region web app. This scenario will be kept simple by restricting the application components to just a web app and [Azure Front Door](#), but the concepts can be expanded and applied to other infrastructure patterns. For example, if your application connects to an Azure database offering or storage account, see [active geo-replication for SQL databases](#) and [redundancy options for storage accounts](#). For a reference architecture for a more detailed scenario, see [Highly available multi-region web application](#).

The following architecture diagram shows the infrastructure you'll be creating during this tutorial. It consists of two identical App Services in separate regions, one being the active or primary region, and the other is the standby or secondary region. Azure Front Door is used to route traffic to the App Services and access restrictions are configured so that direct access to the apps from the internet is blocked. The dotted line indicates that traffic will only be sent to the standby region if the active region goes down.

Azure provides various options for load balancing and traffic routing. Azure Front Door was selected for this use case because it involves internet facing web apps hosted on Azure App Service deployed in multiple regions. To help you decide what to use for your use case if it differs from this tutorial, see the [decision tree for load balancing in Azure](#).



With this architecture:

- Identical App Service apps are deployed in two separate regions.
- Public traffic directly to the App Service apps is blocked.
- Azure Front Door is used route traffic to the primary/active region. The secondary region has an App Service that's up and running and ready to serve traffic if needed.

What you'll learn:

- ✓ Create identical App Services in separate regions.
- ✓ Create Azure Front Door with access restrictions that block public access to the App Services.

Prerequisites

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

To complete this tutorial:

- Use the Bash environment in [Azure Cloud Shell](#). For more information, see [Quickstart for Bash in Azure Cloud Shell](#).
- [Launch Cloud Shell](#)
- If you prefer to run CLI reference commands locally, [install](#) the Azure CLI. If you're running on Windows or macOS, consider running Azure CLI in a Docker container.

For more information, see [How to run the Azure CLI in a Docker container](#).

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

Create two instances of a web app

You'll need two instances of a web app that run in different Azure regions for this tutorial. You'll use the [region pair](#) East US/West US as your two regions and create two empty web apps. Feel free to choose your own regions if needed.

To make management and clean-up simpler, you'll use a single resource group for all resources in this tutorial. Consider using separate resource groups for each region/resource to further isolate your resources in a disaster recovery situation.

Run the following command to create your resource group.

```
Azure CLI
```

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

Create App Service plans

Run the following commands to create the App Service plans. Replace the placeholders for `<app-service-plan-east-us>` and `<app-service-plan-west-us>` with two unique names where you can easily identify the region they're in.

```
Azure CLI
```

```
az appservice plan create --name <app-service-plan-east-us> --resource-group  
myresourcegroup --is-linux --location eastus  
  
az appservice plan create --name <app-service-plan-west-us> --resource-group  
myresourcegroup --is-linux --location westus
```

Create web apps

Once the App Service plans are created, run the following commands to create the web apps. Replace the placeholders for `<web-app-east-us>` and `<web-app-west-us>` with two globally unique names (valid characters are `a-z`, `0-9`, and `-`) and be sure to pay attention to the `--plan` parameter so that you place one app in each plan (and therefore in each region). Replace the `<runtime>` parameter with the language version of your app. Run `az webapp list-runtimes` for the list of available runtimes. If you plan on using the sample Node.js app given in this tutorial in the following sections, use "NODE:18-Its" as your runtime.

Azure CLI

```
az webapp create --name <web-app-east-us> --resource-group myresourcegroup --plan <app-service-plan-east-us> --runtime <runtime>

az webapp create --name <web-app-west-us> --resource-group myresourcegroup --plan <app-service-plan-west-us> --runtime <runtime>
```

Make note of the default hostname of each web app so you can define the backend addresses when you deploy the Front Door in the next step. It should be in the format `<web-app-name>.azurewebsites.net`. These hostnames can be found by running the following command or by navigating to the app's [Overview page](#) in the [Azure portal](#).

Azure CLI

```
az webapp show --name <web-app-name> --resource-group myresourcegroup --query "hostNames"
```

Create an Azure Front Door

A multi-region deployment can use an active-active or active-passive configuration. An active-active configuration distributes requests across multiple active regions. An active-passive configuration keeps running instances in the secondary region, but doesn't send traffic there unless the primary region fails. Azure Front Door has a built-in feature that allows you to enable these configurations. For more information on designing apps for high availability and fault tolerance, see [Architect Azure applications for resiliency and availability](#).

Create an Azure Front Door profile

You'll now create an [Azure Front Door Premium](#) to route traffic to your apps.

Run `az afd profile create` to create an Azure Front Door profile.

ⓘ Note

If you want to deploy Azure Front Door Standard instead of Premium, substitute the value of the `--sku` parameter with `Standard_AzureFrontDoor`. You won't be able to deploy managed rules with WAF Policy if you choose the Standard SKU. For a detailed comparison of the SKUs, see [Azure Front Door tier comparison](#).

Azure CLI

```
az afd profile create --profile-name myfrontdoorprofile --resource-group  
myresourcegroup --sku Premium_AzureFrontDoor
```

Parameter	Value	Description
profile-name	myfrontdoorprofile	Name for the Azure Front Door profile, which is unique within the resource group.
resource-group	myresourcegroup	The resource group that contains the resources from this tutorial.
sku	Premium_AzureFrontDoor	The pricing tier of the Azure Front Door profile.

Add an endpoint

Run `az afd endpoint create` to create an endpoint in your profile. You can create multiple endpoints in your profile after finishing the create experience.

Azure CLI

```
az afd endpoint create --resource-group myresourcegroup --endpoint-name  
myendpoint --profile-name myfrontdoorprofile --enabled-state Enabled
```

Parameter	Value	Description
endpoint-name	myendpoint	Name of the endpoint under the profile, which is unique globally.
enabled-state	Enabled	Whether to enable this endpoint.

Create an origin group

Run `az afd origin-group create` to create an origin group that contains your two web apps.

Azure CLI

```
az afd origin-group create --resource-group myresourcegroup --origin-group-name myorigingroup --profile-name myfrontdoorprofile --probe-request-type GET --probe-protocol Http --probe-interval-in-seconds 60 --probe-path / --sample-size 4 --successful-samples-required 3 --additional-latency-in-milliseconds 50
```

Parameter	Value	Description
origin-group-name	myorigingroup	Name of the origin group.
probe-request-type	GET	The type of health probe request that is made.
probe-protocol	Http	Protocol to use for health probe.
probe-interval-in-seconds	60	The number of seconds between health probes.
probe-path	/	The path relative to the origin that is used to determine the health of the origin.
sample-size	4	The number of samples to consider for load balancing decisions.
successful-samples-required	3	The number of samples within the sample period that must succeed.
additional-latency-in-milliseconds	50	The additional latency in milliseconds for probes to fall into the lowest latency bucket.

Add an origin to the group

Run [az afd origin create](#) to add an origin to your origin group. For the `--host-name` parameter, replace the placeholder for `<web-app-east-us>` with your app name in that region. Notice the `--priority` parameter is set to "1", which indicates all traffic will be sent to your primary app.

Azure CLI

```
az afd origin create --resource-group myresourcegroup --host-name <web-app-east-us>.azurewebsites.net --profile-name myfrontdoorprofile --origin-group-name myorigingroup --origin-name primaryapp --origin-host-header <web-app-east-us>.azurewebsites.net --priority 1 --weight 1000 --enabled-state Enabled --http-port 80 --https-port 443
```

Parameter	Value	Description
-----------	-------	-------------

Parameter	Value	Description
host-name	<web-app-east-us>.azurewebsites.net	The hostname of the primary web app.
origin-name	primaryapp	Name of the origin.
origin-host-header	<web-app-east-us>.azurewebsites.net	The host header to send for requests to this origin. If you leave this blank, the request hostname determines this value. Azure CDN origins, such as Web Apps, Blob Storage, and Cloud Services require this host header value to match the origin hostname by default.
priority	1	Set this parameter to 1 to direct all traffic to the primary web app.
weight	1000	Weight of the origin in given origin group for load balancing. Must be between 1 and 1000.
enabled-state	Enabled	Whether to enable this origin.
http-port	80	The port used for HTTP requests to the origin.
https-port	443	The port used for HTTPS requests to the origin.

Repeat this step to add your second origin. Pay attention to the `--priority` parameter. For this origin, it's set to "2". This priority setting tells Azure Front Door to direct all traffic to the primary origin unless the primary goes down. If you set the priority for this origin to "1", Azure Front Door will treat both origins as active and direct traffic to both regions. Be sure to replace both instances of the placeholder for `<web-app-west-us>` with the name of that web app.

Azure CLI

```
az afd origin create --resource-group myresourcegroup --host-name <web-app-west-us>.azurewebsites.net --profile-name myfrontdoorprofile --origin-group-name myorigingroup --origin-name secondaryapp --origin-host-header <web-app-west-us>.azurewebsites.net --priority 2 --weight 1000 --enabled-state Enabled --http-port 80 --https-port 443
```

Add a route

Run `az afd route create` to map your endpoint to the origin group. This route forwards requests from the endpoint to your origin group.

Azure CLI

```
az afd route create --resource-group myresourcegroup --profile-name myfrontdoorprofile --endpoint-name myendpoint --forwarding-protocol MatchRequest --route-name route --https-redirect Enabled --origin-group myorigingroup --supported-protocols HttpHttps --link-to-default-domain Enabled
```

Parameter	Value	Description
endpoint-name	myendpoint	Name of the endpoint.
forwarding-protocol	MatchRequest	Protocol this rule will use when forwarding traffic to backends.
route-name	route	Name of the route.
https-redirect	Enabled	Whether to automatically redirect HTTP traffic to HTTPS traffic.
supported-protocols	HttpHttps	List of supported protocols for this route.
link-to-default-domain	Enabled	Whether this route will be linked to the default endpoint domain.

Allow about 15 minutes for this step to complete as it takes some time for this change to propagate globally. After this period, your Azure Front Door will be fully functional.

Restrict access to web apps to the Azure Front Door instance

If you try to access your apps directly using their URLs at this point, you'll still be able to. To ensure traffic can only reach your apps through Azure Front Door, you'll set access restrictions on each of your apps. Front Door's features work best when traffic only flows through Front Door. You should configure your origins to block traffic that hasn't been sent through Front Door. Otherwise, traffic might bypass Front Door's web application firewall, DDoS protection, and other security features. Traffic from Azure Front Door to your applications originates from a well known set of IP ranges defined in the [AzureFrontDoor.Backend service tag](#). By using a service tag restriction rule, you can [restrict traffic to only originate from Azure Front Door](#).

Before setting up the App Service access restrictions, take note of the *Front Door ID* by running the following command. This ID will be needed to ensure traffic only originates from your specific Front Door instance. The access restriction further filters the incoming requests based on the unique HTTP header that your Azure Front Door sends.

Azure CLI

```
az afd profile show --resource-group myresourcegroup --profile-name  
myfrontdoorprofile --query "frontDoorId"
```

Run the following commands to set the access restrictions on your web apps. Replace the placeholder for <front-door-id> with the result from the previous command. Replace the placeholders for the app names.

Azure CLI

```
az webapp config access-restriction add --resource-group myresourcegroup -n  
<web-app-east-us> --priority 100 --service-tag AzureFrontDoor.Backend --  
http-header x-azure-fdid=<front-door-id>  
  
az webapp config access-restriction add --resource-group myresourcegroup -n  
<web-app-west-us> --priority 100 --service-tag AzureFrontDoor.Backend --  
http-header x-azure-fdid=<front-door-id>
```

Test the Front Door

When you create the Azure Front Door Standard/Premium profile, it takes a few minutes for the configuration to be deployed globally. Once completed, you can access the frontend host you created.

Run [az afd endpoint show](#) to get the hostname of the Front Door endpoint.

Azure CLI

```
az afd endpoint show --resource-group myresourcegroup --profile-name  
myfrontdoorprofile --endpoint-name myendpoint --query "hostName"
```

In a browser, go to the endpoint hostname that the previous command returned: <myendpoint>-<hash>.z01.azurefd.net. Your request will automatically get routed to the primary app in East US.

To test instant global failover:

1. Open a browser and go to the endpoint hostname: <myendpoint>-<hash>.z01.azurefd.net.
2. Stop the primary app by running [az webapp stop](#).

Azure CLI

```
az webapp stop --name <web-app-east-us> --resource-group  
myresourcegroup
```

3. Refresh your browser. You should see the same information page because traffic is now directed to the running app in West US.

 Tip

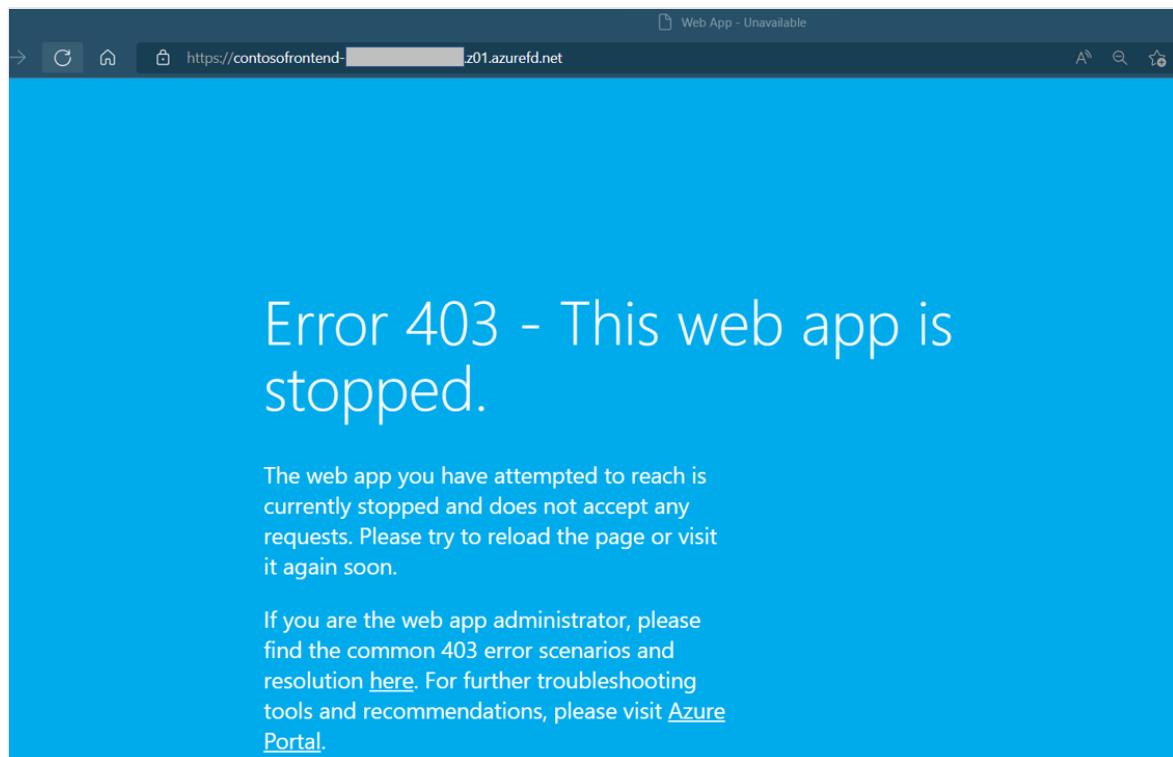
You might need to refresh the page a couple times as failover may take a couple seconds.

4. Now stop the secondary app.

Azure CLI

```
az webapp stop --name <web-app-west-us> --resource-group  
myresourcegroup
```

5. Refresh your browser. This time, you should see an error message.



6. Restart one of the Web Apps by running `az webapp start`. Refresh your browser and you should see the app again.

Azure CLI

```
az webapp start --name <web-app-east-us> --resource-group  
myresourcegroup
```

You've now validated that you can access your apps through Azure Front Door and that failover functions as intended. Restart your other app if you're done with failover testing.

To test your access restrictions and ensure your apps can only be reached through Azure Front Door, open a browser and navigate to each of your app's URLs. To find the URLs, run the following commands:

Azure CLI

```
az webapp show --name <web-app-east-us> --resource-group myresourcegroup --  
query "hostNames"  
  
az webapp show --name <web-app-west-us> --resource-group myresourcegroup --  
query "hostNames"
```

You should see an error page indicating that the apps aren't accessible.

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.

Azure CLI

```
az group delete --name myresourcegroup
```

This command may take a few minutes to run.

Deploy from ARM/Bicep

The resources you created in this tutorial can be deployed using an ARM/Bicep template. The [Highly available multi-region web app Bicep template](#) allows you to create a secure, highly available, multi-region end to end solution with two web apps in different regions behind Azure Front Door.

To learn how to deploy ARM/Bicep templates, see [How to deploy resources with Bicep and Azure CLI](#).

Frequently asked questions

In this tutorial so far, you've deployed the baseline infrastructure to enable a multi-region web app. App Service provides features that can help you ensure you're running applications following security best practices and recommendations.

This section contains frequently asked questions that can help you further secure your apps and deploy and manage your resources using best practices.

What is the recommended method for managing and deploying application infrastructure and Azure resources?

For this tutorial, you used the Azure CLI to deploy your infrastructure resources. Consider configuring a continuous deployment mechanism to manage your application infrastructure. Since you're deploying resources in different regions, you'll need to independently manage those resources across the regions. To ensure the resources are identical across each region, infrastructure as code (IaC) such as [Azure Resource Manager templates](#) or [Terraform](#) should be used with deployment pipelines such as [Azure Pipelines](#) or [GitHub Actions](#). This way, if configured appropriately, any change to resources would trigger updates across all regions you're deployed to. For more information, see [Continuous deployment to Azure App Service](#).

How can I use staging slots to practice safe deployment to production?

Deploying your application code directly to production apps/slots isn't recommended. This is because you'd want to have a safe place to test your apps and validate changes you've made before pushing to production. Use a combination of staging slots and slot swap to move code from your testing environment to production.

You already created the baseline infrastructure for this scenario. You'll now create deployment slots for each instance of your app and configure continuous deployment to these staging slots with GitHub Actions. As with infrastructure management, configuring continuous deployment for your application source code is also recommended to ensure changes across regions are in sync. If you don't configure continuous deployment, you'll need to manually update each app in each region every time there's a code change.

For the remaining steps in this tutorial, you should have an app ready to deploy to your App Services. If you need a sample app, you can use the [Node.js Hello World sample app](#). Fork that repository so you have your own copy.

Be sure to set the App Service stack settings for your apps. Stack settings refer to the language or runtime used for your app. This setting can be configured using the Azure CLI with the `az webapp config set` command or in the portal with the following steps. If you use the Node.js sample, set the stack settings to "Node 18 LTS".

1. Going to your app and selecting **Configuration** in the left-hand table of contents.
2. Select the **General settings** tab.
3. Under **Stack settings**, choose the appropriate value for your app.
4. Select **Save** and then **Continue** to confirm the update.
5. Repeat these steps for your other apps.

Run the following commands to create staging slots called "stage" for each of your apps. Replace the placeholders for `<web-app-east-us>` and `<web-app-west-us>` with your app names.

Azure CLI

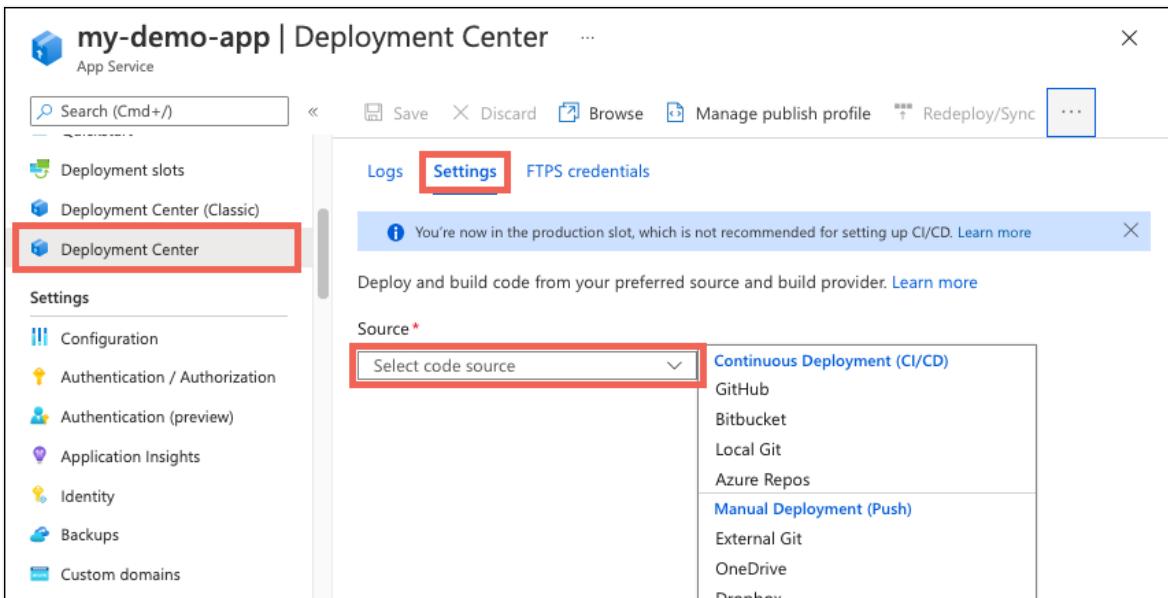
```
az webapp deployment slot create --resource-group myresourcegroup --name <web-app-east-us> --slot stage --configuration-source <web-app-east-us>

az webapp deployment slot create --resource-group myresourcegroup --name <web-app-west-us> --slot stage --configuration-source <web-app-west-us>
```

To set up continuous deployment, you should use the Azure portal. For detailed guidance on how to configure continuous deployment with providers such as GitHub Actions, see [Continuous deployment to Azure App Service](#).

To configure continuous deployment with GitHub Actions, complete the following steps for each of your staging slots.

1. In the [Azure portal](#), go to the management page for one of your App Service app slots.
2. In the left pane, select **Deployment Center**. Then select **Settings**.
3. In the **Source** box, select "GitHub" from the CI/CD options:



4. If you're deploying from GitHub for the first time, select **Authorize** and follow the authorization prompts. If you want to deploy from a different user's repository, select **Change Account**.
 5. After you authorize your Azure account with GitHub, select the **Organization**, **Repository**, and **Branch** to configure CI/CD for. If you can't find an organization or repository, you might need to enable more permissions on GitHub. For more information, see [Managing access to your organization's repositories](#).
- a. If you're using the Node.js sample app, use the following settings.

Setting	Value
Organization	<your-GitHub-organization>
Repository	nodejs-docs-hello-world
Branch	main

6. Select **Save**.

New commits in the selected repository and branch now deploy continuously into your App Service app slot. You can track the commits and deployments on the **Logs** tab.

A default workflow file that uses a publish profile to authenticate to App Service is added to your GitHub repository. You can view this file by going to the `<repo-name>/.github/workflows/` directory.

How do I disable basic auth on App Service?

Consider [disabling basic auth on App Service](#), which limits access to the FTP and SCM endpoints to users that are backed by Azure Active Directory (Azure AD). If using a continuous deployment tool to deploy your application source code, disabling basic auth will require [extra steps to configure continuous deployment](#). For example, you won't be able to use a publish profile since that authentication mechanism doesn't use Azure AD backed credentials. Instead, you'll need to use either a [service principal](#) or [OpenID Connect](#).

To disable basic auth for your App Service, run the following commands for each app and slot by replacing the placeholders for <web-app-east-us> and <web-app-west-us> with your app names. The first set of commands disables FTP access for the production sites and staging slots, and the second set of commands disables basic auth access to the WebDeploy port and SCM site for the production sites and staging slots.

Azure CLI

```
az resource update --resource-group myresourcegroup --name ftp --namespace Microsoft.Web --resource-type basicPublishingCredentialsPolicies --parent sites/<web-app-east-us> --set properties.allow=false

az resource update --resource-group myresourcegroup --name ftp --namespace Microsoft.Web --resource-type basicPublishingCredentialsPolicies --parent sites/<web-app-east-us>/slots/stage --set properties.allow=false

az resource update --resource-group myresourcegroup --name ftp --namespace Microsoft.Web --resource-type basicPublishingCredentialsPolicies --parent sites/<web-app-west-us> --set properties.allow=false

az resource update --resource-group myresourcegroup --name ftp --namespace Microsoft.Web --resource-type basicPublishingCredentialsPolicies --parent sites/<web-app-west-us>/slots/stage --set properties.allow=false
```

Azure CLI

```
az resource update --resource-group myresourcegroup --name scm --namespace Microsoft.Web --resource-type basicPublishingCredentialsPolicies --parent sites/<web-app-east-us> --set properties.allow=false

az resource update --resource-group myresourcegroup --name scm --namespace Microsoft.Web --resource-type basicPublishingCredentialsPolicies --parent sites/<web-app-east-us>/slots/stage --set properties.allow=false

az resource update --resource-group myresourcegroup --name scm --namespace Microsoft.Web --resource-type basicPublishingCredentialsPolicies --parent sites/<web-app-west-us> --set properties.allow=false

az resource update --resource-group myresourcegroup --name scm --namespace Microsoft.Web --resource-type basicPublishingCredentialsPolicies --parent sites/<web-app-west-us>/slots/stage --set properties.allow=false
```

For more information on disabling basic auth including how to test and monitor logins, see [Disabling basic auth on App Service](#).

How do I deploy my code using continuous deployment if I disabled basic auth?

If you choose to allow basic auth on your App Service apps, you can use any of the available deployment methods on App Service, including using the publish profile that was configured in the [staging slots](#) section.

If you disable basic auth for your App Services, continuous deployment requires a service principal or OpenID Connect for authentication. If you use GitHub Actions as your code repository, see the [step-by-step tutorial for using a service principal or OpenID Connect to deploy to App Service using GitHub Actions](#) or complete the steps in the following section.

Create the service principal and configure credentials with GitHub Actions

To configure continuous deployment with GitHub Actions and a service principal, use the following steps.

1. Run the following command to create the [service principal](#). Replace the placeholders with your <subscription-id> and app names. The output is a JSON object with the role assignment credentials that provide access to your App Service apps. Copy this JSON object for the next step. It will include your client secret, which will only be visible at this time. It's always a good practice to grant minimum access. The scope in this example is limited to just the apps, not the entire resource group.

Bash

```
az ad sp create-for-rbac --name "myApp" --role contributor --scopes /subscriptions/<subscription-id>/resourceGroups/myresourcegroup/providers/Microsoft.Web/sites/<web-app-east-us> /subscriptions/<subscription-id>/resourceGroups/myresourcegroup/providers/Microsoft.Web/sites/<web-app-west-us> --sdk-auth
```

2. You need to provide your service principal's credentials to the Azure Login action as part of the GitHub Action workflow you'll be using. These values can either be

provided directly in the workflow or can be stored in GitHub secrets and referenced in your workflow. Saving the values as GitHub secrets is the more secure option.

- a. Open your GitHub repository and go to **Settings > Security > Secrets and variables > Actions**
- b. Select **New repository secret** and create a secret for each of the following values. The values can be found in the json output you copied earlier.

Name	Value
AZURE_APP_ID	<application/client-id>
AZURE_PASSWORD	<client-secret>
AZURE_TENANT_ID	<tenant-id>
AZURE_SUBSCRIPTION_ID	<subscription-id>

Create the GitHub Actions workflow

Now that you have a service principal that can access your App Service apps, edit the default workflows that were created for your apps when you configured continuous deployment. Authentication must be done using your service principal instead of the publish profile. For sample workflows, see the "Service principal" tab in [Deploy to App Service](#). The following sample workflow can be used for the Node.js sample app that was provided.

1. Open your app's GitHub repository and go to the `<repo-name>/.github/workflows/` directory. You'll see the autogenerated workflows.
2. For each workflow file, select the "pencil" button in the top right to edit the file. Replace the contents with the following text, which assumes you created the GitHub secrets earlier for your credential. Update the placeholder for `<web-app-name>` under the "env" section, and then commit directly to the main branch. This commit will trigger the GitHub Action to run again and deploy your code, this time using the service principal to authenticate.

```
yml
```

```
name: Build and deploy Node.js app to Azure Web App
```

```
on:
```

```
push:
  branches:
    - main
  workflow_dispatch:

env:
  AZURE_WEBAPP_NAME: <web-app-name>      # set this to your application's name
  NODE_VERSION: '18.x'                      # set this to the node version to use
  AZURE_WEBAPP_PACKAGE_PATH: '.'           # set this to the path to your web app project, defaults to the repository root
  AZURE_WEBAPP_SLOT_NAME: stage          # set this to your application's slot name

jobs:
  build:
    runs-on: ubuntu-latest

    steps:
      - uses: actions/checkout@v2

      - name: Set up Node.js version
        uses: actions/setup-node@v1
        with:
          node-version: ${{ env.NODE_VERSION }}

      - name: npm install, build
        run: |
          npm install
          npm run build --if-present

      - name: Upload artifact for deployment job
        uses: actions/upload-artifact@v2
        with:
          name: node-app
          path: .

deploy:
  runs-on: ubuntu-latest
  needs: build
  environment:
    name: 'stage'
    url: ${{ steps.deploy-to-webapp.outputs.webapp-url }}

  steps:
    - name: Download artifact from build job
      uses: actions/download-artifact@v2
      with:
        name: node-app

    - uses: azure/login@v1
      with:
        creds: |
        {
```

```
        "clientId": "${{ secrets.AZURE_APP_ID }}",
        "clientSecret": "${{ secrets.AZURE_PASSWORD }}",
        "subscriptionId": "${{ secrets.AZURE_SUBSCRIPTION_ID }}",
        "tenantId": "${{ secrets.AZURE_TENANT_ID }}"
    }

    - name: 'Deploy to Azure Web App'
      id: deploy-to-webapp
      uses: azure/webapps-deploy@v2
      with:
        app-name: ${{ env.AZURE_WEBAPP_NAME }}
        slot-name: ${{ env.AZURE_WEBAPP_SLOT_NAME }}
        package: ${{ env.AZURE_WEBAPP_PACKAGE_PATH }}

    - name: logout
      run: |
        az logout
```

How does slot traffic routing allow me to test updates that I make to my apps?

Traffic routing with slots allows you to direct a pre-defined portion of your user traffic to each slot. Initially, 100% of traffic is directed to the production site. However, you have the ability, for example, to send 10% of your traffic to your staging slot. If you configure slot traffic routing in this way, when users try to access your app, 10% of them will automatically be routed to the staging slot with no changes to your Front Door instance. To learn more about slot swaps and staging environments in App Service, see [Set up staging environments in Azure App Service](#).

How do I move my code from my staging slot to my production slot?

Once you're done testing and validating in your staging slots, you can perform a [slot swap](#) from your staging slot to your production site. You'll need to do this swap for all instances of your app in each region. During a slot swap, the App Service platform ensures the target slot doesn't experience downtime.

To perform the swap, run the following command for each app. Replace the placeholder for `<web-app-name>`.

Azure CLI

```
az webapp deployment slot swap --resource-group MyResourceGroup -name <web-app-name> --slot stage --target-slot production
```

After a few minutes, you can navigate to your Front Door's endpoint to validate the slot swap succeeded.

At this point, your apps are up and running and any changes you make to your application source code will automatically trigger an update to both of your staging slots. You can then repeat the slot swap process when you're ready to move that code into production.

How else can I use Azure Front Door in my multi-region deployments?

If you're concerned about potential disruptions or issues with continuity across regions, as in some customers seeing one version of your app while others see another version, or if you're making significant changes to your apps, you can temporarily remove the site that's undergoing the slot swap from your Front Door's origin group. All traffic will then be directed to the other origin. Navigate to the **Update origin group** pane and **Delete** the origin that is undergoing the change. Once you've made all of your changes and are ready to serve traffic there again, you can return to the same pane and select **+ Add an origin** to readd the origin.

The screenshot shows the Azure Front Door management interface. On the left, the 'Origin groups' section is selected in the navigation menu. A specific origin group named 'MyOriginGroup' is highlighted with a red box. On the right, a detailed 'Update origin group' dialog box is open, also showing 'MyOriginGroup' as the name. The dialog box lists two origins: 'myapp-icuvvu3amn5fk.azurewebsites.net' and 'mysecondaryapp-icuvvu3amn5fk.azurewebsites.net', both marked as 'Enabled'. A 'Delete' button with a red box is visible next to the second origin. Below the origins, sections for 'Health probes' and 'Load balancing' are shown, along with 'Update' and 'Cancel' buttons at the bottom.

If you'd prefer to not delete and then readd origins, you can create extra origin groups for your Front Door instance. You can then associate the route to the origin group pointing to the intended origin. For example, you can create two new origin groups, one

for your primary region, and one for your secondary region. When your primary region is undergoing a change, associate the route with your secondary region and vice versa when your secondary region is undergoing a change. When all changes are complete, you can associate the route with your original origin group that contains both regions. This method works because a route can only be associated with one origin group at a time.

To demonstrate working with multiple origins, in the following screenshot, there are three origin groups. "MyOriginGroup" consists of both web apps, and the other two origin groups each consist of the web app in their respective region. In the example, the app in the primary region is undergoing a change. Before that change was started, the route was associated with "MySecondaryRegion" so all traffic would be sent to the app in the secondary region during the change period. You can update the route by selecting "Unassociated", which will bring up the **Associate routes** pane.

Name	Endpoint	Routes	Provisioning state	Actions
MyOriginGroup		2 associated routes	Succeeded	...
MyPrimaryRegion		1 associated route	Succeeded	...
MySecondaryRegion	afd-jcuypu3gmn5fk	MyRoute	Succeeded	...

How do I restrict access to the advanced tools site?

With Azure App service, the SCM/advanced tools site is used to manage your apps and deploy application source code. Consider [locking down the SCM/advanced tools site](#) since this site will most likely not need to be reached through Front Door. For example, you can set up access restrictions that only allow you to conduct your testing and enable continuous deployment from your tool of choice. If you're using deployment slots, for production slots specifically, you can deny almost all access to the SCM site since your testing and validation will be done with your staging slots.

Next steps

[How to deploy a highly available multi-region web app](#)

[Highly available zone-redundant web application](#)

Tutorial: Create a secure n-tier app in Azure App Service

Article • 03/16/2023

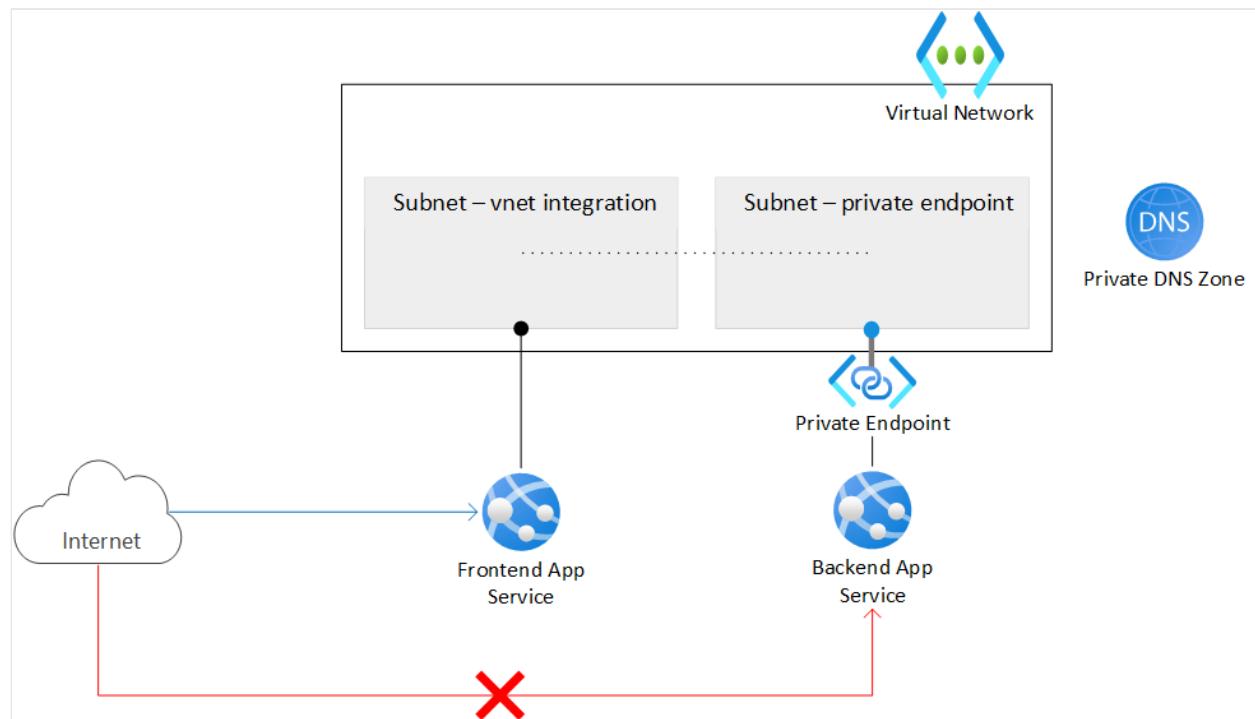
Many applications have more than a single component. For example, you may have a front end that is publicly accessible and connects to a back-end API or web app which in turn connects to a database, storage account, key vault, another VM, or a combination of these resources. This architecture makes up an N-tier application. It's important that applications like this are architected to protect back-end resources to the greatest extent possible.

In this tutorial, you learn how to deploy a secure N-tier application, with a front-end web app that connects to another network-isolated web app. All traffic is isolated within your Azure Virtual Network using [Virtual Network integration](#) and [private endpoints](#). For more comprehensive guidance that includes other scenarios, see:

- [Multi-region N-tier application](#)
- [Reliable web app pattern planning \(.NET\)](#).

Scenario architecture

The following diagram shows the architecture you'll create during this tutorial.



- **Virtual network** Contains two subnets, one is integrated with the front-end web app, and the other has a private endpoint for the back-end web app. The virtual

network blocks all inbound network traffic, except for the front-end app that's integrated with it.

- **Front-end web app** Integrated into the virtual network and accessible from the public internet.
- **Back-end web app** Accessible only through the private endpoint in the virtual network.
- **Private endpoint** Integrates with the back-end web app and makes the web app accessible with a private IP address.
- **Private DNS zone** Lets you resolve a DNS name to the private endpoint's IP address.

ⓘ Note

Virtual network integration and private endpoints are available all the way down to the **Basic** tier in App Service. The **Free** tier doesn't support these features. With this architecture:

- Public traffic to the back-end app is blocked.
- Outbound traffic from App Service is routed to the virtual network and can reach the back-end app.
- App Service is able to perform DNS resolution to the back-end app.

This scenario shows one of the possible N-tier scenarios in App Service. You can use the concepts covered in this tutorial to build more complex N-tier apps.

What you'll learn:

- ✓ Create a virtual network and subnets for App Service virtual network integration.
- ✓ Create private DNS zones.
- ✓ Create private endpoints.
- ✓ Configure virtual network integration in App Service.
- ✓ Disable basic auth in app service.
- ✓ Continuously deploy to a locked down backend web app.

Prerequisites

The tutorial uses two sample Node.js apps that are hosted on GitHub. If you don't already have a GitHub account, [create an account for free ↗](#).

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

To complete this tutorial:

- Use the Bash environment in [Azure Cloud Shell](#). For more information, see [Quickstart for Bash in Azure Cloud Shell](#).

 [Launch Cloud Shell](#) 

- If you prefer to run CLI reference commands locally, [install](#) the Azure CLI. If you're running on Windows or macOS, consider running Azure CLI in a Docker container. For more information, see [How to run the Azure CLI in a Docker container](#).
 - 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 other sign-in options, see [Sign in with the Azure CLI](#).
 - When you're prompted, install the Azure CLI extension 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](#).

1. Create two instances of a web app

You need two instances of a web app, one for the frontend and one for the backend. You need to use at least the **Basic** tier in order to use virtual network integration and private endpoints. You'll configure the virtual network integration and other configurations later on.

1. Create a resource group to manage all of the resources you're creating in this tutorial.

Azure CLI

```
# Save resource group name and region as variables for convenience
groupName=myresourcegroup
region=eastus
az group create --name $groupName --location $region
```

2. Create an App Service plan. Replace <app-service-plan-name> with a unique name. Modify the `--sku` parameter if you need to use a different SKU. Ensure that you aren't using the free tier since that SKU doesn't support the required networking features.

Azure CLI

```
# Save App Service plan name as a variable for convenience
aspName=<app-service-plan-name>
az appservice plan create --name $aspName --resource-group $groupName --
-is-linux --location $region --sku P1V3
```

3. Create the web apps. Replace <frontend-app-name> and <backend-app-name> with two globally unique names (valid characters are a-z, 0-9, and -). For this tutorial, you're provided with sample Node.js apps. If you'd like to use your own apps, change the --runtime parameter accordingly. Run az webapp list-runtimes for the list of available runtimes.

Azure CLI

```
az webapp create --name <frontend-app-name> --resource-group $groupName
--plan $aspName --runtime "NODE:18-lts"
az webapp create --name <backend-app-name> --resource-group $groupName
--plan $aspName --runtime "NODE:18-lts"
```

2. Create network infrastructure

You'll create the following network resources:

- A virtual network.
- A subnet for the App Service virtual network integration.
- A subnet for the private endpoint.
- A private DNS zone.
- A private endpoint.

1. Create a *virtual network*. Replace <virtual-network-name> with a unique name.

Azure CLI

```
# Save vnet name as variable for convenience
vnetName=<virtual-network-name>
az network vnet create --resource-group $groupName --location $region --
-name $vnetName --address-prefixes 10.0.0.0/16
```

2. Create a *subnet for the App Service virtual network integration*.

Azure CLI

```
az network vnet subnet create --resource-group $groupName --vnet-name
$vnetName --name vnet-integration-subnet --address-prefixes 10.0.0.0/24
```

```
--delegations Microsoft.Web/serverfarms --disable-private-endpoint-network-policies false
```

For App Service, the virtual network integration subnet is recommended to [have a CIDR block of /26 at a minimum](#). /24 is more than sufficient. `--delegations Microsoft.Web/serverfarms` specifies that the subnet is [delegated for App Service virtual network integration](#).

3. Create another *subnet for the private endpoints*.

Azure CLI

```
az network vnet subnet create --resource-group $groupName --vnet-name $vnetName --name private-endpoint-subnet --address-prefixes 10.0.1.0/24 --disable-private-endpoint-network-policies true
```

For private endpoint subnets, you must [disable private endpoint network policies](#) by setting `--disable-private-endpoint-network-policies` to `true`.

4. Create the *private DNS zone*.

Azure CLI

```
az network private-dns zone create --resource-group $groupName --name privatelink.azurewebsites.net
```

For more information on these settings, see [Azure Private Endpoint DNS configuration](#).

(!) Note

If you create the private endpoint using the portal, a private DNS zone is created automatically for you and you don't need to create it separately. For consistency with this tutorial, you create the private DNS zone and private endpoint separately using the Azure CLI.

5. Link the private DNS zone to the virtual network.

Azure CLI

```
az network private-dns link vnet create --resource-group $groupName --name myDnsLink --zone-name privatelink.azurewebsites.net --virtual-network $vnetName --registration-enabled False
```

6. In the private endpoint subnet of your virtual network, create a *private endpoint* for your backend web app. Replace <backend-app-name> with your backend web app name.

Azure CLI

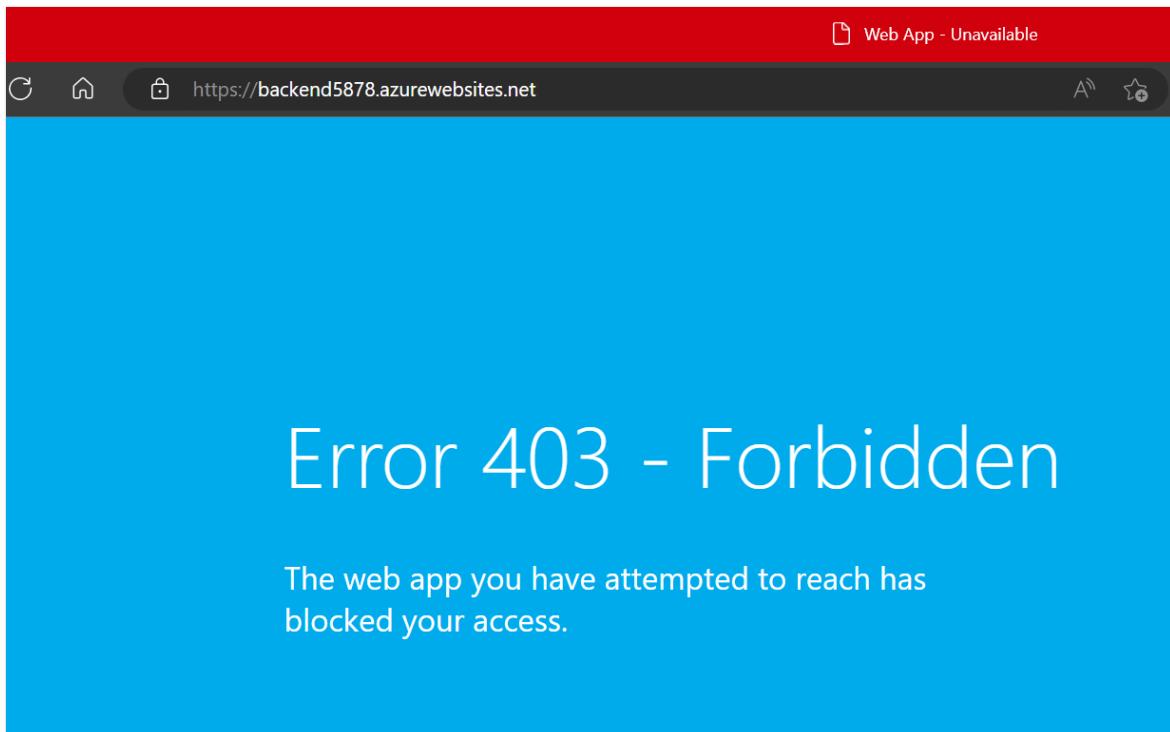
```
# Get backend web app resource ID
resourceId=$(az webapp show --resource-group $groupName --name
<backend-app-name> --query id --output tsv)
az network private-endpoint create --resource-group $groupName --name
myPrivateEndpoint --location $region --connection-name myConnection --
private-connection-resource-id $resourceId --group-id sites --vnet-name
$vnetName --subnet private-endpoint-subnet
```

7. Link the private endpoint to the private DNS zone with a DNS zone group for the backend web app private endpoint. This DNS zone group helps you to auto-update the private DNS Zone when there's an update to the private endpoint.

Azure CLI

```
az network private-endpoint dns-zone-group create --resource-group
$groupName --endpoint-name myPrivateEndpoint --name myZoneGroup --
private-dns-zone privatelink.azurewebsites.net --zone-name
privatelink.azurewebsites.net
```

8. When you create a private endpoint for an App Service, public access gets implicitly disabled. If you try to access your backend web app using its default URL, your access is denied. From a browser, navigate to <backend-app-name>.azurewebsites.net to confirm this behavior.



For more information on App Service access restrictions with private endpoints, see [Azure App Service access restrictions](#).

3. Configure virtual network integration in your frontend web app

Enable virtual network integration on your app. Replace <frontend-app-name> with your frontend web app name.

Azure CLI

```
az webapp vnet-integration add --resource-group $groupName --name <frontend-app-name> --vnet $vnetName --subnet vnet-integration-subnet
```

Virtual network integration allows outbound traffic to flow directly into the virtual network. By default, only local IP traffic defined in [RFC-1918](#) is routed to the virtual network, which is what you need for the private endpoints. To route all your traffic to the virtual network, see [Manage virtual network integration routing](#). Routing all traffic can also be used if you want to route internet traffic through your virtual network, such as through an [Azure Virtual Network NAT](#) or an [Azure Firewall](#).

4. Enable deployment to back-end web app from internet

Since your backend web app isn't publicly accessible, you must let your continuous deployment tool reach your app by [making the SCM site publicly accessible](#). The main web app itself can continue to deny all traffic.

1. Enable public access for the back-end web app.

```
Azure CLI
```

```
az webapp update --resource-group $groupName --name <backend-app-name>
--set publicNetworkAccess=Enabled
```

2. Set the unmatched rule action for the main web app to deny all traffic. This setting denies public access to the main web app even though the general app access setting is set to allow public access.

```
Azure CLI
```

```
az resource update --resource-group $groupName --name <backend-app-name> --namespace Microsoft.Web --resource-type sites --set
properties.siteConfig.ipSecurityRestrictionsDefaultAction=Deny
```

3. Set the unmatched rule action for the SCM site to allow all traffic.

```
Azure CLI
```

```
az resource update --resource-group $groupName --name <backend-app-name> --namespace Microsoft.Web --resource-type sites --set
properties.siteConfig.scmIpSecurityRestrictionsDefaultAction=Allow
```

5. Lock down FTP and SCM access

Now that the back-end SCM site is publicly accessible, you need to lock it down with better security.

1. Disable FTP access for both the front-end and back-end web apps. Replace `<frontend-app-name>` and `<backend-app-name>` with your app names.

```
Azure CLI
```

```
az resource update --resource-group $groupName --name ftp --namespace
Microsoft.Web --resource-type basicPublishingCredentialsPolicies --
parent sites/<frontend-app-name> --set properties.allow=false
az resource update --resource-group $groupName --name ftp --namespace
Microsoft.Web --resource-type basicPublishingCredentialsPolicies --
parent sites/<backend-app-name> --set properties.allow=false
```

2. Disable basic auth access to the WebDeploy ports and SCM/advanced tool sites for both web apps. Replace <frontend-app-name> and <backend-app-name> with your app names.

Azure CLI

```
az resource update --resource-group $groupName --name scm --namespace Microsoft.Web --resource-type basicPublishingCredentialsPolicies --parent sites/<frontend-app-name> --set properties.allow=false  
az resource update --resource-group $groupName --name scm --namespace Microsoft.Web --resource-type basicPublishingCredentialsPolicies --parent sites/<backend-app-name> --set properties.allow=false
```

Disabling basic auth on App Service [\[2\]](#) limits access to the FTP and SCM endpoints to users that are backed by Azure Active Directory, which further secures your apps. For more information on disabling basic auth including how to test and monitor logins, see [Disabling basic auth on App Service \[2\]](#).

6. Configure continuous deployment using GitHub Actions

1. Navigate to the [Node.js backend sample app \[2\]](#). This app is a simple Hello World app.
2. Select the **Fork** button in the upper right on the GitHub page.
3. Select the **Owner** and leave the default Repository name.
4. Select **Create fork**.
5. Repeat the same process for the [Node.js frontend sample app \[2\]](#). This app is a basic web app that accesses a remote URL.
6. Create a [service principal](#). Replace <subscription-id>, <frontend-app-name>, and <backend-app-name> with your values.

Azure CLI

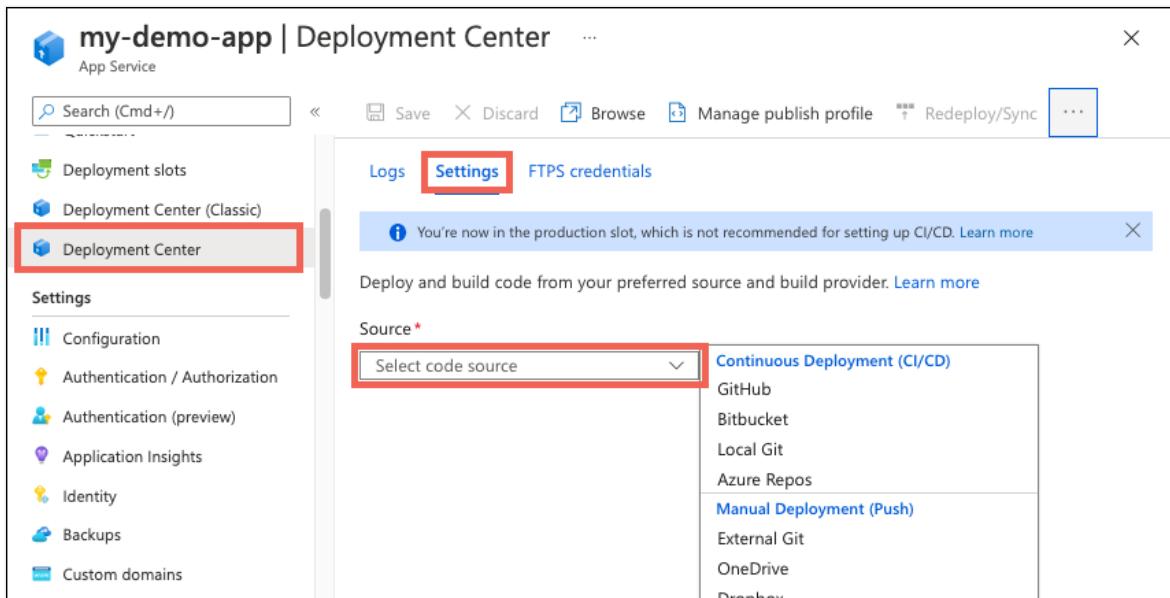
```
az ad sp create-for-rbac --name "myApp" --role contributor --scopes /subscriptions/<subscription-id>/resourceGroups/$groupName/providers/Microsoft.Web/sites/<frontend-app-name> /subscriptions/<subscription-id>/resourceGroups/$groupName/providers/Microsoft.Web/sites/<backend-app-name> --sdk-auth
```

The output is a JSON object with the role assignment credentials that provide access to your App Service apps. Copy this JSON object for the next step. It includes your client secret, which is only visible at this time. It's always a good practice to grant minimum access. The scope in this example is limited to just the apps, not the entire resource group.

7. To store the service principal's credentials as GitHub secrets, go to one of the forked sample repositories in GitHub and go to **Settings > Security > Secrets and variables > Actions**.
8. Select **New repository secret** and create a secret for each of the following values. The values can be found in the json output you copied earlier.

Name	Value
AZURE_APP_ID	<application/client-id>
AZURE_PASSWORD	<client-secret>
AZURE_TENANT_ID	<tenant-id>
AZURE_SUBSCRIPTION_ID	<subscription-id>

9. Repeat this process for the other forked sample repository.
10. To set up continuous deployment with GitHub Actions, sign in to the [Azure portal ↗](#).
11. Navigate to the **Overview** page for your front-end web app.
12. In the left pane, select **Deployment Center**. Then select **Settings**.
13. In the **Source** box, select "GitHub" from the CI/CD options.



14. If you're deploying from GitHub for the first time, select **Authorize** and follow the authorization prompts. If you want to deploy from a different user's repository, select **Change Account**.
15. If you're using the Node.js sample app that was forked as part of the prerequisites, use the following settings for **Organization**, **Repository**, and **Branch**.

Setting	Value
Organization	<your-GitHub-organization>
Repository	nodejs-frontend
Branch	main

16. Select **Save**.
17. Repeat the same steps for your back-end web app. The Deployment Center settings are given in the following table.

Setting	Value
Organization	<your-GitHub-organization>
Repository	nodejs-backend
Branch	main

7. Use a service principal for GitHub Actions deployment

Your Deployment Center configuration has created a default workflow file in each of your sample repositories, but it uses a publish profile by default, which uses basic auth. Since you've disabled basic auth, if you check the **Logs** tab in Deployment Center, you'll see that the automatically triggered deployment results in an error. You must modify the workflow file to use the service principal to authenticate with App Service. For sample workflows, see [Deploy to App Service](#).

1. Open one of your forked GitHub repositories and go to the `<repo-name>/.github/workflows/` directory.
2. Select the auto-generated workflow file and then select the "pencil" button in the top right to edit the file. Replace the contents with the following text, which assumes you created the GitHub secrets earlier for your credential. Update the placeholder for `<web-app-name>` under the "env" section, and then commit directly to the main branch. This commit triggers the GitHub Action to run again and deploy your code, this time using the service principal to authenticate.

```
yml

name: Build and deploy Node.js app to Azure Web App

on:
  push:
    branches:
      - main
  workflow_dispatch:

env:
  AZURE_WEBAPP_NAME: <web-app-name>      # set this to your application's name
  NODE_VERSION: '18.x'                      # set this to the node version to use
  AZURE_WEBAPP_PACKAGE_PATH: '.'           # set this to the path to your web app project, defaults to the repository root

jobs:
  build:
    runs-on: ubuntu-latest

    steps:
      - uses: actions/checkout@v2

      - name: Set up Node.js version
        uses: actions/setup-node@v1
        with:
          node-version: ${{ env.NODE_VERSION }}

      - name: npm install, build
        run: |
```

```

    npm install
    npm run build --if-present

    - name: Upload artifact for deployment job
      uses: actions/upload-artifact@v2
      with:
        name: node-app
        path: .

  deploy:
    runs-on: ubuntu-latest
    needs: build
    environment:
      url: ${{ steps.deploy-to-webapp.outputs.webapp-url }}

  steps:
    - name: Download artifact from build job
      uses: actions/download-artifact@v2
      with:
        name: node-app
    - uses: azure/login@v1
      with:
        creds: |
          {
            "clientId": "${{ secrets.AZURE_APP_ID }}",
            "clientSecret": "${{ secrets.AZURE_PASSWORD }}",
            "subscriptionId": "${{ secrets.AZURE_SUBSCRIPTION_ID }}",
            "tenantId": "${{ secrets.AZURE_TENANT_ID }}"
          }

    - name: 'Deploy to Azure Web App'
      id: deploy-to-webapp
      uses: azure/webapps-deploy@v2
      with:
        app-name: ${{ env.AZURE_WEBAPP_NAME }}
        package: ${{ env.AZURE_WEBAPP_PACKAGE_PATH }}

    - name: logout
      run: |
        az logout

```

3. Repeat this process for the workflow file in your other forked GitHub repository.

The new GitHub commits trigger another deployment for each of your apps. This time, deployment should succeed since the workflow uses the service principal to authenticate with the apps' SCM sites.

For detailed guidance on how to configure continuous deployment with providers such as GitHub Actions, see [Continuous deployment to Azure App Service](#).

8. Validate connections and app access

1. Browse to the front-end web app with its URL: `https://<frontend-app-name>.azurewebsites.net`.
2. In the textbox, input the URL for your backend web app: `https://<backend-app-name>.azurewebsites.net`. If you set up the connections properly, you should get the message "Hello from the backend web app!", which is the entire content of the backend web app. All *outbound* traffic from the front-end web app is routed through the virtual network. Your frontend web app is securely connecting to your backend web app through the private endpoint. If something is wrong with your connections, your frontend web app crashes.
3. Try navigating directly to your backend web app with its URL: `https://<backend-app-name>.azurewebsites.net`. You should see the message `Web App - Unavailable`. If you can reach the app, ensure you've configured the private endpoint and that the access restrictions for your app are set to deny all traffic for the main web app.
4. To further validate that the frontend web app is reaching the backend web app over private link, SSH to one of your front end's instances. To SSH, run the following command, which establishes an SSH session to the web container of your app and opens a remote shell in your browser.

Azure CLI

```
az webapp ssh --resource-group $groupName --name <frontend-app-name>
```

5. When the shell opens in your browser, run `nslookup` to confirm your back-end web app is being reached using the private IP of your backend web app. You can also run `curl` to validate the site content again. Replace `<backend-app-name>` with your backend web app name.

Bash

```
nslookup <backend-app-name>.azurewebsites.net
curl https://<backend-app-name>.azurewebsites.net
```

Last login: Wed Feb 15 03:29:33 2023 from 169.254.129.2

APP SERVICE ON LINUX

Documentation: <http://aka.ms/webapp-linux>
NodeJS quickstart: <https://aka.ms/node-qs>
NodeJS Version : v18.12.1
Note: Any data outside '/home' is not persisted

```
root@8e114a2c33d9:/home# nslookup backend-web-app.azurewebsites.net
Server: 127.0.0.11
Address: 127.0.0.11#53

Non-authoritative answer:
  backend-web-app.azurewebsites.net canonical name = backend-web-app.privatelink.azurewebsites.net.
Name: backend-web-app.privatelink.azurewebsites.net
Address: 10.0.1.4

root@8e114a2c33d9:/home# curl https://backend-web-app.azurewebsites.net
Hello from the backend web app!root@8e114a2c33d9:/home#
```

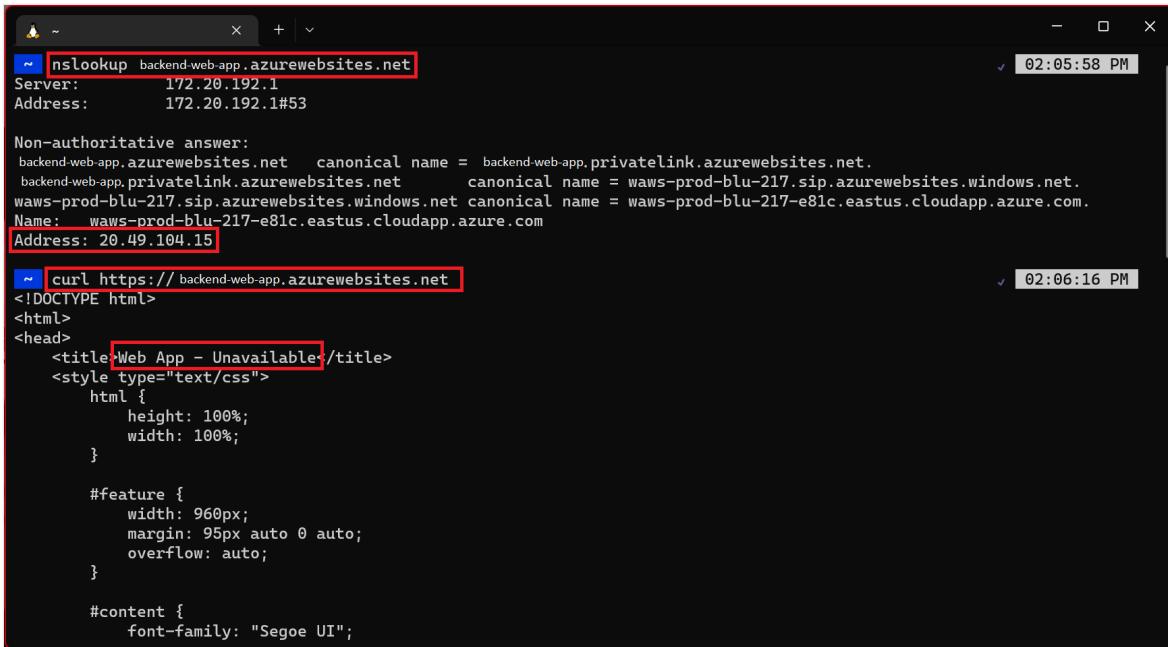
The `nslookup` should resolve to the private IP address of your back-end web app. The private IP address should be an address from your virtual network. To confirm your private IP, go to the [Networking](#) page for your back-end web app.

The screenshot shows the Microsoft Azure portal interface for managing the networking configuration of a web application named "backend". The left sidebar lists various settings categories, with "Networking" highlighted. The main content area is titled "backend | Networking" and contains three main sections: "Inbound Traffic", "Web App", and "Outbound Traffic".

- Inbound Traffic:** Features include Access restriction (On), App assigned address (N/A), and Private endpoints (On). An "Inbound address" input field contains "10.0.1.4", which is highlighted with a red box.
- Web App:** Describes custom domains directing traffic to the web app. A "Domains" section lists "jordanntierbackend.azurewebsites.net".
- Outbound Traffic:** Features include VNet integration (Off) and Hybrid connections (Off). It also lists outbound addresses: "20.120.68.244,20.120.69.26,20.120.69.48,20.120.69.94".

At the top right, there is a search bar and a feedback link. A note at the top states: "When you enable private endpoints, app assigned address inbound traffic feature is no longer applicable and will be turned off."

6. Repeat the same `nslookup` and `curl` commands from another terminal (one that isn't an SSH session on your front end's instances).



```
~ nslookup backend-web-app.azurewebsites.net
Server: 172.20.192.1
Address: 172.20.192.1#53

Non-authoritative answer:
backend-web-app.azurewebsites.net canonical name = backend-web-app.privatelink.azurewebsites.net.
backend-web-app.privatelink.azurewebsites.net canonical name = waws-prod-blu-217.sip.azurewebsites.windows.net.
waws-prod-blu-217.sip.azurewebsites.windows.net canonical name = waws-prod-blu-217-e81c.eastus.cloudapp.azure.com.
Name: waws-prod-blu-217-e81c.eastus.cloudapp.azure.com
Address: 20.49.104.15

curl https://backend-web-app.azurewebsites.net
<!DOCTYPE html>
<html>
<head>
    <title>Web App - Unavailable</title>
    <style type="text/css">
        html {
            height: 100%;
            width: 100%;
        }

        #feature {
            width: 960px;
            margin: 95px auto 0 auto;
            overflow: auto;
        }

        #content {
            font-family: "Segoe UI";
        }
    </style>

```

The `nslookup` returns the public IP for the back-end web app. Since public access to the back-end web app is disabled, if you try to reach the public IP, you get an access denied error. This error means this site isn't accessible from the public internet, which is the intended behavior. The `nslookup` doesn't resolve to the private IP because that can only be resolved from within the virtual network through the private DNS zone. Only the front-end web app is within the virtual network. If you try to run `curl` on the back-end web app from the external terminal, the HTML that is returned contains `Web App - Unavailable`. This error displays the HTML for the error page you saw earlier when you tried navigating to the back-end web app in your browser.

9. 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.

Azure CLI

```
az group delete --name myresourcegroup
```

This command may take a few minutes to run.

Frequently asked questions

- [Is there an alternative to deployment using a service principal?](#)

- What happens when I configure GitHub Actions deployment in App Service?
- Is it safe to leave the back-end SCM publicly accessible?
- Is there a way to deploy without opening up the back-end SCM site at all?
- How can I deploy this architecture with ARM/Bicep?

Is there an alternative to deployment using a service principal?

Since in this tutorial you've [disabled basic auth](#), you can't authenticate with the back end SCM site with a username and password, and neither can you with a publish profile. Instead of a service principal, you can also use [OpenID Connect](#).

What happens when I configure GitHub Actions deployment in App Service?

Azure auto-generates a workflow file in your repository. New commits in the selected repository and branch now deploy continuously into your App Service app. You can track the commits and deployments on the [Logs](#) tab.

A default workflow file that uses a publish profile to authenticate to App Service is added to your GitHub repository. You can view this file by going to the `<repo-name>/ .github/workflows/` directory.

Is it safe to leave the back-end SCM publicly accessible?

When you [lock down FTP and SCM access](#), it ensures that only Azure AD backed principals can access the SCM endpoint even though it's publicly accessible. This setting should reassure you that your backend web app is still secure.

Is there a way to deploy without opening up the back-end SCM site at all?

If you're concerned about enabling public access to the SCM site, or you're restricted by policy, consider other App Service deployment options like [running from a ZIP package](#).

How can I deploy this architecture with ARM/Bicep?

The resources you created in this tutorial can be deployed using an ARM/Bicep template. The [App connected to a backend web app Bicep template](#) allows you to create a secure N-tier app solution.

To learn how to deploy ARM/Bicep templates, see [How to deploy resources with Bicep and Azure CLI](#).

Next steps

[Integrate your app with an Azure virtual network](#)

[App Service networking features](#)

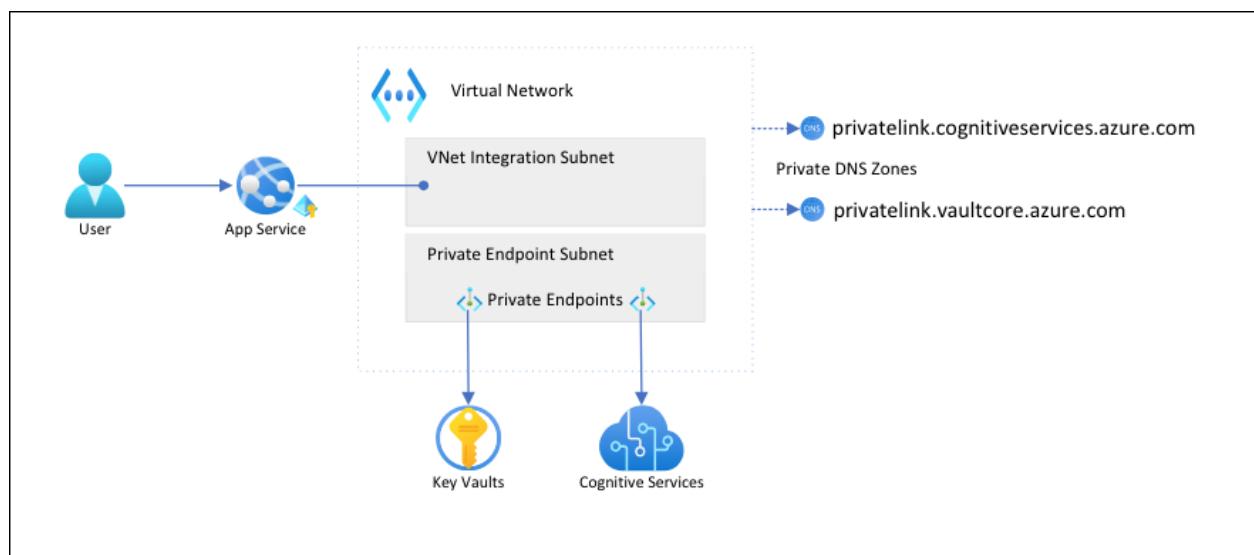
[Reliable web app pattern planning \(.NET\)](#)

Tutorial: Isolate back-end communication in Azure App Service with Virtual Network integration

Article • 03/09/2023

In this article you will configure an App Service app with secure, network-isolated communication to backend services. The example scenario used is in [Tutorial: Secure Cognitive Service connection from App Service using Key Vault](#). When you're finished, you have an App Service app that accesses both Key Vault and Cognitive Services through an [Azure virtual network](#), and no other traffic is allowed to access those back-end resources. All traffic will be isolated within your virtual network using [virtual network integration](#) and [private endpoints](#).

As a multi-tenanted service, outbound network traffic from your App Service app to other Azure services shares the same environment with other apps or even other subscriptions. While the traffic itself can be encrypted, certain scenarios may require an extra level of security by isolating back-end communication from other network traffic. These scenarios are typically accessible to large enterprises with a high level of expertise, but App Service puts it within reach with virtual network integration.



With this architecture:

- Public traffic to the back-end services is blocked.
- Outbound traffic from App Service is routed to the virtual network and can reach the back-end services.
- App Service is able to perform DNS resolution to the back-end services through the private DNS zones.

What you will learn:

- ✓ Create a virtual network and subnets for App Service virtual network integration
- ✓ Create private DNS zones
- ✓ Create private endpoints
- ✓ Configure virtual network integration in App Service

Prerequisites

The tutorial assumes that you have followed the [Tutorial: Secure Cognitive Service connection from App Service using Key Vault](#) and created the language detector app.

The tutorial continues to use the following environment variables from the previous tutorial. Make sure you set them properly.

Azure CLI

```
groupName=myKVResourceGroup
region=westeurope
csResourceName=<cs-resource-name>
appName=<app-name>
vaultName=<vault-name>
```

Create virtual network and subnets

1. Create a virtual network. Replace <*virtual-network-name*> with a unique name.

Azure CLI

```
# Save vnet name as variable for convenience
vnetName=<virtual-network-name>

az network vnet create --resource-group $groupName --location $region -
-name $vnetName --address-prefixes 10.0.0.0/16
```

2. Create a subnet for the App Service virtual network integration.

Azure CLI

```
az network vnet subnet create --resource-group $groupName --vnet-name
$vnetName --name vnet-integration-subnet --address-prefixes 10.0.0.0/24
--delegations Microsoft.Web/serverfarms --disable-private-endpoint-
network-policies false
```

For App Service, the virtual network integration subnet is recommended to have a CIDR block of /26 at a minimum (see [Virtual network integration subnet requirements](#)). /24 is more than sufficient. `--delegations Microsoft.Web/serverfarms` specifies that the subnet is delegated for App Service virtual network integration.

3. Create another subnet for the private endpoints.

Azure CLI

```
az network vnet subnet create --resource-group $groupName --vnet-name $vnetName --name private-endpoint-subnet --address-prefixes 10.0.1.0/24 --disable-private-endpoint-network-policies true
```

For private endpoint subnets, you must [disable private endpoint network policies](#).

Create private DNS zones

Because your Key Vault and Cognitive Services resources will sit behind [private endpoints](#), you need to define [private DNS zones](#) for them. These zones are used to host the DNS records for private endpoints and allow the clients to find the back-end services by name.

1. Create two private DNS zones, one for your Cognitive Services resource and one for your key vault.

Azure CLI

```
az network private-dns zone create --resource-group $groupName --name privatelink.cognitiveservices.azure.com  
az network private-dns zone create --resource-group $groupName --name privatelink.vaultcore.azure.net
```

For more information on these settings, see [Azure Private Endpoint DNS configuration](#)

2. Link the private DNS zones to the virtual network.

Azure CLI

```
az network private-dns link vnet create --resource-group $groupName --name cognitiveservices-zonelink --zone-name privatelink.cognitiveservices.azure.com --virtual-network $vnetName --registration-enabled False  
az network private-dns link vnet create --resource-group $groupName --
```

```
name vaultcore-zonelink --zone-name privatelink.vaultcore.azure.net --  
virtual-network $vnetName --registration-enabled False
```

Create private endpoints

1. In the private endpoint subnet of your virtual network, create a private endpoint for your Cognitive Service.

Azure CLI

```
# Get Cognitive Services resource ID  
csResourceId=$(az cognitiveservices account show --resource-group  
$groupName --name $csResourceName --query id --output tsv)  
  
az network private-endpoint create --resource-group $groupName --name  
securecstext-pe --location $region --connection-name securecstext-pc --  
private-connection-resource-id $csResourceId --group-id account --vnet-  
name $vnetName --subnet private-endpoint-subnet
```

2. Create a DNS zone group for the Cognitive Services private endpoint. DNS zone group is a link between the private DNS zone and the private endpoint. This link helps you to auto update the private DNS Zone when there is an update to the private endpoint.

Azure CLI

```
az network private-endpoint dns-zone-group create --resource-group  
$groupName --endpoint-name securecstext-pe --name securecstext-zg --  
private-dns-zone privatelink.cognitiveservices.azure.com --zone-name  
privatelink.cognitiveservices.azure.com
```

3. Block public traffic to the Cognitive Services resource.

Azure CLI

```
az rest --uri $csResourceId?api-version=2021-04-30 --method PATCH --  
body '{"properties":{"publicNetworkAccess":"Disabled"}}' --headers  
'Content-Type=application/json'  
  
# Repeat following command until output is "Succeeded"  
az cognitiveservices account show --resource-group $groupName --name  
$csResourceName --query properties.provisioningState
```

ⓘ Note

Make sure the provisioning state of your change is "Succeeded". Then you can observe the behavior change in the sample app. You can still load the app, but if you try click the **Detect** button, you get an `HTTP 500` error. The app has lost its connectivity to the Cognitive Services resource through the shared networking.

4. Repeat the steps above for the key vault.

Azure CLI

```
# Create private endpoint for key vault
vaultResourceId=$(az keyvault show --name $vaultName --query id --output tsv)
az network private-endpoint create --resource-group $groupName --name securekeyvault-pe --location $region --connection-name securekeyvault-pe --private-connection-resource-id $vaultResourceId --group-id vault --vnet-name $vnetName --subnet private-endpoint-subnet
# Create DNS zone group for the endpoint
az network private-endpoint dns-zone-group create --resource-group $groupName --endpoint-name securekeyvault-pe --name securekeyvault-zg --private-dns-zone privatelink.vaultcore.azure.net --zone-name privatelink.vaultcore.azure.net
# Block public traffic to key vault
az keyvault update --name $vaultName --default-action Deny
```

5. Force an immediate refetch of the [key vault references](#) in your app by resetting the app settings (for more information, see [Rotation](#)).

Azure CLI

```
az webapp config appsettings set --resource-group $groupName --name $appName --settings CS_ACCOUNT_NAME="@Microsoft.KeyVault(SecretUri=$csResourceKVUri)" CS_ACCOUNT_KEY="@Microsoft.KeyVault(SecretUri=$csKeyKVUri)"
```

⚠ Note

Again, you can observe the behavior change in the sample app. You can no longer load the app because it can no longer access the key vault references. The app has lost its connectivity to the key vault through the shared networking.

The two private endpoints are only accessible to clients inside the virtual network you created. You can't even access the secrets in the key vault through [Secrets](#) page in the

Azure portal, because the portal accesses them through the public internet (see [Manage the locked down resources](#)).

Configure virtual network integration in your app

1. Scale the app up to a supported pricing tier (see [Integrate your app with an Azure virtual network](#)).

Azure CLI

```
az appservice plan update --name $appName --resource-group $groupName -  
-sku S1
```

2. Unrelated to our scenario but also important, enforce HTTPS for inbound requests.

Azure CLI

```
az webapp update --resource-group $groupName --name $appName --https-only
```

3. Enable virtual network integration on your app.

Azure CLI

```
az webapp vnet-integration add --resource-group $groupName --name  
$appName --vnet $vnetName --subnet vnet-integration-subnet
```

Virtual network integration allows outbound traffic to flow directly into the virtual network. By default, only local IP traffic defined in [RFC-1918](#) is routed to the virtual network, which is what you need for the private endpoints. To route all your traffic to the virtual network, see [Manage virtual network integration routing](#).

Routing all traffic can also be used if you want to route internet traffic through your virtual network, such as through an [Azure Virtual Network NAT](#) or an [Azure Firewall](#).

4. In the browser, navigate to `<app-name>.azurewebsites.net` again and wait for the integration to take effect. If you get an HTTP 500 error, wait a few minutes and try again. If you can load the page and get detection results, then you're connecting to the Cognitive Services endpoint with key vault references.

 Note

If keep getting HTTP 500 errors after a long time, it may help to force a refetch of the **key vault references** again, like so:

Azure CLI

```
az webapp config appsettings set --resource-group $groupName --name $appName --settings CS_ACCOUNT_NAME="@Microsoft.KeyVault(SecretUri=$csResourceKVUri)" CS_ACCOUNT_KEY="@Microsoft.KeyVault(SecretUri=$csKeyKVUri)"
```

Manage the locked down resources

Depending on your scenarios, you may not be able to manage the private endpoint protected resources through the Azure portal, Azure CLI, or Azure PowerShell (for example, Key Vault). These tools all make REST API calls to access the resources through the public internet, and are blocked by your configuration. Here are a few options for accessing the locked down resources:

- For Key Vault, add the public IP of your local machine to view or update the private endpoint protected secrets.
- If your on premises network is extended into the Azure virtual network through a [VPN gateway](#) or [ExpressRoute](#), you can manage the private endpoint protected resources directly from your on premises network.
- Manage the private endpoint protected resources from a [jump server](#) in the virtual network.
- [Deploy Cloud Shell into the virtual network](#).

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:

Azure CLI

```
az group delete --name $groupName
```

This command may take a minute to run.

Next steps

- Integrate your app with an Azure virtual network
- App Service networking features

Disable basic authentication in App Service deployments

Article • 03/01/2024

This article shows you how to disable basic authentication (username and password authentication) when deploying code to App Service apps.

App Service provides basic authentication for FTP and WebDeploy clients to connect to it by using [deployment credentials](#). These APIs are great for browsing your site's file system, uploading drivers and utilities, and deploying with MsBuild. However, enterprises often require more secure deployment methods than basic authentication, such as [Microsoft Entra ID authentication](#) (see [Authentication types by deployment methods in Azure App Service](#)). Microsoft Entra uses OAuth 2.0 token-based authorization and has many benefits and improvements that help mitigate the issues in basic authentication. For example, OAuth access tokens have a limited usable lifetime, and are specific to the applications and resources for which they're issued, so they can't be reused. Microsoft Entra also lets you deploy from other Azure services using managed identities.

Disable basic authentication

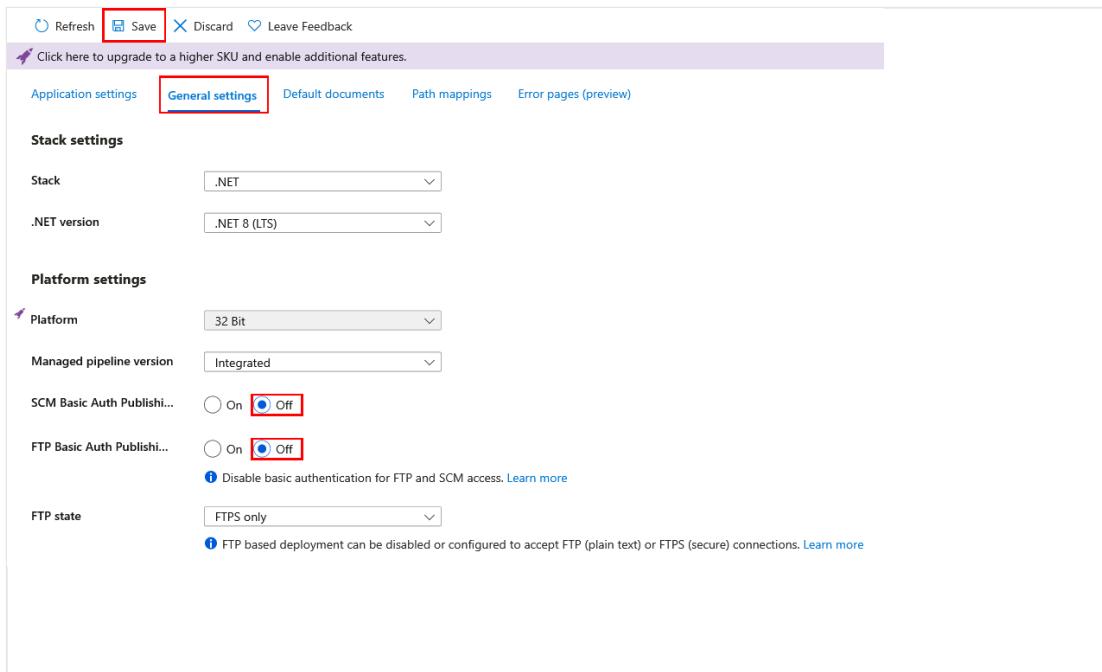
Two different controls for basic authentication are available. Specifically:

- For [FTP deployment](#), basic authentication is controlled by the `basicPublishingCredentialsPolicies/ftp` flag ([FTP Basic Auth Publishing Credentials](#) option in the portal).
- For other deployment methods that use basic authentication, such as Visual Studio, local Git, and GitHub, basic authentication is controlled by the `basicPublishingCredentialsPolicies/scm` flag ([SCM Basic Auth Publishing Credentials](#) option in the portal).

Azure portal

1. In the [Azure portal](#), search for and select **App Services**, and then select your app.
2. In the app's left menu, select **Configuration > General settings**.

3. For SCM Basic Auth Publishing Credentials or FTP Basic Auth Publishing Credentials, select Off, then select Save.



To confirm that FTP access is blocked, try [connecting to your app using FTP/S](#). You should get a `401 Unauthenticated` message.

To confirm that Git access is blocked, try [local Git deployment](#). You should get an `Authentication failed` message.

Deployment without basic authentication

When you disable basic authentication, deployment methods that depend on basic authentication stop working.

The following table shows how various deployment methods behave when basic authentication is disabled, and if there's any fallback mechanism. For more information, see [Authentication types by deployment methods in Azure App Service](#).

[+] Expand table

Deployment method	When basic authentication is disabled
Visual Studio deployment	Doesn't work.
FTP	Doesn't work.
Local Git	Doesn't work.

Deployment method	When basic authentication is disabled
Azure CLI	<p>In Azure CLI 2.48.1 or higher, the following commands fall back to Microsoft Entra authentication:</p> <ul style="list-style-type: none"> - az webapp up - az webapp deploy - az webapp log deployment show - az webapp log deployment list - az webapp log download - az webapp log tail - az webapp browse - az webapp create-remote-connection - az webapp ssh - az functionapp deploy - az functionapp log deployment list - az functionapp log deployment show - az functionapp deployment source config-zip
Maven plugin ↗ or Gradle plugin ↗	Works.
GitHub with App Service Build Service	Doesn't work.
GitHub Actions	<ul style="list-style-type: none"> - An existing GitHub Actions workflow that uses basic authentication can't authenticate. In the Deployment Center, disconnect the existing GitHub configuration and create a new GitHub Actions configuration with the user-assigned identity option instead. - If the existing GitHub Actions deployment is manually configured, try using a service principal or OpenID Connect instead. - For new GitHub Actions configuration in the Deployment Center, use the user-assigned identity option.
Deployment in create wizard ↗	When Basic authentication is set to Disable and Continuous deployment set to Enable , GitHub Actions is configured with the user-assigned identity option (OpenID Connect).
Azure Repos with App Service Build Service	Doesn't work.
Bitbucket	Doesn't work.
Azure Pipelines with AzureWebApp task	Works.
Azure Pipelines with AzureRmWebAppDeployment task	<ul style="list-style-type: none"> - Use the latest AzureRmWebAppDeployment task to get fallback behavior. - The Publish Profile (PublishProfile) connection type doesn't work, because it uses basic authentication. Change the

Deployment method	When basic authentication is disabled
	<p>connection type to Azure Resource Manager (AzureRM).</p> <ul style="list-style-type: none"> - On non-Windows Pipelines agents, authentication works. - On Windows agents, the deployment method used by the task might need to be modified. When Web Deploy is used (<code>DeploymentType: 'webDeploy'</code>) and basic authentication is disabled, the task authenticates with a Microsoft Entra token. There are additional requirements if you're not using the <code>windows-latest</code> agent or if you're using a self-hosted agent. For more information, see I can't Web Deploy to my Azure App Service using Microsoft Entra authentication from my Windows agent. - Other deployment methods work, such as zip deploy or run from package.

Create a custom role with no permissions for basic authentication

To prevent a lower-privileged user from enabling basic authentication for any app, you can create a custom role and assign the user to the role.

Azure portal

1. In the Azure portal, in the top menu, search for and select the subscription you want to create the custom role in.
2. From the left navigation, select **Access Control (IAM)** > **Add** > **Add custom role**.
3. Set the **Basic** tab as you wish, then select **Next**.
4. In the **Permissions** tab, and select **Exclude permissions**.
5. Find and select **Microsoft Web Apps**, then search for the following operations:

[\[\] Expand table](#)

Operation	Description
<code>microsoft.web/sites/basicPublishingCredentialsPolicies/ftp</code>	FTP publishing credentials for App Service apps.

Operation	Description
microsoft.web/sites/basicPublishingCredentialsPolicies/scm	SCM publishing credentials for App Service apps.
microsoft.web/sites/slots/basicPublishingCredentialsPolicies/ftp	FTP publishing credentials for App Service slots.
microsoft.web/sites/slots/basicPublishingCredentialsPolicies/scm	SCM publishing credentials for App Service slots.

6. Under each of these operations, select the box for **Write**, then select **Add**. This step adds the operation as **NotActions** for the role.

Your Permissions tab should look like the following screenshot:

Create a custom role ...

Basics Permissions Assignable scopes JSON Review + create

+ Add permissions + Exclude permissions

Click Add permissions to select the permissions you want to add to this custom role.
To add a wildcard (*) permission, you must manually add the permission on the JSON tab. [Learn more](#)
To exclude specific permissions from a wildcard permission, click Exclude permissions. [Learn more](#)

Permission	↑↓	Description	↑↓	Permission type	↑
Microsoft.Web/sites/basicPublishingCredentialsPolicies/ftp/Write		Update whether FTP ...		NotAction	
Microsoft.Web/sites/basicPublishingCredentialsPolicies/scm/Write		Update whether SC... ...		NotAction	
Microsoft.Web/sites/slots/basicPublishingCredentialsPolicies/ftp/Write		Update whether FTP ...		NotAction	
Microsoft.Web/sites/slots/basicPublishingCredentialsPolicies/scm/Write		Update whether SC... ...		NotAction	

Definitions

Control plane
Actions specify the operations that are allowed to perform. NotActions specify operations that are excluded from allowed Actions (this is useful if a wildcards).

Data plane
DataActions specify the operation

Review + create Previous Next Feedback

7. Select **Review + create**, then select **Create**.

8. You can now assign this role to your organization's users.

For more information, see [Create or update Azure custom roles using the Azure portal](#)

Monitor for basic authentication attempts

All successful and attempted logins are logged to the Azure Monitor `AppServiceAuditLogs` log type. To audit the attempted and successful logins on FTP and WebDeploy, follow the steps at [Send logs to Azure Monitor](#) and enable shipping of the `AppServiceAuditLogs` log type.

To confirm that the logs are shipped to your selected service(s), try logging in via FTP or WebDeploy. The following example shows a Storage Account log.

```
{  
  "time": "2023-10-16T17:42:32.9322528Z",  
  "ResourceId": "/SUBSCRIPTIONS/aaaa0a0a-bb1b-cc2c-dd3d-  
eeeeee4e4e4e/RESOURCEGROUPS/MYRESOURCEGROUP/PROVIDERS/MICROSOFT.WEB/SITES/MY  
-DEMO-APP",  
  "Category": "AppServiceAuditLogs",  
  "OperationName": "Authorization",  
  "Properties": {  
    "User": "$my-demo-app",  
    "UserDisplayName": "$my-demo-app",  
    "UserAddress": "24.19.191.170",  
    "Protocol": "FTP"  
  }  
}
```

Basic authentication related policies

[Azure Policy](#) can help you enforce organizational standards and to assess compliance at-scale. You can use Azure Policy to audit for any apps that still use basic authentication, and remediate any noncompliant resources. The following are built-in policies for auditing and remediating basic authentication on App Service:

- [Audit policy for FTP ↗](#)
- [Audit policy for SCM ↗](#)
- [Remediation policy for FTP ↗](#)
- [Remediation policy for SCM ↗](#)

The following are corresponding policies for slots:

- [Audit policy for FTP ↗](#)
- [Audit policy for SCM ↗](#)
- [Remediation policy for FTP ↗](#)
- [Remediation policy for SCM ↗](#)

Frequently asked questions

Why do I get a warning in Visual Studio saying that basic authentication is disabled?

Visual Studio requires basic authentication to deploy to Azure App Service. The warning reminds you that the configuration on your app changed and you can no longer deploy to it. Either you disabled basic authentication on the app yourself, or your organization policy enforces that basic authentication is disabled for App Service apps.

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

Encryption at rest using customer-managed keys

Article • 03/24/2022

Encrypting your web app's application data at rest requires an Azure Storage Account and an Azure Key Vault. These services are used when you run your app from a deployment package.

- [Azure Storage provides encryption at rest](#). You can use system-provided keys or your own, customer-managed keys. This is where your application data is stored when it's not running in a web app in Azure.
- [Running from a deployment package](#) is a deployment feature of App Service. It allows you to deploy your site content from an Azure Storage Account using a Shared Access Signature (SAS) URL.
- [Key Vault references](#) are a security feature of App Service. It allows you to import secrets at runtime as application settings. Use this to encrypt the SAS URL of your Azure Storage Account.

Set up encryption at rest

Create an Azure Storage account

First, [create an Azure Storage account](#) and [encrypt it with customer-managed keys](#). Once the storage account is created, use the [Azure Storage Explorer](#) to upload package files.

Next, use the Storage Explorer to [generate an SAS](#).

Note

Save this SAS URL, this is used later to enable secure access of the deployment package at runtime.

Configure running from a package from your storage account

Once you upload your file to Blob storage and have an SAS URL for the file, set the `WEBSITE_RUN_FROM_PACKAGE` application setting to the SAS URL. The following example

does it by using Azure CLI:

```
az webapp config appsettings set --name <app-name> --resource-group <resource-group-name> --settings WEBSITE_RUN_FROM_PACKAGE="<your-SAS-URL>"
```

Adding this application setting causes your web app to restart. After the app has restarted, browse to it and make sure that the app has started correctly using the deployment package. If the application didn't start correctly, see the [Run from package troubleshooting guide](#).

Encrypt the application setting using Key Vault references

Now you can replace the value of the `WEBSITE_RUN_FROM_PACKAGE` application setting with a Key Vault reference to the SAS-encoded URL. This keeps the SAS URL encrypted in Key Vault, which provides an extra layer of security.

1. Use the following [az keyvault create](#) command to create a Key Vault instance.

Azure CLI

```
az keyvault create --name "Contoso-Vault" --resource-group <group-name> --location eastus
```

2. Follow [these instructions to grant your app access](#) to your key vault:

3. Use the following [az keyvault secret set](#) command to add your external URL as a secret in your key vault:

Azure CLI

```
az keyvault secret set --vault-name "Contoso-Vault" --name "external-url" --value "<SAS-URL>"
```

4. Use the following [az webapp config appsettings set](#) command to create the `WEBSITE_RUN_FROM_PACKAGE` application setting with the value as a Key Vault reference to the external URL:

Azure CLI

```
az webapp config appsettings set --settings WEBSITE_RUN_FROM_PACKAGE="@Microsoft.KeyVault(SecretUri=https://Contoso-Vault.vault.azure.net/secrets/external-url/<secret-version>")"
```

The <secret-version> will be in the output of the previous `az keyvault secret set` command.

Updating this application setting causes your web app to restart. After the app has restarted, browse to it make sure it has started correctly using the Key Vault reference.

How to rotate the access token

It is best practice to periodically rotate the SAS key of your storage account. To ensure the web app does not inadvertently lose access, you must also update the SAS URL in Key Vault.

1. Rotate the SAS key by navigating to your storage account in the Azure portal.
Under **Settings > Access keys**, click the icon to rotate the SAS key.
2. Copy the new SAS URL, and use the following command to set the updated SAS URL in your key vault:

Azure CLI

```
az keyvault secret set --vault-name "Contoso-Vault" --name "external-url" --value "<SAS-URL>"
```

3. Update the key vault reference in your application setting to the new secret version:

Azure CLI

```
az webapp config appsettings set --settings  
WEBSITE_RUN_FROM_PACKAGE="@Microsoft.KeyVault(SecretUri=https://Contoso  
-Vault.vault.azure.net/secrets/external-url/<secret-version>")
```

The <secret-version> will be in the output of the previous `az keyvault secret set` command.

How to revoke the web app's data access

There are two methods to revoke the web app's access to the storage account.

Rotate the SAS key for the Azure Storage account

If the SAS key for the storage account is rotated, the web app will no longer have access to the storage account, but it will continue to run with the last downloaded version of the package file. Restart the web app to clear the last downloaded version.

Remove the web app's access to Key Vault

You can revoke the web app's access to the site data by disabling the web app's access to Key Vault. To do this, remove the access policy for the web app's identity. This is the same identity you created earlier while configuring key vault references.

Summary

Your application files are now encrypted at rest in your storage account. When your web app starts, it retrieves the SAS URL from your key vault. Finally, the web app loads the application files from the storage account.

If you need to revoke the web app's access to your storage account, you can either revoke access to the key vault or rotate the storage account keys, which invalidates the SAS URL.

Frequently Asked Questions

Is there any additional charge for running my web app from the deployment package?

Only the cost associated with the Azure Storage Account and any applicable egress charges.

How does running from the deployment package affect my web app?

- Running your app from the deployment package makes `wwwroot/` read-only. Your app receives an error when it attempts to write to this directory.
- TAR and GZIP formats are not supported.
- This feature is not compatible with local cache.

Next steps

- [Key Vault references for App Service](#)

- Azure Storage encryption for data at rest

OS and runtime patching in Azure App Service

Article • 03/09/2023

This article shows you how to get certain version information regarding the OS or software in [App Service](#).

App Service is a Platform-as-a-Service, which means that the OS and application stack are managed for you by Azure; you only manage your application and its data. More control over the OS and application stack is available for you in [Azure Virtual Machines](#). With that in mind, it is nevertheless helpful for you as an App Service user to know more information, such as:

- How and when are OS updates applied?
- How is App Service patched against significant vulnerabilities (such as zero-day)?
- Which OS and runtime versions are running your apps?

For security reasons, certain specifics of security information are not published. However, the article aims to alleviate concerns by maximizing transparency on the process, and how you can stay up-to-date on security-related announcements or runtime updates.

How and when are OS updates applied?

Azure manages OS patching on two levels, the physical servers and the guest virtual machines (VMs) that run the App Service resources. Both are updated monthly, which aligns to the monthly [Patch Tuesday](#) schedule. These updates are applied automatically, in a way that guarantees the high-availability SLA of Azure services.

For detailed information on how updates are applied, see [Demystifying the magic behind App Service OS updates](#).

How does Azure deal with significant vulnerabilities?

When severe vulnerabilities require immediate patching, such as [zero-day vulnerabilities](#), the high-priority updates are handled on a case-by-case basis.

Stay current with critical security announcements in Azure by visiting [Azure Security Blog](#).

When are supported language runtimes updated, added, or deprecated?

New stable versions of supported language runtimes (major, minor, or patch) are periodically added to App Service instances. Some updates overwrite the existing installation, while others are installed side by side with existing versions. An overwrite installation means that your app automatically runs on the updated runtime. A side-by-side installation means you must manually migrate your app to take advantage of a new runtime version. For more information, see one of the subsections.

ⓘ Note

Information here applies to language runtimes that are built into an App Service app. A custom runtime you upload to App Service, for example, remains unchanged unless you manually upgrade it.

New patch updates

Patch updates to .NET, PHP, Java SDK, or Tomcat version are applied automatically by overwriting the existing installation with the latest version. Node.js patch updates are installed side by side with the existing versions (similar to major and minor versions in the next section). New Python patch versions can be installed manually through [site extensions](#), side by side with the built-in Python installations.

New major and minor versions

When a new major or minor version is added, it is installed side by side with the existing versions. You can manually upgrade your app to the new version. If you configured the runtime version in a configuration file (such as `web.config` and `package.json`), you need to upgrade with the same method. If you used an App Service setting to configure your runtime version, you can change it in the [Azure portal](#) or by running an [Azure CLI](#) command in the [Cloud Shell](#), as shown in the following examples:

Azure CLI

```
az webapp config set --net-framework-version v4.7 --resource-group <groupname> --name <appname>
az webapp config set --php-version 7.0 --resource-group <groupname> --name <appname>
az webapp config appsettings set --settings WEBSITE_NODE_DEFAULT_VERSION=~14 --resource-group <groupname> --name <appname>
```

```
az webapp config set --python-version 3.8 --resource-group <groupname> --name <appname>
az webapp config set --java-version 1.8 --java-container Tomcat --java-container-version 9.0 --resource-group <groupname> --name <appname>
```

ⓘ Note

This example uses the recommended "tilde syntax" to target the latest available version of Node.js 16 runtime on Windows App Service.

How can I query OS and runtime update status on my instances?

While critical OS information is locked down from access (see [Operating system functionality on Azure App Service](#)), the [Kudu console](#) enables you to query your App Service instance regarding the OS version and runtime versions.

The following table shows how to the versions of Windows and of the language runtime that are running your apps:

Information	Where to find it
Windows version	See <a href="https://<appname>.scm.azurewebsites.net/Env.cshtml">https://<appname>.scm.azurewebsites.net/Env.cshtml (under System info)
.NET version	At <a href="https://<appname>.scm.azurewebsites.net/DebugConsole">https://<appname>.scm.azurewebsites.net/DebugConsole , run the following command in the command prompt: <code>reg query "HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\NET Framework Setup\NDP\v4\Full"</code>
.NET Core version	At <a href="https://<appname>.scm.azurewebsites.net/DebugConsole">https://<appname>.scm.azurewebsites.net/DebugConsole , run the following command in the command prompt: <code>dotnet --version</code>
PHP version	At <a href="https://<appname>.scm.azurewebsites.net/DebugConsole">https://<appname>.scm.azurewebsites.net/DebugConsole , run the following command in the command prompt: <code>php --version</code>
Default Node.js version	In the Cloud Shell , run the following command: <code>az webapp config appsettings list --resource-group <groupname> --name <appname> --query "[?name=='WEBSITE_NODE_DEFAULT_VERSION']"</code>
Python version	At <a href="https://<appname>.scm.azurewebsites.net/DebugConsole">https://<appname>.scm.azurewebsites.net/DebugConsole , run the following command in the command prompt: <code>python --version</code>

Information Where to find it

Java version At <https://<appname>.scm.azurewebsites.net/DebugConsole>, run the following command in the command prompt:
`java -version`

ⓘ Note

Access to registry location

`HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Component Based Servicing\Packages`, where information on "KB" patches is stored, is locked down.

More resources

[Trust Center: Security ↗](#)

[64 bit ASP.NET Core on Azure App Service ↗](#)

Name resolution (DNS) in App Service

Article • 01/17/2024

Your app uses DNS when making calls to dependent resources. Resources could be Azure services such as Key Vault, Storage or Azure SQL, but it could also be web APIs that your app depends on. When you want to make a call to for example *myservice.com*, you're using DNS to resolve the name to an IP. This article describes how App Service is handling name resolution and how it determines what DNS servers to use. The article also describes settings you can use to configure DNS resolution.

How name resolution works in App Service

If you aren't integrating your app with a virtual network and custom DNS servers aren't configured, your app uses [Azure DNS](#). If you integrate your app with a virtual network, your app uses the DNS configuration of the virtual network. The default for virtual network is also to use Azure DNS. Through the virtual network, it's also possible to link to [Azure DNS private zones](#) and use that for private endpoint resolution or private domain name resolution.

If you configured your virtual network with a list of custom DNS servers, name resolution uses these servers. If your virtual network is using custom DNS servers and you're using private endpoints, you should read [this article](#) carefully. You also need to consider that your custom DNS servers are able to resolve any public DNS records used by your app. Your DNS configuration needs to either forward requests to a public DNS server, include a public DNS server like Azure DNS in the list of custom DNS servers or specify an alternative server at the app level.

When your app needs to resolve a domain name using DNS, the app sends a name resolution request to all configured DNS servers. If the first server in the list returns a response within the timeout limit, you get the result returned immediately. If not, the app waits for the other servers to respond within the timeout period and evaluates the DNS server responses in the order you configured the servers. If none of the servers respond within the timeout and you configured retry, you repeat the process.

Configuring DNS servers

The individual app allows you to override the DNS configuration by specifying the `dnsServers` property in the `dnsConfiguration` site property object. You can specify up to five custom DNS servers. You can configure custom DNS servers using the Azure CLI:

Azure CLI

```
az resource update --resource-group <group-name> --name <app-name> --  
resource-type "Microsoft.Web/sites" --set  
properties.dnsConfiguration.dnsServers="['168.63.129.16','xxx.xxx.xxx.xxx']"
```

DNS app settings

App Service has existing app settings to configure DNS servers and name resolution behavior. Site properties override the app settings if both exist. Site properties have the advantage of being auditable with Azure Policy and validated at the time of configuration. We recommend you to use site properties.

You can still use the existing `WEBSITE_DNS_SERVER` app setting, and you can add custom DNS servers with either setting. If you want to add multiple DNS servers using the app setting, you must separate the servers by commas with no blank spaces added.

Using the app setting `WEBSITE_DNS_ALT_SERVER`, you append the specific DNS server to the list of DNS servers configured. The alternative DNS server is appended to both explicitly configured DNS servers and DNS servers inherited from the virtual network.

App settings also exist for configuring name resolution behavior and are named `WEBSITE_DNS_MAX_CACHE_TIMEOUT`, `WEBSITE_DNS_TIMEOUT` and `WEBSITE_DNS_ATTEMPTS`.

Configure name resolution behavior

If you require fine-grained control over name resolution, App Service allows you to modify the default behavior. You can modify retry attempts, retry timeout and cache timeout. Changing behavior like disabling or lowering cache duration can influence performance.

[] Expand table

Property name	Windows default value	Linux default value	Allowed values	Description
dnsRetryAttemptCount	1	5	1-5	Defines the number of attempts to resolve where one means no retries.
dnsMaxCacheTimeout	30	0	0-60	DNS results are cached according to the individual

Property name	Windows default value	Linux default value	Allowed values	Description
				records TTL, but no longer than the defined max cache timeout. Setting cache to zero means caching is disabled.
dnsRetryAttemptTimeout	3	1	1-30	Timeout before retrying or failing. Timeout also defines the time to wait for secondary server results if the primary doesn't respond.

ⓘ Note

- Changing name resolution behavior is not supported on Windows Container apps.
- To configure `dnsMaxCacheTimeout`, you need to ensure that caching is enabled by adding the app setting `WEBSITE_ENABLE_DNS_CACHE="true"`. If you enable caching, but do not configure `dnsMaxCacheTimeout`, the timeout will be set to 30.

Configure the name resolution behavior by using these CLI commands:

Azure CLI

```
az resource update --resource-group <group-name> --name <app-name> --set properties.dnsConfiguration.dnsMaxCacheTimeout=[0-60] --resource-type "Microsoft.Web/sites"
az resource update --resource-group <group-name> --name <app-name> --set properties.dnsConfiguration.dnsRetryAttemptCount=[1-5] --resource-type "Microsoft.Web/sites"
az resource update --resource-group <group-name> --name <app-name> --set properties.dnsConfiguration.dnsRetryAttemptTimeout=[1-30] --resource-type "Microsoft.Web/sites"
```

Validate the settings by using this CLI command:

Azure CLI

```
az resource show --resource-group <group-name> --name <app-name> --query properties.dnsConfiguration --resource-type "Microsoft.Web/sites"
```

Next steps

- Configure virtual network integration
- Name resolution for resources in Azure virtual networks
- General networking overview

Mitigating subdomain takeovers in Azure App Service

Article • 06/06/2023

Subdomain takeovers are a common threat for organizations that regularly create and delete many resources. A subdomain takeover can occur when you have a DNS record that points to a deprovisioned Azure resource. Such DNS records are also known as "dangling DNS" entries. Subdomain takeovers enable malicious actors to redirect traffic intended for an organization's domain to a site performing malicious activity.

The risks of subdomain takeover include:

- Loss of control over the content of the subdomain
- Cookie harvesting from unsuspecting visitors
- Phishing campaigns
- Further risks of classic attacks such as XSS, CSRF, CORS bypass

Learn more about Subdomain Takeover at [Dangling DNS and subdomain takeover](#).

Azure App Service provides [Name Reservation Service](#) and [domain verification tokens](#) to prevent subdomain takeovers.

How App Service prevents subdomain takeovers

Upon deletion of an App Service app or App Service Environment (ASE), immediate reuse of the corresponding DNS is forbidden except for subscriptions belonging to the tenant of the subscription that originally owned the DNS. Thus, the customer is afforded some time to either clean-up any associations/pointers to the said DNS or reclaim the DNS in Azure by recreating the resource with the same name. This behavior is enabled by default on Azure App Service for "`*.azurewebsites.net`" and "`*.appserviceenvironment.net`" resources, so it doesn't require any customer configuration.

Example scenario

Subscription 'A' and subscription 'B' are the only subscriptions belonging to tenant 'AB'. Subscription 'A' contains an App Service web app 'test' with DNS name '`test.azurewebsites.net`'. Upon deletion of the app, only subscription 'A' or subscription

'B' will be able to immediately reuse the DNS name 'test.azurewebsites.net' by creating a web app named 'test'. No other subscriptions will be allowed to claim the name right after the resource deletion.

How you can prevent subdomain takeovers

When creating DNS entries for Azure App Service, create an asuid.{subdomain} TXT record with the Domain Verification ID. When such a TXT record exists, no other Azure Subscription can validate the Custom Domain or take it over unless they add their token verification ID to the DNS entries.

These records prevent the creation of another App Service app using the same name from your CNAME entry. Without the ability to prove ownership of the domain name, threat actors can't receive traffic or control the content.

DNS records should be updated before the site deletion to ensure bad actors can't take over the domain between the period of deletion and re-creation.

To get a domain verification ID, see the [Map a custom domain tutorial](#)

Azure security baseline for App Service

Article • 09/20/2023

This security baseline applies guidance from the [Microsoft cloud security benchmark version 1.0](#) to App Service. The Microsoft cloud security benchmark provides recommendations on how you can secure your cloud solutions on Azure. The content is grouped by the security controls defined by the Microsoft cloud security benchmark and the related guidance applicable to App Service.

You can monitor this security baseline and its recommendations using Microsoft Defender for Cloud. Azure Policy definitions will be listed in the Regulatory Compliance section of the Microsoft Defender for Cloud portal page.

When a feature has relevant Azure Policy Definitions, they are listed in this baseline to help you measure compliance with the Microsoft cloud security benchmark controls and recommendations. Some recommendations may require a paid Microsoft Defender plan to enable certain security scenarios.

ⓘ Note

Features not applicable to App Service have been excluded. To see how App Service completely maps to the Microsoft cloud security benchmark, see the [full App Service security baseline mapping file](#).

Security profile

The security profile summarizes high-impact behaviors of App Service, which may result in increased security considerations.

Service Behavior Attribute	Value
Product Category	Compute, Web
Customer can access HOST / OS	No Access
Service can be deployed into customer's virtual network	True
Stores customer content at rest	True

Network security

For more information, see the [Microsoft cloud security benchmark: Network security](#).

NS-1: Establish network segmentation boundaries

Features

Virtual Network Integration

Description: Service supports deployment into customer's private Virtual Network (VNet). [Learn more](#).

Supported	Enabled By Default	Configuration Responsibility
True	False	Customer

Feature notes: Virtual Network Integration is configured by default when using App Service Environments but must be configured manually when using the public multi-tenant offering.

Configuration Guidance: Ensure a stable IP for outbound communications towards internet addresses: You can provide a stable outbound IP by using the Virtual Network integration feature. This allows the receiving party to allow-list based on IP, should that be needed.

When using App Service in the Isolated pricing tier, also called an App Service Environment (ASE), you can deploy directly into a subnet within your Azure Virtual Network. Use network security groups to secure your Azure App Service Environment by blocking inbound and outbound traffic to resources in your virtual network, or to restrict access to apps in an App Service Environment.

In the multi-tenant App Service (an app not in Isolated tier), enable your apps to access resources in or through a Virtual Network with the Virtual Network Integration feature. You can then use network security groups to control outbound traffic from your app. When using Virtual Network Integration, you can enable the 'Route All' configuration to make all outbound traffic subject to network security groups and user-defined routes on the integration subnet. This feature can also be used to block outbound traffic to public addresses from the app. Virtual Network Integration cannot be used to provide inbound access to an app.

For communications towards Azure Services often there's no need to depend on the IP address and mechanics like Service Endpoints should be used instead.

Note: For App Service Environments, by default, network security groups include an implicit deny rule at the lowest priority and requires you to add explicit allow rules. Add allow rules for your network security group based on a least privileged networking approach. The underlying virtual machines that are used to host the App Service Environment are not directly accessible because they are in a Microsoft-managed subscription.

When using Virtual Network Integration feature with virtual networks in the same region, use network security groups and route tables with user-defined routes. User-defined routes can be placed on the integration subnet to send outbound traffic as intended.

Reference: [Integrate your app with an Azure virtual network](#)

Network Security Group Support

Description: Service network traffic respects Network Security Groups rule assignment on its subnets. [Learn more](#).

Supported	Enabled By Default	Configuration Responsibility
True	True	Microsoft

Feature notes: Network Security Group support is available for all customers using App Service Environments but is only available on VNet integrated apps for customers using the public multi-tenant offering.

Configuration Guidance: No additional configurations are required as this is enabled on a default deployment.

Reference: [App Service Environment networking](#)

NS-2: Secure cloud services with network controls

Features

Azure Private Link

Description: Service native IP filtering capability for filtering network traffic (not to be confused with NSG or Azure Firewall). [Learn more](#).

Supported	Enabled By Default	Configuration Responsibility
True	False	Customer

Configuration Guidance: Use private endpoints for your Azure Web Apps to allow clients located in your private network to securely access the apps over Private Link. The private endpoint uses an IP address from your Azure VNet address space. Network traffic between a client on your private network and the Web App traverses over the VNet and a Private Link on the Microsoft backbone network, eliminating exposure from the public Internet.

Note: Private Endpoint is only used for incoming flows to your Web App. Outgoing flows won't use this Private Endpoint. You can inject outgoing flows to your network in a different subnet through the VNet integration feature. The use of private endpoints for services on the receiving end of App Service traffic avoids SNAT from happening and provides a stable outbound IP range.

Additional Guidance: If running containers on App Service which are stored in Azure Container Registry (ACR) ensure those images are pulled over a private network. Do this by configuring a private endpoint on the ACR storing those images in conjunction with setting the "WEBSITE_PULL_IMAGE_OVER_VNET" application setting on your web application.

Reference: [Using Private Endpoints for Azure Web App](#)

Disable Public Network Access

Description: Service supports disabling public network access either through using service-level IP ACL filtering rule (not NSG or Azure Firewall) or using a 'Disable Public Network Access' toggle switch. [Learn more](#).

Supported	Enabled By Default	Configuration Responsibility
True	False	Customer

Configuration Guidance: Disable Public Network Access' using either service-level IP ACL filtering rules or private endpoints or by setting the `publicNetworkAccess` property to disabled in ARM.

Reference: [Set up Azure App Service access restrictions](#)

NS-5: Deploy DDOS protection

Other guidance for NS-5

Enable DDOS Protection Standard on the virtual network hosting your App Service's Web Application Firewall. Azure provides DDoS Basic protection on its network, which can be improved with intelligent DDoS Standard capabilities which learns about normal traffic patterns and can detect unusual behavior. DDoS Standard applies to a Virtual Network so it must be configured for the network resource in front of the app, such as Application Gateway or an NVA.

NS-6: Deploy web application firewall

Other guidance for NS-6

Avoid WAF being bypassed for your applications. Make sure the WAF cannot be bypassed by locking down access to only the WAF. Use a combination of Access Restrictions, Service Endpoints and Private Endpoints.

Additionally, protect an App Service Environment by routing traffic through a Web Application Firewall (WAF) enabled Azure Application Gateway or Azure Front Door.

For the multi-tenant offering, secure inbound traffic to your app with:

- Access Restrictions: a series of allow or deny rules that control inbound access
- Service Endpoints: can deny inbound traffic from outside of specified virtual networks or subnets
- Private Endpoints: expose your app to your Virtual Network with a private IP address. With the Private Endpoints enabled on your app, it is no longer internet-accessible

Consider implementing an Azure Firewall to centrally create, enforce, and log application and network connectivity policies across your subscriptions and virtual networks. Azure Firewall uses a static public IP address for virtual network resources, which allows outside firewalls to identify traffic that originates from your virtual network.

Identity management

For more information, see the [Microsoft cloud security benchmark: Identity management](#).

IM-1: Use centralized identity and authentication system

Features

Azure AD Authentication Required for Data Plane Access

Description: Service supports using Azure AD authentication for data plane access.

[Learn more.](#)

Supported	Enabled By Default	Configuration Responsibility
True	False	Customer

Configuration Guidance: For authenticated web applications, only use well-known established identity providers to authenticate and authorize user access. In case your app should only be accessed by users of your own organization, or otherwise your users are all using Azure Active Directory (Azure AD), configure Azure AD as the default authentication method to control your data plane access.

Reference: [Authentication and authorization in Azure App Service and Azure Functions](#)

Local Authentication Methods for Data Plane Access

Description: Local authentications methods supported for data plane access, such as a local username and password. [Learn more.](#)

Supported	Enabled By Default	Configuration Responsibility
True	False	Customer

Feature notes: Avoid the usage of local authentication methods or accounts, these should be disabled wherever possible. Instead use Azure AD to authenticate where possible.

Configuration Guidance: Restrict the use of local authentication methods for data plane access. Instead, use Azure Active Directory (Azure AD) as the default authentication method to control your data plane access.

Reference: [Authentication and authorization in Azure App Service and Azure Functions](#)

IM-3: Manage application identities securely and automatically

Features

Managed Identities

Description: Data plane actions support authentication using managed identities. [Learn more.](#)

Supported	Enabled By Default	Configuration Responsibility
True	False	Customer

Configuration Guidance: Use Azure managed identities instead of service principals when possible, which can authenticate to Azure services and resources that support Azure Active Directory (Azure AD) authentication. Managed identity credentials are fully managed, rotated, and protected by the platform, avoiding hard-coded credentials in source code or configuration files.

A common scenario to use a managed identity with App Service is to access other Azure PaaS services such as Azure SQL Database, Azure Storage, or Key Vault.

Reference: [How to use managed identities for App Service and Azure Functions](#)

Service Principals

Description: Data plane supports authentication using service principals. [Learn more.](#)

Supported	Enabled By Default	Configuration Responsibility
True	False	Customer

Additional Guidance: Though service principals are supported by the service as a pattern for authentication, we recommend using Managed Identities where possible instead.

Microsoft Defender for Cloud monitoring

Azure Policy built-in definitions - Microsoft.Web:

Name (Azure portal)	Description	Effect(s)	Version (GitHub)
App Service apps should use managed identity ↗	Use a managed identity for enhanced authentication security	AuditIfNotExists, Disabled	3.0.0 ↗
Function apps should use managed identity ↗	Use a managed identity for enhanced authentication security	AuditIfNotExists, Disabled	3.0.0 ↗

IM-7: Restrict resource access based on conditions

Features

Conditional Access for Data Plane

Description: Data plane access can be controlled using Azure AD Conditional Access Policies. [Learn more.](#)

Supported	Enabled By Default	Configuration Responsibility
True	False	Customer

Configuration Guidance: Define the applicable conditions and criteria for Azure Active Directory (Azure AD) conditional access in the workload. Consider common use cases such as blocking or granting access from specific locations, blocking risky sign-in behavior, or requiring organization-managed devices for specific applications.

IM-8: Restrict the exposure of credential and secrets

Features

Service Credential and Secrets Support Integration and Storage in Azure Key Vault

Description: Data plane supports native use of Azure Key Vault for credential and secrets store. [Learn more.](#)

Supported	Enabled By Default	Configuration Responsibility
True	False	Customer

Configuration Guidance: Ensure that app secrets and credentials are stored in secure locations such as Azure Key Vault, instead of embedding them into code or configuration files. Use a managed identity on your app to then access credentials, or secrets stored in Key Vault in a secure fashion.

Reference: [Use Key Vault references for App Service and Azure Functions](#)

Privileged access

For more information, see the [Microsoft cloud security benchmark: Privileged access](#).

PA-7: Follow just enough administration (least privilege) principle

Features

Azure RBAC for Data Plane

Description: Azure Role-Based Access Control (Azure RBAC) can be used to manage access to service's data plane actions. [Learn more](#).

Supported	Enabled By Default	Configuration Responsibility
False	Not Applicable	Not Applicable

Configuration Guidance: This feature is not supported to secure this service.

PA-8: Determine access process for cloud provider support

Features

Customer Lockbox

Description: Customer Lockbox can be used for Microsoft support access. [Learn more](#).

Supported	Enabled By Default	Configuration Responsibility
True	False	Customer

Configuration Guidance: In support scenarios where Microsoft needs to access your data, use Customer Lockbox to review, then approve or reject each of Microsoft's data access requests.

Data protection

For more information, see the [Microsoft cloud security benchmark: Data protection](#).

DP-1: Discover, classify, and label sensitive data

Features

Sensitive Data Discovery and Classification

Description: Tools (such as Azure Purview or Azure Information Protection) can be used for data discovery and classification in the service. [Learn more.](#)

Supported	Enabled By Default	Configuration Responsibility
False	Not Applicable	Not Applicable

Feature notes: Implement Credential Scanner in your build pipeline to identify credentials within code. Credential Scanner will also encourage moving discovered credentials to more secure locations such as Azure Key Vault.

Configuration Guidance: This feature is not supported to secure this service.

DP-2: Monitor anomalies and threats targeting sensitive data

Features

Data Leakage/Loss Prevention

Description: Service supports DLP solution to monitor sensitive data movement (in customer's content). [Learn more.](#)

Supported	Enabled By Default	Configuration Responsibility
False	Not Applicable	Not Applicable

Feature notes: While data identification, classification, and loss prevention features are not yet available for App Service, you can reduce the data exfiltration risk from the virtual network by removing all rules where the destination uses a 'tag' for Internet or Azure services.

Microsoft manages the underlying infrastructure for App Service and has implemented strict controls to prevent the loss or exposure of your data.

Use tags to assist in tracking App Service resources that store or process sensitive information.

Configuration Guidance: This feature is not supported to secure this service.

DP-3: Encrypt sensitive data in transit

Features

Data in Transit Encryption

Description: Service supports data in-transit encryption for data plane. [Learn more.](#)

Supported	Enabled By Default	Configuration Responsibility
True	False	Customer

Configuration Guidance: Use and enforce the default minimum version of TLS v1.2, configured in TLS/SSL settings, for encrypting all information in transit. Also ensure that all HTTP connection requests are redirected to HTTPS.

Reference: [Add a TLS/SSL certificate in Azure App Service](#)

Microsoft Defender for Cloud monitoring

Azure Policy built-in definitions - Microsoft.Web:

Name (Azure portal)	Description	Effect(s)	Version (GitHub)
App Service apps should only be accessible over HTTPS ↗	Use of HTTPS ensures server/service authentication and protects data in transit from network layer eavesdropping attacks.	Audit, Disabled, Deny	4.0.0 ↗
App Service apps should require FTPS only ↗	Enable FTPS enforcement for enhanced security.	AuditIfNotExists, Disabled	3.0.0 ↗
App Service apps should use the latest TLS version ↗	Periodically, newer versions are released for TLS either due to security flaws, include additional functionality, and enhance speed. Upgrade to the latest TLS version for App Service apps to take advantage of security fixes, if any, and/or new functionalities of the latest version.	AuditIfNotExists, Disabled	2.0.1 ↗

Name (Azure portal)	Description	Effect(s)	Version (GitHub)
Function apps should only be accessible over HTTPS ↴	Use of HTTPS ensures server/service authentication and protects data in transit from network layer eavesdropping attacks.	Audit, Disabled, Deny	5.0.0 ↴
Function apps should require FTPS only ↴	Enable FTPS enforcement for enhanced security.	AuditIfNotExists, Disabled	3.0.0 ↴
Function apps should use the latest TLS version ↴	Periodically, newer versions are released for TLS either due to security flaws, include additional functionality, and enhance speed. Upgrade to the latest TLS version for Function apps to take advantage of security fixes, if any, and/or new functionalities of the latest version.	AuditIfNotExists, Disabled	2.0.1 ↴

DP-4: Enable data at rest encryption by default

Features

Data at Rest Encryption Using Platform Keys

Description: Data at-rest encryption using platform keys is supported, any customer content at rest is encrypted with these Microsoft managed keys. [Learn more](#).

Supported	Enabled By Default	Configuration Responsibility
True	True	Microsoft

Feature notes: Web site content in an App Service app, such as files, are stored in Azure Storage, which automatically encrypts the content at rest. Choose to store application secrets in Key Vault and retrieve them at runtime.

Customer supplied secrets are encrypted at rest while stored in App Service configuration databases.

Note that while locally attached disks can be used optionally by websites as temporary storage, (for example, D:\local and %TMP%), they are only encrypted at rest in the public multi-tenant App Service offering where the Pv3 SKU may be used. For older public multi-tenant scale units where the Pv3 SKU is unavailable, the customer must create a new resource group and redeploy their resources there.

Additionally, the customer has the option to run their application in App Service directly from a ZIP package. For more information, please visit: [Run your app in Azure App Service directly from a ZIP package](#).

Configuration Guidance: No additional configurations are required as this is enabled on a default deployment.

DP-5: Use customer-managed key option in data at rest encryption when required

Features

Data at Rest Encryption Using CMK

Description: Data at-rest encryption using customer-managed keys is supported for customer content stored by the service. [Learn more](#).

Supported	Enabled By Default	Configuration Responsibility
True	False	Customer

Configuration Guidance: If required for regulatory compliance, define the use case and service scope where encryption using customer-managed keys are needed. Enable and implement data at rest encryption using customer-managed key for those services.

Note: Web site content in an App Service app, such as files, are stored in Azure Storage, which automatically encrypts the content at rest. Choose to store application secrets in Key Vault and retrieve them at runtime.

Customer supplied secrets are encrypted at rest while stored in App Service configuration databases.

Note that while locally attached disks can be used optionally by websites as temporary storage, (for example, D:\local and %TMP%), they are not encrypted at rest.

Reference: [Encryption at rest using customer-managed keys](#)

DP-6: Use a secure key management process

Features

Key Management in Azure Key Vault

Description: The service supports Azure Key Vault integration for any customer keys, secrets, or certificates. [Learn more.](#)

Supported	Enabled By Default	Configuration Responsibility
True	False	Customer

Configuration Guidance: Use Azure Key Vault to create and control the life cycle of your encryption keys, including key generation, distribution, and storage. Rotate and revoke your keys in Azure Key Vault and your service based on a defined schedule or when there is a key retirement or compromise. When there is a need to use customer-managed key (CMK) in the workload, service, or application level, ensure you follow the best practices for key management: Use a key hierarchy to generate a separate data encryption key (DEK) with your key encryption key (KEK) in your key vault. Ensure keys are registered with Azure Key Vault and referenced via key IDs from the service or application. If you need to bring your own key (BYOK) to the service (such as importing HSM-protected keys from your on-premises HSMs into Azure Key Vault), follow recommended guidelines to perform initial key generation and key transfer.

Reference: [Use Key Vault references for App Service and Azure Functions](#)

DP-7: Use a secure certificate management process

Features

Certificate Management in Azure Key Vault

Description: The service supports Azure Key Vault integration for any customer certificates. [Learn more.](#)

Supported	Enabled By Default	Configuration Responsibility
True	False	Customer

Configuration Guidance: App Service can be configured with SSL/TLS and other certificates, which can be configured directly on App Service or referenced from Key Vault. To ensure central management of all certificates and secrets, store any certificates used by App Service in Key Vault instead of deploying them locally on App Service directly. When this is configured App Service will automatically download the latest certificate from Azure Key Vault. Ensure the certificate generation follows defined

standards without using any insecure properties, such as: insufficient key size, overly long validity period, insecure cryptography. Setup automatic rotation of the certificate in Azure Key Vault based on a defined schedule or when there is a certificate expiration.

Reference: [Add a TLS/SSL certificate in Azure App Service](#)

Asset management

For more information, see the [Microsoft cloud security benchmark: Asset management](#).

AM-2: Use only approved services

Features

Azure Policy Support

Description: Service configurations can be monitored and enforced via Azure Policy.

[Learn more.](#)

Supported	Enabled By Default	Configuration Responsibility
True	False	Customer

Configuration Guidance: Use Microsoft Defender for Cloud to configure Azure Policy to audit and enforce configurations of your Azure resources. Use Azure Monitor to create alerts when there is a configuration deviation detected on the resources. Use Azure Policy [deny] and [deploy if not exists] effects to enforce secure configuration across Azure resources.

Note: Define and implement standard security configurations for your App Service deployed apps with Azure Policy. Use built-in Azure Policy definitions as well as Azure Policy aliases in the "Microsoft.Web" namespace to create custom policies to alert, audit, and enforce system configurations. Develop a process and pipeline for managing policy exceptions.

Reference: [Azure Policy Regulatory Compliance controls for Azure App Service](#)

AM-4: Limit access to asset management

Other guidance for AM-4

Isolate systems that process sensitive information. To do so, use separate App Service Plans or App Service Environments and consider the use of different subscriptions or management groups.

Logging and threat detection

For more information, see the [Microsoft cloud security benchmark: Logging and threat detection](#).

LT-1: Enable threat detection capabilities

Features

Microsoft Defender for Service / Product Offering

Description: Service has an offering-specific Microsoft Defender solution to monitor and alert on security issues. [Learn more](#).

Supported	Enabled By Default	Configuration Responsibility
True	False	Customer

Configuration Guidance: Use Microsoft Defender for App Service to identify attacks targeting applications running over App Service. When you enable Microsoft Defender for App Service, you immediately benefit from the following services offered by this Defender plan:

- **Secure:** Defender for App Service assesses the resources covered by your App Service plan and generates security recommendations based on its findings. Use the detailed instructions in these recommendations to harden your App Service resources.
- **Detect:** Defender for App Service detects a multitude of threats to your App Service resources by monitoring the VM instance in which your App Service is running and its management interface, the requests and responses sent to and from your App Service apps, the underlying sandboxes and VMs, and App Service internal logs.

Reference: [Protect your web apps and APIs](#)

LT-4: Enable logging for security investigation

Features

Azure Resource Logs

Description: Service produces resource logs that can provide enhanced service-specific metrics and logging. The customer can configure these resource logs and send them to their own data sink like a storage account or log analytics workspace. [Learn more.](#)

Supported	Enabled By Default	Configuration Responsibility
True	False	Customer

Configuration Guidance: Enable resource logs for your web apps on App Service.

Reference: [Enable diagnostics logging for apps in Azure App Service](#)

Posture and vulnerability management

For more information, see the [Microsoft cloud security benchmark: Posture and vulnerability management](#).

PV-2: Audit and enforce secure configurations

Other guidance for PV-2

Turn off remote debugging, remote debugging must not be turned on for production workloads as this opens additional ports on the service which increases the attack surface.

Microsoft Defender for Cloud monitoring

Azure Policy built-in definitions - Microsoft.Web:

Name (Azure portal)	Description	Effect(s)	Version (GitHub)
App Service apps should have 'Client Certificates (Incoming client certificates)' enabled ↗	Client certificates allow for the app to request a certificate for incoming requests. Only clients that have a valid certificate will be able to reach the app.	Audit, Disabled	3.0.0 ↗

Name (Azure portal)	Description	Effect(s)	Version (GitHub)
App Service apps should have remote debugging turned off ↗	Remote debugging requires inbound ports to be opened on an App Service app. Remote debugging should be turned off.	AuditIfNotExists, Disabled	2.0.0 ↗
App Service apps should not have CORS configured to allow every resource to access your apps ↗	Cross-Origin Resource Sharing (CORS) should not allow all domains to access your app. Allow only required domains to interact with your app.	AuditIfNotExists, Disabled	2.0.0 ↗
Function apps should have 'Client Certificates (Incoming client certificates)' enabled ↗	Client certificates allow for the app to request a certificate for incoming requests. Only clients with valid certificates will be able to reach the app.	Audit, Disabled	3.0.0 ↗
Function apps should have remote debugging turned off ↗	Remote debugging requires inbound ports to be opened on Function apps. Remote debugging should be turned off.	AuditIfNotExists, Disabled	2.0.0 ↗
Function apps should not have CORS configured to allow every resource to access your apps ↗	Cross-Origin Resource Sharing (CORS) should not allow all domains to access your Function app. Allow only required domains to interact with your Function app.	AuditIfNotExists, Disabled	2.0.0 ↗

PV-7: Conduct regular red team operations

Other guidance for PV-7

Conduct regular penetration test on your web applications following the [penetration testing rules of engagement](#) ↗ .

Backup and recovery

For more information, see the [Microsoft cloud security benchmark: Backup and recovery](#).

BR-1: Ensure regular automated backups

Features

Azure Backup

Description: The service can be backed up by the Azure Backup service. [Learn more.](#)

Supported	Enabled By Default	Configuration Responsibility
True	False	Customer

Configuration Guidance: Where possible, implement stateless application design to simplify recovery and backup scenarios with App Service.

If you really do need to maintain a stateful application, enable the Backup and Restore feature in App Service which 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. Ensure that regular and automated back-ups occur at a frequency as defined by your organizational policies.

Note: App Service can back up the following information to an Azure storage account and container, which you have configured your app to use:

- App configuration
- File content
- Database connected to your app

Reference: [Back up your app in Azure](#)

Service Native Backup Capability

Description: Service supports its own native backup capability (if not using Azure Backup). [Learn more.](#)

Supported	Enabled By Default	Configuration Responsibility
False	Not Applicable	Not Applicable

Configuration Guidance: This feature is not supported to secure this service.

DevOps security

For more information, see the [Microsoft cloud security benchmark: DevOps security](#).

DS-6: Enforce security of workload throughout DevOps lifecycle

Other guidance for DS-6

Deploy code to App Service from a controlled and trusted environment, like a well-managed and secured DevOps deployment pipeline. This avoids code that was not version controlled and verified to be deployed from a malicious host.

Next steps

- See the [Microsoft cloud security benchmark overview](#)
- Learn more about [Azure security baselines](#)

Azure Policy Regulatory Compliance controls for Azure App Service

Article • 10/22/2024

Regulatory Compliance in Azure Policy provides Microsoft created and managed initiative definitions, known as *built-ins*, for the **compliance domains** and **security controls** related to different compliance standards. This page lists the **compliance domains** and **security controls** for Azure App Service. You can assign the built-ins for a **security control** individually to help make your Azure resources compliant with the specific standard.

The title of each built-in policy definition links to the policy definition in the Azure portal. Use the link in the **Policy Version** column to view the source on the [Azure Policy GitHub repo](#).

Important

Each control is associated with one or more [Azure Policy](#) definitions. These policies might help you [assess compliance](#) with the control. However, there often isn't a one-to-one or complete match between a control and one or more policies. As such, **Compliant** in Azure Policy refers only to the policies themselves. This doesn't ensure that you're fully compliant with all requirements of a control. In addition, the compliance standard includes controls that aren't addressed by any Azure Policy definitions at this time. Therefore, compliance in Azure Policy is only a partial view of your overall compliance status. The associations between controls and Azure Policy Regulatory Compliance definitions for these compliance standards can change over time.

Australian Government ISM PROTECTED

To review how the available Azure Policy built-ins for all Azure services map to this compliance standard, see [Azure Policy Regulatory Compliance - Australian Government ISM PROTECTED](#). For more information about this compliance standard, see [Australian Government ISM PROTECTED](#).

 Expand table

Domain	Control ID	Control title	Policy (Azure portal)	Policy version (GitHub)
Guidelines for Cryptography - Transport Layer Security	1139	Using Transport Layer Security - 1139	App Service apps should use the latest TLS version ↗	2.1.0 ↗
Guidelines for Cryptography - Transport Layer Security	1139	Using Transport Layer Security - 1139	Function apps should use the latest TLS version ↗	2.1.0 ↗
Guidelines for System Management - System administration	1386	Restriction of management traffic flows - 1386	App Service apps should have remote debugging turned off ↗	2.0.0 ↗
Guidelines for System Management - System administration	1386	Restriction of management traffic flows - 1386	Function apps should have remote debugging turned off ↗	2.0.0 ↗
Guidelines for Software Development - Web application development	1424	Web browser-based security controls - 1424	App Service apps should not have CORS configured to allow every resource to access your apps ↗	2.0.0 ↗
Guidelines for Software Development - Web application development	1552	Web application interactions - 1552	App Service apps should only be accessible over HTTPS ↗	4.0.0 ↗
Guidelines for Software Development - Web application development	1552	Web application interactions - 1552	Function apps should only be accessible over HTTPS ↗	5.0.0 ↗

Canada Federal PBMM

To review how the available Azure Policy built-ins for all Azure services map to this compliance standard, see [Azure Policy Regulatory Compliance - Canada Federal PBMM](#). For more information about this compliance standard, see [Canada Federal PBMM ↗](#).

[Expand table](#)

Domain	Control ID	Control title	Policy (Azure portal)	Policy version (GitHub)
Access Control	AC-4	Information Flow Enforcement	App Service apps should not have CORS configured to allow every	2.0.0 ↗

Domain	Control ID	Control title	Policy (Azure portal)	Policy version (GitHub)
			resource to access your apps ↗	
Access Control	AC-17(1)	Remote Access Automated Monitoring / Control	App Service apps should have remote debugging turned off ↗	2.0.0 ↗
Access Control	AC-17(1)	Remote Access Automated Monitoring / Control	Function apps should have remote debugging turned off ↗	2.0.0 ↗
System and Communications Protection	SC-8(1)	Transmission Confidentiality and Integrity Cryptographic or Alternate Physical Protection	App Service apps should only be accessible over HTTPS ↗	4.0.0 ↗
System and Communications Protection	SC-8(1)	Transmission Confidentiality and Integrity Cryptographic or Alternate Physical Protection	Function apps should only be accessible over HTTPS ↗	5.0.0 ↗

CIS Microsoft Azure Foundations Benchmark 1.1.0

To review how the available Azure Policy built-ins for all Azure services map to this compliance standard, see [Azure Policy Regulatory Compliance - CIS Microsoft Azure Foundations Benchmark 1.1.0](#). For more information about this compliance standard, see [CIS Microsoft Azure Foundations Benchmark](#) ↗.

↔ [Expand table](#)

Domain	Control ID	Control title	Policy (Azure portal)	Policy version (GitHub)
9 AppService	9.1	Ensure App Service Authentication is set on Azure App Service	App Service apps should have authentication enabled ↗	2.0.1 ↗
9 AppService	9.1	Ensure App Service Authentication is set on Azure App Service	Function apps should have authentication enabled ↗	3.0.0 ↗

Domain	Control ID	Control title	Policy (Azure portal)	Policy version (GitHub)
9 AppService	9.10	Ensure that 'HTTP Version' is the latest, if used to run the web app	App Service apps should use latest 'HTTP Version'	4.0.0 ↗
9 AppService	9.10	Ensure that 'HTTP Version' is the latest, if used to run the web app	Function apps should use latest 'HTTP Version'	4.0.0 ↗
9 AppService	9.2	Ensure web app redirects all HTTP traffic to HTTPS in Azure App Service	App Service apps should only be accessible over HTTPS	4.0.0 ↗
9 AppService	9.3	Ensure web app is using the latest version of TLS encryption	App Service apps should use the latest TLS version	2.1.0 ↗
9 AppService	9.3	Ensure web app is using the latest version of TLS encryption	Function apps should use the latest TLS version	2.1.0 ↗
9 AppService	9.4	Ensure the web app has 'Client Certificates (Incoming client certificates)' set to 'On'	[Deprecated]: Function apps should have 'Client Certificates (Incoming client certificates)' enabled	3.1.0-deprecated ↗
9 AppService	9.4	Ensure the web app has 'Client Certificates (Incoming client certificates)' set to 'On'	App Service apps should have Client Certificates (Incoming client certificates) enabled	1.0.0 ↗
9 AppService	9.5	Ensure that Register with Azure Active Directory is enabled on App Service	App Service apps should use managed identity	3.0.0 ↗
9 AppService	9.5	Ensure that Register with Azure Active Directory is enabled on App Service	Function apps should use managed identity	3.0.0 ↗

CIS Microsoft Azure Foundations Benchmark 1.3.0

To review how the available Azure Policy built-ins for all Azure services map to this compliance standard, see [Azure Policy Regulatory Compliance - CIS Microsoft Azure Foundations Benchmark 1.3.0](#). For more information about this compliance standard, see [CIS Microsoft Azure Foundations Benchmark](#).

[Expand table](#)

Domain	Control ID	Control title	Policy (Azure portal)	Policy version (GitHub)
5 Logging and Monitoring	5.3	Ensure that Diagnostic Logs are enabled for all services which support it.	App Service apps should have resource logs enabled ↗	2.0.1 ↗
9 AppService	9.1	Ensure App Service Authentication is set on Azure App Service	App Service apps should have authentication enabled ↗	2.0.1 ↗
9 AppService	9.1	Ensure App Service Authentication is set on Azure App Service	Function apps should have authentication enabled ↗	3.0.0 ↗
9 AppService	9.10	Ensure FTP deployments are disabled	App Service apps should require FTPS only ↗	3.0.0 ↗
9 AppService	9.10	Ensure FTP deployments are disabled	Function apps should require FTPS only ↗	3.0.0 ↗
9 AppService	9.2	Ensure web app redirects all HTTP traffic to HTTPS in Azure App Service	App Service apps should only be accessible over HTTPS ↗	4.0.0 ↗
9 AppService	9.3	Ensure web app is using the latest version of TLS encryption	App Service apps should use the latest TLS version ↗	2.1.0 ↗
9 AppService	9.3	Ensure web app is using the latest version of TLS encryption	Function apps should use the latest TLS version ↗	2.1.0 ↗
9 AppService	9.4	Ensure the web app has 'Client Certificates (Incoming client certificates)' set to 'On'	[Deprecated]: Function apps should have 'Client Certificates (Incoming client certificates)' enabled ↗	3.1.0-deprecated ↗
9 AppService	9.4	Ensure the web app has 'Client Certificates (Incoming client certificates)' set to 'On'	App Service apps should have Client Certificates (Incoming client certificates) enabled ↗	1.0.0 ↗
9 AppService	9.5	Ensure that Register with Azure Active Directory is enabled on App Service	App Service apps should use managed identity ↗	3.0.0 ↗
9 AppService	9.5	Ensure that Register with Azure Active Directory is enabled on App Service	Function apps should use managed identity ↗	3.0.0 ↗

Domain	Control ID	Control title	Policy (Azure portal)	Policy version (GitHub)
9 AppService	9.9	Ensure that 'HTTP Version' is the latest, if used to run the web app	App Service apps should use latest 'HTTP Version' ↗	4.0.0 ↗
9 AppService	9.9	Ensure that 'HTTP Version' is the latest, if used to run the web app	Function apps should use latest 'HTTP Version' ↗	4.0.0 ↗

CIS Microsoft Azure Foundations Benchmark 1.4.0

To review how the available Azure Policy built-ins for all Azure services map to this compliance standard, see [Azure Policy Regulatory Compliance details for CIS v1.4.0](#). For more information about this compliance standard, see [CIS Microsoft Azure Foundations Benchmark](#) ↗.

[+] Expand table

Domain	Control ID	Control title	Policy (Azure portal)	Policy version (GitHub)
5 Logging and Monitoring	5.3	Ensure that Diagnostic Logs Are Enabled for All Services that Support it.	App Service apps should have resource logs enabled ↗	2.0.1 ↗
9 AppService	9.1	Ensure App Service Authentication is set up for apps in Azure App Service	App Service apps should have authentication enabled ↗	2.0.1 ↗
9 AppService	9.1	Ensure App Service Authentication is set up for apps in Azure App Service	Function apps should have authentication enabled ↗	3.0.0 ↗
9 AppService	9.10	Ensure FTP deployments are Disabled	App Service apps should require FTPS only ↗	3.0.0 ↗
9 AppService	9.10	Ensure FTP deployments are Disabled	Function apps should require FTPS only ↗	3.0.0 ↗

Domain	Control ID	Control title	Policy (Azure portal)	Policy version (GitHub)
9 AppService	9.2	Ensure Web App Redirects All HTTP traffic to HTTPS in Azure App Service	App Service apps should only be accessible over HTTPS ↗	4.0.0 ↗
9 AppService	9.3	Ensure Web App is using the latest version of TLS encryption	App Service apps should use the latest TLS version ↗	2.1.0 ↗
9 AppService	9.3	Ensure Web App is using the latest version of TLS encryption	Function apps should use the latest TLS version ↗	2.1.0 ↗
9 AppService	9.4	Ensure the web app has 'Client Certificates (Incoming client certificates)' set to 'On'	[Deprecated]: Function apps should have 'Client Certificates (Incoming client certificates)' enabled ↗	3.1.0-deprecated ↗
9 AppService	9.4	Ensure the web app has 'Client Certificates (Incoming client certificates)' set to 'On'	App Service apps should have Client Certificates (Incoming client certificates) enabled ↗	1.0.0 ↗
9 AppService	9.5	Ensure that Register with Azure Active Directory is enabled on App Service	App Service apps should use managed identity ↗	3.0.0 ↗
9 AppService	9.5	Ensure that Register with Azure Active Directory is enabled on App Service	Function apps should use managed identity ↗	3.0.0 ↗
9 AppService	9.9	Ensure that 'HTTP Version' is the Latest, if Used to Run the Web App	App Service apps should use latest 'HTTP Version' ↗	4.0.0 ↗
9 AppService	9.9	Ensure that 'HTTP Version' is the Latest, if Used to Run the Web App	Function apps should use latest 'HTTP Version' ↗	4.0.0 ↗

CIS Microsoft Azure Foundations Benchmark 2.0.0

To review how the available Azure Policy built-ins for all Azure services map to this compliance standard, see [Azure Policy Regulatory Compliance details for CIS v2.0.0](#). For

more information about this compliance standard, see [CIS Microsoft Azure Foundations Benchmark](#).

[+] Expand table

Domain	Control ID	Control title	Policy (Azure portal)	Policy version (GitHub)
5	5.4	Ensure that Azure Monitor Resource Logging is Enabled for All Services that Support it	App Service apps should have resource logs enabled	2.0.1 ↗
9	9.1	Ensure App Service Authentication is set up for apps in Azure App Service	App Service apps should have authentication enabled	2.0.1 ↗
9	9.1	Ensure App Service Authentication is set up for apps in Azure App Service	Function apps should have authentication enabled	3.0.0 ↗
9	9.10	Ensure FTP deployments are Disabled	App Service apps should require FTPS only	3.0.0 ↗
9	9.10	Ensure FTP deployments are Disabled	Function apps should require FTPS only	3.0.0 ↗
9	9.2	Ensure Web App Redirects All HTTP traffic to HTTPS in Azure App Service	App Service apps should only be accessible over HTTPS	4.0.0 ↗
9	9.3	Ensure Web App is using the latest version of TLS encryption	App Service apps should use the latest TLS version	2.1.0 ↗
9	9.3	Ensure Web App is using the latest version of TLS encryption	Function apps should use the latest TLS version	2.1.0 ↗
9	9.4	Ensure the web app has 'Client Certificates (Incoming client certificates)' set to 'On'	[Deprecated]: App Service apps should have 'Client Certificates (Incoming client certificates)' enabled	3.1.0-deprecated ↗
9	9.4	Ensure the web app has 'Client Certificates (Incoming client certificates)' set to 'On'	[Deprecated]: Function apps should have 'Client Certificates (Incoming client certificates)' enabled	3.1.0-deprecated ↗
9	9.5	Ensure that Register with Azure Active Directory is enabled on App Service	App Service apps should use managed identity	3.0.0 ↗

Domain	Control ID	Control title	Policy (Azure portal)	Policy version (GitHub)
9	9.5	Ensure that Register with Azure Active Directory is enabled on App Service	Function apps should use managed identity ↗	3.0.0 ↗
9	9.6	Ensure That 'PHP version' is the Latest, If Used to Run the Web App	App Service app slots that use PHP should use a specified 'PHP version' ↗	1.0.0 ↗
9	9.6	Ensure That 'PHP version' is the Latest, If Used to Run the Web App	App Service apps that use PHP should use a specified 'PHP version' ↗	3.2.0 ↗
9	9.7	Ensure that 'Python version' is the Latest Stable Version, if Used to Run the Web App	App Service app slots that use Python should use a specified 'Python version' ↗	1.0.0 ↗
9	9.7	Ensure that 'Python version' is the Latest Stable Version, if Used to Run the Web App	App Service apps that use Python should use a specified 'Python version' ↗	4.1.0 ↗
9	9.8	Ensure that 'Java version' is the latest, if used to run the Web App	Function app slots that use Java should use a specified 'Java version' ↗	1.0.0 ↗
9	9.8	Ensure that 'Java version' is the latest, if used to run the Web App	Function apps that use Java should use a specified 'Java version' ↗	3.1.0 ↗
9	9.9	Ensure that 'HTTP Version' is the Latest, if Used to Run the Web App	App Service apps should use latest 'HTTP Version' ↗	4.0.0 ↗
9	9.9	Ensure that 'HTTP Version' is the Latest, if Used to Run the Web App	Function apps should use latest 'HTTP Version' ↗	4.0.0 ↗

CMMC Level 3

To review how the available Azure Policy built-ins for all Azure services map to this compliance standard, see [Azure Policy Regulatory Compliance - CMMC Level 3](#). For more information about this compliance standard, see [Cybersecurity Maturity Model Certification \(CMMC\) ↗](#).

 [Expand table](#)

Domain	Control ID	Control title	Policy (Azure portal)	Policy version (GitHub)
Access Control	AC.1.001	Limit information system access to authorized users, processes acting on behalf of authorized users, and devices (including other information systems).	App Service apps should have remote debugging turned off ↗	2.0.0 ↗
Access Control	AC.1.001	Limit information system access to authorized users, processes acting on behalf of authorized users, and devices (including other information systems).	App Service apps should not have CORS configured to allow every resource to access your apps ↗	2.0.0 ↗
Access Control	AC.1.001	Limit information system access to authorized users, processes acting on behalf of authorized users, and devices (including other information systems).	Function apps should have remote debugging turned off ↗	2.0.0 ↗
Access Control	AC.1.001	Limit information system access to authorized users, processes acting on behalf of authorized users, and devices (including other information systems).	Function apps should not have CORS configured to allow every resource to access your apps ↗	2.0.0 ↗
Access Control	AC.1.002	Limit information system access to the types of transactions and functions that authorized users are permitted to execute.	App Service apps should not have CORS configured to allow every resource to access your apps ↗	2.0.0 ↗
Access Control	AC.1.002	Limit information system access to the types of transactions and functions that authorized users are permitted to execute.	App Service apps should only be accessible over HTTPS ↗	4.0.0 ↗
Access Control	AC.1.002	Limit information system access to the types of transactions and functions that authorized users are permitted to execute.	Function apps should not have CORS configured to allow every resource to access your apps ↗	2.0.0 ↗
Access Control	AC.1.002	Limit information system access to the types of transactions and	Function apps should only be	5.0.0 ↗

Domain	Control ID	Control title	Policy (Azure portal)	Policy version (GitHub)
		functions that authorized users are permitted to execute.	accessible over HTTPS ↗	
Access Control	AC.2.013	Monitor and control remote access sessions.	App Service apps should have remote debugging turned off ↗	2.0.0 ↗
Access Control	AC.2.013	Monitor and control remote access sessions.	Function apps should have remote debugging turned off ↗	2.0.0 ↗
Access Control	AC.2.016	Control the flow of CUI in accordance with approved authorizations.	Function apps should not have CORS configured to allow every resource to access your apps ↗	2.0.0 ↗
Audit and Accountability	AU.3.048	Collect audit information (e.g., logs) into one or more central repositories.	App Service apps should have resource logs enabled ↗	2.0.1 ↗
Configuration Management	CM.3.068	Restrict, disable, or prevent the use of nonessential programs, functions, ports, protocols, and services.	App Service apps should have remote debugging turned off ↗	2.0.0 ↗
Configuration Management	CM.3.068	Restrict, disable, or prevent the use of nonessential programs, functions, ports, protocols, and services.	App Service apps should not have CORS configured to allow every resource to access your apps ↗	2.0.0 ↗
Configuration Management	CM.3.068	Restrict, disable, or prevent the use of nonessential programs, functions, ports, protocols, and services.	Function apps should have remote debugging turned off ↗	2.0.0 ↗
Configuration Management	CM.3.068	Restrict, disable, or prevent the use of nonessential programs, functions, ports, protocols, and services.	Function apps should not have CORS configured to allow every resource to access your apps ↗	2.0.0 ↗

Domain	Control ID	Control title	Policy (Azure portal)	Policy version (GitHub)
Identification and Authentication	IA.3.084	Employ replay-resistant authentication mechanisms for network access to privileged and nonprivileged accounts.	App Service apps should only be accessible over HTTPS ↗	4.0.0 ↗
Identification and Authentication	IA.3.084	Employ replay-resistant authentication mechanisms for network access to privileged and nonprivileged accounts.	App Service apps should use the latest TLS version ↗	2.1.0 ↗
Identification and Authentication	IA.3.084	Employ replay-resistant authentication mechanisms for network access to privileged and nonprivileged accounts.	Function apps should only be accessible over HTTPS ↗	5.0.0 ↗
Identification and Authentication	IA.3.084	Employ replay-resistant authentication mechanisms for network access to privileged and nonprivileged accounts.	Function apps should use the latest TLS version ↗	2.1.0 ↗
System and Communications Protection	SC.1.175	Monitor, control, and protect communications (i.e., information transmitted or received by organizational systems) at the external boundaries and key internal boundaries of organizational systems.	App Service apps should only be accessible over HTTPS ↗	4.0.0 ↗
System and Communications Protection	SC.1.175	Monitor, control, and protect communications (i.e., information transmitted or received by organizational systems) at the external boundaries and key internal boundaries of organizational systems.	App Service apps should use the latest TLS version ↗	2.1.0 ↗
System and Communications Protection	SC.1.175	Monitor, control, and protect communications (i.e., information transmitted or received by organizational systems) at the external boundaries and key internal boundaries of organizational systems.	Function apps should only be accessible over HTTPS ↗	5.0.0 ↗
System and Communications Protection	SC.1.175	Monitor, control, and protect communications (i.e., information transmitted or received by organizational systems) at the	Function apps should use the latest TLS version ↗	2.1.0 ↗

Domain	Control ID	Control title	Policy (Azure portal)	Policy version (GitHub)
		external boundaries and key internal boundaries of organizational systems.		
System and Communications Protection	SC.3.183	Deny network communications traffic by default and allow network communications traffic by exception (i.e., deny all, permit by exception).	App Service apps should not have CORS configured to allow every resource to access your apps ↗	2.0.0 ↗
System and Communications Protection	SC.3.183	Deny network communications traffic by default and allow network communications traffic by exception (i.e., deny all, permit by exception).	Function apps should not have CORS configured to allow every resource to access your apps ↗	2.0.0 ↗
System and Communications Protection	SC.3.185	Implement cryptographic mechanisms to prevent unauthorized disclosure of CUI during transmission unless otherwise protected by alternative physical safeguards.	App Service apps should only be accessible over HTTPS ↗	4.0.0 ↗
System and Communications Protection	SC.3.185	Implement cryptographic mechanisms to prevent unauthorized disclosure of CUI during transmission unless otherwise protected by alternative physical safeguards.	App Service apps should use the latest TLS version ↗	2.1.0 ↗
System and Communications Protection	SC.3.185	Implement cryptographic mechanisms to prevent unauthorized disclosure of CUI during transmission unless otherwise protected by alternative physical safeguards.	Function apps should only be accessible over HTTPS ↗	5.0.0 ↗
System and Communications Protection	SC.3.185	Implement cryptographic mechanisms to prevent unauthorized disclosure of CUI during transmission unless otherwise protected by alternative physical safeguards.	Function apps should use the latest TLS version ↗	2.1.0 ↗

Domain	Control ID	Control title	Policy (Azure portal)	Policy version (GitHub)
System and Communications Protection	SC.3.190	Protect the authenticity of communications sessions.	App Service apps should only be accessible over HTTPS ↗	4.0.0 ↗
System and Communications Protection	SC.3.190	Protect the authenticity of communications sessions.	App Service apps should use the latest TLS version ↗	2.1.0 ↗
System and Communications Protection	SC.3.190	Protect the authenticity of communications sessions.	Function apps should only be accessible over HTTPS ↗	5.0.0 ↗
System and Communications Protection	SC.3.190	Protect the authenticity of communications sessions.	Function apps should use the latest TLS version ↗	2.1.0 ↗
System and Information Integrity	SI.1.210	Identify, report, and correct information and information system flaws in a timely manner.	App Service apps should use latest 'HTTP Version' ↗	4.0.0 ↗
System and Information Integrity	SI.1.210	Identify, report, and correct information and information system flaws in a timely manner.	App Service apps should use the latest TLS version ↗	2.1.0 ↗
System and Information Integrity	SI.1.210	Identify, report, and correct information and information system flaws in a timely manner.	Function apps should use latest 'HTTP Version' ↗	4.0.0 ↗
System and Information Integrity	SI.1.210	Identify, report, and correct information and information system flaws in a timely manner.	Function apps should use the latest TLS version ↗	2.1.0 ↗

FedRAMP High

To review how the available Azure Policy built-ins for all Azure services map to this compliance standard, see [Azure Policy Regulatory Compliance - FedRAMP High](#). For more information about this compliance standard, see [FedRAMP High](#)↗.

 Expand table

Domain	Control ID	Control title	Policy (Azure portal)	Policy version (GitHub)
Access Control	AC-2	Account Management	App Service apps should use managed identity ↗	3.0.0 ↗
Access Control	AC-2	Account Management	Function apps should use managed identity ↗	3.0.0 ↗
Access Control	AC-3	Access Enforcement	App Service apps should use managed identity ↗	3.0.0 ↗
Access Control	AC-3	Access Enforcement	Function apps should use managed identity ↗	3.0.0 ↗
Access Control	AC-4	Information Flow Enforcement	App Service apps should not have CORS configured to allow every resource to access your apps ↗	2.0.0 ↗
Access Control	AC-17	Remote Access	App Service apps should have remote debugging turned off ↗	2.0.0 ↗
Access Control	AC-17	Remote Access	Function apps should have remote debugging turned off ↗	2.0.0 ↗
Access Control	AC-17 (1)	Automated Monitoring / Control	App Service apps should have remote debugging turned off ↗	2.0.0 ↗
Access Control	AC-17 (1)	Automated Monitoring / Control	Function apps should have remote debugging turned off ↗	2.0.0 ↗
Audit And Accountability	AU-6 (4)	Central Review And Analysis	App Service apps should have resource logs enabled ↗	2.0.1 ↗
Audit And Accountability	AU-6 (5)	Integration / Scanning And Monitoring Capabilities	App Service apps should have resource logs enabled ↗	2.0.1 ↗
Audit And Accountability	AU-12	Audit Generation	App Service apps should have resource logs enabled ↗	2.0.1 ↗
Audit And Accountability	AU-12 (1)	System-Wide / Time-Correlated Audit Trail	App Service apps should have resource logs enabled ↗	2.0.1 ↗
Configuration Management	CM-6	Configuration Settings	App Service apps should have Client Certificates	1.0.0 ↗

Domain	Control ID	Control title	Policy (Azure portal)	Policy version (GitHub)
			(Incoming client certificates) enabled ↗	
Configuration Management	CM-6	Configuration Settings	App Service apps should have remote debugging turned off ↗	2.0.0 ↗
Configuration Management	CM-6	Configuration Settings	App Service apps should not have CORS configured to allow every resource to access your apps ↗	2.0.0 ↗
Configuration Management	CM-6	Configuration Settings	Function apps should have Client Certificates (Incoming client certificates) enabled ↗	1.0.0 ↗
Configuration Management	CM-6	Configuration Settings	Function apps should have remote debugging turned off ↗	2.0.0 ↗
Configuration Management	CM-6	Configuration Settings	Function apps should not have CORS configured to allow every resource to access your apps ↗	2.0.0 ↗
Identification And Authentication	IA-2	Identification And Authentication (Organizational Users)	App Service apps should use managed identity ↗	3.0.0 ↗
Identification And Authentication	IA-2	Identification And Authentication (Organizational Users)	Function apps should use managed identity ↗	3.0.0 ↗
Identification And Authentication	IA-4	Identifier Management	App Service apps should use managed identity ↗	3.0.0 ↗
Identification And Authentication	IA-4	Identifier Management	Function apps should use managed identity ↗	3.0.0 ↗
System And Communications Protection	SC-8	Transmission Confidentiality And Integrity	App Service apps should only be accessible over HTTPS ↗	4.0.0 ↗
System And Communications Protection	SC-8	Transmission Confidentiality And Integrity	App Service apps should require FTPS only ↗	3.0.0 ↗
System And Communications	SC-8	Transmission Confidentiality And	App Service apps should use the latest TLS version ↗	2.1.0 ↗

Domain	Control ID	Control title	Policy (Azure portal)	Policy version (GitHub)
Protection		Integrity		
System And Communications Protection	SC-8	Transmission Confidentiality And Integrity	Function apps should only be accessible over HTTPS ↗	5.0.0 ↗
System And Communications Protection	SC-8	Transmission Confidentiality And Integrity	Function apps should require FTPS only ↗	3.0.0 ↗
System And Communications Protection	SC-8	Transmission Confidentiality And Integrity	Function apps should use the latest TLS version ↗	2.1.0 ↗
System And Communications Protection	SC-8 (1)	Cryptographic Or Alternate Physical Protection	App Service apps should only be accessible over HTTPS ↗	4.0.0 ↗
System And Communications Protection	SC-8 (1)	Cryptographic Or Alternate Physical Protection	App Service apps should require FTPS only ↗	3.0.0 ↗
System And Communications Protection	SC-8 (1)	Cryptographic Or Alternate Physical Protection	App Service apps should use the latest TLS version ↗	2.1.0 ↗
System And Communications Protection	SC-8 (1)	Cryptographic Or Alternate Physical Protection	Function apps should only be accessible over HTTPS ↗	5.0.0 ↗
System And Communications Protection	SC-8 (1)	Cryptographic Or Alternate Physical Protection	Function apps should require FTPS only ↗	3.0.0 ↗
System And Communications Protection	SC-8 (1)	Cryptographic Or Alternate Physical Protection	Function apps should use the latest TLS version ↗	2.1.0 ↗
System And Communications Protection	SC-28	Protection Of Information At Rest	App Service Environment should have internal encryption enabled ↗	1.0.1 ↗
System And Communications Protection	SC-28 (1)	Cryptographic Protection	App Service Environment should have internal encryption enabled ↗	1.0.1 ↗
System And Information Integrity	SI-2	Flaw Remediation	App Service apps should use latest 'HTTP Version' ↗	4.0.0 ↗

Domain	Control ID	Control title	Policy (Azure portal)	Policy version (GitHub)
System And Information Integrity	SI-2	Flaw Remediation	Function apps should use latest 'HTTP Version' ↗	4.0.0 ↗

FedRAMP Moderate

To review how the available Azure Policy built-ins for all Azure services map to this compliance standard, see [Azure Policy Regulatory Compliance - FedRAMP Moderate](#). For more information about this compliance standard, see [FedRAMP Moderate](#) ↗.

[Expand table](#)

Domain	Control ID	Control title	Policy (Azure portal)	Policy version (GitHub)
Access Control	AC-2	Account Management	App Service apps should use managed identity ↗	3.0.0 ↗
Access Control	AC-2	Account Management	Function apps should use managed identity ↗	3.0.0 ↗
Access Control	AC-3	Access Enforcement	App Service apps should use managed identity ↗	3.0.0 ↗
Access Control	AC-3	Access Enforcement	Function apps should use managed identity ↗	3.0.0 ↗
Access Control	AC-4	Information Flow Enforcement	App Service apps should not have CORS configured to allow every resource to access your apps ↗	2.0.0 ↗
Access Control	AC-17	Remote Access	App Service apps should have remote debugging turned off ↗	2.0.0 ↗
Access Control	AC-17	Remote Access	Function apps should have remote debugging turned off ↗	2.0.0 ↗
Access Control	AC-17 (1)	Automated Monitoring / Control	App Service apps should have remote debugging turned off ↗	2.0.0 ↗
Access Control	AC-17 (1)	Automated Monitoring / Control	Function apps should have remote debugging	2.0.0 ↗

Domain	Control ID	Control title	Policy (Azure portal)	Policy version (GitHub)
			turned off ↗	
Audit And Accountability	AU-12	Audit Generation	App Service apps should have resource logs enabled ↗	2.0.1 ↗
Configuration Management	CM-6	Configuration Settings	[Deprecated]: Function apps should have 'Client Certificates (Incoming client certificates)' enabled ↗	3.1.0-deprecated ↗
Configuration Management	CM-6	Configuration Settings	App Service apps should have Client Certificates (Incoming client certificates) enabled ↗	1.0.0 ↗
Configuration Management	CM-6	Configuration Settings	App Service apps should have remote debugging turned off ↗	2.0.0 ↗
Configuration Management	CM-6	Configuration Settings	App Service apps should not have CORS configured to allow every resource to access your apps ↗	2.0.0 ↗
Configuration Management	CM-6	Configuration Settings	Function apps should have remote debugging turned off ↗	2.0.0 ↗
Configuration Management	CM-6	Configuration Settings	Function apps should not have CORS configured to allow every resource to access your apps ↗	2.0.0 ↗
Identification And Authentication	IA-2	Identification And Authentication (Organizational Users)	App Service apps should use managed identity ↗	3.0.0 ↗
Identification And Authentication	IA-2	Identification And Authentication (Organizational Users)	Function apps should use managed identity ↗	3.0.0 ↗
Identification And Authentication	IA-4	Identifier Management	App Service apps should use managed identity ↗	3.0.0 ↗
Identification And Authentication	IA-4	Identifier Management	Function apps should use managed identity ↗	3.0.0 ↗

Domain	Control ID	Control title	Policy (Azure portal)	Policy version (GitHub)
System And Communications Protection	SC-8	Transmission Confidentiality And Integrity	App Service apps should only be accessible over HTTPS ↗	4.0.0 ↗
System And Communications Protection	SC-8	Transmission Confidentiality And Integrity	App Service apps should require FTPS only ↗	3.0.0 ↗
System And Communications Protection	SC-8	Transmission Confidentiality And Integrity	App Service apps should use the latest TLS version ↗	2.1.0 ↗
System And Communications Protection	SC-8	Transmission Confidentiality And Integrity	Function apps should only be accessible over HTTPS ↗	5.0.0 ↗
System And Communications Protection	SC-8	Transmission Confidentiality And Integrity	Function apps should require FTPS only ↗	3.0.0 ↗
System And Communications Protection	SC-8	Transmission Confidentiality And Integrity	Function apps should use the latest TLS version ↗	2.1.0 ↗
System And Communications Protection	SC-8 (1)	Cryptographic Or Alternate Physical Protection	App Service apps should only be accessible over HTTPS ↗	4.0.0 ↗
System And Communications Protection	SC-8 (1)	Cryptographic Or Alternate Physical Protection	App Service apps should require FTPS only ↗	3.0.0 ↗
System And Communications Protection	SC-8 (1)	Cryptographic Or Alternate Physical Protection	App Service apps should use the latest TLS version ↗	2.1.0 ↗
System And Communications Protection	SC-8 (1)	Cryptographic Or Alternate Physical Protection	Function apps should only be accessible over HTTPS ↗	5.0.0 ↗
System And Communications Protection	SC-8 (1)	Cryptographic Or Alternate Physical Protection	Function apps should require FTPS only ↗	3.0.0 ↗
System And Communications Protection	SC-8 (1)	Cryptographic Or Alternate Physical Protection	Function apps should use the latest TLS version ↗	2.1.0 ↗

Domain	Control ID	Control title	Policy (Azure portal)	Policy version (GitHub)
System And Communications Protection	SC-28	Protection Of Information At Rest	App Service Environment should have internal encryption enabled ↗	1.0.1 ↗
System And Communications Protection	SC-28 (1)	Cryptographic Protection	App Service Environment should have internal encryption enabled ↗	1.0.1 ↗
System And Information Integrity	SI-2	Flaw Remediation	App Service apps should use latest 'HTTP Version' ↗	4.0.0 ↗
System And Information Integrity	SI-2	Flaw Remediation	Function apps should use latest 'HTTP Version' ↗	4.0.0 ↗

HIPAA HITRUST 9.2

To review how the available Azure Policy built-ins for all Azure services map to this compliance standard, see [Azure Policy Regulatory Compliance - HIPAA HITRUST 9.2](#). For more information about this compliance standard, see [HIPAA HITRUST 9.2 ↗](#).

[Expand table](#)

Domain	Control ID	Control title	Policy (Azure portal)	Policy version (GitHub)
Identification of Risks Related to External Parties	1402.05i1Organizational.45 - 05.i	Remote access connections between the organization and external parties are encrypted.	Function apps should only be accessible over HTTPS ↗	5.0.0 ↗
Identification of Risks Related to External Parties	1403.05i1Organizational.67 - 05.i	Access granted to external parties is limited to the minimum necessary and granted only for the duration required.	App Service apps should only be accessible over HTTPS ↗	4.0.0 ↗

Domain	Control ID	Control title	Policy (Azure portal)	Policy version (GitHub)
06 Configuration Management	0662.09sCSPOrganizational.2-09.s	0662.09sCSPOrganizational.2-09.s 09.08 Exchange of Information	App Service apps should have Client Certificates (Incoming client certificates) enabled ↗	1.0.0 ↗
08 Network Protection	0805.01m1Organizational.12-01.m	0805.01m1Organizational.12-01.m 01.04 Network Access Control	App Service apps should use a virtual network service endpoint ↗	2.0.1 ↗
08 Network Protection	0806.01m2Organizational.12356-01.m	0806.01m2Organizational.12356-01.m 01.04 Network Access Control	App Service apps should use a virtual network service endpoint ↗	2.0.1 ↗
08 Network Protection	0809.01n2Organizational.1234-01.n	0809.01n2Organizational.1234-01.n 01.04 Network Access Control	App Service apps should only be accessible over HTTPS ↗	4.0.0 ↗
08 Network Protection	0809.01n2Organizational.1234-01.n	0809.01n2Organizational.1234-01.n 01.04 Network Access Control	App Service apps should use the latest TLS version ↗	2.1.0 ↗

Domain	Control ID	Control title	Policy (Azure portal)	Policy version (GitHub)
08 Network Protection	0809.01n2Organizational.1234-01.n	0809.01n2Organizational.1234-01.n 01.04 Network Access Control	Function apps should only be accessible over HTTPS ↗	5.0.0 ↗
08 Network Protection	0809.01n2Organizational.1234-01.n	0809.01n2Organizational.1234-01.n 01.04 Network Access Control	Function apps should use the latest TLS version ↗	2.1.0 ↗
08 Network Protection	0810.01n2Organizational.5-01.n	0810.01n2Organizational.5-01.n 01.04 Network Access Control	App Service apps should only be accessible over HTTPS ↗	4.0.0 ↗
08 Network Protection	0810.01n2Organizational.5-01.n	0810.01n2Organizational.5-01.n 01.04 Network Access Control	App Service apps should use the latest TLS version ↗	2.1.0 ↗
08 Network Protection	0810.01n2Organizational.5-01.n	0810.01n2Organizational.5-01.n 01.04 Network Access Control	Function apps should only be accessible over HTTPS ↗	5.0.0 ↗
08 Network Protection	0810.01n2Organizational.5-01.n	0810.01n2Organizational.5-01.n 01.04 Network Access Control	Function apps should use the latest TLS version ↗	2.1.0 ↗
08 Network Protection	0811.01n2Organizational.6-01.n	0811.01n2Organizational.6-01.n 01.04 Network Access Control	App Service apps should	4.0.0 ↗

Domain	Control ID	Control title	Policy (Azure portal)	Policy version (GitHub)
			only be accessible over HTTPS ↗	
08 Network Protection	0811.01n2Organizational.6-01.n	0811.01n2Organizational.6-01.n 01.04 Network Access Control	App Service apps should use the latest TLS version ↗	2.1.0 ↗
08 Network Protection	0811.01n2Organizational.6-01.n	0811.01n2Organizational.6-01.n 01.04 Network Access Control	Function apps should only be accessible over HTTPS ↗	5.0.0 ↗
08 Network Protection	0811.01n2Organizational.6-01.n	0811.01n2Organizational.6-01.n 01.04 Network Access Control	Function apps should use the latest TLS version ↗	2.1.0 ↗
08 Network Protection	0812.01n2Organizational.8-01.n	0812.01n2Organizational.8-01.n 01.04 Network Access Control	App Service apps should only be accessible over HTTPS ↗	4.0.0 ↗
08 Network Protection	0812.01n2Organizational.8-01.n	0812.01n2Organizational.8-01.n 01.04 Network Access Control	App Service apps should use the latest TLS version ↗	2.1.0 ↗
08 Network Protection	0812.01n2Organizational.8-01.n	0812.01n2Organizational.8-01.n 01.04 Network Access Control	Function apps should only be accessible	5.0.0 ↗

Domain	Control ID	Control title	Policy (Azure portal)	Policy version (GitHub)
			over HTTPS ↗	
08 Network Protection	0812.01n2Organizational.8-01.n	0812.01n2Organizational.8-01.n 01.04 Network Access Control	Function apps should use the latest TLS version ↗	2.1.0 ↗
08 Network Protection	0814.01n1Organizational.12-01.n	0814.01n1Organizational.12-01.n 01.04 Network Access Control	App Service apps should only be accessible over HTTPS ↗	4.0.0 ↗
08 Network Protection	0814.01n1Organizational.12-01.n	0814.01n1Organizational.12-01.n 01.04 Network Access Control	App Service apps should use the latest TLS version ↗	2.1.0 ↗
08 Network Protection	0814.01n1Organizational.12-01.n	0814.01n1Organizational.12-01.n 01.04 Network Access Control	Function apps should only be accessible over HTTPS ↗	5.0.0 ↗
08 Network Protection	0814.01n1Organizational.12-01.n	0814.01n1Organizational.12-01.n 01.04 Network Access Control	Function apps should use the latest TLS version ↗	2.1.0 ↗
08 Network Protection	0861.09m2Organizational.67-09.m	0861.09m2Organizational.67-09.m 09.06 Network Security Management	App Service apps should use a virtual network service endpoint ↗	2.0.1 ↗

Domain	Control ID	Control title	Policy (Azure portal)	Policy version (GitHub)
08 Network Protection	0894.01m2Organizational.7-01.m	0894.01m2Organizational.7-01.m 01.04 Network Access Control	App Service apps should use a virtual network service endpoint ↗	2.0.1 ↗
09 Transmission Protection	0901.09s1Organizational.1-09.s	0901.09s1Organizational.1-09.s 09.08 Exchange of Information	App Service apps should not have CORS configured to allow every resource to access your apps ↗	2.0.0 ↗
09 Transmission Protection	0902.09s2Organizational.13-09.s	0902.09s2Organizational.13-09.s 09.08 Exchange of Information	Function apps should not have CORS configured to allow every resource to access your apps ↗	2.0.0 ↗
09 Transmission Protection	0912.09s1Organizational.4-09.s	0912.09s1Organizational.4-09.s 09.08 Exchange of Information	App Service apps should have remote debugging turned off ↗	2.0.0 ↗
09 Transmission Protection	0913.09s1Organizational.5-09.s	0913.09s1Organizational.5-09.s 09.08 Exchange of Information	Function apps should have remote	2.0.0 ↗

Domain	Control ID	Control title	Policy (Azure portal)	Policy version (GitHub)
			debugging turned off ↗	
09 Transmission Protection	0915.09s2Organizational.2-09.s	0915.09s2Organizational.2-09.s 09.08 Exchange of Information	App Service apps should have Client Certificates (Incoming client certificates) enabled ↗	1.0.0 ↗
09 Transmission Protection	0916.09s2Organizational.4-09.s	0916.09s2Organizational.4-09.s 09.08 Exchange of Information	App Service apps should not have CORS configured to allow every resource to access your apps ↗	2.0.0 ↗
09 Transmission Protection	0949.09y2Organizational.5-09.y	0949.09y2Organizational.5-09.y 09.09 Electronic Commerce Services	App Service apps should only be accessible over HTTPS ↗	4.0.0 ↗
09 Transmission Protection	0949.09y2Organizational.5-09.y	0949.09y2Organizational.5-09.y 09.09 Electronic Commerce Services	App Service apps should use the latest TLS version ↗	2.1.0 ↗
09 Transmission Protection	0949.09y2Organizational.5-09.y	0949.09y2Organizational.5-09.y 09.09 Electronic Commerce Services	Function apps should only be accessible	5.0.0 ↗

Domain	Control ID	Control title	Policy (Azure portal)	Policy version (GitHub)
		over HTTPS ↗		
09 Transmission Protection	0949.09y2Organizational.5-09.y	0949.09y2Organizational.5-09.y 09.09 Electronic Commerce Services	Function apps should use the latest TLS version ↗	2.1.0 ↗
09 Transmission Protection	0960.09sCSPOrganizational.1- 09.s	0960.09sCSPOrganizational.1- 09.s 09.08 Exchange of Information	Function apps should not have CORS configured to allow every resource to access your apps ↗	2.0.0 ↗
11 Access Control	1194.01l2Organizational.2-01.l	1194.01l2Organizational.2-01.l 01.04 Network Access Control	App Service apps should have remote debugging turned off ↗	2.0.0 ↗
11 Access Control	1195.01l3Organizational.1-01.l	1195.01l3Organizational.1-01.l 01.04 Network Access Control	Function apps should have remote debugging turned off ↗	2.0.0 ↗
12 Audit Logging & Monitoring	1209.09aa3System.2-09.aa	1209.09aa3System.2-09.aa 09.10 Monitoring	App Service apps should have resource logs enabled ↗	2.0.1 ↗
13 Education, Training and	1325.09s1Organizational.3-09.s	1325.09s1Organizational.3-09.s 09.08 Exchange of Information	Function apps	2.0.0 ↗

Domain	Control ID	Control title	Policy (Azure portal)	Policy version (GitHub)
Awareness			should have remote debugging turned off ↗	

IRS 1075 September 2016

To review how the available Azure Policy built-ins for all Azure services map to this compliance standard, see [Azure Policy Regulatory Compliance - IRS 1075 September 2016](#). For more information about this compliance standard, see [IRS 1075 September 2016 \[↗\]\(#\)](#).

[\[+\] Expand table](#)

Domain	Control ID	Control title	Policy (Azure portal)	Policy version (GitHub)
Access Control	9.3.1.12	Remote Access (AC-17)	App Service apps should have remote debugging turned off ↗	2.0.0 ↗
Access Control	9.3.1.12	Remote Access (AC-17)	Function apps should have remote debugging turned off ↗	2.0.0 ↗
Access Control	9.3.1.4	Information Flow Enforcement (AC-4)	App Service apps should not have CORS configured to allow every resource to access your apps ↗	2.0.0 ↗
System and Communications Protection	9.3.16.6	Transmission Confidentiality and Integrity (SC-8)	App Service apps should only be accessible over HTTPS ↗	4.0.0 ↗
System and Communications Protection	9.3.16.6	Transmission Confidentiality and Integrity (SC-8)	Function apps should only be accessible over HTTPS ↗	5.0.0 ↗

ISO 27001:2013

To review how the available Azure Policy built-ins for all Azure services map to this compliance standard, see [Azure Policy Regulatory Compliance - ISO 27001:2013](#). For more information about this compliance standard, see [ISO 27001:2013 \[↗\]\(#\)](#).

[Expand table](#)

Domain	Control ID	Control title	Policy (Azure portal)	Policy version (GitHub)
Cryptography	10.1.1	Policy on the use of cryptographic controls	App Service apps should only be accessible over HTTPS	4.0.0 ↗
Cryptography	10.1.1	Policy on the use of cryptographic controls	Function apps should only be accessible over HTTPS	5.0.0 ↗

Microsoft cloud security benchmark

The [Microsoft cloud security benchmark](#) provides recommendations on how you can secure your cloud solutions on Azure. To see how this service completely maps to the Microsoft cloud security benchmark, see the [Azure Security Benchmark mapping files](#).

To review how the available Azure Policy built-ins for all Azure services map to this compliance standard, see [Azure Policy Regulatory Compliance - Microsoft cloud security benchmark](#).

[Expand table](#)

Domain	Control ID	Control title	Policy (Azure portal)	Policy version (GitHub)
Network Security	NS-8	Detect and disable insecure services and protocols	App Service apps should use the latest TLS version	2.1.0 ↗
Network Security	NS-8	Detect and disable insecure services and protocols	Function apps should use the latest TLS version	2.1.0 ↗
Identity Management	IM-3	Manage application identities securely and automatically	App Service apps should use managed identity	3.0.0 ↗
Identity Management	IM-3	Manage application identities securely and automatically	Function apps should use managed identity	3.0.0 ↗
Data Protection	DP-3	Encrypt sensitive data in transit	App Service apps should only be accessible over HTTPS	4.0.0 ↗

Domain	Control ID	Control title	Policy (Azure portal)	Policy version (GitHub)
Data Protection	DP-3	Encrypt sensitive data in transit	App Service apps should require FTPS only ↗	3.0.0 ↗
Data Protection	DP-3	Encrypt sensitive data in transit	App Service apps should use the latest TLS version ↗	2.1.0 ↗
Data Protection	DP-3	Encrypt sensitive data in transit	Function apps should only be accessible over HTTPS ↗	5.0.0 ↗
Data Protection	DP-3	Encrypt sensitive data in transit	Function apps should require FTPS only ↗	3.0.0 ↗
Data Protection	DP-3	Encrypt sensitive data in transit	Function apps should use the latest TLS version ↗	2.1.0 ↗
Logging and Threat Detection	LT-3	Enable logging for security investigation	App Service apps should have resource logs enabled ↗	2.0.1 ↗
Posture and Vulnerability Management	PV-2	Audit and enforce secure configurations	[Deprecated]: Function apps should have 'Client Certificates (Incoming client certificates)' enabled ↗	3.1.0-deprecated ↗
Posture and Vulnerability Management	PV-2	Audit and enforce secure configurations	App Service apps should have Client Certificates (Incoming client certificates) enabled ↗	1.0.0 ↗
Posture and Vulnerability Management	PV-2	Audit and enforce secure configurations	App Service apps should have remote debugging turned off ↗	2.0.0 ↗
Posture and Vulnerability Management	PV-2	Audit and enforce secure configurations	App Service apps should not have CORS configured to allow every resource to access your apps ↗	2.0.0 ↗
Posture and Vulnerability Management	PV-2	Audit and enforce secure configurations	Function apps should have remote debugging turned off ↗	2.0.0 ↗
Posture and Vulnerability Management	PV-2	Audit and enforce secure configurations	Function apps should not have CORS configured to allow every resource to access your apps ↗	2.0.0 ↗

To review how the available Azure Policy built-ins for all Azure services map to this compliance standard, see [Azure Policy Regulatory Compliance - NIST SP 800-171 R2](#). For more information about this compliance standard, see [NIST SP 800-171 R2](#).

[Expand table](#)

Domain	Control ID	Control title	Policy (Azure portal)	Policy version (GitHub)
Access Control	3.1.1	Limit system access to authorized users, processes acting on behalf of authorized users, and devices (including other systems).	App Service apps should have remote debugging turned off	2.0.0
Access Control	3.1.1	Limit system access to authorized users, processes acting on behalf of authorized users, and devices (including other systems).	App Service apps should use managed identity	3.0.0
Access Control	3.1.1	Limit system access to authorized users, processes acting on behalf of authorized users, and devices (including other systems).	Function apps should have remote debugging turned off	2.0.0
Access Control	3.1.1	Limit system access to authorized users, processes acting on behalf of authorized users, and devices (including other systems).	Function apps should use managed identity	3.0.0
Access Control	3.1.12	Monitor and control remote access sessions.	App Service apps should have remote debugging turned off	2.0.0
Access Control	3.1.12	Monitor and control remote access sessions.	Function apps should have remote debugging turned off	2.0.0
Access Control	3.1.2	Limit system access to the types of transactions and functions that authorized users are permitted to execute.	App Service apps should have remote debugging turned off	2.0.0

Domain	Control ID	Control title	Policy (Azure portal)	Policy version (GitHub)
Access Control	3.1.2	Limit system access to the types of transactions and functions that authorized users are permitted to execute.	App Service apps should use managed identity	3.0.0 ↗
Access Control	3.1.2	Limit system access to the types of transactions and functions that authorized users are permitted to execute.	Function apps should have remote debugging turned off	2.0.0 ↗
Access Control	3.1.2	Limit system access to the types of transactions and functions that authorized users are permitted to execute.	Function apps should use managed identity	3.0.0 ↗
Access Control	3.1.3	Control the flow of CUI in accordance with approved authorizations.	App Service apps should not have CORS configured to allow every resource to access your apps	2.0.0 ↗
System and Communications Protection	3.13.16	Protect the confidentiality of CUI at rest.	App Service Environment should have internal encryption enabled	1.0.1 ↗
System and Communications Protection	3.13.8	Implement cryptographic mechanisms to prevent unauthorized disclosure of CUI during transmission unless otherwise protected by alternative physical safeguards.	App Service apps should only be accessible over HTTPS	4.0.0 ↗
System and Communications Protection	3.13.8	Implement cryptographic mechanisms to prevent unauthorized disclosure of CUI during transmission unless otherwise protected by alternative physical safeguards.	App Service apps should require FTPS only	3.0.0 ↗

Domain	Control ID	Control title	Policy (Azure portal)	Policy version (GitHub)
System and Communications Protection	3.13.8	Implement cryptographic mechanisms to prevent unauthorized disclosure of CUI during transmission unless otherwise protected by alternative physical safeguards.	App Service apps should use the latest TLS version	2.1.0 ↗
System and Communications Protection	3.13.8	Implement cryptographic mechanisms to prevent unauthorized disclosure of CUI during transmission unless otherwise protected by alternative physical safeguards.	Function apps should only be accessible over HTTPS	5.0.0 ↗
System and Communications Protection	3.13.8	Implement cryptographic mechanisms to prevent unauthorized disclosure of CUI during transmission unless otherwise protected by alternative physical safeguards.	Function apps should require FTPS only	3.0.0 ↗
System and Communications Protection	3.13.8	Implement cryptographic mechanisms to prevent unauthorized disclosure of CUI during transmission unless otherwise protected by alternative physical safeguards.	Function apps should use the latest TLS version	2.1.0 ↗
System and Information Integrity	3.14.1	Identify, report, and correct system flaws in a timely manner.	App Service apps should use latest 'HTTP Version'	4.0.0 ↗
System and Information Integrity	3.14.1	Identify, report, and correct system flaws in a timely manner.	Function apps should use latest 'HTTP Version'	4.0.0 ↗
Audit and Accountability	3.3.1	Create and retain system audit logs and records to the extent needed to enable the monitoring, analysis, investigation, and reporting of unlawful or unauthorized system activity	App Service apps should have resource logs enabled	2.0.1 ↗

Domain	Control ID	Control title	Policy (Azure portal)	Policy version (GitHub)
Audit and Accountability	3.3.2	Ensure that the actions of individual system users can be uniquely traced to those users, so they can be held accountable for their actions.	App Service apps should have resource logs enabled	2.0.1 ↗
Configuration Management	3.4.1	Establish and maintain baseline configurations and inventories of organizational systems (including hardware, software, firmware, and documentation) throughout the respective system development life cycles.	[Deprecated]: Function apps should have 'Client Certificates (Incoming client certificates)' enabled	3.1.0-deprecated ↗
Configuration Management	3.4.1	Establish and maintain baseline configurations and inventories of organizational systems (including hardware, software, firmware, and documentation) throughout the respective system development life cycles.	App Service apps should have Client Certificates (Incoming client certificates) enabled	1.0.0 ↗
Configuration Management	3.4.1	Establish and maintain baseline configurations and inventories of organizational systems (including hardware, software, firmware, and documentation) throughout the respective system development life cycles.	App Service apps should have remote debugging turned off	2.0.0 ↗
Configuration Management	3.4.1	Establish and maintain baseline configurations and inventories of organizational systems (including hardware, software, firmware, and documentation) throughout the respective system development life cycles.	App Service apps should not have CORS configured to allow every resource to access your apps	2.0.0 ↗
Configuration Management	3.4.1	Establish and maintain baseline configurations and inventories of organizational systems (including hardware, software, firmware, and documentation) throughout	Function apps should have remote debugging turned off	2.0.0 ↗

Domain	Control ID	Control title	Policy (Azure portal)	Policy version (GitHub)
		the respective system development life cycles.		
Configuration Management	3.4.1	Establish and maintain baseline configurations and inventories of organizational systems (including hardware, software, firmware, and documentation) throughout the respective system development life cycles.	Function apps should not have CORS configured to allow every resource to access your apps	2.0.0 ↗
Configuration Management	3.4.2	Establish and enforce security configuration settings for information technology products employed in organizational systems.	[Deprecated]: Function apps should have 'Client Certificates (Incoming client certificates)' enabled	3.1.0-deprecated ↗
Configuration Management	3.4.2	Establish and enforce security configuration settings for information technology products employed in organizational systems.	App Service apps should have Client Certificates (Incoming client certificates) enabled	1.0.0 ↗
Configuration Management	3.4.2	Establish and enforce security configuration settings for information technology products employed in organizational systems.	App Service apps should have remote debugging turned off	2.0.0 ↗
Configuration Management	3.4.2	Establish and enforce security configuration settings for information technology products employed in organizational systems.	App Service apps should not have CORS configured to allow every resource to access your apps	2.0.0 ↗
Configuration Management	3.4.2	Establish and enforce security configuration settings for information technology products employed in organizational systems.	Function apps should have remote debugging turned off	2.0.0 ↗
Configuration Management	3.4.2	Establish and enforce security configuration settings for information technology	Function apps should not have CORS configured to	2.0.0 ↗

Domain	Control ID	Control title	Policy (Azure portal)	Policy version (GitHub)
		products employed in organizational systems.	allow every resource to access your apps ↗	
Identification and Authentication	3.5.1	Identify system users, processes acting on behalf of users, and devices.	App Service apps should use managed identity ↗	3.0.0 ↗
Identification and Authentication	3.5.1	Identify system users, processes acting on behalf of users, and devices.	Function apps should use managed identity ↗	3.0.0 ↗
Identification and Authentication	3.5.2	Authenticate (or verify) the identities of users, processes, or devices, as a prerequisite to allowing access to organizational systems.	App Service apps should use managed identity ↗	3.0.0 ↗
Identification and Authentication	3.5.2	Authenticate (or verify) the identities of users, processes, or devices, as a prerequisite to allowing access to organizational systems.	Function apps should use managed identity ↗	3.0.0 ↗
Identification and Authentication	3.5.5	Prevent reuse of identifiers for a defined period.	App Service apps should use managed identity ↗	3.0.0 ↗
Identification and Authentication	3.5.5	Prevent reuse of identifiers for a defined period.	Function apps should use managed identity ↗	3.0.0 ↗
Identification and Authentication	3.5.6	Disable identifiers after a defined period of inactivity.	App Service apps should use managed identity ↗	3.0.0 ↗
Identification and Authentication	3.5.6	Disable identifiers after a defined period of inactivity.	Function apps should use managed identity ↗	3.0.0 ↗

NIST SP 800-53 Rev. 4

To review how the available Azure Policy built-ins for all Azure services map to this compliance standard, see [Azure Policy Regulatory Compliance - NIST SP 800-53 Rev. 4](#). For more information about this compliance standard, see [NIST SP 800-53 Rev. 4](#).

[Expand table](#)

Domain	Control ID	Control title	Policy (Azure portal)	Policy version (GitHub)
Access Control	AC-2	Account Management	App Service apps should use managed identity ↗	3.0.0 ↗
Access Control	AC-2	Account Management	Function apps should use managed identity ↗	3.0.0 ↗
Access Control	AC-3	Access Enforcement	App Service apps should use managed identity ↗	3.0.0 ↗
Access Control	AC-3	Access Enforcement	Function apps should use managed identity ↗	3.0.0 ↗
Access Control	AC-4	Information Flow Enforcement	App Service apps should not have CORS configured to allow every resource to access your apps ↗	2.0.0 ↗
Access Control	AC-17	Remote Access	App Service apps should have remote debugging turned off ↗	2.0.0 ↗
Access Control	AC-17	Remote Access	Function apps should have remote debugging turned off ↗	2.0.0 ↗
Access Control	AC-17 (1)	Automated Monitoring / Control	App Service apps should have remote debugging turned off ↗	2.0.0 ↗
Access Control	AC-17 (1)	Automated Monitoring / Control	Function apps should have remote debugging turned off ↗	2.0.0 ↗
Audit And Accountability	AU-6 (4)	Central Review And Analysis	App Service apps should have resource logs enabled ↗	2.0.1 ↗
Audit And Accountability	AU-6 (5)	Integration / Scanning And Monitoring Capabilities	App Service apps should have resource logs enabled ↗	2.0.1 ↗
Audit And Accountability	AU-12	Audit Generation	App Service apps should have resource logs enabled ↗	2.0.1 ↗

Domain	Control ID	Control title	Policy (Azure portal)	Policy version (GitHub)
Audit And Accountability	AU-12 (1)	System-Wide / Time-Correlated Audit Trail	App Service apps should have resource logs enabled ↗	2.0.1 ↗
Configuration Management	CM-6	Configuration Settings	[Deprecated]: Function apps should have 'Client Certificates (Incoming client certificates)' enabled ↗	3.1.0-deprecated ↗
Configuration Management	CM-6	Configuration Settings	App Service apps should have Client Certificates (Incoming client certificates) enabled ↗	1.0.0 ↗
Configuration Management	CM-6	Configuration Settings	App Service apps should have remote debugging turned off ↗	2.0.0 ↗
Configuration Management	CM-6	Configuration Settings	App Service apps should not have CORS configured to allow every resource to access your apps ↗	2.0.0 ↗
Configuration Management	CM-6	Configuration Settings	Function apps should have remote debugging turned off ↗	2.0.0 ↗
Configuration Management	CM-6	Configuration Settings	Function apps should not have CORS configured to allow every resource to access your apps ↗	2.0.0 ↗
Identification And Authentication	IA-2	Identification And Authentication (Organizational Users)	App Service apps should use managed identity ↗	3.0.0 ↗
Identification And Authentication	IA-2	Identification And Authentication (Organizational Users)	Function apps should use managed identity ↗	3.0.0 ↗
Identification And Authentication	IA-4	Identifier Management	App Service apps should use managed identity ↗	3.0.0 ↗
Identification And Authentication	IA-4	Identifier Management	Function apps should use managed identity ↗	3.0.0 ↗

Domain	Control ID	Control title	Policy (Azure portal)	Policy version (GitHub)
System And Communications Protection	SC-8	Transmission Confidentiality And Integrity	App Service apps should only be accessible over HTTPS ↗	4.0.0 ↗
System And Communications Protection	SC-8	Transmission Confidentiality And Integrity	App Service apps should require FTPS only ↗	3.0.0 ↗
System And Communications Protection	SC-8	Transmission Confidentiality And Integrity	App Service apps should use the latest TLS version ↗	2.1.0 ↗
System And Communications Protection	SC-8	Transmission Confidentiality And Integrity	Function apps should only be accessible over HTTPS ↗	5.0.0 ↗
System And Communications Protection	SC-8	Transmission Confidentiality And Integrity	Function apps should require FTPS only ↗	3.0.0 ↗
System And Communications Protection	SC-8	Transmission Confidentiality And Integrity	Function apps should use the latest TLS version ↗	2.1.0 ↗
System And Communications Protection	SC-8 (1)	Cryptographic Or Alternate Physical Protection	App Service apps should only be accessible over HTTPS ↗	4.0.0 ↗
System And Communications Protection	SC-8 (1)	Cryptographic Or Alternate Physical Protection	App Service apps should require FTPS only ↗	3.0.0 ↗
System And Communications Protection	SC-8 (1)	Cryptographic Or Alternate Physical Protection	App Service apps should use the latest TLS version ↗	2.1.0 ↗
System And Communications Protection	SC-8 (1)	Cryptographic Or Alternate Physical Protection	Function apps should only be accessible over HTTPS ↗	5.0.0 ↗
System And Communications Protection	SC-8 (1)	Cryptographic Or Alternate Physical Protection	Function apps should require FTPS only ↗	3.0.0 ↗
System And Communications Protection	SC-8 (1)	Cryptographic Or Alternate Physical Protection	Function apps should use the latest TLS version ↗	2.1.0 ↗

Domain	Control ID	Control title	Policy (Azure portal)	Policy version (GitHub)
System And Communications Protection	SC-28	Protection Of Information At Rest	App Service Environment should have internal encryption enabled	1.0.1 ↗
System And Communications Protection	SC-28 (1)	Cryptographic Protection	App Service Environment should have internal encryption enabled	1.0.1 ↗
System And Information Integrity	SI-2	Flaw Remediation	App Service apps should use latest 'HTTP Version'	4.0.0 ↗
System And Information Integrity	SI-2	Flaw Remediation	Function apps should use latest 'HTTP Version'	4.0.0 ↗
System And Information Integrity	SI-2 (6)	Removal of Previous Versions of Software / Firmware	App Service apps should use latest 'HTTP Version'	4.0.0 ↗
System And Information Integrity	SI-2 (6)	Removal of Previous Versions of Software / Firmware	Function apps should use latest 'HTTP Version'	4.0.0 ↗

NIST SP 800-53 Rev. 5

To review how the available Azure Policy built-ins for all Azure services map to this compliance standard, see [Azure Policy Regulatory Compliance - NIST SP 800-53 Rev. 5](#). For more information about this compliance standard, see [NIST SP 800-53 Rev. 5](#).

[Expand table](#)

Domain	Control ID	Control title	Policy (Azure portal)	Policy version (GitHub)
Access Control	AC-2	Account Management	App Service apps should use managed identity	3.0.0 ↗
Access Control	AC-2	Account Management	Function apps should use managed identity	3.0.0 ↗
Access Control	AC-3	Access Enforcement	App Service apps should use managed identity	3.0.0 ↗

Domain	Control ID	Control title	Policy (Azure portal)	Policy version (GitHub)
Access Control	AC-3	Access Enforcement	Function apps should use managed identity ↗	3.0.0 ↗
Access Control	AC-4	Information Flow Enforcement	App Service apps should not have CORS configured to allow every resource to access your apps ↗	2.0.0 ↗
Access Control	AC-17	Remote Access	App Service apps should have remote debugging turned off ↗	2.0.0 ↗
Access Control	AC-17	Remote Access	Function apps should have remote debugging turned off ↗	2.0.0 ↗
Access Control	AC-17 (1)	Monitoring and Control	App Service apps should have remote debugging turned off ↗	2.0.0 ↗
Access Control	AC-17 (1)	Monitoring and Control	Function apps should have remote debugging turned off ↗	2.0.0 ↗
Audit and Accountability	AU-6 (4)	Central Review and Analysis	App Service apps should have resource logs enabled ↗	2.0.1 ↗
Audit and Accountability	AU-6 (5)	Integrated Analysis of Audit Records	App Service apps should have resource logs enabled ↗	2.0.1 ↗
Audit and Accountability	AU-12	Audit Record Generation	App Service apps should have resource logs enabled ↗	2.0.1 ↗
Audit and Accountability	AU-12 (1)	System-wide and Time-correlated Audit Trail	App Service apps should have resource logs enabled ↗	2.0.1 ↗
Configuration Management	CM-6	Configuration Settings	[Deprecated]: Function apps should have 'Client Certificates (Incoming client certificates)' enabled ↗	3.1.0-deprecated ↗
Configuration Management	CM-6	Configuration Settings	App Service apps should have Client Certificates (Incoming client certificates) enabled ↗	1.0.0 ↗

Domain	Control ID	Control title	Policy (Azure portal)	Policy version (GitHub)
Configuration Management	CM-6	Configuration Settings	App Service apps should have remote debugging turned off ↗	2.0.0 ↗
Configuration Management	CM-6	Configuration Settings	App Service apps should not have CORS configured to allow every resource to access your apps ↗	2.0.0 ↗
Configuration Management	CM-6	Configuration Settings	Function apps should have remote debugging turned off ↗	2.0.0 ↗
Configuration Management	CM-6	Configuration Settings	Function apps should not have CORS configured to allow every resource to access your apps ↗	2.0.0 ↗
Identification and Authentication	IA-2	Identification and Authentication (organizational Users)	App Service apps should use managed identity ↗	3.0.0 ↗
Identification and Authentication	IA-2	Identification and Authentication (organizational Users)	Function apps should use managed identity ↗	3.0.0 ↗
Identification and Authentication	IA-4	Identifier Management	App Service apps should use managed identity ↗	3.0.0 ↗
Identification and Authentication	IA-4	Identifier Management	Function apps should use managed identity ↗	3.0.0 ↗
System and Communications Protection	SC-8	Transmission Confidentiality and Integrity	App Service apps should only be accessible over HTTPS ↗	4.0.0 ↗
System and Communications Protection	SC-8	Transmission Confidentiality and Integrity	App Service apps should require FTPS only ↗	3.0.0 ↗
System and Communications Protection	SC-8	Transmission Confidentiality and Integrity	App Service apps should use the latest TLS version ↗	2.1.0 ↗
System and Communications Protection	SC-8	Transmission Confidentiality and Integrity	Function apps should only be accessible over HTTPS ↗	5.0.0 ↗

Domain	Control ID	Control title	Policy (Azure portal)	Policy version (GitHub)
System and Communications Protection	SC-8	Transmission Confidentiality and Integrity	Function apps should require FTPS only ↗	3.0.0 ↗
System and Communications Protection	SC-8	Transmission Confidentiality and Integrity	Function apps should use the latest TLS version ↗	2.1.0 ↗
System and Communications Protection	SC-8 (1)	Cryptographic Protection	App Service apps should only be accessible over HTTPS ↗	4.0.0 ↗
System and Communications Protection	SC-8 (1)	Cryptographic Protection	App Service apps should require FTPS only ↗	3.0.0 ↗
System and Communications Protection	SC-8 (1)	Cryptographic Protection	App Service apps should use the latest TLS version ↗	2.1.0 ↗
System and Communications Protection	SC-8 (1)	Cryptographic Protection	Function apps should only be accessible over HTTPS ↗	5.0.0 ↗
System and Communications Protection	SC-8 (1)	Cryptographic Protection	Function apps should require FTPS only ↗	3.0.0 ↗
System and Communications Protection	SC-8 (1)	Cryptographic Protection	Function apps should use the latest TLS version ↗	2.1.0 ↗
System and Communications Protection	SC-28	Protection of Information at Rest	App Service Environment should have internal encryption enabled ↗	1.0.1 ↗
System and Communications Protection	SC-28 (1)	Cryptographic Protection	App Service Environment should have internal encryption enabled ↗	1.0.1 ↗
System and Information Integrity	SI-2	Flaw Remediation	App Service apps should use latest 'HTTP Version' ↗	4.0.0 ↗
System and Information Integrity	SI-2	Flaw Remediation	Function apps should use latest 'HTTP Version' ↗	4.0.0 ↗

Domain	Control ID	Control title	Policy (Azure portal)	Policy version (GitHub)
System and Information Integrity	SI-2 (6)	Removal of Previous Versions of Software and Firmware	App Service apps should use latest 'HTTP Version'	4.0.0 ↗
System and Information Integrity	SI-2 (6)	Removal of Previous Versions of Software and Firmware	Function apps should use latest 'HTTP Version'	4.0.0 ↗

NL BIO Cloud Theme

To review how the available Azure Policy built-ins for all Azure services map to this compliance standard, see [Azure Policy Regulatory Compliance details for NL BIO Cloud Theme](#). For more information about this compliance standard, see [Baseline Information Security Government Cybersecurity - Digital Government \(digitaleoverheid.nl\)](#).

[] Expand table

Domain	Control ID	Control title	Policy (Azure portal)	Policy version (GitHub)
C.04.3 Technical vulnerability management - Timelines	C.04.3	If the probability of abuse and the expected damage are both high, patches are installed no later than within a week.	App Service apps should use latest 'HTTP Version'	4.0.0 ↗
C.04.3 Technical vulnerability management - Timelines	C.04.3	If the probability of abuse and the expected damage are both high, patches are installed no later than within a week.	App Service apps that use Java should use a specified 'Java version'	3.1.0 ↗
C.04.3 Technical vulnerability management - Timelines	C.04.3	If the probability of abuse and the expected damage are both high, patches are installed no later than within a week.	App Service apps that use PHP should use a specified 'PHP version'	3.2.0 ↗
C.04.3 Technical vulnerability management - Timelines	C.04.3	If the probability of abuse and the expected damage are both high, patches are installed no later than within a week.	App Service apps that use Python should use a specified 'Python version'	4.1.0 ↗

Domain	Control ID	Control title	Policy (Azure portal)	Policy version (GitHub)
C.04.3 Technical vulnerability management - Timelines	C.04.3	If the probability of abuse and the expected damage are both high, patches are installed no later than within a week.	Function apps should use latest 'HTTP Version'	4.0.0 ↗
C.04.3 Technical vulnerability management - Timelines	C.04.3	If the probability of abuse and the expected damage are both high, patches are installed no later than within a week.	Function apps that use Java should use a specified 'Java version'	3.1.0 ↗
C.04.3 Technical vulnerability management - Timelines	C.04.3	If the probability of abuse and the expected damage are both high, patches are installed no later than within a week.	Function apps that use Python should use a specified 'Python version'	4.1.0 ↗
C.04.6 Technical vulnerability management - Timelines	C.04.6	Technical weaknesses can be remedied by performing patch management in a timely manner.	App Service apps should use latest 'HTTP Version'	4.0.0 ↗
C.04.6 Technical vulnerability management - Timelines	C.04.6	Technical weaknesses can be remedied by performing patch management in a timely manner.	App Service apps that use Java should use a specified 'Java version'	3.1.0 ↗
C.04.6 Technical vulnerability management - Timelines	C.04.6	Technical weaknesses can be remedied by performing patch management in a timely manner.	App Service apps that use PHP should use a specified 'PHP version'	3.2.0 ↗
C.04.6 Technical vulnerability management - Timelines	C.04.6	Technical weaknesses can be remedied by performing patch management in a timely manner.	App Service apps that use Python should use a specified 'Python version'	4.1.0 ↗
C.04.6 Technical vulnerability management - Timelines	C.04.6	Technical weaknesses can be remedied by performing patch management in a timely manner.	Function apps should use latest 'HTTP Version'	4.0.0 ↗
C.04.6 Technical vulnerability management - Timelines	C.04.6	Technical weaknesses can be remedied by performing patch management in a timely manner.	Function apps that use Java should use a specified 'Java version'	3.1.0 ↗

Domain	Control ID	Control title	Policy (Azure portal)	Policy version (GitHub)
C.04.6 Technical vulnerability management - Timelines	C.04.6	Technical weaknesses can be remedied by performing patch management in a timely manner.	Function apps that use Python should use a specified 'Python version'	4.1.0 ↗
C.04.7 Technical vulnerability management - Evaluated	C.04.7	Evaluations of technical vulnerabilities are recorded and reported.	App Service apps should have remote debugging turned off	2.0.0 ↗
C.04.7 Technical vulnerability management - Evaluated	C.04.7	Evaluations of technical vulnerabilities are recorded and reported.	App Service apps should not have CORS configured to allow every resource to access your apps	2.0.0 ↗
C.04.7 Technical vulnerability management - Evaluated	C.04.7	Evaluations of technical vulnerabilities are recorded and reported.	App Service apps should use latest 'HTTP Version'	4.0.0 ↗
C.04.7 Technical vulnerability management - Evaluated	C.04.7	Evaluations of technical vulnerabilities are recorded and reported.	App Service apps that use Java should use a specified 'Java version'	3.1.0 ↗
C.04.7 Technical vulnerability management - Evaluated	C.04.7	Evaluations of technical vulnerabilities are recorded and reported.	App Service apps that use PHP should use a specified 'PHP version'	3.2.0 ↗
C.04.7 Technical vulnerability management - Evaluated	C.04.7	Evaluations of technical vulnerabilities are recorded and reported.	App Service apps that use Python should use a specified 'Python version'	4.1.0 ↗
C.04.7 Technical vulnerability management - Evaluated	C.04.7	Evaluations of technical vulnerabilities are recorded and reported.	Function apps should have remote debugging turned off	2.0.0 ↗
C.04.7 Technical vulnerability management - Evaluated	C.04.7	Evaluations of technical vulnerabilities are recorded and reported.	Function apps should not have CORS configured to allow every resource to access your apps	2.0.0 ↗
C.04.7 Technical vulnerability	C.04.7	Evaluations of technical vulnerabilities are recorded	Function apps should use latest 'HTTP	4.0.0 ↗

Domain	Control ID	Control title	Policy (Azure portal)	Policy version (GitHub)
management - Evaluated		and reported.	Version' ↗	
C.04.7 Technical vulnerability management - Evaluated	C.04.7	Evaluations of technical vulnerabilities are recorded and reported.	Function apps that use Java should use a specified 'Java version' ↗	3.1.0 ↗
C.04.7 Technical vulnerability management - Evaluated	C.04.7	Evaluations of technical vulnerabilities are recorded and reported.	Function apps that use Python should use a specified 'Python version' ↗	4.1.0 ↗
U.05.1 Data protection - Cryptographic measures	U.05.1	Data transport is secured with cryptography where key management is carried out by the CSC itself if possible.	App Service apps should only be accessible over HTTPS ↗	4.0.0 ↗
U.05.1 Data protection - Cryptographic measures	U.05.1	Data transport is secured with cryptography where key management is carried out by the CSC itself if possible.	App Service apps should require FTPS only ↗	3.0.0 ↗
U.05.1 Data protection - Cryptographic measures	U.05.1	Data transport is secured with cryptography where key management is carried out by the CSC itself if possible.	App Service apps should use the latest TLS version ↗	2.1.0 ↗
U.05.1 Data protection - Cryptographic measures	U.05.1	Data transport is secured with cryptography where key management is carried out by the CSC itself if possible.	Function apps should only be accessible over HTTPS ↗	5.0.0 ↗
U.05.1 Data protection - Cryptographic measures	U.05.1	Data transport is secured with cryptography where key management is carried out by the CSC itself if possible.	Function apps should require FTPS only ↗	3.0.0 ↗
U.05.1 Data protection - Cryptographic measures	U.05.1	Data transport is secured with cryptography where key management is carried out by the CSC itself if possible.	Function apps should use the latest TLS version ↗	2.1.0 ↗
U.07.3 Data separation - Management features	U.07.3	U.07.3 - The privileges to view or modify CSC data and/or encryption keys are granted in a controlled manner and use is logged.	App Service apps should use managed identity ↗	3.0.0 ↗

Domain	Control ID	Control title	Policy (Azure portal)	Policy version (GitHub)
U.07.3 Data separation - Management features	U.07.3	U.07.3 - The privileges to view or modify CSC data and/or encryption keys are granted in a controlled manner and use is logged.	App Service Environment should have internal encryption enabled	1.0.1 ↗
U.07.3 Data separation - Management features	U.07.3	U.07.3 - The privileges to view or modify CSC data and/or encryption keys are granted in a controlled manner and use is logged.	Function apps should use managed identity	3.0.0 ↗
U.09.3 Malware Protection - Detection, prevention and recovery	U.09.3	The malware protection runs on different environments.	Function apps should use latest 'HTTP Version'	4.0.0 ↗
U.10.2 Access to IT services and data - Users	U.10.2	Under the responsibility of the CSP, access is granted to administrators.	App Service apps should use managed identity	3.0.0 ↗
U.10.2 Access to IT services and data - Users	U.10.2	Under the responsibility of the CSP, access is granted to administrators.	Function apps should use managed identity	3.0.0 ↗
U.10.3 Access to IT services and data - Users	U.10.3	Only users with authenticated equipment can access IT services and data.	App Service apps should have Client Certificates (Incoming client certificates) enabled	1.0.0 ↗
U.10.3 Access to IT services and data - Users	U.10.3	Only users with authenticated equipment can access IT services and data.	App Service apps should use managed identity	3.0.0 ↗
U.10.3 Access to IT services and data - Users	U.10.3	Only users with authenticated equipment can access IT services and data.	Function apps should use managed identity	3.0.0 ↗
U.10.5 Access to IT services and data - Competent	U.10.5	Access to IT services and data is limited by technical measures and has been implemented.	App Service apps should use managed identity	3.0.0 ↗
U.10.5 Access to IT services and data - Competent	U.10.5	Access to IT services and data is limited by technical	Function apps should use managed identity	3.0.0 ↗

Domain	Control ID	Control title	Policy (Azure portal)	Policy version (GitHub)
		measures and has been implemented.		
U.11.1 Cryptoservices - Policy	U.11.1	In the cryptography policy, at least the subjects in accordance with BIO have been elaborated.	App Service apps should only be accessible over HTTPS ↗	4.0.0 ↗
U.11.1 Cryptoservices - Policy	U.11.1	In the cryptography policy, at least the subjects in accordance with BIO have been elaborated.	App Service apps should require FTPS only ↗	3.0.0 ↗
U.11.1 Cryptoservices - Policy	U.11.1	In the cryptography policy, at least the subjects in accordance with BIO have been elaborated.	App Service apps should use the latest TLS version ↗	2.1.0 ↗
U.11.1 Cryptoservices - Policy	U.11.1	In the cryptography policy, at least the subjects in accordance with BIO have been elaborated.	Function apps should only be accessible over HTTPS ↗	5.0.0 ↗
U.11.1 Cryptoservices - Policy	U.11.1	In the cryptography policy, at least the subjects in accordance with BIO have been elaborated.	Function apps should require FTPS only ↗	3.0.0 ↗
U.11.1 Cryptoservices - Policy	U.11.1	In the cryptography policy, at least the subjects in accordance with BIO have been elaborated.	Function apps should use the latest TLS version ↗	2.1.0 ↗
U.11.2 Cryptoservices - Cryptographic measures	U.11.2	In case of PKoverheid certificates use PKoverheid requirements for key management. In other situations use ISO11770.	App Service apps should only be accessible over HTTPS ↗	4.0.0 ↗
U.11.2 Cryptoservices - Cryptographic measures	U.11.2	In case of PKoverheid certificates use PKoverheid requirements for key management. In other situations use ISO11770.	App Service apps should require FTPS only ↗	3.0.0 ↗
U.11.2 Cryptoservices - Cryptographic measures	U.11.2	In case of PKoverheid certificates use PKoverheid requirements for key	App Service apps should use the latest TLS version ↗	2.1.0 ↗

Domain	Control ID	Control title	Policy (Azure portal)	Policy version (GitHub)
		management. In other situations use ISO11770.		
U.11.2 Cryptoservices - Cryptographic measures	U.11.2	In case of PKoverheid certificates use PKoverheid requirements for key management. In other situations use ISO11770.	Function apps should only be accessible over HTTPS ↗	5.0.0 ↗
U.11.2 Cryptoservices - Cryptographic measures	U.11.2	In case of PKoverheid certificates use PKoverheid requirements for key management. In other situations use ISO11770.	Function apps should require FTPS only ↗	3.0.0 ↗
U.11.2 Cryptoservices - Cryptographic measures	U.11.2	In case of PKoverheid certificates use PKoverheid requirements for key management. In other situations use ISO11770.	Function apps should use the latest TLS version ↗	2.1.0 ↗
U.15.1 Logging and monitoring - Events logged	U.15.1	The violation of the policy rules is recorded by the CSP and the CSC.	App Service apps should have resource logs enabled ↗	2.0.1 ↗

PCI DSS 3.2.1

To review how the available Azure Policy built-ins for all Azure services map to this compliance standard, see [PCI DSS 3.2.1](#). For more information about this compliance standard, see [PCI DSS 3.2.1 ↗](#).

[Expand table](#)

Domain	Control ID	Control title	Policy (Azure portal)	Policy version (GitHub)
Requirement 3	3.4	PCI DSS requirement 3.4	App Service apps should only be accessible over HTTPS ↗	4.0.0 ↗
Requirement 3	3.4	PCI DSS requirement 3.4	Function apps should only be accessible over HTTPS ↗	5.0.0 ↗
Requirement 4	4.1	PCI DSS requirement 4.1	App Service apps should only be accessible over HTTPS ↗	4.0.0 ↗

Domain	Control ID	Control title	Policy (Azure portal)	Policy version (GitHub)
Requirement 4	4.1	PCI DSS requirement 4.1	Function apps should only be accessible over HTTPS ↗	5.0.0 ↗
Requirement 6	6.5.3	PCI DSS requirement 6.5.3	App Service apps should only be accessible over HTTPS ↗	4.0.0 ↗
Requirement 6	6.5.3	PCI DSS requirement 6.5.3	Function apps should only be accessible over HTTPS ↗	5.0.0 ↗

PCI DSS v4.0

To review how the available Azure Policy built-ins for all Azure services map to this compliance standard, see [Azure Policy Regulatory Compliance details for PCI DSS v4.0](#). For more information about this compliance standard, see [PCI DSS v4.0](#).

 Expand table

Domain	Control ID	Control title	Policy (Azure portal)	Policy version (GitHub)
Requirement 03: Protect Stored Account Data	3.5.1	Primary account number (PAN) is secured wherever it is stored	App Service apps should only be accessible over HTTPS ↗	4.0.0 ↗
Requirement 03: Protect Stored Account Data	3.5.1	Primary account number (PAN) is secured wherever it is stored	Function apps should only be accessible over HTTPS ↗	5.0.0 ↗
Requirement 06: Develop and Maintain Secure Systems and Software	6.2.4	Bespoke and custom software are developed securely	App Service apps should only be accessible over HTTPS ↗	4.0.0 ↗
Requirement 06: Develop and Maintain Secure Systems and Software	6.2.4	Bespoke and custom software are developed securely	Function apps should only be accessible over HTTPS ↗	5.0.0 ↗

Reserve Bank of India - IT Framework for NBFC

To review how the available Azure Policy built-ins for all Azure services map to this compliance standard, see [Azure Policy Regulatory Compliance - Reserve Bank of India - IT](#)

Framework for NBFC. For more information about this compliance standard, see [Reserve Bank of India - IT Framework for NBFC](#).

[Expand table](#)

Domain	Control ID	Control title	Policy (Azure portal)	Policy version (GitHub)
Information and Cyber Security	3.1.b	Segregation of Functions-3.1	[Deprecated]: Function apps should have 'Client Certificates (Incoming client certificates)' enabled	3.1.0-deprecated
Information and Cyber Security	3.1.b	Segregation of Functions-3.1	App Service apps should have remote debugging turned off	2.0.0
Information and Cyber Security	3.1.b	Segregation of Functions-3.1	Function apps should have remote debugging turned off	2.0.0
Information and Cyber Security	3.1.h	Public Key Infrastructure (PKI)-3.1	App Service apps should only be accessible over HTTPS	4.0.0
Information and Cyber Security	3.1.h	Public Key Infrastructure (PKI)-3.1	App Service apps should use the latest TLS version	2.1.0
Information and Cyber Security	3.1.h	Public Key Infrastructure (PKI)-3.1	App Service Environment should have internal encryption enabled	1.0.1
Information and Cyber Security	3.1.h	Public Key Infrastructure (PKI)-3.1	Function apps should only be accessible over HTTPS	5.0.0
Information and Cyber Security	3.1.h	Public Key Infrastructure (PKI)-3.1	Function apps should use the latest TLS version	2.1.0
Information and Cyber Security	3.8	Digital Signatures-3.8	[Deprecated]: Function apps should have 'Client Certificates (Incoming client certificates)' enabled	3.1.0-deprecated
Information and Cyber Security	3.8	Digital Signatures-3.8	App Service apps should have Client Certificates (Incoming client certificates) enabled	1.0.0

Reserve Bank of India IT Framework for Banks v2016

To review how the available Azure Policy built-ins for all Azure services map to this compliance standard, see [Azure Policy Regulatory Compliance - RBI ITF Banks v2016](#). For more information about this compliance standard, see [RBI ITF Banks v2016 \(PDF\)](#).

[Expand table](#)

Domain	Control ID	Control title	Policy (Azure portal)	Policy version (GitHub)
Advanced Real-Timethreat Defenceand Management	Advanced Real-Timethreat Defenceand Management-13.1	Advanced Real-Timethreat Defenceand Management-13.1	[Deprecated]: Function apps should have 'Client Certificates (Incoming client certificates)' enabled	3.1.0-deprecated
Network Management And Security	Network Device Configuration Management-4.3	Network Device Configuration Management-4.3	App Service apps should have Client Certificates (Incoming client certificates) enabled	1.0.0
Network Management And Security	Network Device Configuration Management-4.3	Network Device Configuration Management-4.3	App Service apps should have remote debugging turned off	2.0.0
Audit Log Settings	Audit Log Settings-17.1	Audit Log Settings-17.1	App Service apps should have resource logs enabled	2.0.1
Advanced Real-Timethreat Defenceand Management	Advanced Real-Timethreat Defenceand Management-13.1	Advanced Real-Timethreat Defenceand Management-13.1	App Service apps should not have CORS configured to allow every resource to access your apps	2.0.0
Secure Mail And Messaging Systems	Secure Mail And Messaging Systems-10.1	Secure Mail And Messaging Systems-10.1	App Service apps should only be accessible over HTTPS	4.0.0
Secure Mail And Messaging Systems	Secure Mail And Messaging Systems-10.1	Secure Mail And Messaging Systems-10.1	App Service apps should require FTPS only	3.0.0
User Access Control / Management	User Access Control / Management-8.4	User Access Control / Management-8.4	App Service apps should use managed identity	3.0.0
Secure Mail And Messaging Systems	Secure Mail And Messaging Systems-10.1	Secure Mail And Messaging Systems-10.1	App Service apps should use the latest TLS version	2.1.0
Network Management And Security	Network Device Configuration Management-4.3	Network Device Configuration Management-4.3	Function apps should have remote debugging turned off	2.0.0

Domain	Control ID	Control title	Policy (Azure portal)	Policy version (GitHub)
Advanced Real-Timethreat Defenceand Management		Advanced Real-Timethreat Defenceand Management-13.1	Function apps should not have CORS configured to allow every resource to access your apps ↗	2.0.0 ↗
Secure Mail And Messaging Systems		Secure Mail And Messaging Systems-10.1	Function apps should only be accessible over HTTPS ↗	5.0.0 ↗
Secure Mail And Messaging Systems		Secure Mail And Messaging Systems-10.1	Function apps should require FTPS only ↗	3.0.0 ↗
User Access Control / Management		User Access Control / Management-8.4	Function apps should use managed identity ↗	3.0.0 ↗
Secure Mail And Messaging Systems		Secure Mail And Messaging Systems-10.1	Function apps should use the latest TLS version ↗	2.1.0 ↗

RMIT Malaysia

To review how the available Azure Policy built-ins for all Azure services map to this compliance standard, see [Azure Policy Regulatory Compliance - RMIT Malaysia](#). For more information about this compliance standard, see [RMIT Malaysia](#) ↗.

 Expand table

Domain	Control ID	Control title	Policy (Azure portal)	Policy version (GitHub)
Cryptography	10.20	Cryptography - 10.20	[Deprecated]: Function apps should have 'Client Certificates (Incoming client certificates)' enabled ↗	3.1.0-deprecated ↗
Cryptography	10.20	Cryptography - 10.20	App Service apps should have Client Certificates (Incoming client certificates) enabled ↗	1.0.0 ↗
Access Control	10.54	Access Control - 10.54	App Service apps should have authentication enabled ↗	2.0.1 ↗

Domain	Control ID	Control title	Policy (Azure portal)	Policy version (GitHub)
Access Control	10.54	Access Control - 10.54	Function apps should have authentication enabled ↗	3.0.0 ↗
Access Control	10.54	Access Control - 10.54	Function apps should use managed identity ↗	3.0.0 ↗
Security of Digital Services	10.66	Security of Digital Services - 10.66	App Service apps should have resource logs enabled ↗	2.0.1 ↗
Security of Digital Services	10.68	Security of Digital Services - 10.68	App Service apps should use the latest TLS version ↗	2.1.0 ↗
Security of Digital Services	10.68	Security of Digital Services - 10.68	Function apps should use the latest TLS version ↗	2.1.0 ↗
Control Measures on Cybersecurity	Appendix 5.3	Control Measures on Cybersecurity - Appendix 5.3	App Service apps should not have CORS configured to allow every resource to access your apps ↗	2.0.0 ↗
Control Measures on Cybersecurity	Appendix 5.3	Control Measures on Cybersecurity - Appendix 5.3	App Service apps should only be accessible over HTTPS ↗	4.0.0 ↗
Control Measures on Cybersecurity	Appendix 5.3	Control Measures on Cybersecurity - Appendix 5.3	App Service apps should require FTPS only ↗	3.0.0 ↗
Control Measures on Cybersecurity	Appendix 5.3	Control Measures on Cybersecurity - Appendix 5.3	App Service apps should use latest 'HTTP Version' ↗	4.0.0 ↗
Control Measures on Cybersecurity	Appendix 5.3	Control Measures on Cybersecurity - Appendix 5.3	Function apps should only be accessible over HTTPS ↗	5.0.0 ↗
Control Measures on Cybersecurity	Appendix 5.3	Control Measures on Cybersecurity - Appendix 5.3	Function apps should require FTPS only ↗	3.0.0 ↗
Control Measures on Cybersecurity	Appendix 5.3	Control Measures on Cybersecurity - Appendix 5.3	Function apps should use latest 'HTTP Version' ↗	4.0.0 ↗
Control Measures on Cybersecurity	Appendix 5.7	Control Measures on Cybersecurity - Appendix 5.7	App Service apps should have remote debugging turned off ↗	2.0.0 ↗

Domain	Control ID	Control title	Policy (Azure portal)	Policy version (GitHub)
Control Measures on Cybersecurity	Appendix 5.7	Control Measures on Cybersecurity - Appendix 5.7	Function apps should have remote debugging turned off ↗	2.0.0 ↗
Control Measures on Cybersecurity	Appendix 5.7	Control Measures on Cybersecurity - Appendix 5.7	Function apps should not have CORS configured to allow every resource to access your apps ↗	2.0.0 ↗

Spain ENS

To review how the available Azure Policy built-ins for all Azure services map to this compliance standard, see [Azure Policy Regulatory Compliance details for Spain ENS](#). For more information about this compliance standard, see [CCN-STIC 884 ↗](#).

[Expand table](#)

Domain	Control ID	Control title	Policy (Azure portal)	Policy version (GitHub)
Protective Measures	mp.s.3	Protection of services	App Service app slots should be injected into a virtual network ↗	1.0.0 ↗
Protective Measures	mp.s.3	Protection of services	App Service app slots should use latest 'HTTP Version' ↗	1.0.0 ↗
Protective Measures	mp.s.3	Protection of services	App Service apps should be injected into a virtual network ↗	3.0.0 ↗
Protective Measures	mp.s.3	Protection of services	App Service apps should have authentication enabled ↗	2.0.1 ↗
Protective Measures	mp.s.3	Protection of services	App Service apps should use latest 'HTTP Version' ↗	4.0.0 ↗
Protective Measures	mp.s.3	Protection of services	Function app slots should use latest 'HTTP Version' ↗	1.0.0 ↗
Protective Measures	mp.s.3	Protection of services	Function apps should use latest 'HTTP Version' ↗	4.0.0 ↗
Protective Measures	mp.sw.2	Protection of IT applications	Function apps that use Python should use a specified 'Python version' ↗	4.1.0 ↗

Domain	Control ID	Control title	Policy (Azure portal)	Policy version (GitHub)
Operational framework	op.exp.4	Operation	App Service apps that use Python should use a specified 'Python version' ↗	4.1.0 ↗
Operational framework	op.exp.7	Operation	App Service apps should have resource logs enabled ↗	2.0.1 ↗
Operational framework	op.exp.8	Operation	App Service app slots should have resource logs enabled ↗	1.0.0 ↗

SWIFT CSP-CSCF v2021

To review how the available Azure Policy built-ins for all Azure services map to this compliance standard, see [Azure Policy Regulatory Compliance details for SWIFT CSP-CSCF v2021](#). For more information about this compliance standard, see [SWIFT CSP CSCF v2021](#).

[\[+\] Expand table](#)

Domain	Control ID	Control title	Policy (Azure portal)	Policy version (GitHub)
SWIFT Environment Protection	1.1	SWIFT Environment Protection	App Service apps should have remote debugging turned off ↗	2.0.0 ↗
SWIFT Environment Protection	1.1	SWIFT Environment Protection	App Service apps should use a virtual network service endpoint ↗	2.0.1 ↗
SWIFT Environment Protection	1.1	SWIFT Environment Protection	Function apps should have remote debugging turned off ↗	2.0.0 ↗
SWIFT Environment Protection	1.2	Operating System Privileged Account Control	App Service apps should have remote debugging turned off ↗	2.0.0 ↗
SWIFT Environment Protection	1.2	Operating System Privileged Account Control	Function apps should have remote debugging turned off ↗	2.0.0 ↗
Reduce Attack Surface and Vulnerabilities	2.1	Internal Data Flow Security	App Service apps should have Client Certificates	1.0.0 ↗

Domain	Control ID	Control title	Policy (Azure portal)	Policy version (GitHub)
			(Incoming client certificates) enabled ↗	
Reduce Attack Surface and Vulnerabilities	2.1	Internal Data Flow Security	App Service apps should only be accessible over HTTPS ↗	4.0.0 ↗
Reduce Attack Surface and Vulnerabilities	2.1	Internal Data Flow Security	App Service apps should use managed identity ↗	3.0.0 ↗
Reduce Attack Surface and Vulnerabilities	2.1	Internal Data Flow Security	App Service apps should use the latest TLS version ↗	2.1.0 ↗
Reduce Attack Surface and Vulnerabilities	2.1	Internal Data Flow Security	Function apps should only be accessible over HTTPS ↗	5.0.0 ↗
Reduce Attack Surface and Vulnerabilities	2.1	Internal Data Flow Security	Function apps should use managed identity ↗	3.0.0 ↗
Reduce Attack Surface and Vulnerabilities	2.1	Internal Data Flow Security	Function apps should use the latest TLS version ↗	2.1.0 ↗
Reduce Attack Surface and Vulnerabilities	2.4A	Back-office Data Flow Security	App Service apps should have Client Certificates (Incoming client certificates) enabled ↗	1.0.0 ↗
Reduce Attack Surface and Vulnerabilities	2.4A	Back-office Data Flow Security	App Service apps should only be accessible over HTTPS ↗	4.0.0 ↗
Reduce Attack Surface and Vulnerabilities	2.4A	Back-office Data Flow Security	Function apps should only be accessible over HTTPS ↗	5.0.0 ↗
Reduce Attack Surface and Vulnerabilities	2.5A	External Transmission Data Protection	App Service apps should only be accessible over HTTPS ↗	4.0.0 ↗
Reduce Attack Surface and Vulnerabilities	2.5A	External Transmission Data Protection	Function apps should only be accessible over HTTPS ↗	5.0.0 ↗
Reduce Attack Surface and Vulnerabilities	2.6	Operator Session Confidentiality and Integrity	App Service apps should use the latest TLS version ↗	2.1.0 ↗
Reduce Attack Surface and Vulnerabilities	2.6	Operator Session Confidentiality and Integrity	Function apps should use the latest TLS version ↗	2.1.0 ↗

Domain	Control ID	Control title	Policy (Azure portal)	Policy version (GitHub)
Manage Identities and Segregate Privileges	5.2	Token Management	App Service apps should use managed identity ↗	3.0.0 ↗
Manage Identities and Segregate Privileges	5.2	Token Management	Function apps should use managed identity ↗	3.0.0 ↗
Manage Identities and Segregate Privileges	5.4	Physical and Logical Password Storage	App Service apps should use managed identity ↗	3.0.0 ↗
Manage Identities and Segregate Privileges	5.4	Physical and Logical Password Storage	Function apps should use managed identity ↗	3.0.0 ↗
Detect Anomalous Activity to Systems or Transaction Records	6.2	Software Integrity	App Service apps should have remote debugging turned off ↗	2.0.0 ↗
Detect Anomalous Activity to Systems or Transaction Records	6.2	Software Integrity	Function apps should have remote debugging turned off ↗	2.0.0 ↗
Detect Anomalous Activity to Systems or Transaction Records	6.5A	Intrusion Detection	App Service apps should have remote debugging turned off ↗	2.0.0 ↗
Detect Anomalous Activity to Systems or Transaction Records	6.5A	Intrusion Detection	App Service apps should not have CORS configured to allow every resource to access your apps ↗	2.0.0 ↗
Detect Anomalous Activity to Systems or Transaction Records	6.5A	Intrusion Detection	Function apps should have remote debugging turned off ↗	2.0.0 ↗
Detect Anomalous Activity to Systems or Transaction Records	6.5A	Intrusion Detection	Function apps should not have CORS configured to allow every resource to access your apps ↗	2.0.0 ↗

SWIFT CSP-CSCF v2022

To review how the available Azure Policy built-ins for all Azure services map to this compliance standard, see [Azure Policy Regulatory Compliance details for SWIFT CSP-CSCF v2022](#). For more information about this compliance standard, see [SWIFT CSP CSCF v2022 ↗](#).

expand table Expand table

Domain	Control ID	Control title	Policy (Azure portal)	Policy version (GitHub)
1. Restrict Internet Access & Protect Critical Systems from General IT Environment	1.1	Ensure the protection of the user's local SWIFT infrastructure from potentially compromised elements of the general IT environment and external environment.	App Service apps should use a virtual network service endpoint	2.0.1 ↗
1. Restrict Internet Access & Protect Critical Systems from General IT Environment	1.5A	Ensure the protection of the customer's connectivity infrastructure from external environment and potentially compromised elements of the general IT environment.	App Service apps should use a virtual network service endpoint	2.0.1 ↗
6. Detect Anomalous Activity to Systems or Transaction Records	6.4	Record security events and detect anomalous actions and operations within the local SWIFT environment.	App Service apps should have resource logs enabled	2.0.1 ↗

System and Organization Controls (SOC) 2

To review how the available Azure Policy built-ins for all Azure services map to this compliance standard, see [Azure Policy Regulatory Compliance details for System and Organization Controls \(SOC\) 2](#). For more information about this compliance standard, see [System and Organization Controls \(SOC\) 2](#).

Expand table

Domain	Control ID	Control title	Policy (Azure portal)	Policy version (GitHub)
Logical and Physical Access Controls	CC6.1	Logical access security software, infrastructure, and architectures	App Service apps should only be accessible over HTTPS	4.0.0 ↗
Logical and Physical Access Controls	CC6.1	Logical access security software, infrastructure, and architectures	App Service apps should require FTPS only	3.0.0 ↗
Logical and Physical Access Controls	CC6.1	Logical access security software,	Function apps should only be accessible over HTTPS	5.0.0 ↗

Domain	Control ID	Control title	Policy (Azure portal)	Policy version (GitHub)
		infrastructure, and architectures		
Logical and Physical Access Controls	CC6.1	Logical access security software, infrastructure, and architectures	Function apps should require FTPS only ↗	3.0.0 ↗
Logical and Physical Access Controls	CC6.1	Logical access security software, infrastructure, and architectures	Function apps should use the latest TLS version ↗	2.1.0 ↗
Logical and Physical Access Controls	CC6.6	Security measures against threats outside system boundaries	App Service apps should only be accessible over HTTPS ↗	4.0.0 ↗
Logical and Physical Access Controls	CC6.6	Security measures against threats outside system boundaries	App Service apps should require FTPS only ↗	3.0.0 ↗
Logical and Physical Access Controls	CC6.6	Security measures against threats outside system boundaries	Function apps should only be accessible over HTTPS ↗	5.0.0 ↗
Logical and Physical Access Controls	CC6.6	Security measures against threats outside system boundaries	Function apps should require FTPS only ↗	3.0.0 ↗
Logical and Physical Access Controls	CC6.6	Security measures against threats outside system boundaries	Function apps should use the latest TLS version ↗	2.1.0 ↗
Logical and Physical Access Controls	CC6.7	Restrict the movement of information to authorized users	App Service apps should only be accessible over HTTPS ↗	4.0.0 ↗
Logical and Physical Access Controls	CC6.7	Restrict the movement of information to authorized users	App Service apps should require FTPS only ↗	3.0.0 ↗
Logical and Physical Access Controls	CC6.7	Restrict the movement of information to authorized users	Function apps should only be accessible over HTTPS ↗	5.0.0 ↗

Domain	Control ID	Control title	Policy (Azure portal)	Policy version (GitHub)
Logical and Physical Access Controls	CC6.7	Restrict the movement of information to authorized users	Function apps should require FTPS only ↗	3.0.0 ↗
Logical and Physical Access Controls	CC6.7	Restrict the movement of information to authorized users	Function apps should use the latest TLS version ↗	2.1.0 ↗
Logical and Physical Access Controls	CC6.8	Prevent or detect against unauthorized or malicious software	[Deprecated]: Function apps should have 'Client Certificates (Incoming client certificates)' enabled ↗	3.1.0-deprecated ↗
Logical and Physical Access Controls	CC6.8	Prevent or detect against unauthorized or malicious software	App Service apps should have Client Certificates (Incoming client certificates) enabled ↗	1.0.0 ↗
Logical and Physical Access Controls	CC6.8	Prevent or detect against unauthorized or malicious software	App Service apps should have remote debugging turned off ↗	2.0.0 ↗
Logical and Physical Access Controls	CC6.8	Prevent or detect against unauthorized or malicious software	App Service apps should not have CORS configured to allow every resource to access your apps ↗	2.0.0 ↗
Logical and Physical Access Controls	CC6.8	Prevent or detect against unauthorized or malicious software	App Service apps should use latest 'HTTP Version' ↗	4.0.0 ↗
Logical and Physical Access Controls	CC6.8	Prevent or detect against unauthorized or malicious software	Function apps should have remote debugging turned off ↗	2.0.0 ↗
Logical and Physical Access Controls	CC6.8	Prevent or detect against unauthorized or malicious software	Function apps should not have CORS configured to allow every resource to access your apps ↗	2.0.0 ↗
Logical and Physical Access Controls	CC6.8	Prevent or detect against unauthorized or malicious software	Function apps should use latest 'HTTP Version' ↗	4.0.0 ↗
Change Management	CC8.1	Changes to infrastructure, data, and software	[Deprecated]: Function apps should have 'Client Certificates (Incoming client certificates)' enabled ↗	3.1.0-deprecated ↗

Domain	Control ID	Control title	Policy (Azure portal)	Policy version (GitHub)
Change Management	CC8.1	Changes to infrastructure, data, and software	App Service apps should have Client Certificates (Incoming client certificates) enabled ↗	1.0.0 ↗
Change Management	CC8.1	Changes to infrastructure, data, and software	App Service apps should have remote debugging turned off ↗	2.0.0 ↗
Change Management	CC8.1	Changes to infrastructure, data, and software	App Service apps should not have CORS configured to allow every resource to access your apps ↗	2.0.0 ↗
Change Management	CC8.1	Changes to infrastructure, data, and software	App Service apps should use latest 'HTTP Version' ↗	4.0.0 ↗
Change Management	CC8.1	Changes to infrastructure, data, and software	Function apps should have remote debugging turned off ↗	2.0.0 ↗
Change Management	CC8.1	Changes to infrastructure, data, and software	Function apps should not have CORS configured to allow every resource to access your apps ↗	2.0.0 ↗
Change Management	CC8.1	Changes to infrastructure, data, and software	Function apps should use latest 'HTTP Version' ↗	4.0.0 ↗

UK OFFICIAL and UK NHS

To review how the available Azure Policy built-ins for all Azure services map to this compliance standard, see [Azure Policy Regulatory Compliance - UK OFFICIAL and UK NHS](#). For more information about this compliance standard, see [UK OFFICIAL](#) ↗.

[Expand table](#)

Domain	Control ID	Control title	Policy (Azure portal)	Policy version (GitHub)
Data in transit protection	1	Data in transit protection	App Service apps should only be accessible over HTTPS ↗	4.0.0 ↗

Domain	Control ID	Control title	Policy (Azure portal)	Policy version (GitHub)
Data in transit protection	1	Data in transit protection	Function apps should only be accessible over HTTPS ↗	5.0.0 ↗
External interface protection	11	External interface protection	App Service apps should have remote debugging turned off ↗	2.0.0 ↗
External interface protection	11	External interface protection	Function apps should have remote debugging turned off ↗	2.0.0 ↗

Next steps

- Learn more about [Azure Policy Regulatory Compliance](#).
- See the built-ins on the [Azure Policy GitHub repo](#) ↗.

Feedback

Was this page helpful?

Yes
 No

[Provide product feedback](#) ↗ | [Get help at Microsoft Q&A](#)

App Service networking features

Article • 03/03/2023

You can deploy applications in Azure App Service in multiple ways. By default, apps hosted in App Service are accessible directly through the internet and can reach only internet-hosted endpoints. But for many applications, you need to control the inbound and outbound network traffic. There are several features in App Service to help you meet those needs. The challenge is knowing which feature to use to solve a given problem. This article will help you determine which feature to use, based on some example use cases.

There are two main deployment types for Azure App Service:

- The multi-tenant public service hosts App Service plans in the Free, Shared, Basic, Standard, Premium, PremiumV2, and PremiumV3 pricing SKUs.
- The single-tenant App Service Environment (ASE) hosts Isolated SKU App Service plans directly in your Azure virtual network.

The features you use will depend on whether you're in the multi-tenant service or in an ASE.

ⓘ Note

Networking features are not available for apps deployed in Azure Arc.

Multi-tenant App Service networking features

Azure App Service is a distributed system. The roles that handle incoming HTTP or HTTPS requests are called *front ends*. The roles that host the customer workload are called *workers*. All the roles in an App Service deployment exist in a multi-tenant network. Because there are many different customers in the same App Service scale unit, you can't connect the App Service network directly to your network.

Instead of connecting the networks, you need features to handle the various aspects of application communication. The features that handle requests *to* your app can't be used to solve problems when you're making calls *from* your app. Likewise, the features that solve problems for calls from your app can't be used to solve problems to your app.

Inbound features

Outbound features

Inbound features	Outbound features
App-assigned address	Hybrid Connections
Access restrictions	Gateway-required virtual network integration
Service endpoints	Virtual network integration
Private endpoints	

Other than noted exceptions, you can use all of these features together. You can mix the features to solve your problems.

Use cases and features

For any given use case, there might be a few ways to solve the problem. Choosing the best feature sometimes goes beyond the use case itself. The following inbound use cases suggest how to use App Service networking features to solve problems with controlling traffic going to your app:

Inbound use case	Feature
Support IP-based SSL needs for your app	App-assigned address
Support unshared dedicated inbound address for your app	App-assigned address
Restrict access to your app from a set of well-defined addresses	Access restrictions
Restrict access to your app from resources in a virtual network	Service endpoints Internal Load Balancer (ILB) ASE Private endpoints
Expose your app on a private IP in your virtual network	ILB ASE Private endpoints Private IP for inbound traffic on an Application Gateway instance with service endpoints
Protect your app with a web application firewall (WAF)	Application Gateway and ILB ASE Application Gateway with private endpoints Application Gateway with service endpoints Azure Front Door with access restrictions
Load balance traffic to your apps in different regions	Azure Front Door with access restrictions

Inbound use case	Feature
Load balance traffic in the same region	Application Gateway with service endpoints

The following outbound use cases suggest how to use App Service networking features to solve outbound access needs for your app:

Outbound use case	Feature
Access resources in an Azure virtual network in the same region	Virtual network integration ASE
Access resources in an Azure virtual network in a different region	virtual network integration and virtual network peering Gateway-required virtual network integration ASE and virtual network peering
Access resources secured with service endpoints	virtual network integration ASE
Access resources in a private network that's not connected to Azure	Hybrid Connections
Access resources across Azure ExpressRoute circuits	virtual network integration ASE
Secure outbound traffic from your web app	virtual network integration and network security groups ASE
Route outbound traffic from your web app	virtual network integration and route tables ASE

Default networking behavior

Azure App Service scale units support many customers in each deployment. The Free and Shared SKU plans host customer workloads on multi-tenant workers. The Basic and higher plans host customer workloads that are dedicated to only one App Service plan. If you have a Standard App Service plan, all the apps in that plan will run on the same worker. If you scale out the worker, all the apps in that App Service plan will be replicated on a new worker for each instance in your App Service plan.

Outbound addresses

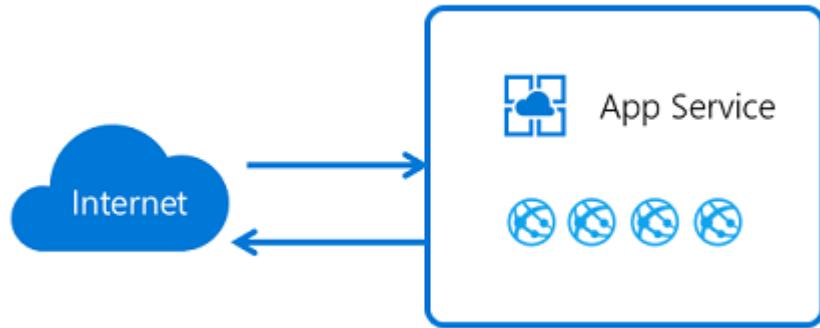
The worker VMs are broken down in large part by the App Service plans. The Free, Shared, Basic, Standard, and Premium plans all use the same worker VM type. The PremiumV2 plan uses another VM type. PremiumV3 uses yet another VM type. When you change the VM family, you get a different set of outbound addresses. If you scale from Standard to PremiumV2, your outbound addresses will change. If you scale from PremiumV2 to PremiumV3, your outbound addresses will change. In some older scale units, both the inbound and outbound addresses will change when you scale from Standard to PremiumV2.

There are many addresses that are used for outbound calls. The outbound addresses used by your app for making outbound calls are listed in the properties for your app. These addresses are shared by all the apps running on the same worker VM family in the App Service deployment. If you want to see all the addresses that your app might use in a scale unit, there's a property called `possibleOutboundAddresses` that will list them.

The screenshot shows the Azure portal's Properties blade for an App Service named "vnet-integration-app". The left sidebar lists various settings: Deployment Center, Configuration, Authentication / Authorization, Application Insights, Identity, Backups, Custom domains, SSL settings, Networking, Scale up (App Service plan), Scale out (App Service plan), WebJobs, Push, MySQL In App, and Properties. The Properties section is currently selected. Key details shown include:

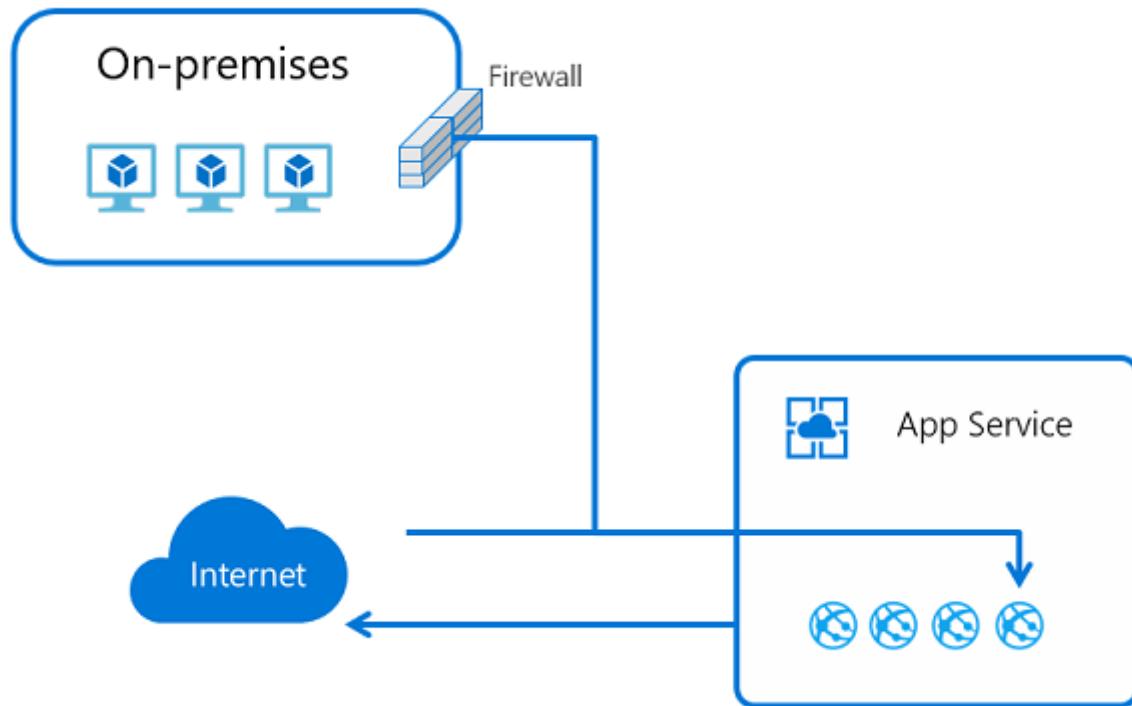
- Status: Running
- URL: vnet-integration-app.azurewebsites.net
- Virtual IP address: No IP-based SSL binding is configured
- Mode: Standard
- Outbound IP addresses: 13.90.143.69, 13.90.251.106, 13.90.192.153, 13.90.199.220, 40.71.205.9
- Additional Outbound IP Addresses: 13.90.143.69, 13.90.251.106, 13.90.192.153, 13.90.199.220, 40.71.205.9, 13.90.197.3, 13.90.197.145
- Deployment Trigger URL: [https://\\$vnet-integration-app:cwQd7bev5y9020aPAtwZlsJjYqYMMMPZM5kZtFpgstZ8svPq0ktyFpjlf8S@vnet-integrati...](https://$vnet-integration-app:cwQd7bev5y9020aPAtwZlsJjYqYMMMPZM5kZtFpgstZ8svPq0ktyFpjlf8S@vnet-integrati...)

App Service has many endpoints that are used to manage the service. Those addresses are published in a separate document and are also in the `AppServiceManagement` IP service tag. The `AppServiceManagement` tag is used only in App Service Environments where you need to allow such traffic. The App Service inbound addresses are tracked in the `AppService` IP service tag. There's no IP service tag that contains the outbound addresses used by App Service.



App-assigned address

The app-assigned address feature is an offshoot of the IP-based SSL capability. You access it by setting up SSL with your app. You can use this feature for IP-based SSL calls. You can also use it to give your app an address that only it has.



When you use an app-assigned address, your traffic still goes through the same front-end roles that handle all the incoming traffic into the App Service scale unit. But the address that's assigned to your app is used only by your app. Use cases for this feature:

- Support IP-based SSL needs for your app.
- Set a dedicated address for your app that's not shared.

To learn how to set an address on your app, see [Add a TLS/SSL certificate in Azure App Service](#).

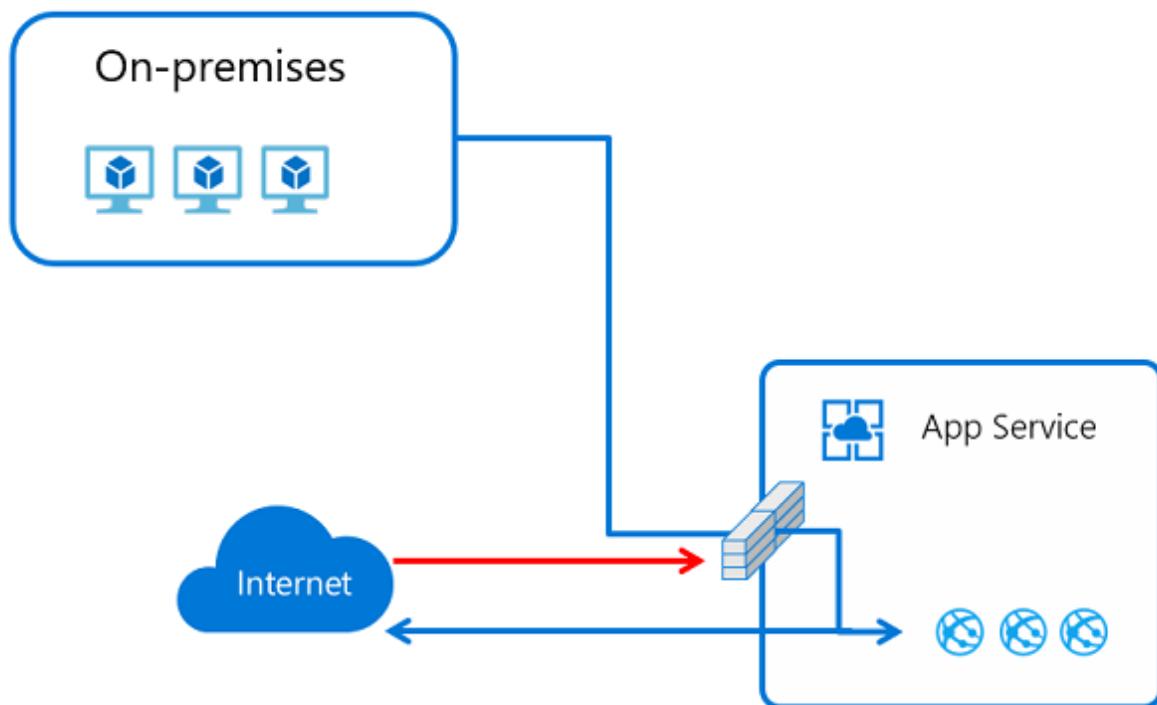
Access restrictions

Access restrictions let you filter *inbound* requests. The filtering action takes place on the front-end roles that are upstream from the worker roles where your apps are running. Because the front-end roles are upstream from the workers, you can think of access restrictions as network-level protection for your apps. For more information about access restrictions, see [Access restrictions overview](#).

This feature allows you to build a list of allow and deny rules that are evaluated in priority order. It's similar to the network security group (NSG) feature in Azure networking. You can use this feature in an ASE or in the multi-tenant service. When you use it with an ILB ASE, you can restrict access from private address blocks. To learn how to enable this feature, see [Configuring access restrictions](#).

 **Note**

Up to 512 access restriction rules can be configured per app.



Private endpoint

Private endpoint is a network interface that connects you privately and securely to your Web App by Azure private link. Private endpoint uses a private IP address from your virtual network, effectively bringing the web app into your virtual network. This feature is

only for *inbound* flows to your web app. For more information, see [Using private endpoints for Azure Web App](#).

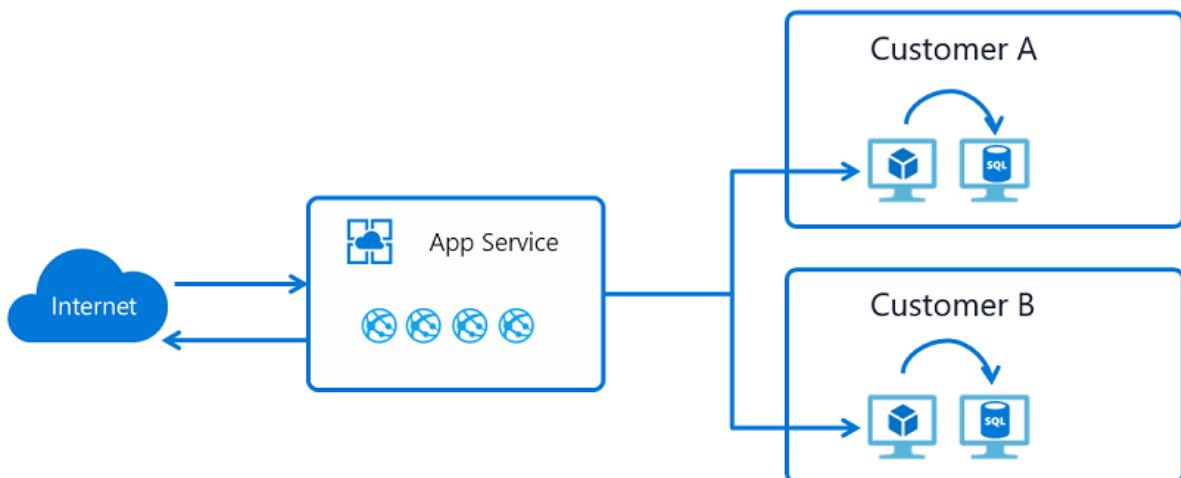
Some use cases for this feature:

- Restrict access to your app from resources in a virtual network.
- Expose your app on a private IP in your virtual network.
- Protect your app with a WAF.

Private endpoints prevent data exfiltration because the only thing you can reach across the private endpoint is the app with which it's configured.

Hybrid Connections

App Service Hybrid Connections enables your apps to make *outbound* calls to specified TCP endpoints. The endpoint can be on-premises, in a virtual network, or anywhere that allows outbound traffic to Azure on port 443. To use the feature, you need to install a relay agent called Hybrid Connection Manager on a Windows Server 2012 or newer host. Hybrid Connection Manager needs to be able to reach Azure Relay at port 443. You can download Hybrid Connection Manager from the App Service Hybrid Connections UI in the portal.



App Service Hybrid Connections is built on the Azure Relay Hybrid Connections capability. App Service uses a specialized form of the feature that only supports making outbound calls from your app to a TCP host and port. This host and port only need to resolve on the host where Hybrid Connection Manager is installed.

When the app, in App Service, does a DNS lookup on the host and port defined in your hybrid connection, the traffic automatically redirects to go through the hybrid connection and out of Hybrid Connection Manager. To learn more, see [App Service Hybrid Connections](#).

This feature is commonly used to:

- Access resources in private networks that aren't connected to Azure with a VPN or ExpressRoute.
- Support the migration of on-premises apps to App Service without the need to move supporting databases.
- Provide access with improved security to a single host and port per hybrid connection. Most networking features open access to a network. With Hybrid Connections, you can only reach the single host and port.
- Cover scenarios not covered by other outbound connectivity methods.
- Perform development in App Service in a way that allows the apps to easily use on-premises resources.

Because this feature enables access to on-premises resources without an inbound firewall hole, it's popular with developers. The other outbound App Service networking features are related to Azure Virtual Network. Hybrid Connections doesn't depend on going through a virtual network. It can be used for a wider variety of networking needs.

App Service Hybrid Connections is unaware of what you're doing on top of it. So you can use it to access a database, a web service, or an arbitrary TCP socket on a mainframe. The feature essentially tunnels TCP packets.

Hybrid Connections is popular for development, but it's also used in production applications. It's great for accessing a web service or database, but it's not appropriate for situations that involve creating many connections.

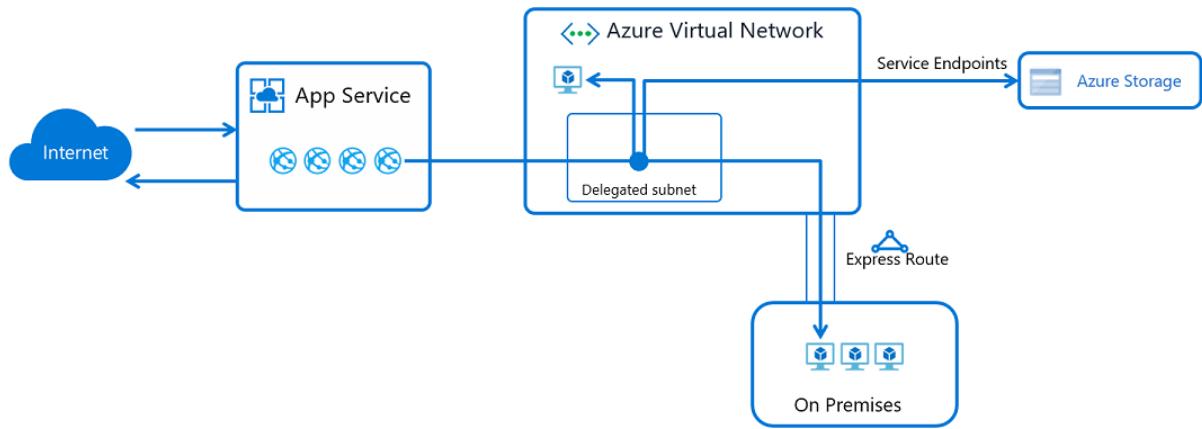
Virtual network integration

App Service virtual network integration enables your app to make *outbound* requests into an Azure virtual network.

The virtual network integration feature enables you to place the back end of your app in a subnet in a Resource Manager virtual network. The virtual network must be in the same region as your app. This feature isn't available from an App Service Environment, which is already in a virtual network. Use cases for this feature:

- Access resources in Resource Manager virtual networks in the same region.
- Access resources in peered virtual networks, including cross region connections.
- Access resources that are secured with service endpoints.
- Access resources that are accessible across ExpressRoute or VPN connections.
- Access resources in private networks without the need and cost of a Virtual Network gateway.
- Help to secure all outbound traffic.

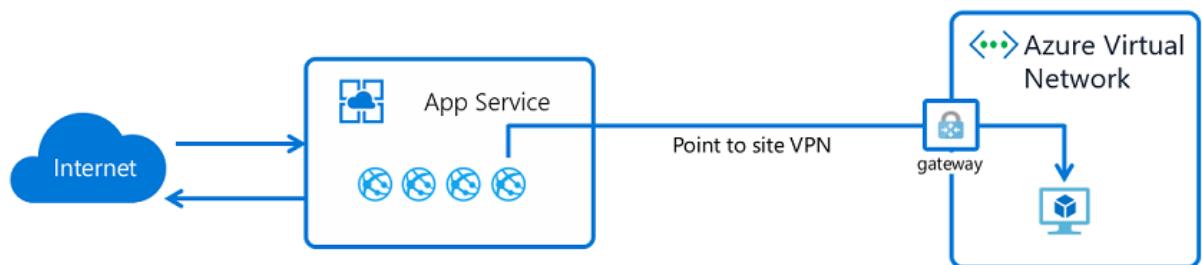
- Force tunnel all outbound traffic.



To learn more, see [App Service virtual network integration](#).

Gateway-required virtual network integration

Gateway-required virtual network integration was the first edition of virtual network integration in App Service. The feature works by connecting the host your app is running on to a Virtual Network gateway on your virtual network by using a point-to-site VPN. When you configure the feature, your app gets one of the point-to-site assigned addresses assigned to each instance.



Gateway required integration allows you to connect directly to a virtual network in another region without peering and to connect to a classic virtual network. The feature is limited to App Service Windows plans and doesn't work with ExpressRoute-connected virtual networks. It's recommended to use the regional virtual network integration. For more information on this feature, see [App Service virtual network integration](#).

App Service Environment

An App Service Environment (ASE) is a single-tenant deployment of the Azure App Service that runs in your virtual network. Some cases such for this feature:

- Access resources in your virtual network.
- Access resources across ExpressRoute.

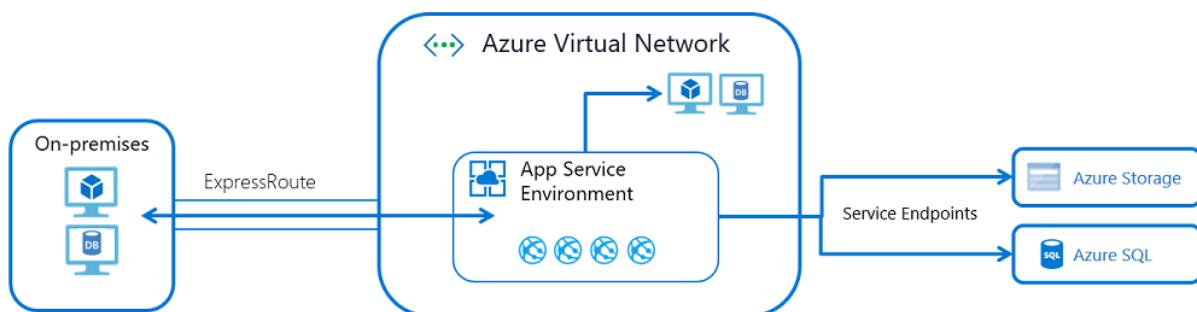
- Expose your apps with a private address in your virtual network.
- Access resources across service endpoints.
- Access resources across private endpoints.

With an ASE, you don't need to use virtual network integration because the ASE is already in your virtual network. If you want to access resources like SQL or Azure Storage over service endpoints, enable service endpoints on the ASE subnet. If you want to access resources in the virtual network or private endpoints in the virtual network, you don't need to do any extra configuration. If you want to access resources across ExpressRoute, you're already in the virtual network, and don't need to configure anything on the ASE or the apps in it.

Because the apps in an ILB ASE can be exposed on a private IP address, you can easily add WAF devices to expose just the apps that you want to the internet and help keep the rest secure. This feature can help make the development of multi-tier applications easier.

Some things aren't currently possible from the multi-tenant service but are possible from an ASE. Here are some examples:

- Host your apps in a single-tenant service.
- Scale up to many more instances than are possible in the multi-tenant service.
- Load private CA client certificates for use by your apps with private CA-secured endpoints.
- Force TLS 1.2 across all apps hosted in the system without any ability to disable it at the app level.



The ASE provides the best story around isolated and dedicated app hosting, but it does involve some management challenges. Some things to consider before you use an operational ASE:

- An ASE runs inside your virtual network, but it does have dependencies outside the virtual network. Those dependencies must be allowed. For more information, see [Networking considerations for an App Service Environment](#).

- An ASE doesn't scale immediately like the multi-tenant service. You need to anticipate scaling needs rather than reactively scaling.
- An ASE does have a higher up-front cost. To get the most out of your ASE, you should plan to put many workloads into one ASE rather than using it for small efforts.
- The apps in an ASE can't selectively restrict access to some apps in the ASE and not others.
- An ASE is in a subnet, and any networking rules apply to all the traffic to and from that ASE. If you want to assign inbound traffic rules for just one app, use access restrictions.

Combining features

The features noted for the multi-tenant service can be used together to solve more elaborate use cases. Two of the more common use cases are described here, but that's just examples. By understanding what the various features do, you can meet nearly all your system architecture needs.

Place an app into a virtual network

You might wonder how to put an app into a virtual network. If you put your app into a virtual network, the inbound and outbound endpoints for the app are within the virtual network. An ASE is the best way to solve this problem. But you can meet most of your needs within the multi-tenant service by combining features. For example, you can host intranet-only applications with private inbound and outbound addresses by:

- Creating an application gateway with private inbound and outbound addresses.
- Securing inbound traffic to your app with service endpoints.
- Using the virtual network integration feature so the back end of your app is in your virtual network.

This deployment style won't give you a dedicated address for outbound traffic to the internet or the ability to lock down all outbound traffic from your app. It will give you a much of what you would only otherwise get with an ASE.

Create multi-tier applications

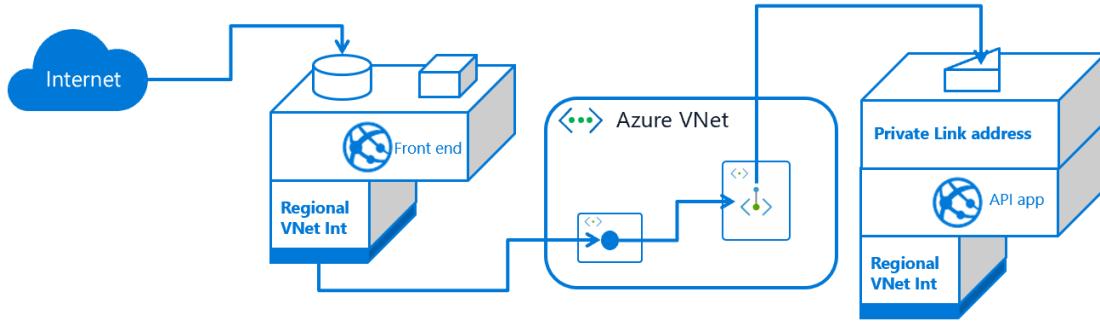
A multi-tier application is an application in which the API back-end apps can be accessed only from the front-end tier. There are two ways to create a multi-tier application. Both start by using virtual network integration to connect your front-end web app to a subnet in a virtual network. Doing so will enable your web app to make

calls into your virtual network. After your front-end app is connected to the virtual network, you need to decide how to lock down access to your API application. You can:

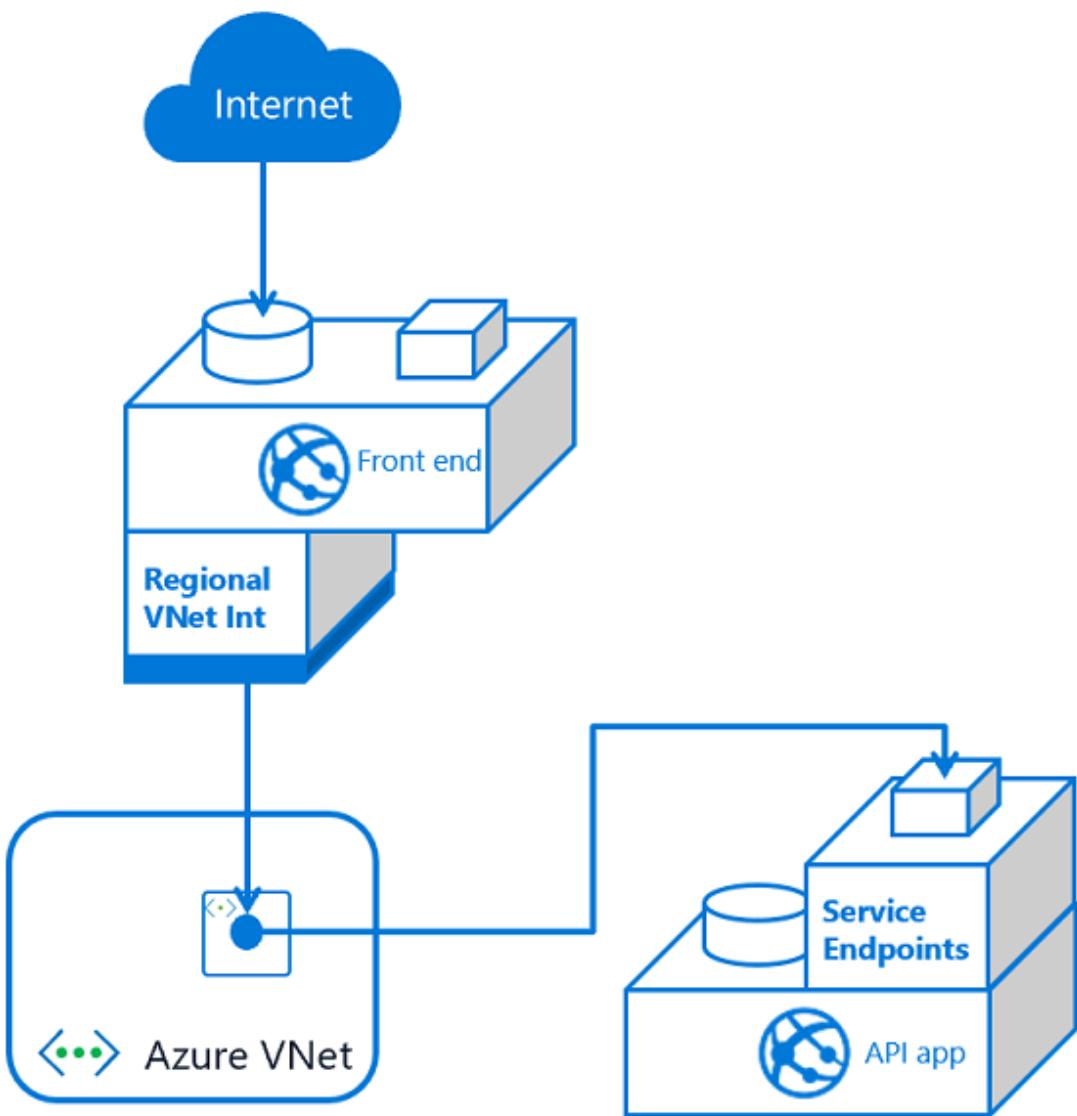
- Host both the front end and the API app in the same ILB ASE, and expose the front-end app to the internet by using an application gateway.
- Host the front end in the multi-tenant service and the back end in an ILB ASE.
- Host both the front end and the API app in the multi-tenant service.

If you're hosting both the front end and API app for a multi-tier application, you can:

- Expose your API application by using private endpoints in your virtual network:



- Use service endpoints to ensure inbound traffic to your API app comes only from the subnet used by your front-end web app:



Here are some considerations to help you decide which method to use:

- When you use service endpoints, you only need to secure traffic to your API app to the integration subnet. Service endpoints help to secure the API app, but you could still have data exfiltration from your front-end app to other apps in the app service.
- When you use private endpoints, you have two subnets at play, which adds complexity. Also, the private endpoint is a top-level resource and adds management overhead. The benefit of using private endpoints is that you don't have the possibility of data exfiltration.

Either method will work with multiple front ends. On a small scale, service endpoints are easier to use because you simply enable service endpoints for the API app on the front-end integration subnet. As you add more front-end apps, you need to adjust every API app to include service endpoints with the integration subnet. When you use private endpoints, there's more complexity, but you don't have to change anything on your API apps after you set a private endpoint.

Line-of-business applications

Line-of-business (LOB) applications are internal applications that aren't normally exposed for access from the internet. These applications are called from inside corporate networks where access can be strictly controlled. If you use an ILB ASE, it's easy to host your line-of-business applications. If you use the multi-tenant service, you can either use private endpoints or use service endpoints combined with an application gateway. There are two reasons to use an application gateway with service endpoints instead of using private endpoints:

- You need WAF protection on your LOB apps.
- You want to load balance to multiple instances of your LOB apps.

If neither of these needs apply, you're better off using private endpoints. With private endpoints available in App Service, you can expose your apps on private addresses in your virtual network. The private endpoint you place in your virtual network can be reached across ExpressRoute and VPN connections.

Configuring private endpoints will expose your apps on a private address, but you'll need to configure DNS to reach that address from on-premises. To make this configuration work, you'll need to forward the Azure DNS private zone that contains your private endpoints to your on-premises DNS servers. Azure DNS private zones don't support zone forwarding, but you can support zone forwarding by using [Azure DNS private resolver](#).

App Service ports

If you scan App Service, you'll find several ports that are exposed for inbound connections. There's no way to block or control access to these ports in the multi-tenant service. Here's the list of exposed ports:

Use	Port or ports
HTTP/HTTPS	80, 443
Management	454, 455
FTP/FTPS	21, 990, 10001-10300
Visual Studio remote debugging	4020, 4022, 4024
Web Deploy service	8172
Infrastructure use	7654, 1221

Inbound and outbound IP addresses in Azure App Service

Article • 05/13/2024

Azure App Service is a multitenant service, except for [App Service Environments](#). Apps that aren't in an App Service environment (not in the [Isolated tier](#)) share network infrastructure with other apps. As a result, the inbound and outbound IP addresses of an app can be different, and can even change in certain situations.

[App Service Environments](#) use dedicated network infrastructures, so apps running in an App Service environment get static, dedicated IP addresses both for inbound and outbound connections.

How IP addresses work in App Service

An App Service app runs in an App Service plan, and App Service plans are deployed into one of the deployment units in the Azure infrastructure (internally called a webspace). Each deployment unit is assigned a set of virtual IP addresses, which includes one public inbound IP address and a set of [outbound IP addresses](#). All App Service plans in the same deployment unit, and app instances that run in them, share the same set of virtual IP addresses. For an App Service Environment (an App Service plan in [Isolated tier](#)), the App Service plan is the deployment unit itself, so the virtual IP addresses are dedicated to it as a result.

Because you're not allowed to move an App Service plan between deployment units, the virtual IP addresses assigned to your app usually remain the same, but there are exceptions.

When inbound IP changes

Regardless of the number of scaled-out instances, each app has a single inbound IP address. The inbound IP address may change when you perform one of the following actions:

- Delete an app and recreate it in a different resource group (deployment unit may change).
- Delete the last app in a resource group *and* region combination and recreate it (deployment unit may change).

- Delete an existing IP-based TLS/SSL binding, such as during certificate renewal (see [Renew certificate](#)).

Find the inbound IP

Just run the following command in a local terminal:

```
Bash
```

```
nslookup <app-name>.azurewebsites.net
```

Get a static inbound IP

Sometimes you might want a dedicated, static IP address for your app. To get a static inbound IP address, you need to [secure a custom DNS name with an IP-based certificate binding](#). If you don't actually need TLS functionality to secure your app, you can even upload a self-signed certificate for this binding. In an IP-based TLS binding, the certificate is bound to the IP address itself, so App Service creates a static IP address to make it happen.

When outbound IPs change

Regardless of the number of scaled-out instances, each app has a set number of outbound IP addresses at any given time. Any outbound connection from the App Service app, such as to a back-end database, uses one of the outbound IP addresses as the origin IP address. The IP address to use is selected randomly at runtime, so your back-end service must open its firewall to all the outbound IP addresses for your app.

The set of outbound IP addresses for your app changes when you perform one of the following actions:

- Delete an app and recreate it in a different resource group (deployment unit may change).
- Delete the last app in a resource group *and* region combination and recreate it (deployment unit may change).
- Scale your app between the lower tiers (**Basic**, **Standard**, and **Premium**), the **PremiumV2** tier, the **PremiumV3** tier, and the **Pmv3** options within the **PremiumV3** tier (IP addresses may be added to or subtracted from the set).

You can find the set of all possible outbound IP addresses your app can use, regardless of pricing tiers, by looking for the `possibleOutboundIpAddresses` property or in the

Additional Outbound IP Addresses field in the [Properties](#) page in the Azure portal. See [Find outbound IPs](#).

The set of all possible outbound IP addresses can increase over time if App Service adds new pricing tiers or options to existing App Service deployments. For example, if App Service adds the **PremiumV3** tier to an existing App Service deployment, then the set of all possible outbound IP addresses increases. Similarly, if App Service adds new **Pmv3** options to a deployment that already supports the **PremiumV3** tier, then the set of all possible outbound IP addresses increases. Adding IP addresses to a deployment has no immediate effect since the outbound IP addresses for running applications don't change when a new pricing tier or option is added to an App Service deployment. However, if applications switch to a new pricing tier or option that wasn't previously available, then new outbound addresses are used and customers need to update downstream firewall rules and IP address restrictions.

Find outbound IPs

To find the outbound IP addresses currently used by your app in the Azure portal, select **Properties** in your app's left-hand navigation. They're listed in the **Outbound IP Addresses** field.

You can find the same information by running the following command in the [Cloud Shell](#).

Azure CLI

```
az webapp show --resource-group <group_name> --name <app_name> --query
outboundIpAddresses --output tsv
```

Azure PowerShell

```
(Get-AzWebApp -ResourceGroup <group_name> -name
<app_name>).OutboundIpAddresses
```

To find *all* possible outbound IP addresses for your app, regardless of pricing tiers, select **Properties** in your app's left-hand navigation. They're listed in the **Additional Outbound IP Addresses** field.

You can find the same information by running the following command in the [Cloud Shell](#).

Azure CLI

```
az webapp show --resource-group <group_name> --name <app_name> --query  
possibleOutboundIpAddresses --output tsv
```

Azure PowerShell

```
(Get-AzWebApp -ResourceGroup <group_name> -name  
<app_name>).PossibleOutboundIpAddresses
```

Get a static outbound IP

You can control the IP address of outbound traffic from your app by using virtual network integration together with a virtual network NAT gateway to direct traffic through a static public IP address. [Virtual network integration](#) is available on **Basic**, **Standard**, **Premium**, **PremiumV2**, and **PremiumV3** App Service plans. To learn more about this setup, see [NAT gateway integration](#).

Service tag

By using the `AppService` service tag, you can define network access for the Azure App Service service without specifying individual IP addresses. The service tag is a group of IP address prefixes that you use to minimize the complexity of creating security rules. When you use service tags, Azure automatically updates the IP addresses as they change for the service. However, the service tag isn't a security control mechanism. The service tag is merely a list of IP addresses.

The `AppService` service tag includes only the inbound IP addresses of multitenant apps. Inbound IP addresses from apps deployed in isolated (App Service Environment) and apps using [IP-based TLS bindings](#) aren't included. Further all outbound IP addresses used in both multitenant and isolated aren't included in the tag.

The tag can be used to allow outbound traffic in a Network security group (NSG) to apps. If the app is using IP-based TLS or the app is deployed in isolated mode, you must use the dedicated IP address instead.

Note

Service tag helps you define network access, but it shouldn't be considered as a replacement for proper network security measures as it doesn't provide granular control over individual IP addresses.

Next steps

- Learn how to [restrict inbound traffic](#) by source IP addresses.
- Learn more about [service tags](#).

Azure App Service access restrictions

Article • 02/13/2024

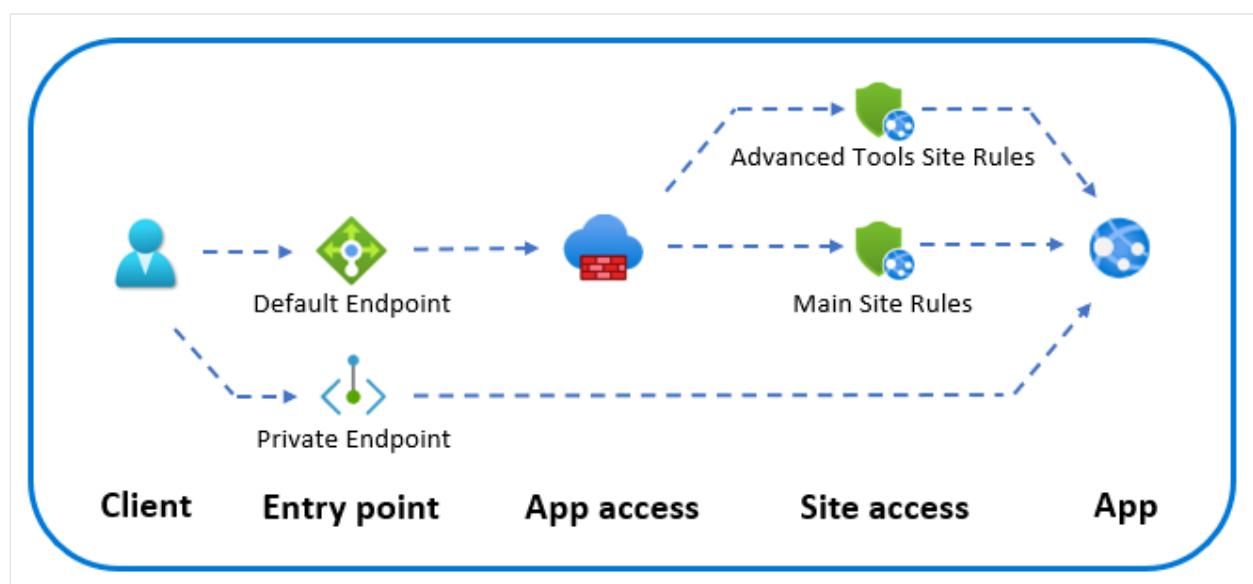
Access restrictions in App Service are equivalent to a firewall allowing you to block and filter traffic. Access restrictions apply to **inbound** access only. Most App Service pricing tiers also have the ability to add private endpoints to the app, which is another entry point to the app. Access restrictions don't apply to traffic entering through a private endpoint. For all apps hosted on App Service, the default entry point is publicly available. The only exception is apps hosted in ILB App Service Environment where the default entry point is internal to the virtual network.

How it works

When traffic reaches App Service, it first evaluates if the traffic originates from a private endpoint or is coming through the default endpoint. If the traffic is sent through a private endpoint, it sends directly to the site without any restrictions. Restrictions to private endpoints are configured using network security groups.

If you send traffic through the default endpoint (often a public endpoint), the traffic is first evaluated at the app access level. Here you can either enable or disable access. If you enable app access, the traffic is evaluated at the site access level. For any app, you have both the main site and the advanced tools site (also known as scm or kudu site).

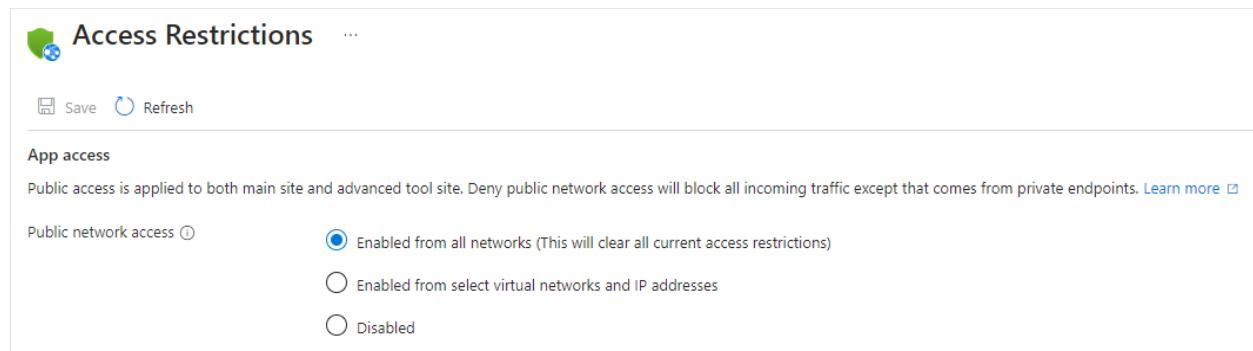
You have the option of configuring a set of access restriction rules for each site. Access restriction rules are evaluated in priority order. If some rules have the same priority, they're evaluated in the order they're listed when returned from the Azure Resource Manager API and in Azure portal before sorting. You can also specify the behavior if no rules are matched. The following sections go into details.



App access

App access allows you to configure if access is available through the default (public) endpoint. You configure this behavior to either be `Disabled` or `Enabled`. When access is enabled, you can add [Site access](#) restriction rules to control access from select virtual networks and IP addresses.

If the setting isn't set (the property is `null`), the default behavior is to enable access unless a private endpoint exists which changes the behavior to disable access. In Azure portal, when the property isn't set, the radio button is also not set and you're then using default behavior.



The screenshot shows the 'Access Restrictions' page in the Azure portal. At the top, there is a save and refresh button. Below that, the 'App access' section is shown with the following details:

Public access is applied to both main site and advanced tool site. Deny public network access will block all incoming traffic except that comes from private endpoints. [Learn more](#)

Public network access Enabled from all networks (This will clear all current access restrictions)

Enabled from select virtual networks and IP addresses

Disabled

In the Azure Resource Manager API, the property controlling app access is called `publicNetworkAccess`. For internal load balancer (ILB) App Service Environment, the default entry point for apps is always internal to the virtual network. Enabling app access (`publicNetworkAccess`) doesn't grant direct public access to the apps; instead, it allows access from the default entry point, which corresponds to the internal IP address of the App Service Environment. If you disable app access on an ILB App Service Environment, you can only access the apps through private endpoints added to the individual apps.

Site access

Site access restrictions let you filter the incoming requests. Site access restrictions allow you to build a list of allow and deny rules that are evaluated in priority order. It's similar to the network security group (NSG) feature in Azure networking.

Site access and rules

Main site Advanced tool site

You can define lists of allow/deny rules to control traffic to your site. Rules are evaluated in priority order. If no created rule is matched to the traffic, the "Unmatched rule action" will control how the traffic is handled. [Learn more](#)

Unmatched rule action Allow Deny

[+ Add](#) [Delete](#)

Filter rules Action : All [X](#)

<input type="checkbox"/>	Priority ↑	Name	Source	Action	HTTP headers
	1	Allow all	Any	<input checked="" type="checkbox"/> Allow	Not configured

Site access restriction has several types of rules that you can apply:

Unmatched rule

You can configure the behavior when no rules are matched (the default action). It's a special rule that always appears as the last rule of the rules collection. If the setting isn't configured, the unmatched rule behavior depends on configured rules. If there are no rules, the unmatched rule behavior is to allow all access, but if one or more rules exists it implicitly changes to deny all access. You can explicitly configure this behavior to either allow or deny access regardless of defined rules.

IP-based access restriction rules

The IP-based access restrictions feature helps when you want to restrict the IP addresses that can be used to reach your app. Both IPv4 and IPv6 are supported. Some use cases for this feature:

- Restrict access to your app from a set of well-defined addresses.
- Restrict access to traffic coming through an external load-balancing service or other network appliances with known egress IP addresses.

To learn how to enable this feature, see [Configuring access restrictions](#).

Note

IP-based access restriction rules only handle virtual network address ranges when your app is in an App Service Environment. If your app is in the multi-tenant service, you need to use **service endpoints** to restrict traffic to select subnets in your virtual network.

Access restriction rules based on service endpoints

Service endpoints allow you to lock down *inbound* access to your app so that the source address must come from a set of subnets that you select. This feature works together with IP access restrictions. Service endpoints aren't compatible with remote debugging. If you want to use remote debugging with your app, your client can't be in a subnet that has service endpoints enabled. The process for setting service endpoints is similar to the process for setting IP access restrictions. You can build an allow/deny list of access rules that includes public addresses and subnets in your virtual networks.

 **Note**

Access restriction rules based on service endpoints are not supported on apps that have private endpoint configured or apps that use IP-based SSL (**App-assigned address**).

To learn more about configuring service endpoints with your app, see [Azure App Service access restrictions](#).

Any service endpoint source

For testing or in specific scenarios, you can allow traffic from any service endpoint enabled subnet. You can do that by defining an IP-based rule with the text "AnyVnets" instead of an IP range. You can't create these rules in the portal, but you can modify an existing IP-based rule and replace the IP address with the "AnyVnets" string.

Access restriction rules based on service tags

[Azure service tags](#) are well defined sets of IP addresses for Azure services. Service tags group the IP ranges used in various Azure services and is often also further scoped to specific regions. This type of rule allows you to filter *inbound* traffic from specific Azure services.

For a full list of tags and more information, visit the service tag link.

To learn how to enable this feature, see [Configuring access restrictions](#).

Multi-source rules

Multi-source rules allow you to combine up to eight IP ranges or eight Service Tags in a single rule. You might use multi-source rules if you have more than 512 IP ranges. You can also use multi-source rules if you want to create logical rules where multiple IP ranges are combined with a single http header filter.

Multi-source rules are defined the same way you define single-source rules, but with each range separated with comma.

You can't create these rules in the portal, but you can modify an existing service tag or IP-based rule and add more sources to the rule.

Http header filtering for site access restriction rules

For any rule, regardless of type, you can add http header filtering. Http header filters allow you to further inspect the incoming request and filter based on specific http header values. Each header can have up to eight values per rule. The following lists the supported http headers:

- **X-Forwarded-For.** [Standard header ↗](#) for identifying the originating IP address of a client connecting through a proxy server. Accepts valid IP addresses.
- **X-Forwarded-Host.** [Standard header ↗](#) for identifying the original host requested by the client. Accepts any string up to 64 characters in length.
- **X-Azure-FDID.** [Custom header](#) for identifying the reverse proxy instance. Azure Front Door sends a guid identifying the instance, but it can also be used for non-Microsoft proxies to identify the specific instance. Accepts any string up to 64 characters in length.
- **X-FD-HealthProbe.** [Custom header](#) for identifying the health probe of the reverse proxy. Azure Front Door sends "1" to uniquely identify a health probe request. The header can also be used for non-Microsoft proxies to identify health probes. Accepts any string up to 64 characters in length.

Some use cases for http header filtering are:

- Restrict access to traffic from proxy servers forwarding the host name
- Restrict access to a specific Azure Front Door instance with a service tag rule and X-Azure-FDID header restriction

Diagnostic logging

App Service can [send various logging categories to Azure Monitor](#). One of those categories is called `IPSecurity Audit logs` and represent the activities in access restrictions. All requests that match a rule (except the unmatched rule), both allow and deny, is logged and can be used to validate configuration of access restrictions. The logging capability is also a powerful tool when troubleshooting rules configuration.

Advanced use cases

Combining the above features allow you to solve some specific use cases that are described in the following sections.

Block a single IP address

If you want to deny/block one or more specific IP addresses, you can add the IP addresses as deny rules and configure the unmatched rule to allow all unmatched traffic.

Restrict access to the advanced tools site

The advanced tools site, which is also known as scm or kudu, has an individual rules collection that you can configure. You can also configure the unmatched rule for this site. A setting allows you to use the rules configured for the main site. You can't selectively allow access to certain advanced tool site features. For example, you can't selectively allow access only to the WebJobs management console in the advanced tools site.

Deploy through a private endpoint

You might have a site that is publicly accessible, but your deployment system is in a virtual network. You can keep the deployment traffic private by adding a private endpoint. You then need to ensure that public app access is enabled. Finally you need to set the unmatched rule for the advanced tools site to deny, which blocks all public traffic to that endpoint.

Allow external partner access to private endpoint protected site

In this scenario, you're accessing your site through a private endpoint and are deploying through a private endpoint. You can temporarily invite an external partner to test the site. You can do that by enabling public app access. Add a rule (IP-based) to identify the client of the partner. Configure unmatched rules action to deny for both main and advanced tools site.

Restrict access to a specific Azure Front Door instance

Traffic from Azure Front Door to your application originates from a well known set of IP ranges defined in the `AzureFrontDoor.Backend` service tag. Using a service tag restriction rule, you can restrict traffic to only originate from Azure Front Door. To ensure traffic only originates from your specific instance, you need to further filter the incoming

requests based on the unique http header that Azure Front Door sends called X-Azure-FDID. You can find the Front Door ID in the portal.

Next steps

 Note

Access restriction rules that block public access to your site can also block services such as log streaming. If you require these, you will need to allow your App Service's IP address in your restrictions.

[How to restrict access](#)

[Private endpoints for App Service apps](#)

Set up Azure App Service access restrictions

Article • 08/21/2024

ⓘ Note

Starting June 1, 2024, all newly created App Service apps will have the option to generate a unique default hostname using the naming convention <app-name>-<random-hash>. <region>.azurewebsites.net. Existing app names will remain unchanged.

Example: myapp-ds27dh7271aah175.westus-01.azurewebsites.net

For further details, refer to [Unique Default Hostname for App Service Resource ↗](#).

By setting up access restrictions, you can define a priority-ordered allow/deny list that controls network access to your app. The list can include IP addresses or Azure Virtual Network subnets. When there are one or more entries, an implicit *deny all* exists at the end of the list. To learn more about access restrictions, go to the [access restrictions overview](#).

The access restriction capability works with all Azure App Service-hosted workloads. The workloads can include web apps, API apps, Linux apps, Linux custom containers and Functions.

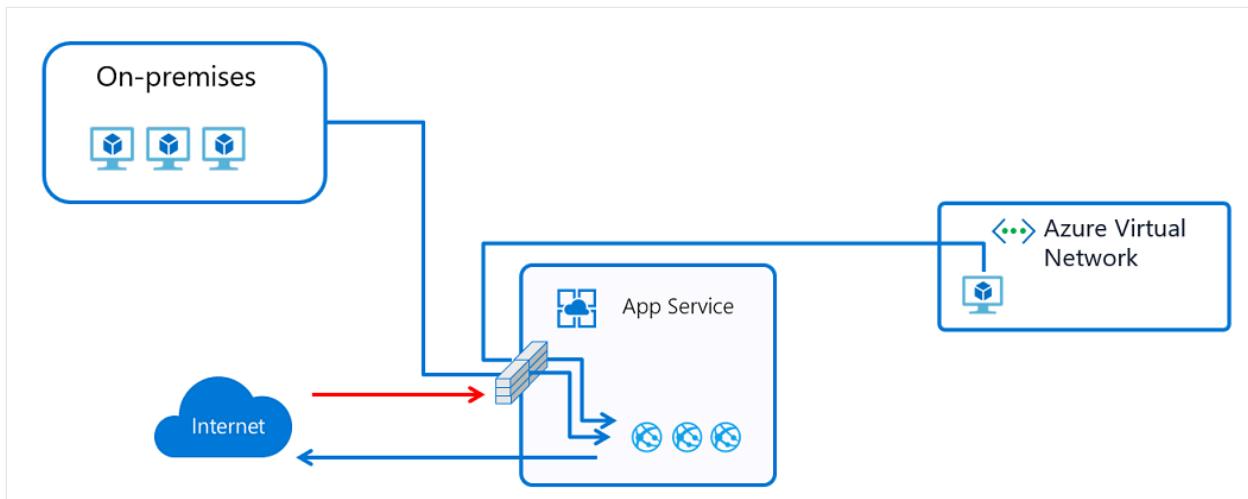
When a request is made to your app, the FROM address is evaluated against the rules in your access restriction list. If the FROM address is in a subnet configured with service endpoints to Microsoft.Web, the source subnet is compared against the virtual network rules in your access restriction list. If the address isn't allowed access based on the rules in the list, the service replies with an [HTTP 403 ↗](#) status code.

The access restriction capability is implemented in the App Service front-end roles, which are upstream of the worker hosts where your code runs. Therefore, access restrictions are effectively network access-control lists (ACLs).

The ability to restrict access to your web app from an Azure virtual network uses [service endpoints](#). With service endpoints, you can restrict access to a multitenant service from selected subnets. It doesn't work to restrict traffic to apps that are hosted in an App Service Environment. If you're in an App Service Environment, you can control access to your app by applying IP address rules.

⚠ Note

The service endpoints must be enabled both on the networking side and for the Azure service that they're being enabled with. For a list of Azure services that support service endpoints, see [Virtual Network service endpoints](#).



Manage access restriction rules in the portal

To add an access restriction rule to your app, do the following steps:

1. Sign in to the Azure portal.
2. Select the app that you want to add access restrictions to.
3. On the left menu, select **Networking**.
4. On the **Networking** page, under **Inbound traffic configuration**, select the **Public network access** setting.

The screenshot shows the Azure portal interface for a 'network-integration-webapp' under the 'Networking' section. On the left, there's a navigation menu with links like Overview, Activity log, Access control (IAM), Tags, Diagnose and solve problems, Microsoft Defender for Cloud, Events (preview), Log stream, Deployment slots, Deployment Center, Environment variables, and Settings. The main content area displays 'Inbound traffic configuration' with details such as Public network access (Enabled with no access restrictions), App assigned address (Not configured), Private endpoints (0 private endpoints), and Inbound addresses (20.105.232.6). It also shows 'Optional inbound services' with Azure Front Door and a 'View details' link.

- On the Access Restrictions page, review the list of access restriction rules that are defined for your app.

The screenshot shows the 'Access Restrictions' page for the same application. It includes sections for 'App access' (Public network access set to 'Enabled from select virtual networks and IP addresses'), 'Site access and rules' (Main site selected, Unmatched rule action set to 'Deny'), and a table of access rules. The table has columns for Priority, Name, Source, Action, and HTTP headers. The rules listed are: Deny example (Priority 80, Source 122.133.144.32/28, Action Deny, Not configured); IP example rule (Priority 100, Source 122.133.144.0/24, Action Allow, Not configured); and Deny all (Priority 2147483647, Source Any, Action Deny, Not configured).

Priority ↑	Name	Source	Action	HTTP headers
80	Deny example	122.133.144.32/28	✖ Deny	Not configured
100	IP example rule	122.133.144.0/24	✓ Allow	Not configured
2147483647	Deny all	Any	✖ Deny	Not configured

The list displays all the current restrictions that are applied to the app. If you have a virtual network restriction on your app, the table shows whether the service endpoints are enabled for Microsoft.Web. If no restrictions are defined on your app and your unmatched rule isn't set to Deny, the app is accessible from anywhere.

Permissions

The following Role-based access control permissions on the subnet or at a higher level are required to configure access restrictions through Azure portal, CLI or when setting the site config properties directly:

[+] Expand table

Action	Description
Microsoft.Web/sites/config/read	Get Web App configuration settings
Microsoft.Web/sites/config/write	Update Web App's configuration settings
Microsoft.Network/virtualNetworks/subnets/joinViaServiceEndpoint/action*	Joins resource such as storage account or SQL database to a subnet
Microsoft.Web/sites/write**	Update Web App settings

*only required when adding a virtual network (service endpoint) rule.

**only required if you're updating access restrictions through Azure portal.

If you're adding a service endpoint-based rule and the virtual network is in a different subscription than the app, you must ensure that the subscription with the virtual network is registered for the `Microsoft.Web` resource provider. You can explicitly register the provider [by following this documentation](#), but also automatically registered when creating the first web app in a subscription.

Add an access restriction rule

To add an access restriction rule to your app, on the **Access Restrictions** page, select **Add**. The rule is only effective after saving.

Rules are enforced in priority order, starting from the lowest number in the **Priority** column. If you don't configure unmatched rule, an implicit *deny all* is in effect after you add even a single rule.

On the **Add Access Restriction** pane, when you create a rule, do the following:

1. Under Action, select either Allow or Deny.

The screenshot shows the 'Add Access Restriction' dialog box. It has the following fields:

- Name**: A text input field containing "Enter name for the IpAddress rule" with a green checkmark icon to its right.
- Action**: A button group with two options: "Allow" (highlighted in blue) and "Deny".
- Priority ***: A text input field containing "Ex. 300".
- Description**: An empty text input field with a green checkmark icon to its right.
- Type**: A dropdown menu set to "IPv4".
- IP Address Block ***: A text input field containing "Enter an IPv4 CIDR. Ex: 208.130.0.0/16".
- Add rule**: A blue button at the bottom left of the form.

2. Optionally, enter a name and description of the rule.

3. In the **Priority** box, enter a priority value.

4. In the **Type** drop-down list, select the type of rule. The different types of rules are described in the following sections.

5. Select **Add rule** after typing in the rule specific input to add the rule to the list.

Finally select **Save** back in the **Access Restrictions** page.

Note

- There is a limit of 512 access restriction rules. If you require more than 512 access restriction rules, we suggest that you consider installing a standalone security product, such as Azure Front Door, Azure App Gateway, or an alternative WAF.

Set an IP address-based rule

Follow the procedure as outlined in the preceding section, but with the following addition:

- For step 4, in the **Type** drop-down list, select **IPv4 or IPv6**.

Specify the **IP Address Block** in Classless Inter-Domain Routing (CIDR) notation for both the IPv4 and IPv6 addresses. To specify an address, you can use something like **1.2.3.4/32**, where the first four octets represent your IP address and **/32** is the mask. The IPv4 CIDR notation for all addresses is **0.0.0.0/0**. To learn more about CIDR notation, see [Classless Inter-Domain Routing](#).

Note

IP-based access restriction rules only handle virtual network address ranges when your app is in an App Service Environment. If your app is in the multi-tenant service, you need to use **service endpoints** to restrict traffic to select subnets in your virtual network.

Set a service endpoint-based rule

- For step 4, in the **Type** drop-down list, select **Virtual Network**.

Add Access Restriction

X

Name ⓘ

Enter name for the ipAddress rule



Action

Allow

Deny

Priority *

Ex. 300

Description



Type

Virtual Network



Subscription *

Purple Demo Subscription



Virtual Network *

networking-demos-vnet



Subnet *

nat-gw-subnet



Selected subnet 'networking-demos-vnet/nat-gw-subnet' does not have service endpoint enabled for Microsoft.Web. Enabling access may take up to 15 minutes to complete.



Ignore missing Microsoft.Web service endpoints

Add rule

Specify the **Subscription**, **Virtual Network**, and **Subnet** drop-down lists, matching what you want to restrict access to.

By using service endpoints, you can restrict access to selected Azure virtual network subnets. If service endpoints aren't already enabled with **Microsoft.Web** for the subnet that you selected, they're automatically enabled unless you select the **Ignore missing Microsoft.Web service endpoints** check box. The scenario where you might want to

enable service endpoints on the app but not the subnet depends mainly on whether you have the permissions to enable them on the subnet.

If you need someone else to enable service endpoints on the subnet, select the **Ignore missing Microsoft.Web service endpoints** check box. Your app is configured for service endpoints in anticipation of having them enabled later on the subnet.

You can't use service endpoints to restrict access to apps that run in an App Service Environment. When your app is in an App Service Environment, you can control access to it by applying IP access rules.

With service endpoints, you can configure your app with application gateways or other web application firewall (WAF) devices. You can also configure multi-tier applications with secure back ends. For more information, see [Networking features and App Service](#) and [Application Gateway integration with service endpoints](#).

 **Note**

- Service endpoints aren't supported for web apps that use IP-based TLS/SSL bindings with a virtual IP (VIP).

Set a service tag-based rule

- For step 4, in the **Type** drop-down list, select **Service Tag**.

Add Access Restriction X

General settings

Name (i)

 ✓

Action

Allow Deny

Priority *

 ✓

Description

 ✓

Source settings

Type

 ▼

Service Tag *

 ^

ActionGroup

ApplicationInsightsAvailability

AzureCloud

AzureCognitiveSearch

AzureEventGrid

AzureFrontDoor.Backend

AzureMachineLearning

AzureTrafficManager

LogicApps

All publicly available service tags are supported in access restriction rules. Each service tag represents a list of IP ranges from Azure services. A list of these services and links to the specific ranges can be found in the [service tag documentation](#). Use Azure Resource Manager templates or scripting to configure more advanced rules like regional scoped rules.

(i) Note

When creating service tag-based rules through Azure portal or Azure CLI you will need read access at the subscription level to get the full list of service tags for selection/validation. In addition, the `Microsoft.Network` resource provider needs to be registered on the subscription.

Edit a rule

1. To begin editing an existing access restriction rule, on the **Access Restrictions** page, select the rule you want to edit.
2. On the **Edit Access Restriction** pane, make your changes, and then select **Update rule**.
3. Select **Save** to save the changes.

Edit rule ×

General settings

Name

Priority *

Action
 Allow Deny

Description

Source settings

IP Address Block *

Update rule

! **Note**

When you edit a rule, you can't switch between rule types.

Delete a rule

1. To delete a rule, on the **Access Restrictions** page, check the rule or rules you want to delete, and then select **Delete**.
2. Select **Save** to save the changes.

The screenshot shows the 'Access Restrictions' page in the Azure portal. At the top, there are 'Save' and 'Refresh' buttons, with 'Save' highlighted with a red box. Below this is a section titled 'App access' with a note about public access. Under 'Public network access', the 'Enabled from select virtual networks and IP addresses' option is selected. In the 'Site access and rules' section, the 'Main site' tab is active. It shows an 'Unmatched rule action' set to 'Deny'. Below this is a table of rules:

Priority ↑	Name	Source	Action	HTTP headers
80	Deny example	122.133.144.32/28	Deny	Not configured
100	IP example rule	122.133.144.0/24	Allow	Not configured
2147483647	Deny all	Any	Deny	Not configured

At the bottom of the table, there is a 'Delete' button highlighted with a red box.

Access restriction advanced scenarios

The following sections describe some advanced scenarios using access restrictions.

Filter by http header

As part of any rule, you can add http header filters. The following http header names are supported:

- X-Forwarded-For
- X-Forwarded-Host
- X-Azure-FDID
- X-FD-HealthProbe

For each header name, you can add up to eight values separated by comma. The http header filters are evaluated after the rule itself and both conditions must be true for the

rule to apply.

Multi-source rules

Multi-source rules allow you to combine up to eight IP ranges or eight Service Tags in a single rule. You use multi-source rules if you have more than 512 IP ranges or you want to create logical rules. Logical rules could be where multiple IP ranges are combined with a single http header filter.

Multi-source rules are defined the same way you define single-source rules, but with each range separated with comma.

PowerShell example:

Azure PowerShell

```
Add-AzWebAppAccessRestrictionRule -ResourceGroupName "ResourceGroup" -  
WebAppName "AppName"  
-Name "Multi-source rule" -IpAddress  
"192.168.1.0/24,192.168.10.0/24,192.168.100.0/24"  
-Priority 100 -Action Allow
```

Block a single IP address

For a scenario where you want to explicitly block a single IP address or a block of IP addresses, but allow access to everything else, add a **Deny** rule for the specific IP address and configure the unmatched rule action to **Allow**.

The screenshot shows the 'Access Restrictions' section of the Azure App Service configuration. It includes settings for 'App access' (Public network access: Enabled from select virtual networks and IP addresses), 'Site access and rules' (Main site selected), and a table of rules:

Priority ↑	Name	Source	Action	HTTP headers
80	Block Me	122.133.144.32/32	Deny	Not configured
2147483647	Allow all	Any	Allow	Not configured

Restrict access to an SCM site

In addition to being able to control access to your app, you can restrict access to the SCM (Advanced tool) site used by your app. The SCM site is both the web deploy endpoint and the Kudu console. You can assign access restrictions to the SCM site from the app separately or use the same set of restrictions for both the app and the SCM site. When you select the **Use main site rules** check box, the rules list is hidden, and it uses the rules from the main site. If you clear the check box, your SCM site settings appear again.

The screenshot shows the 'Access Restrictions' page in the Azure portal. At the top, there are 'Save' and 'Refresh' buttons. Below that, the 'App access' section is shown, with a note that public access is applied to both main site and advanced tool site. It says that denying public network access will block all incoming traffic except from private endpoints. There are three options for public network access: 'Enabled from all networks' (radio button), 'Enabled from select virtual networks and IP addresses' (selected radio button), and 'Disabled'. The 'Site access and rules' section follows, with tabs for 'Main site' and 'Advanced tool site' (selected). A note says you can define lists of allow/deny rules to control traffic. The 'Unmatched rule action' is set to 'Deny' (radio button). The 'Use main site rules' checkbox is checked. Below this is a table of rules:

Priority ↑	Name	Source	Action	HTTP headers
200	Deployment server	200.100.50.0/32	Allow	Not configured
2147483647	Deny all	Any	Deny	Not configured

Restrict access to a specific Azure Front Door instance

Traffic from Azure Front Door to your application originates from a well known set of IP ranges defined in the `AzureFrontDoor.Backend` service tag. Using a service tag restriction rule, you can restrict traffic to only originate from Azure Front Door. To ensure traffic only originates from your specific instance, you need to further filter the incoming requests based on the unique http header that Azure Front Door sends.

Add Access Restriction

X

General settings

Name ⓘ

MyAzureFrontDoorRule



Action

Allow Deny

Priority *

100



Description



Source settings

Type

Service Tag



Service Tag *

AzureFrontDoor.Backend



HTTP headers filter settings

X-Forwarded-Host ⓘ

Ex. exampleOne.com, exampleTwo.com

X-Forwarded-For ⓘ

Enter IPv4 or IPv6 CIDR addresses.

X-Azure-FDID ⓘ

xxxxxxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxx



X-FD-HealthProbe ⓘ

Ex. 1

PowerShell example:

Azure PowerShell

```
$afd = Get-AzFrontDoor -Name "MyFrontDoorInstanceName"  
Add-AzWebAppAccessRestrictionRule -ResourceGroupName "ResourceGroup" -  
WebAppName "AppName"  
-Name "Front Door example rule" -Priority 100 -Action Allow -ServiceTag
```

```
AzureFrontDoor.Backend  
-HTTPHeader @{'x-azure-fdid' = $afd.FrontDoorId}
```

Manage access restriction programmatically

You can manage access restriction programmatically, below you can find examples of how to add rules to access restrictions and how to change *Unmatched rule action* for both *Main site* and *Advanced tool site*.

Add access restrictions rules for main site

You can add access restrictions rules for *Main site* programmatically by choosing one of the following options:

Azure CLI

You can run the following command in the [Cloud Shell](#). For more information about `az webapp config access-restriction` command, visit [this page](#).

Azure CLI

```
az webapp config access-restriction add --resource-group ResourceGroup -  
-name AppName \  
--rule-name 'IP example rule' --action Allow --ip-address  
122.133.144.0/24 --priority 100  
  
az webapp config access-restriction add --resource-group ResourceGroup -  
-name AppName \  
--rule-name "Azure Front Door example" --action Allow --priority 200 -  
-service-tag AzureFrontDoor.Backend \  
--http-header x-azure-fdid=xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx
```

Add access restrictions rules for advanced tool site

You can add access restrictions rules for *Advanced tool site* programmatically by choosing one of the following options:

Azure CLI

You can run the following command in the [Cloud Shell](#). For more information about `az webapp config access-restriction` command, visit [this page](#).

Azure CLI

```
az webapp config access-restriction add --resource-group ResourceGroup --name AppName \
--rule-name 'IP example rule' --action Allow --ip-address 122.133.144.0/24 --priority 100 --scm-site true
```

Change unmatched rule action for main site

You can change *Unmatched rule action* for *Main site* programmatically by choosing one of the following options:

Azure CLI

You can run the following command in the [Cloud Shell](#). For more information about `az resource` command, visit [this page](#). Accepted values for `ipSecurityRestrictionsDefaultAction` are `Allow` or `Deny`.

Azure CLI

```
az resource update --resource-group ResourceGroup --name AppName --resource-type "Microsoft.Web/sites" \
--set properties.siteConfig.ipSecurityRestrictionsDefaultAction=Allow
```

Change unmatched rule action for advanced tool site

You can change *Unmatched rule action* for *Advanced tool site* programmatically by choosing one of the following options:

Azure CLI

You can run the following command in the [Cloud Shell](#). For more information about `az resource` command, visit [this page](#). Accepted values for `scmIpSecurityRestrictionsDefaultAction` are `Allow` or `Deny`.

Azure CLI

```
az resource update --resource-group ResourceGroup --name AppName --resource-type "Microsoft.Web/sites" \
```

```
--set  
properties.siteConfig.scmIpSecurityRestrictionsDefaultAction=Allow
```

Set up Azure Functions access restrictions

Access restrictions are also available for function apps with the same functionality as App Service plans. When you enable access restrictions, you also disable the Azure portal code editor for any disallowed IPs.

Next steps

[Access restrictions for Azure Functions](#)

[Application Gateway integration with service endpoints](#)

[Advanced access restriction scenarios in Azure App Service - blog post ↗](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

Integrate your app with an Azure virtual network

Article • 04/05/2024

This article describes the Azure App Service virtual network integration feature and how to set it up with apps in [App Service](#). With [Azure virtual networks](#), you can place many of your Azure resources in a non-internet-routable network. The App Service virtual network integration feature enables your apps to access resources in or through a virtual network.

ⓘ Note

Information about Gateway-required virtual network integration has [moved to a new location](#).

App Service has two variations:

- The dedicated compute pricing tiers, which include the Basic, Standard, Premium, Premium v2, and Premium v3.
- The App Service Environment, which deploys directly into your virtual network with dedicated supporting infrastructure and is using the Isolated and Isolated v2 pricing tiers.

The virtual network integration feature is used in Azure App Service dedicated compute pricing tiers. If your app is in an [App Service Environment](#), it already integrates with a virtual network and doesn't require you to configure virtual network integration feature to reach resources in the same virtual network. For more information on all the networking features, see [App Service networking features](#).

Virtual network integration gives your app access to resources in your virtual network, but it doesn't grant inbound private access to your app from the virtual network. Private site access refers to making an app accessible only from a private network, such as from within an Azure virtual network. Virtual network integration is used only to make outbound calls from your app into your virtual network. Refer to [private endpoint](#) for inbound private access.

The virtual network integration feature:

- Requires a [supported Basic or Standard](#), Premium, Premium v2, Premium v3, or Elastic Premium App Service pricing tier.

- Supports TCP and UDP.
- Works with App Service apps, function apps and Logic apps.

There are some things that virtual network integration doesn't support, like:

- Mounting a drive.
- Windows Server Active Directory domain join.
- NetBIOS.

Virtual network integration supports connecting to a virtual network in the same region. Using virtual network integration enables your app to access:

- Resources in the virtual network you're integrated with.
- Resources in virtual networks peered to the virtual network your app is integrated with including global peering connections.
- Resources across Azure ExpressRoute connections.
- Service endpoint-secured services.
- Private endpoint-enabled services.

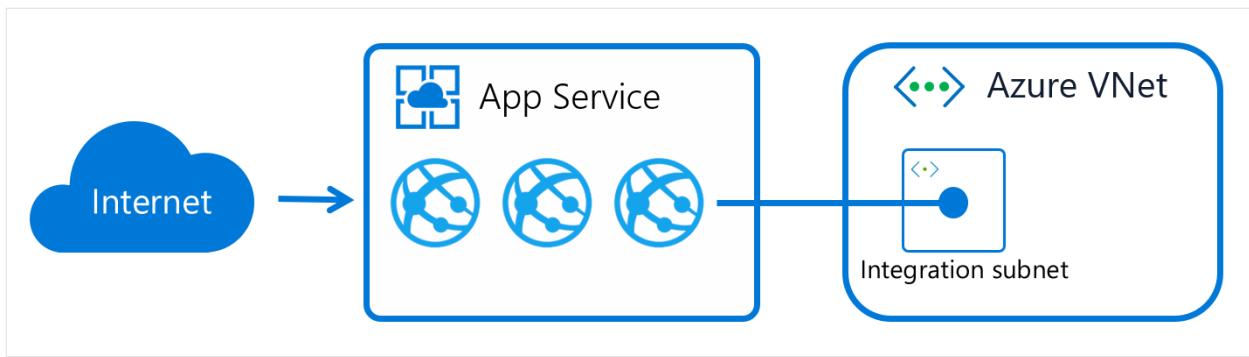
When you use virtual network integration, you can use the following Azure networking features:

- **Network security groups (NSGs)**: You can block outbound traffic with an NSG that you use on your integration subnet. The inbound rules don't apply because you can't use virtual network integration to provide inbound access to your app.
- **Route tables (UDRs)**: You can place a route table on the integration subnet to send outbound traffic where you want.
- **NAT gateway**: You can use [NAT gateway](#) to get a dedicated outbound IP and mitigate SNAT port exhaustion.

Learn [how to enable virtual network integration](#).

How virtual network integration works

Apps in App Service are hosted on worker roles. Virtual network integration works by mounting virtual interfaces to the worker roles with addresses in the delegated subnet. The virtual interfaces used aren't resources customers have direct access to. Because the from address is in your virtual network, it can access most things in or through your virtual network like a VM in your virtual network would.



When virtual network integration is enabled, your app makes outbound calls through your virtual network. The outbound addresses that are listed in the app properties portal are the addresses still used by your app. However, if your outbound call is to a virtual machine or private endpoint in the integration virtual network or peered virtual network, the outbound address is an address from the integration subnet. The private IP assigned to an instance is exposed via the environment variable, WEBSITE_PRIVATE_IP.

When all traffic routing is enabled, all outbound traffic is sent into your virtual network. If all traffic routing isn't enabled, only private traffic (RFC1918) and service endpoints configured on the integration subnet is sent into the virtual network. Outbound traffic to the internet is routed directly from the app.

The virtual network integration feature supports two virtual interfaces per worker. Two virtual interfaces per worker mean two virtual network integrations per App Service plan. In other words, an App Service plan can have virtual network integrations with up to two subnets/virtual networks. The apps in the same App Service plan can only use one of the virtual network integrations to a specific subnet, meaning an app can only have a single virtual network integration at a given time.

Subnet requirements

Virtual network integration depends on a dedicated subnet. When you create a subnet, the Azure subnet consumes five IPs from the start. One address is used from the integration subnet for each App Service plan instance. If you scale your app to four instances, then four addresses are used.

When you scale up/down in instance size, the amount of IP addresses used by the App Service plan is temporarily doubled while the scale operation completes. The new instances need to be fully operational before the existing instances are deprovisioned. The scale operation affects the real, available supported instances for a given subnet size. Platform upgrades need free IP addresses to ensure upgrades can happen without interruptions to outbound traffic. Finally, after scale up, down, or in operations complete, there might be a short period of time before IP addresses are released. In rare cases, this operation can be up to 12 hours.

Because subnet size can't be changed after assignment, use a subnet that's large enough to accommodate whatever scale your app might reach. You should also reserve IP addresses for platform upgrades. To avoid any issues with subnet capacity, use a /26 with 64 addresses. When you're creating subnets in Azure portal as part of integrating with the virtual network, a minimum size of /27 is required. If the subnet already exists before integrating through the portal, you can use a /28 subnet.

With multi plan subnet join (MPSJ) you can join multiple App Service plans in to the same subnet. All App Service plans must be in the same subscription but the virtual network/subnet can be in a different subscription. Each instance from each App Service plan requires an IP address from the subnet and to use MPSJ a minimum size of /26 subnet is required. If you plan to join many and/or large scale plans, you should plan for larger subnet ranges.

Note

Multi plan subnet join is currently in public preview. During preview the following known limitations should be observed:

- The minimum requirement for subnet size of /26 is currently not enforced, but will be enforced at GA.
- There is currently no validation if the subnet has available IPs, so you might be able to join N+1 plan, but the instances will not get an IP. You can view available IPs in the Virtual network integration page in Azure portal in apps that are already connected to the subnet.

Windows Containers specific limits

Windows Containers uses an additional IP address per app for each App Service plan instance, and you need to size the subnet accordingly. If you have for example 10 Windows Container App Service plan instances with 4 apps running, you will need 50 IP addresses and additional addresses to support horizontal (in/out) scale.

Sample calculation:

For each App Service plan instance, you need: 4 Windows Container apps = 4 IP addresses
1 IP address per App Service plan instance $4 + 1 = 5$ IP addresses

For 10 instances: $5 \times 10 = 50$ IP addresses per App Service plan

Since you have 1 App Service plan, $1 \times 50 = 50$ IP addresses.

You are in addition limited by the number of cores available in the worker SKU used. Each core adds three "networking units". The worker itself uses one unit and each virtual network connection uses one unit. The remaining units can be used for apps.

Sample calculation:

App Service plan instance with 4 apps running and using virtual network integration. The Apps are connected to two different subnets (virtual network connections). This will require 7 networking units (1 worker + 2 connections + 4 apps). The minimum size for running this configuration would be I2v2 (4 cores x 3 units = 12 units).

With I1v2 you can run a maximum of 4 apps using the same (1) connection or 3 apps using 2 connections.

Permissions

You must have at least the following Role-based access control permissions on the subnet or at a higher level to configure virtual network integration through Azure portal, CLI or when setting the `virtualNetworkSubnetId` site property directly:

[+] Expand table

Action	Description
Microsoft.Network/virtualNetworks/read	Read the virtual network definition
Microsoft.Network/virtualNetworks/subnets/read	Read a virtual network subnet definition
Microsoft.Network/virtualNetworks/subnets/join/action	Joins a virtual network

If the virtual network is in a different subscription than the app, you must ensure that the subscription with the virtual network is registered for the `Microsoft.Web` resource provider. You can explicitly register the provider [by following this documentation](#), but it also automatically registers when creating the first web app in a subscription.

Routes

You can control what traffic goes through the virtual network integration. There are three types of routing to consider when you configure virtual network integration. [Application routing](#) defines what traffic is routed from your app and into the virtual network. [Configuration routing](#) affects operations that happen before or during startup of your app. Examples are container image pull and [app settings with Key Vault](#)

reference. Network routing is the ability to handle how both app and configuration traffic are routed from your virtual network and out.

Through application routing or configuration routing options, you can configure what traffic is sent through the virtual network integration. Traffic is only subject to [network routing](#) if sent through the virtual network integration.

Application routing

Application routing applies to traffic that is sent from your app after it starts. See [configuration routing](#) for traffic during startup. When you configure application routing, you can either route all traffic or only private traffic (also known as [RFC1918](#) traffic) into your virtual network. You configure this behavior through the outbound internet traffic setting. If outbound internet traffic routing is disabled, your app only routes private traffic into your virtual network. If you want to route all your outbound app traffic into your virtual network, make sure that outbound internet traffic is enabled.

- Only traffic configured in application or configuration routing is subject to the NSGs and UDRs that are applied to your integration subnet.
- When outbound internet traffic routing is enabled, the source address for your outbound traffic from your app is still one of the IP addresses that are listed in your app properties. If you route your traffic through a firewall or a NAT gateway, the source IP address originates from this service.

Learn [how to configure application routing](#).

Note

Outbound SMTP connectivity (port 25) is supported for App Service when the SMTP traffic is routed through the virtual network integration. The supportability is determined by a setting on the subscription where the virtual network is deployed. For virtual networks/subnets created before 1. August 2022 you need to initiate a temporary configuration change to the virtual network/subnet for the setting to be synchronized from the subscription. An example could be to add a temporary subnet, associate/dissociate an NSG temporarily or configure a service endpoint temporarily. For more information, see [Troubleshoot outbound SMTP connectivity problems in Azure](#).

Configuration routing

When you're using virtual network integration, you can configure how parts of the configuration traffic are managed. By default, configuration traffic goes directly over the public route, but for the mentioned individual components, you can actively configure it to be routed through the virtual network integration.

Content share

Bringing your own storage for content is often used in Functions where [content share](#) is configured as part of the Functions app.

To route content share traffic through the virtual network integration, you must ensure that the routing setting is configured. Learn [how to configure content share routing](#).

In addition to configuring the routing, you must also ensure that any firewall or Network Security Group configured on traffic from the subnet allow traffic to port 443 and 445.

Container image pull

When using custom containers, you can pull the container over the virtual network integration. To route the container pull traffic through the virtual network integration, you must ensure that the routing setting is configured. Learn [how to configure image pull routing](#).

Backup/restore

App Service has built-in backup/restore, but if you want to back up to your own storage account, you can use the custom backup/restore feature. If you want to route the traffic to the storage account through the virtual network integration, you must configure the route setting. Database backup isn't supported over the virtual network integration.

App settings using Key Vault references

App settings using Key Vault references attempt to get secrets over the public route. If the Key Vault is blocking public traffic and the app is using virtual network integration, an attempt is made to get the secrets through the virtual network integration.

Note

- Configure SSL/TLS certificates from private Key Vaults is currently not supported.

- App Service Logs to private storage accounts is currently not supported. We recommend using Diagnostics Logging and allowing Trusted Services for the storage account.

Routing app settings

App Service has existing app settings to configure application and configuration routing. Site properties override the app settings if both exist. Site properties have the advantage of being auditable with Azure Policy and validated at the time of configuration. We recommend you to use site properties.

You can still use the existing `WEBSITE_VNET_ROUTE_ALL` app setting to configure application routing.

App settings also exist for some configuration routing options. These app settings are named `WEBSITE_CONTENTOVERVNET` and `WEBSITE_PULL_IMAGE_OVER_VNET`.

Network routing

You can use route tables to route outbound traffic from your app without restriction. Common destinations can include firewall devices or gateways. You can also use a [network security group](#) (NSG) to block outbound traffic to resources in your virtual network or the internet. An NSG that you apply to your integration subnet is in effect regardless of any route tables applied to your integration subnet.

Route tables and network security groups only apply to traffic routed through the virtual network integration. See [application routing](#) and [configuration routing](#) for details.

Routes don't apply to replies from inbound app requests and inbound rules in an NSG don't apply to your app. Virtual network integration affects only outbound traffic from your app. To control inbound traffic to your app, use the [access restrictions](#) feature or [private endpoints](#).

When configuring network security groups or route tables that applies to outbound traffic, you must make sure you consider your application dependencies. Application dependencies include endpoints that your app needs during runtime. Besides APIs and services the app is calling, these endpoints could also be derived endpoints like certificate revocation list (CRL) check endpoints and identity/authentication endpoint, for example Microsoft Entra ID. If you're using [continuous deployment in App Service](#), you might also need to allow endpoints depending on type and language. Specifically

for Linux continuous deployment [↗](#), you need to allow `oryx-cdn.microsoft.io:443`. For Python you additionally need to allow `files.pythonhosted.org`, `pypi.org`.

When you want to route outbound traffic on-premises, you can use a route table to send outbound traffic to your Azure ExpressRoute gateway. If you do route traffic to a gateway, set routes in the external network to send any replies back. Border Gateway Protocol (BGP) routes also affect your app traffic. If you have BGP routes from something like an ExpressRoute gateway, your app outbound traffic is affected. Similar to user-defined routes, BGP routes affect traffic according to your routing scope setting.

Service endpoints

Virtual network integration enables you to reach Azure services that are secured with service endpoints. To access a service endpoint-secured service, follow these steps:

1. Configure virtual network integration with your web app to connect to a specific subnet for integration.
2. Go to the destination service and configure service endpoints against the integration subnet.

Private endpoints

If you want to make calls to [private endpoints](#), make sure that your DNS lookups resolve to the private endpoint. You can enforce this behavior in one of the following ways:

- Integrate with Azure DNS private zones. When your virtual network doesn't have a custom DNS server, the integration is done automatically when the zones are linked to the virtual network.
- Manage the private endpoint in the DNS server used by your app. To manage the configuration, you must know the private endpoint IP address. Then point the endpoint you're trying to reach to that address by using an A record.
- Configure your own DNS server to forward to Azure DNS private zones.

Azure DNS private zones

After your app integrates with your virtual network, it uses the same DNS server that your virtual network is configured with. If no custom DNS is specified, it uses Azure default DNS and any private zones linked to the virtual network.

Limitations

There are some limitations with using virtual network integration:

- The feature is available from all App Service deployments in Premium v2 and Premium v3. It's also available in Basic and Standard tier but only from newer App Service deployments. If you're on an older deployment, you can only use the feature from a Premium v2 App Service plan. If you want to make sure you can use the feature in a Basic or Standard App Service plan, create your app in a Premium v3 App Service plan. Those plans are only supported on our newest deployments. You can scale down if you want after the plan is created.
- The feature isn't available for Isolated plan apps in an App Service Environment.
- You can't reach resources across peering connections with classic virtual networks.
- The feature requires an unused subnet that's an IPv4 /28 block or larger in an Azure Resource Manager virtual network. MPSJ requires a /26 block or larger.
- The app and the virtual network must be in the same region.
- The integration virtual network can't have IPv6 address spaces defined.
- The integration subnet can't have [service endpoint policies](#) enabled.
- You can't delete a virtual network with an integrated app. Remove the integration before you delete the virtual network.
- You can't have more than two virtual network integrations per App Service plan. Multiple apps in the same App Service plan can use the same virtual network integration.
- You can't change the subscription of an app or a plan while there's an app that's using virtual network integration.

Access on-premises resources

No extra configuration is required for the virtual network integration feature to reach through your virtual network to on-premises resources. You simply need to connect your virtual network to on-premises resources by using ExpressRoute or a site-to-site VPN.

Peering

If you use peering with virtual network integration, you don't need to do any more configuration.

Manage virtual network integration

Connecting and disconnecting with a virtual network is at an app level. Operations that can affect virtual network integration across multiple apps are at the App Service plan

level. From the app > **Networking** > **VNet integration** portal, you can get details on your virtual network. You can see similar information at the App Service plan level in the **App Service plan** > **Networking** > **VNet integration** portal.

In the app view of your virtual network integration instance, you can disconnect your app from the virtual network and you can configure application routing. To disconnect your app from a virtual network, select **Disconnect**. Your app is restarted when you disconnect from a virtual network. Disconnecting doesn't change your virtual network. The subnet isn't removed. If you then want to delete your virtual network, first disconnect your app from the virtual network.

The private IP assigned to the instance is exposed via the environment variable `WEBSITE_PRIVATE_IP`. Kudu console UI also shows the list of environment variables available to the web app. This IP is assigned from the address range of the integrated subnet. This IP is used by the web app to connect to the resources through the Azure virtual network.

 **Note**

The value of `WEBSITE_PRIVATE_IP` is bound to change. However, it will be an IP within the address range of the integration subnet, so you'll need to allow access from the entire address range.

Pricing details

The virtual network integration feature has no extra charge for use beyond the App Service plan pricing tier charges.

Troubleshooting

The feature is easy to set up, but that doesn't mean your experience is problem free. If you encounter problems accessing your desired endpoint, there are various steps you can take depending on what you are observing. For more information, see [virtual network integration troubleshooting guide](#).

 **Note**

- Virtual network integration isn't supported for Docker Compose scenarios in App Service.

- Access restrictions does not apply to traffic coming through a private endpoint.

Deleting the App Service plan or app before disconnecting the network integration

If you deleted the app or the App Service plan without disconnecting the virtual network integration first, you aren't able to do any update/delete operations on the virtual network or subnet that was used for the integration with the deleted resource. A subnet delegation 'Microsoft.Web/serverFarms' remains assigned to your subnet and prevents the update and delete operations.

In order to do update/delete the subnet or virtual network again, you need to re-create the virtual network integration, and then disconnect it:

1. Re-create the App Service plan and app (it's mandatory to use the exact same web app name as before).
2. Navigate to **Networking** on the app in Azure portal and configure the virtual network integration.
3. After the virtual network integration is configured, select the 'Disconnect' button.
4. Delete the App Service plan or app.
5. Update/Delete the subnet or virtual network.

If you still encounter issues with the virtual network integration after following these steps, contact Microsoft Support.

Enable virtual network integration in Azure App Service

Article • 09/18/2023

Through integrating with an Azure virtual network from your [Azure App Service app](#), you can reach private resources from your app within the virtual network.

Prerequisites

The virtual network integration feature requires:

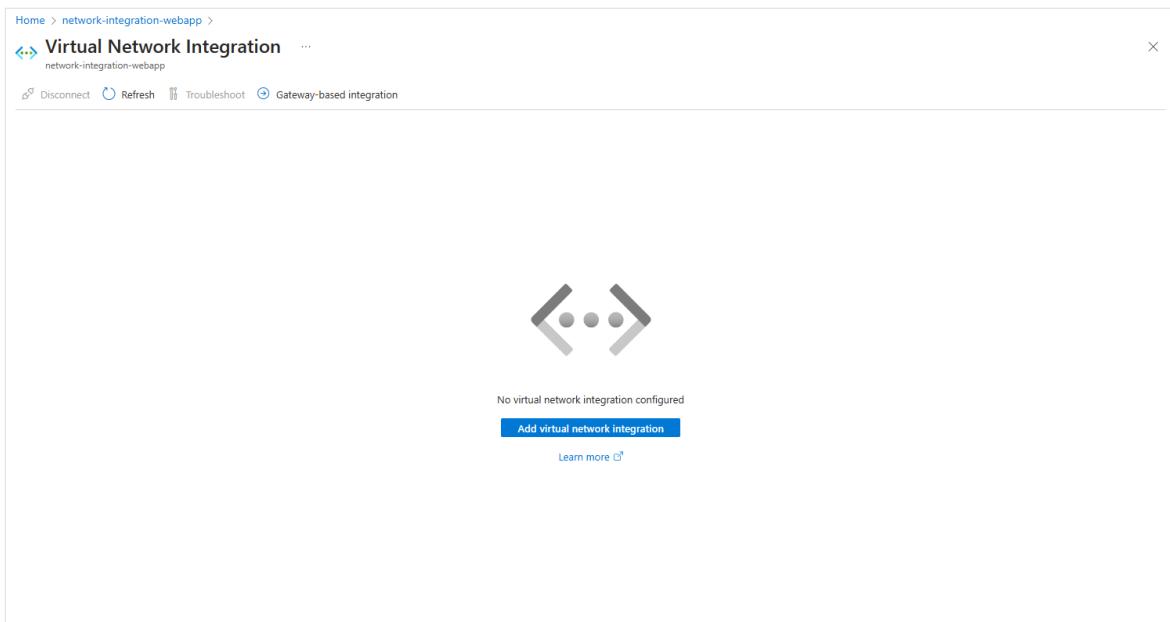
- An App Service pricing tier [that supports virtual network integration](#).
- A virtual network in the same region with an empty subnet.

The subnet must be delegated to Microsoft.Web/serverFarms. If the delegation isn't done before integration, the provisioning process configures this delegation. The subnet must be allocated an IPv4 /28 block (16 addresses). We recommend that you have a minimum of 64 addresses (IPv4 /26 block) to allow for maximum horizontal scale.

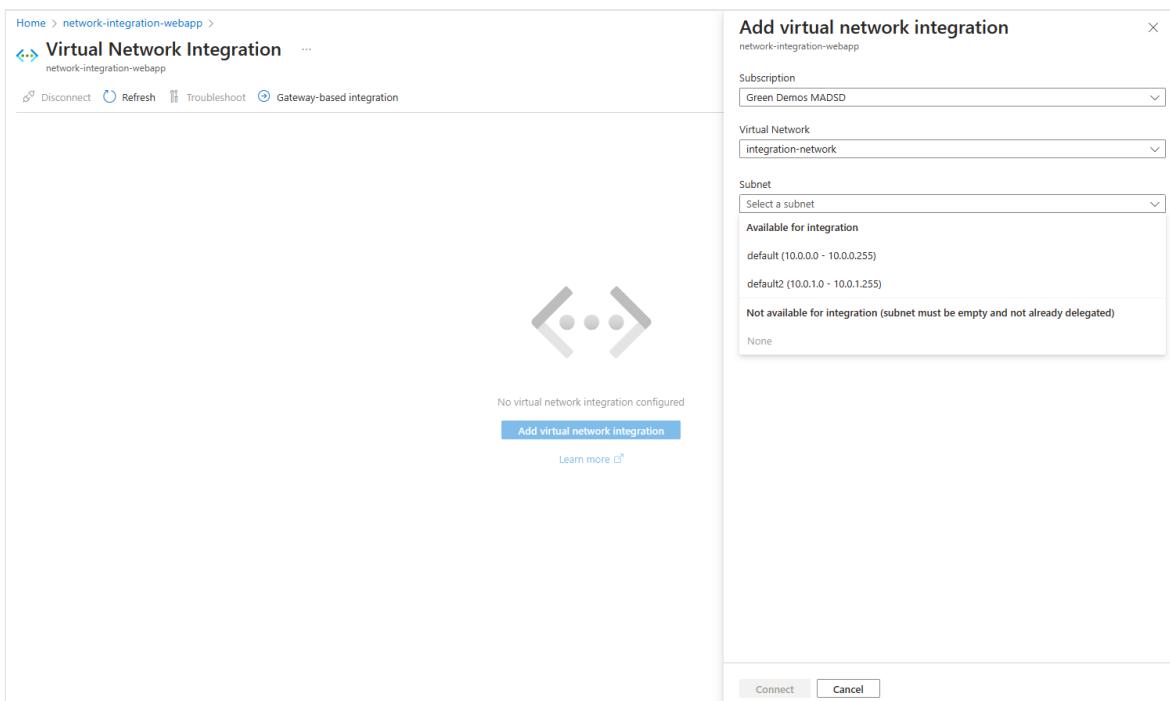
If the virtual network is in a different subscription than the app, you must ensure that the subscription with the virtual network is registered for the `Microsoft.Web` resource provider. You can explicitly register the provider [by following this documentation](#), but it's automatically registered when creating the first web app in a subscription.

Configure in the Azure portal

1. Go to **Networking** in the App Service portal. Under **Outbound traffic configuration**, select **Virtual network integration**.
2. Select **Add virtual network integration**.



3. The dropdown list contains all the virtual networks in your subscription in the same region. Select an empty pre-existing subnet or create a new subnet.



During the integration, your app is restarted. When integration is finished, you see details on the virtual network you're integrated with.

Configure with the Azure CLI

You can also configure virtual network integration by using the Azure CLI:

```
Azure CLI
az webapp vnet-integration add --resource-group <group-name> --name <app-
```

```
name> --vnet <vnet-name> --subnet <subnet-name>
```

ⓘ Note

The command checks if the subnet is delegated to Microsoft.Web/serverFarms and applies the necessary delegation if it isn't configured. If the subnet was configured, and you don't have permissions to check it, or if the virtual network is in another subscription, you can use the `--skip-delegation-check` parameter to bypass the validation.

Configure with Azure PowerShell

Prepare parameters.

Azure PowerShell

```
$siteName = '<app-name>'  
$vNetResourceGroupName = '<group-name>'  
$webAppResourceGroupName = '<group-name>'  
$vNetName = '<vnet-name>'  
$integrationSubnetName = '<subnet-name>'  
$vNetSubscriptionId = '<subscription-guid>'
```

ⓘ Note

If the virtual network is in another subscription than webapp, you can use the `Set-AzContext -Subscription "xxxx-xxxx-xxxx-xxxx"` command to set the current subscription context. Set the current subscription context to the subscription where the virtual network was deployed.

Check if the subnet is delegated to Microsoft.Web/serverFarms.

Azure PowerShell

```
$vnet = Get-AzVirtualNetwork -Name $vNetName -ResourceGroupName  
$vNetResourceGroupName  
$subnet = Get-AzVirtualNetworkSubnetConfig -Name $integrationSubnetName -  
VirtualNetwork $vnet  
Get-AzDelegation -Subnet $subnet
```

If your subnet isn't delegated to Microsoft.Web/serverFarms, add delegation using below commands.

Azure PowerShell

```
$subnet = Add-AzDelegation -Name "myDelegation" -ServiceName  
"Microsoft.Web/serverFarms" -Subnet $subnet  
Set-AzVirtualNetwork -VirtualNetwork $vnet
```

Configure virtual network integration.

ⓘ Note

If the webapp is in another subscription than virtual network, you can use the `Set-AzContext -Subscription "xxxx-xxxx-xxxx-xxxx"` command to set the current subscription context. Set the current subscription context to the subscription where the web app was deployed.

Azure PowerShell

```
$subnetResourceId =  
"/subscriptions/$vNetSubscriptionId/resourceGroups/$vNetResourceGroupName/pr  
oviders/Microsoft.Network/virtualNetworks/$vNetName/subnets/$integrationSubn  
etName"  
$webApp = Get-AzResource -ResourceType Microsoft.Web/sites -  
ResourceGroupName $webAppResourceGroupName -ResourceName $siteName  
$webApp.Properties.virtualNetworkSubnetId = $subnetResourceId  
$webApp.Properties.vnetRouteAllEnabled = 'true'  
$webApp | Set-AzResource -Force
```

Next steps

- Configure virtual network integration routing
- General networking overview

Manage Azure App Service virtual network integration routing

Article • 09/18/2023

Through application routing or configuration routing options, you can configure what traffic is sent through the virtual network integration. For more information, see the [overview section](#).

Prerequisites

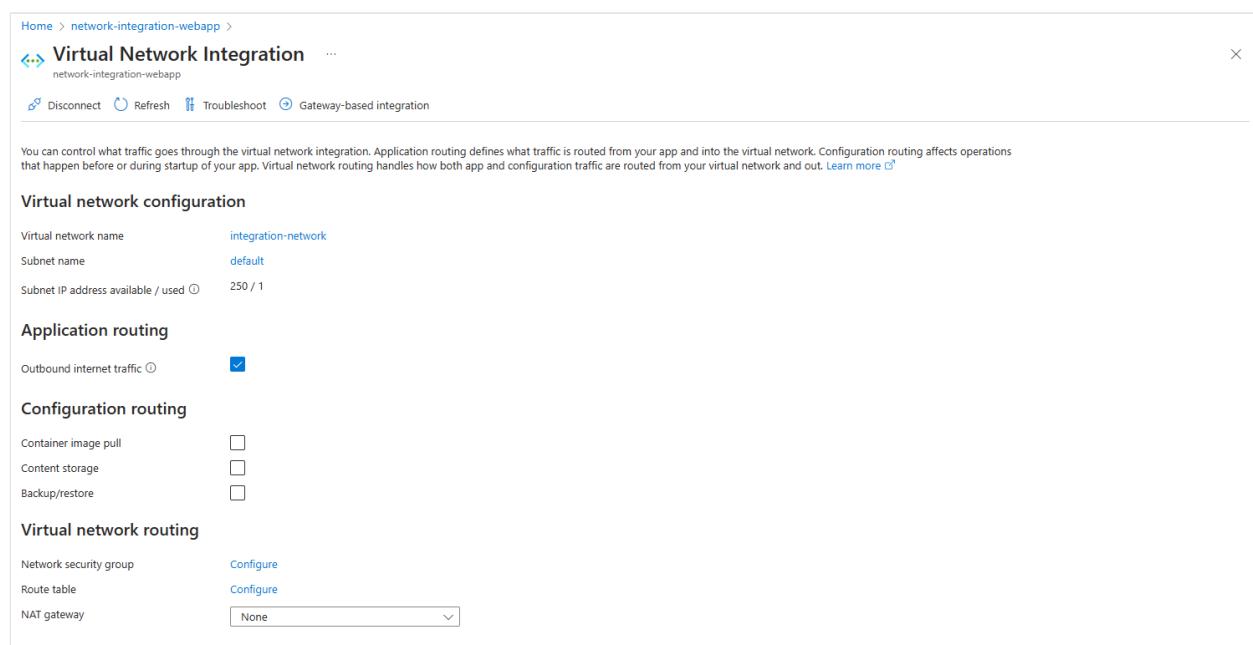
Your app is already integrated using the regional virtual network integration feature.

Configure application routing

Application routing defines what traffic is routed from your app and into the virtual network. We recommend that you use the `vnetRouteAllEnabled` site setting to enable routing of all traffic. Using the configuration setting allows you to audit the behavior with a [built-in policy](#). The existing `WEBSITE_VNET_ROUTE_ALL` app setting can still be used, and you can enable all traffic routing with either setting.

Configure in the Azure portal

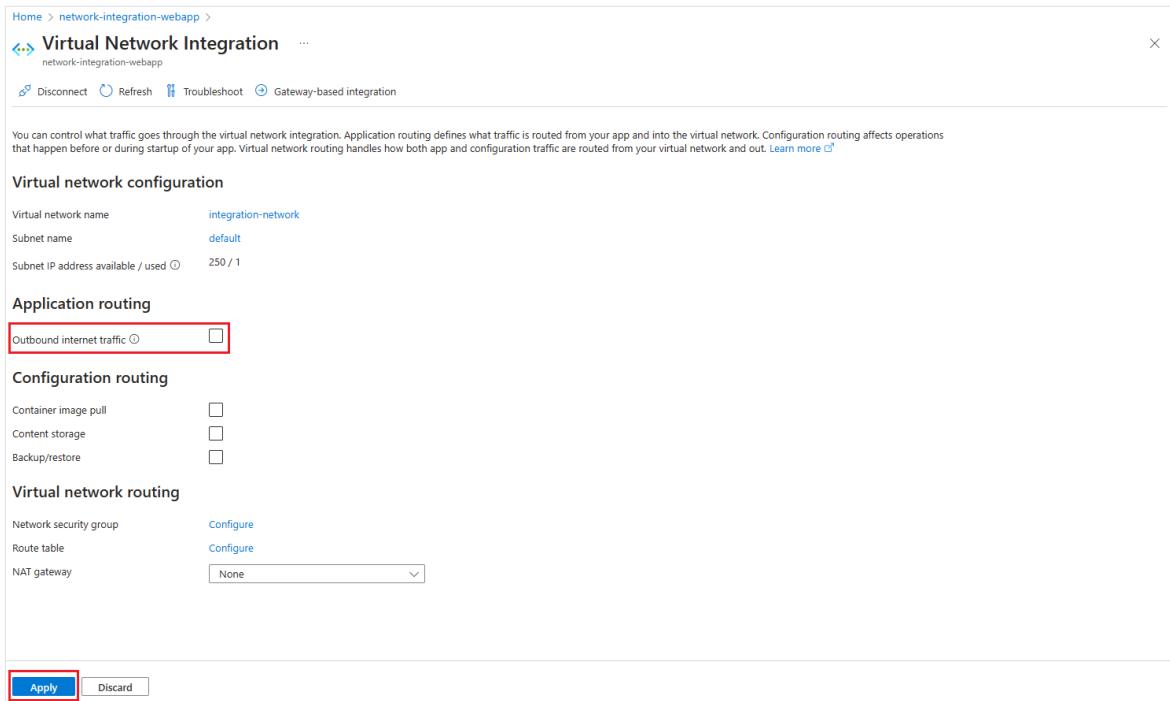
Follow these steps to disable outbound internet traffic routing in your app through the portal.



The screenshot shows the 'Virtual Network Integration' blade for a web app named 'network-integration-webapp'. The 'Virtual network configuration' section shows a selected virtual network named 'integration-network' and its default subnet. Under 'Application routing', the 'Outbound internet traffic' checkbox is checked. In the 'Configuration routing' section, three checkboxes for 'Container image pull', 'Content storage', and 'Backup/restore' are unselected. The 'Virtual network routing' section includes fields for 'Network security group', 'Route table', and 'NAT gateway', all currently set to 'None'. A note at the top states: 'You can control what traffic goes through the virtual network integration. Application routing defines what traffic is routed from your app and into the virtual network. Configuration routing affects operations that happen before or during startup of your app. Virtual network routing handles how both app and configuration traffic are routed from your virtual network and out.' A 'Learn more' link is provided.

1. Go to **Networking > Virtual network integration** in your app portal.

2. Uncheck the **Outbound internet traffic** setting.



The screenshot shows the 'Virtual Network Integration' settings for an app named 'network-integration-webapp'. Under 'Application routing', the 'Outbound internet traffic' checkbox is checked and highlighted with a red box. Other options like 'Container image pull', 'Content storage', and 'Backup/restore' have empty checkboxes. Under 'Virtual network routing', there are sections for 'Network security group', 'Route table', and 'NAT gateway', each with configuration links. At the bottom are 'Apply' and 'Discard' buttons, with 'Apply' also highlighted with a red box.

3. Select **Apply** to confirm.

Configure with the Azure CLI

You can also configure **Outbound internet traffic** by using the Azure CLI.



```
az resource update --resource-group <group-name> --name <app-name> --resource-type "Microsoft.Web/sites" --set properties.vnetRouteAllEnabled=[true|false]
```

Configure configuration routing

When you're using virtual network integration, you can configure how parts of the configuration traffic are managed. By default, configuration traffic goes directly over the public route, but for the mentioned individual components, you can actively configure it to be routed through the virtual network integration.

Container image pull

Routing container image pull over virtual network integration can be configured using the Azure CLI.

Azure CLI

```
az resource update --resource-group <group-name> --name <app-name> --  
resource-type "Microsoft.Web/sites" --set properties.vnetImagePullEnabled=[true|false]
```

We recommend that you use the site property to enable routing image pull traffic through the virtual network integration. Using the configuration setting allows you to audit the behavior with Azure Policy. The existing `WEBSITE_PULL_IMAGE_OVER_VNET` app setting with the value `true` can still be used, and you can enable routing through the virtual network with either setting.

Content share

Routing content share over virtual network integration can be configured using the Azure CLI. In addition to enabling the feature, you must also ensure that any firewall or Network Security Group configured on traffic from the subnet allow traffic to port 443 and 445.

Azure CLI

```
az resource update --resource-group <group-name> --name <app-name> --  
resource-type "Microsoft.Web/sites" --set  
properties.vnetContentShareEnabled=[true|false]
```

We recommend that you use the site property to enable content share traffic through the virtual network integration. Using the configuration setting allows you to audit the behavior with Azure Policy. The existing `WEBSITE_CONTENTOVERVNET` app setting with the value `1` can still be used, and you can enable routing through the virtual network with either setting.

Backup/restore

Routing backup traffic over virtual network integration can be configured using the Azure CLI. Database backup isn't supported over the virtual network integration.

Azure CLI

```
az resource update --resource-group <group-name> --name <app-name> --  
resource-type "Microsoft.Web/sites" --set  
properties.vnetBackupRestoreEnabled=[true|false]
```

Next steps

- Enable virtual network integration
- General networking overview

Configure gateway-required virtual network integration

Article • 10/17/2023

Gateway-required virtual network integration supports connecting to a virtual network in another region or to a classic virtual network. Gateway-required virtual network integration only works for Windows plans. We recommend using [regional virtual network integration](#) to integrate with virtual networks.

Gateway-required virtual network integration:

- Enables an app to connect to only one virtual network at a time.
- Enables up to five virtual networks to be integrated within an App Service plan.
- Allows the same virtual network to be used by multiple apps in an App Service plan without affecting the total number that can be used by an App Service plan. If you have six apps using the same virtual network in the same App Service plan that counts as one virtual network being used.
- SLA on the gateway can affect the overall [SLA](#).
- Enables your apps to use the DNS that the virtual network is configured with.
- Requires a virtual network route-based gateway configured with an SSTP point-to-site VPN before it can be connected to an app.

You can't use gateway-required virtual network integration:

- With a virtual network connected with ExpressRoute.
- From a Linux app.
- From a [Windows container](#).
- To access service endpoint-secured resources.
- To resolve App Settings referencing a network protected Key Vault.
- With a coexistence gateway that supports both ExpressRoute and point-to-site or site-to-site VPNs.

[Regional virtual network integration](#) mitigates the above mentioned limitations.

Set up a gateway in your Azure virtual network

To create a gateway:

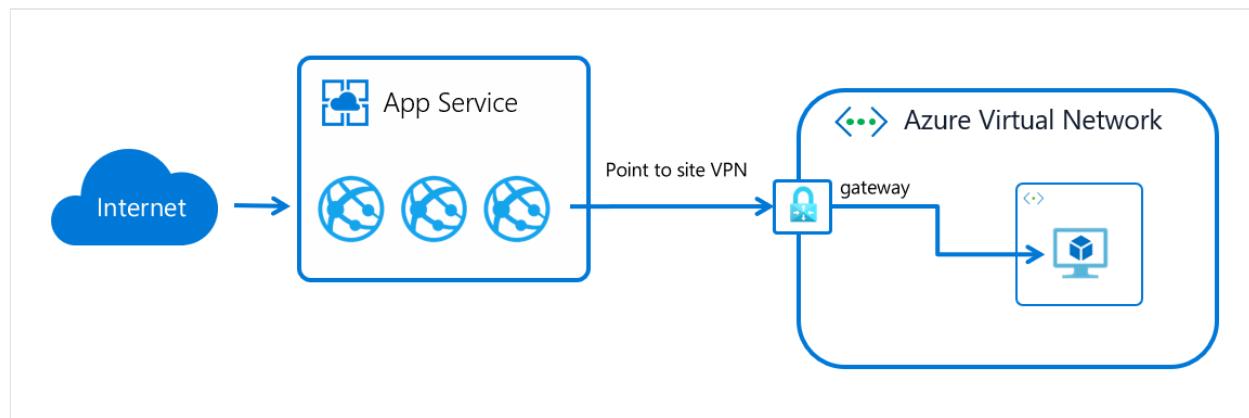
1. [Create the VPN gateway and subnet](#). Select a route-based VPN type.

2. [Set the point-to-site addresses](#). If the gateway isn't in the basic SKU, then IKEV2 must be disabled in the point-to-site configuration and SSTP must be selected. The point-to-site address space must be in the RFC 1918 address blocks 10.0.0.0/8, 172.16.0.0/12, and 192.168.0.0/16.

If you create the gateway for use with gateway-required virtual network integration, you don't need to upload a certificate. Creating the gateway can take 30 minutes. You won't be able to integrate your app with your virtual network until the gateway is created.

How gateway-required virtual network integration works

Gateway-required virtual network integration is built on top of point-to-site VPN technology. Point-to-site VPNs limit network access to the virtual machine that hosts the app. Apps are restricted to send traffic out to the internet only through hybrid connections or through virtual network integration. When your app is configured with the portal to use gateway-required virtual network integration, a complex negotiation is managed on your behalf to create and assign certificates on the gateway and the application side. The result is that the workers used to host your apps can directly connect to the virtual network gateway in the selected virtual network.



Access on-premises resources

Apps can access on-premises resources by integrating with virtual networks that have site-to-site connections. If you use gateway-required virtual network integration, update your on-premises VPN gateway routes with your point-to-site address blocks. When the site-to-site VPN is first set up, the scripts used to configure it should set up routes properly. If you add the point-to-site addresses after you create your site-to-site VPN, you need to update the routes manually. Details on how to do that varies per gateway and aren't described here.

BGP routes from on-premises won't be propagated automatically into App Service. You need to manually propagate them on the point-to-site configuration using the steps in this document [Advertise custom routes for P2S VPN clients](#).

Note

The gateway-required virtual network integration feature doesn't integrate an app with a virtual network that has an ExpressRoute gateway. Even if the ExpressRoute gateway is configured in **coexistence mode**, the virtual network integration doesn't work. If you need to access resources through an ExpressRoute connection, use the regional virtual network integration feature or an **App Service Environment**, which runs in your virtual network.

Peering

If you use gateway-required virtual network integration with peering, you need to configure a few more items. To configure peering to work with your app:

1. Add a peering connection on the virtual network your app connects to. When you add the peering connection, enable **Allow virtual network access** and select **Allow forwarded traffic** and **Allow gateway transit**.
2. Add a peering connection on the virtual network that's being peered to the virtual network you're connected to. When you add the peering connection on the destination virtual network, enable **Allow virtual network access** and select **Allow forwarded traffic** and **Allow remote gateways**.
3. Go to **App Service plan > Networking > VNet integration** in the portal. Select the virtual network your app connects to. Under the routing section, add the address range of the virtual network that's peered with the virtual network your app is connected to.

Manage virtual network integration

Connecting and disconnecting with a virtual network is at an app level. Operations that can affect virtual network integration across multiple apps are at the App Service plan level. From the **app > Networking > VNet integration** portal, you can get details on your virtual network. You can see similar information at the App Service plan level in the **App Service plan > Networking > VNet integration** portal.

The only operation you can take in the app view of your virtual network integration instance is to disconnect your app from the virtual network it's currently connected to.

To disconnect your app from a virtual network, select **Disconnect**. Your app is restarted when you disconnect from a virtual network. Disconnecting doesn't change your virtual network. The subnet or gateway isn't removed. If you then want to delete your virtual network, first disconnect your app from the virtual network and delete the resources in it, such as gateways.

The App Service plan virtual network integration UI shows you all the virtual network integrations used by the apps in your App Service plan. To see details on each virtual network, select the virtual network you're interested in. There are two actions you can perform here for gateway-required virtual network integration:

- **Sync network:** The sync network operation is used only for the gateway-required virtual network integration feature. Performing a sync network operation ensures that your certificates and network information are in sync. If you add or change the DNS of your virtual network, perform a sync network operation. This operation restarts any apps that use this virtual network. This operation won't work if you're using an app and a virtual network belonging to different subscriptions.
- **Add routes:** Adding routes drives outbound traffic into your virtual network.

The private IP assigned to the instance is exposed via the environment variable `WEBSITE_PRIVATE_IP`. Kudu console UI also shows the list of environment variables available to the web app. This IP is an IP from the address range of the point-to-site address pool configured on the virtual network gateway. This IP will be used by the web app to connect to the resources through the Azure virtual network.

 **Note**

The value of `WEBSITE_PRIVATE_IP` is bound to change. However, it will be an IP within the address range of the point-to-site address range, so you'll need to allow access from the entire address range.

Gateway-required virtual network integration routing

The routes that are defined in your virtual network are used to direct traffic into your virtual network from your app. To send more outbound traffic into the virtual network, add those address blocks here. This capability only works with gateway-required virtual network integration. Route tables don't affect your app traffic when you use gateway-required virtual network integration.

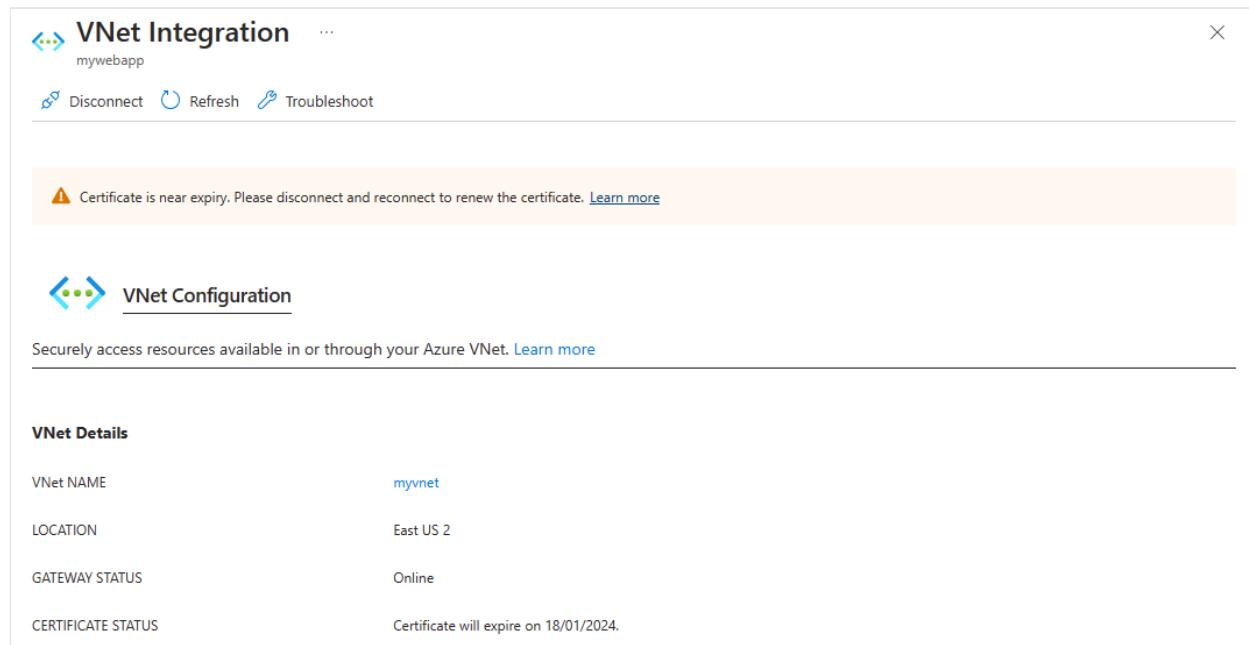
Gateway-required virtual network integration certificates

When gateway-required virtual network integration is enabled, there's a required exchange of certificates to ensure the security of the connection. Along with the certificates are the DNS configuration, routes, and other similar things that describe the network.

If certificates or network information is changed, select **Sync Network**. When you select **Sync Network**, you cause a brief outage in connectivity between your app and your virtual network. Your app isn't restarted, but the loss of connectivity could cause your site to not function properly.

Certificate renewal

The certificate used by the gateway-required virtual network integration has a lifespan of 8 years. If you have apps with gateway-required virtual network integrations that live longer you will have to renew the certificate. You can validate if your certificate has expired or has less than 6 month to expiry by visiting the VNet Integration page in Azure portal.



The screenshot shows the 'VNet Integration' blade for an app named 'mywebapp'. At the top, there are buttons for 'Disconnect', 'Refresh', and 'Troubleshoot'. A yellow warning bar at the top states: '⚠ Certificate is near expiry. Please disconnect and reconnect to renew the certificate. [Learn more](#)'. Below this, there's a section titled 'VNet Configuration' with the sub-instruction: 'Securely access resources available in or through your Azure VNet. [Learn more](#)'. The 'VNet Details' section lists the following information:

VNet NAME	myvnet
LOCATION	East US 2
GATEWAY STATUS	Online
CERTIFICATE STATUS	Certificate will expire on 18/01/2024.

You can renew your certificate when the portal shows a near expiry or expired certificate. To renew the certificate you need to disconnect and reconnect the virtual network. Reconnecting will cause a brief outage in connectivity between your app and your virtual network. Your app isn't restarted, but the loss of connectivity could cause your site to not function properly.

Pricing details

Three charges are related to the use of the gateway-required virtual network integration feature:

- **App Service plan pricing tier charges:** Your apps need to be in a Basic, Standard, Premium, Premium v2, or Premium v3 App Service plan. For more information on those costs, see [App Service pricing ↗](#).
- **Data transfer costs:** There's a charge for data egress, even if the virtual network is in the same datacenter. Those charges are described in [Data transfer pricing details ↗](#).
- **VPN gateway costs:** There's a cost to the virtual network gateway that's required for the point-to-site VPN. For more information, see [VPN gateway pricing ↗](#).

Troubleshooting

Many things can prevent your app from reaching a specific host and port. Most of the time it's one of these things:

- **A firewall is in the way.** If you have a firewall in the way, you hit the TCP timeout. The TCP timeout is 21 seconds in this case. Use the `tcpping` tool to test connectivity. TCP timeouts can be caused by many things beyond firewalls, but start there.
- **DNS isn't accessible.** The DNS timeout is 3 seconds per DNS server. If you have two DNS servers, the timeout is 6 seconds. Use `nameresolver` to see if DNS is working. You can't use `nslookup`, because that doesn't use the DNS your virtual network is configured with. If inaccessible, you could have a firewall or NSG blocking access to DNS or it could be down.

If those items don't answer your problems, look first for things like:

- Is the point-to-site address range in the RFC 1918 ranges (10.0.0.0-10.255.255.255 / 172.16.0.0-172.31.255.255 / 192.168.0.0-192.168.255.255)?
- Does the gateway show as being up in the portal? If your gateway is down, then bring it back up.
- Do certificates show as being in sync, or do you suspect that the network configuration was changed? If your certificates are out of sync or you suspect that a change was made to your virtual network configuration that wasn't synced with your ASPs, select **Sync Network**.
- If you're going across a VPN, is the on-premises gateway configured to route traffic back up to Azure? If you can reach endpoints in your virtual network but not on-premises, check your routes.

- Are you trying to use a coexistence gateway that supports both point to site and ExpressRoute? Coexistence gateways aren't supported with virtual network integration.

Debugging networking issues is a challenge because you can't see what's blocking access to a specific host:port combination. Some causes include:

- You have a firewall up on your host that prevents access to the application port from your point-to-site IP range. Crossing subnets often requires public access.
- Your target host is down.
- Your application is down.
- You had the wrong IP or hostname.
- Your application is listening on a different port than what you expected. You can match your process ID with the listening port by using "netstat -aon" on the endpoint host.
- Your network security groups are configured in such a manner that they prevent access to your application host and port from your point-to-site IP range.

You don't know what address your app actually uses. It could be any address in the point-to-site address range, so you need to allow access from the entire address range.

More debug steps include:

- Connect to a VM in your virtual network and attempt to reach your resource host:port from there. To test for TCP access, use the PowerShell command **Test-NetConnection**. The syntax is:

```
PowerShell
```

```
Test-NetConnection hostname [optional: -Port]
```

- Bring up an application on a VM and test access to that host and port from the console from your app by using **tcpping**.

On-premises resources

If your app can't reach a resource on-premises, check if you can reach the resource from your virtual network. Use the **Test-NetConnection** PowerShell command to check for TCP access. If your VM can't reach your on-premises resource, your VPN or ExpressRoute connection might not be configured properly.

If your virtual network-hosted VM can reach your on-premises system but your app can't, the cause is likely one of the following reasons:

- Your routes aren't configured with your subnet or point-to-site address ranges in your on-premises gateway.
- Your network security groups are blocking access for your point-to-site IP range.
- Your on-premises firewalls are blocking traffic from your point-to-site IP range.
- You're trying to reach a non-RFC 1918 address by using the regional virtual network integration feature.

For more information, see [virtual network integration troubleshooting guide](#).

Using private endpoints for App Service apps

Article • 02/10/2025

ⓘ Note

Starting June 1, 2024, newly created App Service apps can generate a unique default hostname that uses the naming convention <app-name>-<random-hash>. <region>.azurewebsites.net. Existing app names remain unchanged. For example:

myapp-ds27dh7271aah175.westus-01.azurewebsites.net

For more information, see [Unique Default Hostname for App Service Resource](#).

ⓘ Important

Private endpoints are available for Windows and Linux apps, containerized or not, hosted on these App Service plans: **Basic**, **Standard**, **PremiumV2**, **PremiumV3**, **IsolatedV2**, **Functions Premium** (sometimes called the *Elastic Premium* plan).

You can use a private endpoint for your App Service apps. The private endpoint allows clients located in your private network to securely access the app over Azure Private Link. The private endpoint uses an IP address from your Azure virtual network address space. Network traffic between a client on your private network and the app traverses over the virtual network and a Private Link on the Microsoft backbone network. This configuration eliminates exposure from the public Internet.

Using a private endpoint for your app enables you to:

- Secure your app by configuring the private endpoint and disable public network access to eliminating public exposure.
- Securely connect to your app from on-premises networks that connect to the virtual network using a VPN or ExpressRoute private peering.
- Avoid any data exfiltration from your virtual network.

Conceptual overview

A private endpoint is a special network interface (NIC) for your App Service app in a subnet in your virtual network. When you create a private endpoint for your app, it

provides secure connectivity between clients on your private network and your app. The private endpoint is assigned an IP Address from the IP address range of your virtual network. The connection between the private endpoint and the app uses a secure [Private Link](#). Private endpoint is only used for incoming traffic to your app. Outgoing traffic doesn't use this private endpoint. You can inject outgoing traffic to your network in a different subnet through the [virtual network integration feature](#).

Each slot of an app is configured separately. You can use up to 100 private endpoints per slot. You can't share a private endpoint between slots. The subresource name of a slot is `sites-<slot-name>`.

The subnet where you plug the private endpoint can have other resources in it. You don't need a dedicated empty subnet. You can also deploy the private endpoint in a different region than your app.

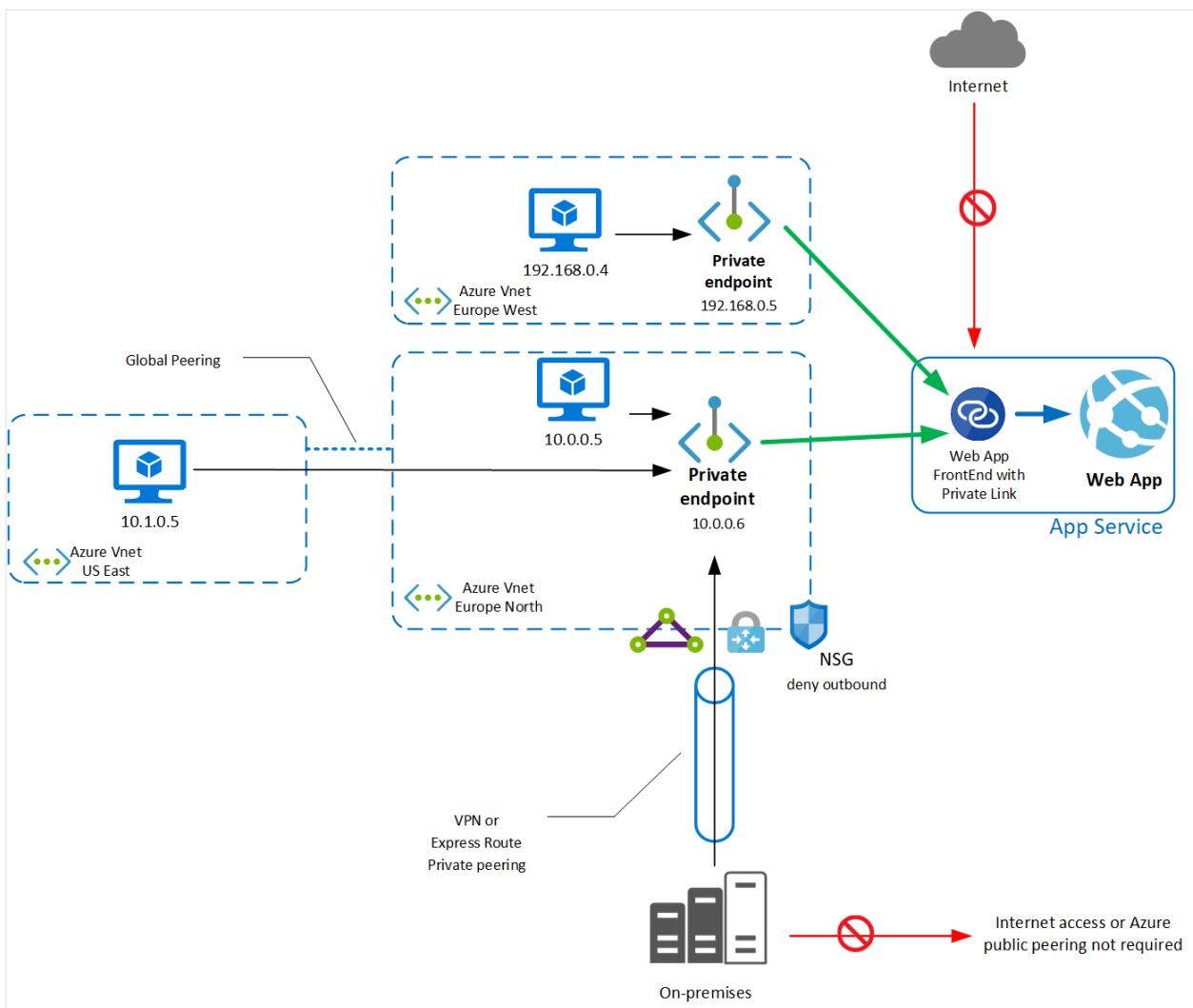
 **Note**

The virtual network integration feature can't use the same subnet as private endpoint.

From a security perspective:

- Private endpoint and public access can coexist on an app. For more information, see [this overview of access restrictions](#).
- To ensure isolation, when you enable private endpoints to your app, be sure that public network access is disabled.
- You can enable multiple private endpoints in others virtual networks and subnets, including virtual network in other regions.
- The access restrictions rules of your app aren't evaluated for traffic through the private endpoint.
- You can eliminate the data exfiltration risk from the virtual network by removing all Network Security Group (NSG) rules where destination is tag Internet or Azure services.

In the Web HTTP logs of your app, you find the client source IP. This feature is implemented using the TCP Proxy protocol, forwarding the client IP property up to the app. For more information, see [Getting connection Information using TCP Proxy v2](#).



DNS

When you use private endpoint for App Service apps, the requested URL must match the name of your app. By default, `<app-name>.azurewebsites.net`. When you use [unique default hostname](#), your app name has the format `<app-name>-<random-hash>. <region>.azurewebsites.net`. In the following examples, *mywebapp* could also represent the full regionalized unique hostname.

By default, without a private endpoint, the public name of your web app is a canonical name to the cluster. For example, the name resolution is:

[Expand table](#)

Name	Type	Value
mywebapp.azurewebsites.net	CNAME	clustername.azurewebsites.windows.net
clustername.azurewebsites.windows.net	CNAME	cloudservicename.cloudapp.net
cloudservicename.cloudapp.net	A	192.0.2.13

When you deploy a private endpoint, the approach updates the DNS entry to point to the canonical name: `mywebapp.privatelink.azurewebsites.net`. For example, the name resolution is:

[\[+\] Expand table](#)

Name	Type	Value	Remark
mywebapp.azurewebsites.net	CNAME	mywebapp.privatelink.azurewebsites.net	
mywebapp.privatelink.azurewebsites.net	CNAME	clustername.azurewebsites.windows.net	
clustername.azurewebsites.windows.net	CNAME	cloudservicename.cloudapp.net	
cloudservicename.cloudapp.net	A	192.0.2.13	<--This public IP isn't your private endpoint, you receive a 403 error

You must set up a private DNS server or an Azure DNS private zone. For tests, you can modify the host entry of your test machine. The DNS zone that you need to create is: `privatelink.azurewebsites.net`. Register the record for your app with a A record and the private endpoint IP. For example, the name resolution is:

[\[+\] Expand table](#)

Name	Type	Value	Remark
mywebapp.azurewebsites.net	CNAME	mywebapp.privatelink.azurewebsites.net	<-- Azure creates this CNAME entry in Azure Public DNS to point the app address to the private

Name	Type	Value	Remark
			endpoint address
mywebapp.privatelink.azurewebsites.net	A	10.10.10.8	<--You manage this entry in your DNS system to point to your private endpoint IP address

After this DNS configuration, you can reach your app privately with the default name mywebapp.azurewebsites.net. You must use this name, because the default certificate is issued for *.azurewebsites.net.

If you need to use a custom DNS name, add the custom name in your app and you must validate the custom name like any custom name, using public DNS resolution. For more information, see [custom DNS validation](#).

For the Kudu console, or Kudu REST API (deployment with Azure DevOps Services self-hosted agents, for example) you must create two records pointing to the private endpoint IP in your Azure DNS private zone or your custom DNS server. The first is for your app and the second is for the SCM of your app.

[] Expand table

Name	Type	Value
mywebapp.privatelink.azurewebsites.net	A	PrivateEndpointIP
mywebapp.scm.privatelink.azurewebsites.net	A	PrivateEndpointIP

App Service Environment v3 special consideration

In order to enable private endpoint for apps hosted in an IsolatedV2 plan (App Service Environment v3), enable the private endpoint support at the App Service Environment

level. You can activate the feature by the Azure portal in the App Service Environment configuration pane, or through the following CLI:

Azure CLI

```
az appservice ase update --name myasename --allow-new-private-endpoint-connections true
```

Specific requirements

If the virtual network is in a different subscription than the app, ensure that the subscription with the virtual network is registered for the `Microsoft.Web` resource provider. To explicitly register the provider, see [Register resource provider](#). You automatically register the provider when you create the first web app in a subscription.

Pricing

For pricing details, see [Azure Private Link pricing](#).

Limitations

- When you use Azure Function in Elastic Premium plan with a private endpoint, to run the function in Azure portal, you must have direct network access. Otherwise, you receive an HTTP 403 error. Your browser must be able to reach the private endpoint to run the function from the Azure portal.
- You can connect up to 100 private endpoints to a particular app.
- Remote Debugging functionality isn't available through the private endpoint. We recommend that you deploy the code to a slot and remote debug it there.
- FTP access is provided through the inbound public IP address. A private endpoint doesn't support FTP access to the app.
- IP-Based TLS isn't supported with private endpoints.
- Apps that you configure with private endpoints can't receive public traffic that comes from subnets with `Microsoft.Web` service endpoint enabled and can't use [service endpoint-based access restriction rules](#).
- Private endpoint naming must follow the rules defined for resources of type `Microsoft.Network/privateEndpoints`. For more information, see [Naming rules and restrictions](#).

For up-to-date information about limitations, see [Limitations](#).

Related content

- To deploy private endpoint for your app through the portal, see [How to connect privately to an app with the Azure portal](#).
 - To deploy private endpoint for your app using Azure CLI, see [How to connect privately to an app with Azure CLI](#).
 - To deploy private endpoint for your app using PowerShell, see [How to connect privately to an app with PowerShell](#).
 - To deploy private endpoint for your app using Azure template, see [How to connect privately to an app with Azure template](#).
 - To see an end-to-end example of how to connect a frontend app to a secured backend app with virtual network integration and private endpoint with ARM template, see this [quickstart](#).
 - To see an end-to-end example of how to connect a frontend app to a secured backend app with virtual network integration and private endpoint with terraform, see this [sample](#).
-

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

Control outbound traffic with Azure Firewall

Article • 01/25/2022

This article shows you how to lock down the outbound traffic from your App Service app to back-end Azure resources or other network resources with [Azure Firewall](#). This configuration helps prevent data exfiltration or the risk of malicious program implantation.

By default, an App Service app can make outbound requests to the public internet (for example, when installing required Node.js packages from NPM.org.). If your app is [integrated with an Azure virtual network](#), you can control outbound traffic with [network security groups](#) to a limited extent, such as the target IP address, port, and protocol. Azure Firewall lets you control outbound traffic at a much more granular level and filter traffic based on real-time threat intelligence from Microsoft Cyber Security. You can centrally create, enforce, and log application and network connectivity policies across subscriptions and virtual networks (see [Azure Firewall features](#)).

For detailed network concepts and security enhancements in App Service, see [Networking features](#) and [Zero to Hero with App Service, Part 6: Securing your web app ↗](#).

Prerequisites

- [Enable regional virtual network integration](#) for your app.
- [Verify that Route All is enabled](#). This setting is enabled by default, which tells App Service to route all outbound traffic through the integrated virtual network. If you disable it, only traffic to private IP addresses will be routed through the virtual network.
- If you want to route access to back-end Azure services through Azure Firewall as well, [disable all service endpoints](#) on the App Service subnet in the integrated virtual network. After Azure Firewall is configured, outbound traffic to Azure Services will be routed through the firewall instead of the service endpoints.

1. Create the required firewall subnet

To deploy a firewall into the integrated virtual network, you need a subnet called `AzureFirewallSubnet`.

1. In the [Azure portal](#), navigate to the virtual network that's integrated with your app.
2. From the left navigation, select **Subnets** > **+ Subnet**.
3. In **Name**, type **AzureFirewallSubnet**.
4. **Subnet address range**, accept the default or specify a range that's [at least /26 in size](#).
5. Select **Save**.

2. Deploy the firewall and get its IP

1. On the [Azure portal](#) menu or from the **Home** page, select **Create a resource**.
2. Type *firewall* in the search box and press **Enter**.
3. Select **Firewall** and then select **Create**.
4. On the **Create a Firewall** page, configure the firewall as shown in the following table:

Setting	Value
Resource group	Same resource group as the integrated virtual network.
Name	Name of your choice
Region	Same region as the integrated virtual network.
Firewall policy	Create one by selecting Add new .
Virtual network	Select the integrated virtual network.
Public IP address	Select an existing address or create one by selecting Add new .

Create a firewall

[Basics](#) [Tags](#) [Review + create](#)

Azure Firewall is a managed cloud-based network security service that protects your Azure Virtual Network resources. It is a fully stateful firewall as a service with built-in high availability and unrestricted cloud scalability. You can centrally create, enforce, and log application and network connectivity policies across subscriptions and virtual networks. Azure Firewall uses a static public IP address for your virtual network resources allowing outside firewalls to identify traffic originating from your virtual network. The service is fully integrated with Azure Monitor for logging and analytics. [Learn more](#).

Project details

Subscription *

Demo Subscription

Resource group *

rg-eus-appservice-networking-demo

[Create new](#)

Instance details

Name *

azfw

Region *

East US

Availability zone ⓘ

None

i Premium firewalls support additional capabilities, such as SSL termination and IDPS. Additional costs may apply. Migrating a Standard firewall to Premium will require some down-time. [Learn more](#)

Firewall tier

Standard

Premium

Firewall management

Use a Firewall Policy to manage this firewall

Use Firewall rules (classic) to manage this firewall

Firewall policy *

(New) azfw-policy

[Add new](#)

Choose a virtual network

Create new

Use existing

Virtual network

vnet-eus-hub (rg-eus-appservice-networking-demo)

Public IP address *

pip-azfw (20.39.40.85)

[Add new](#)

Forced tunneling ⓘ

Disabled

[Review + create](#)

[Previous](#)

[Next : Tags >](#)

[Download a template for automation](#)

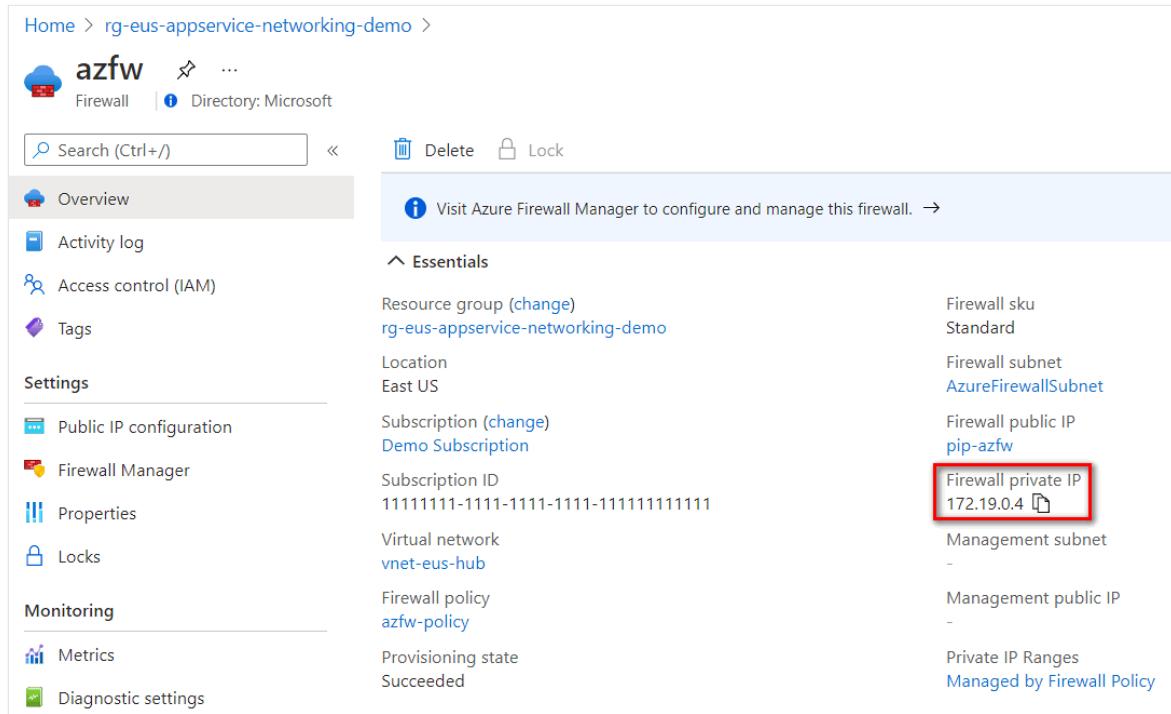
5. Click **Review + create**.

6. Select **Create** again.

This will take a few minutes to deploy.

7. After deployment completes, go to your resource group, and select the firewall.

8. In the firewall's **Overview** page, copy private IP address. The private IP address will be used as next hop address in the routing rule for the virtual network.



The screenshot shows the Azure Firewall Manager Overview page for a resource group named 'rg-eus-appservice-networking-demo'. The left sidebar lists navigation options: Overview (selected), Activity log, Access control (IAM), Tags, Settings (with Public IP configuration, Firewall Manager, Properties, Locks), and Monitoring (with Metrics, Diagnostic settings). The main content area displays essential details about the firewall, including its Resource group ('rg-eus-appservice-networking-demo'), Location ('East US'), Subscription ('Demo Subscription'), Subscription ID ('11111111-1111-1111-1111-111111111111'), Virtual network ('vnet-eus-hub'), Firewall policy ('azfw-policy'), Provisioning state ('Succeeded'), Firewall sku ('Standard'), Firewall subnet ('AzureFirewallSubnet'), Firewall public IP ('pip-azfw'), and Firewall private IP ('172.19.0.4'). The 'Firewall private IP' field is highlighted with a red box.

3. Route all traffic to the firewall

When you create a virtual network, Azure automatically creates a [default route table](#) for each of its subnets and adds system default routes to the table. In this step, you create a user-defined route table that routes all traffic to the firewall, and then associate it with the App Service subnet in the integrated virtual network.

1. On the [Azure portal](#) menu, select **All services** or search for and select **All services** from any page.
2. Under **Networking**, select **Route tables**.
3. Select **Add**.
4. Configure the route table like the following example:

Create Route table

Basics Tags Review + create

Project details

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

Subscription * ⓘ

Demo Subscription

Resource group * ⓘ

rg-eus-appservice-networking-demo

[Create new](#)

Instance details

Region * ⓘ

East US

Name * ⓘ

rt-app-service-snet

Propagate gateway routes * ⓘ

Yes

No

Make sure you select the same region as the firewall you created.

5. Select **Review + create**.

6. Select **Create**.

7. After deployment completes, select **Go to resource**.

8. From the left navigation, select **Routes > Add**.

9. Configure the new route as shown in the following table:

Setting	Value
Address prefix	0.0.0.0/0
Next hop type	Virtual appliance
Next hop address	The private IP address for the firewall that you copied in 2. Deploy the firewall and get its IP .

10. From the left navigation, select **Subnets > Associate**.

11. In **Virtual network**, select your integrated virtual network.

12. In **Subnet**, select the App Service subnet.

13. Select OK.

4. Configure firewall policies

Outbound traffic from your app is now routed through the integrated virtual network to the firewall. To control App Service outbound traffic, add an application rule to firewall policy.

1. Navigate to the firewall's overview page and select its firewall policy.
2. In the firewall policy page, from the left navigation, select **Application Rules > Add a rule collection**.
3. In **Rules**, add a network rule with the App Service subnet as the source address, and specify an FQDN destination. In the screenshot below, the destination FQDN is set to `api.my-ip.io`.

Name *	Source type	Source	Protocol *	TLS inspection	Destination Type *	Destination *
allow-sites	IP Address	172.21.0.0/28	https	<input type="checkbox"/> TLS inspection	FQDN	api.my-ip.io
	IP Address	*.192.168.10.1, 192...	http80,https,mssql...	<input type="checkbox"/> TLS inspection	FQDN	*.*.microsoft.com,...

! Note

Instead of specifying the App Service subnet as the source address, you can also use the private IP address of the app in the subnet directly. You can find

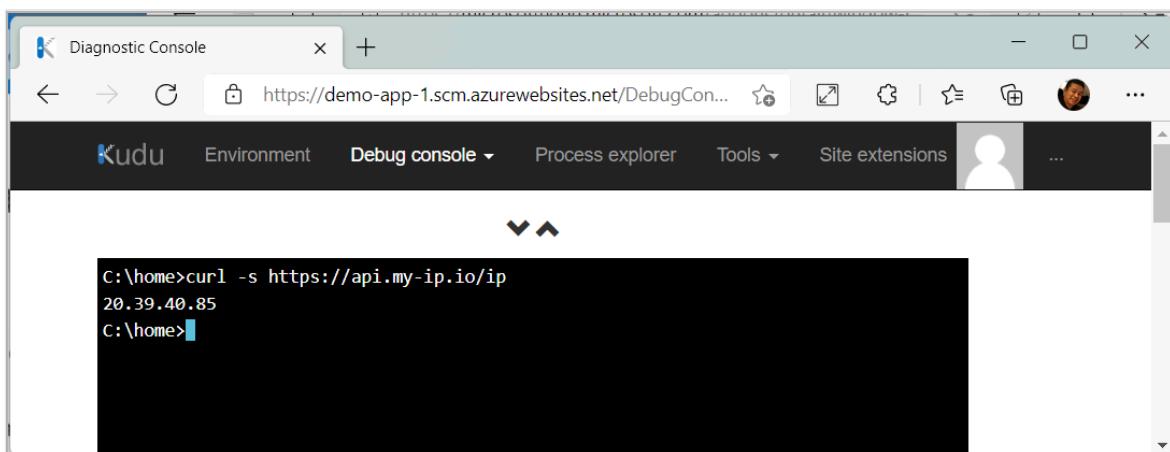
your app's private IP address in the subnet by using the `WEBSITE_PRIVATE_IP` environment variable.

4. Select Add.

5. Verify the outbound traffic

An easy way to verify your configuration is to use the `curl` command from your app's SCM debug console to verify the outbound connection.

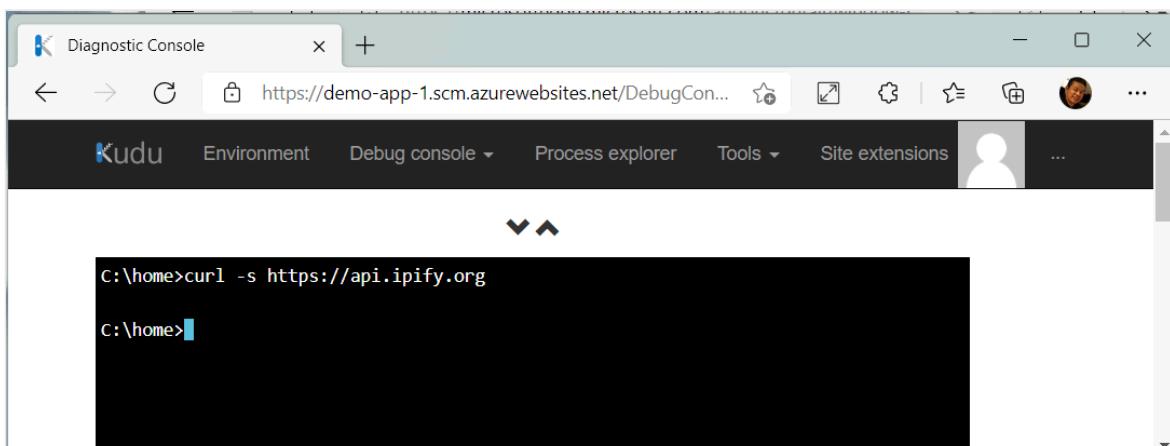
1. In a browser, navigate to `https://<app-name>.scm.azurewebsites.net/DebugConsole`.
2. In the console, run `curl -s <protocol>://<fqdn-address>` with a URL that matches the application rule you configured. To continue example in the previous screenshot, you can use `curl -s https://api.my-ip.io/ip`. The following screenshot shows a successful response from the API, showing the public IP address of your App Service app.



The screenshot shows the Azure Kudu Diagnostic Console interface. The title bar says "Diagnostic Console". The address bar shows the URL `https://demo-app-1.scm.azurewebsites.net/DebugCon...`. The main window has tabs for "Kudu", "Environment", "Debug console", "Process explorer", "Tools", and "Site extensions". The "Debug console" tab is selected. The console output window displays the following command and its result:

```
C:\home>curl -s https://api.my-ip.io/ip
20.39.40.85
C:\home>
```

3. Run `curl -s <protocol>://<fqdn-address>` again with a URL that doesn't match the application rule you configured. In the following screenshot, you get no response, which indicates that your firewall has blocked the outbound request from the app.



The screenshot shows the Azure Kudu Diagnostic Console interface, identical to the one above. The "Debug console" tab is selected. The console output window displays the following command and its result:

```
C:\home>curl -s https://api.ipify.org
C:\home>
```

💡 Tip

Because these outbound requests are going through the firewall, you can capture them in the firewall logs by **enabling diagnostic logging for the firewall** (enable the `AzureFirewallApplicationRule`).

If you run the `curl` commands with diagnostic logs enabled, you can find them in the firewall logs.

1. In the Azure portal, navigate to your firewall.
2. From the left navigation, select **Logs**.
3. Close the welcome message by selecting X.
4. From All Queries, select **Firewall Logs > Application rule log data**.
5. Click **Run**. You can see these two access logs in query result.

The screenshot shows the Azure Firewall Logs interface. On the left, the navigation pane includes links for Overview, Activity log, Access control (IAM), Tags, Public IP configuration, Firewall Manager, Properties, Locks, Monitoring, Metrics, Diagnostic settings, and Logs. The Logs link is highlighted with a red box. The main area displays a query editor with the following code:

```
1 // Application rule log data
2 // Parses the application rule log data.
3 AzureDiagnostics
4 log | where category == "AzureFirewallApplicationRule"
5 //this first parse statement is valid for all entries as they all start with this format
6 | parse msg_s with "request from \"SourceIP:\" \"SourcePort:\""
7 | parse msg_s with "to \"DestIP:\" \"DestPort:\""
8 | parse kind-regex Flags-U msg_s with " * ".Action": " Action \"\"
9 // case1: Action: A, Reason: R
10 | parse kind-regex Flags-U msg_s with "(\"L_Reason\":) \" Reason \"\"
11 // case2: Action: R, Reason: L
12 | parse kind-regex Flags-U msg_s with "Action: \"A\" RuleCollectionGroup: \"RG\" RuleCollection: \"RC\" Rule: \"R\"
```

The results table shows two entries:

TimeGenerated (UTC)	msg.s	Protocol	SourceIP	SourcePort	PGDN	TargetPort	Action
11/3/2021, 10:03:32.324 PM	HTTPS request from 172.21.0.14.50569 to api.ipify.org:443; Action: Allow; Reason: "User"	HTTPS	172.21.0.14	50569	api.ipify.org	443	Allow
11/3/2021, 10:02:19.855 PM	HTTPS request from 172.21.0.14.50494 to api.my-ip.io:443; Action: Allow; Reason: "User"	HTTPS	172.21.0.14	50494	api.my-ip.io	443	Allow

More resources

[Monitor Azure Firewall logs and metrics.](#)

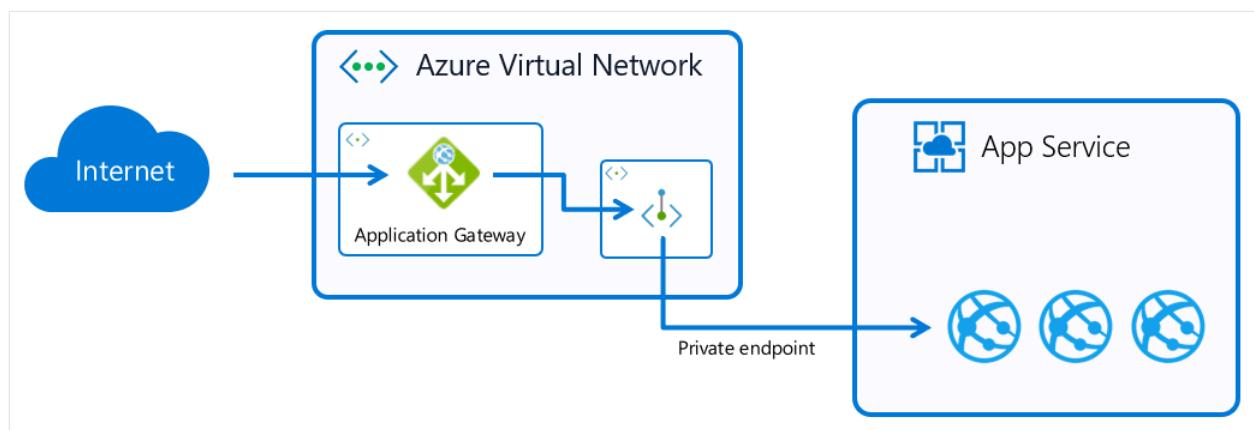
Application Gateway integration

Article • 01/07/2025

This article walks through how to configure Application Gateway with App Service by using private endpoints to secure traffic. The article also discusses considerations around using service endpoints and integrating with internal and external App Service Environments. Finally, the article describes how to set access restrictions on a Source Control Manager (SCM) site.

Integration with App Service

You can use private endpoints to secure traffic between Application Gateway and your App Service app. You need to ensure that Application Gateway can use DNS to resolve the private IP address of the App Service apps. Alternatively, you can use the private IP address in the back-end pool and override the host name in the HTTP settings.



Application Gateway caches the DNS lookup results. If you use fully qualified domain names (FQDNs) and rely on DNS lookup to get the private IP address, you might need to restart the application gateway if the DNS update or the link to an Azure private DNS zone happened after you configured the back-end pool.

To restart the application gateway, stop and start it by using the Azure CLI:

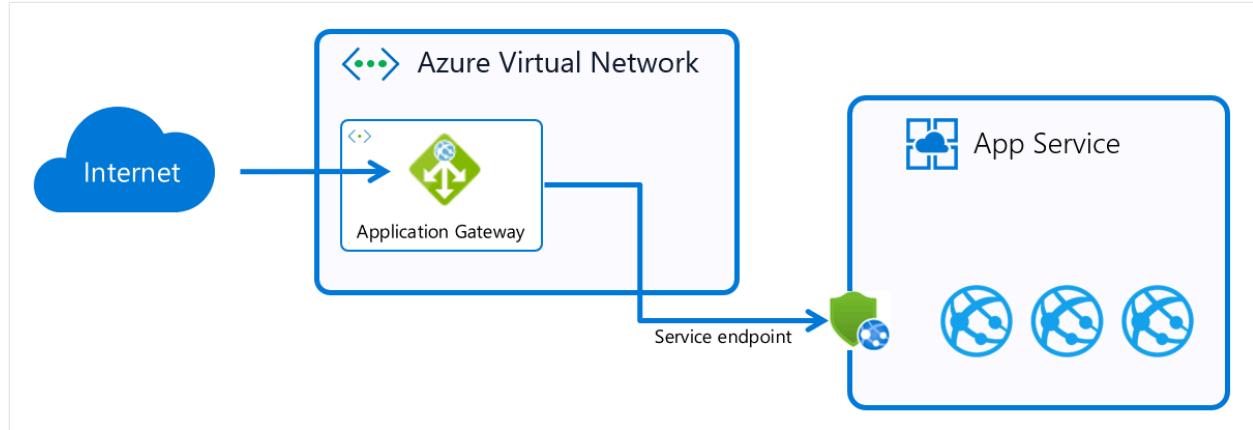
Azure CLI

```
az network application-gateway stop --resource-group myRG --name myAppGw  
az network application-gateway start --resource-group myRG --name myAppGw
```

Learn more about [configuring an App Service app with private endpoint](#).

Considerations for using service endpoints

As an alternative to private endpoints, you can use service endpoints to secure the traffic from Application Gateway. By using [service endpoints](#), you can allow traffic from only a specific subnet within an Azure virtual network and block everything else. In the following scenario, you use this functionality to ensure that an App Service instance can receive traffic from only a specific application gateway.



There are two parts to this configuration, aside from creating the App Service app instance and the application gateway. The first part is enabling service endpoints in the subnet of the virtual network where the application gateway is deployed. Service endpoints ensure that all network traffic leaving the subnet toward App Service is tagged with the specific subnet ID.

The second part is to set an access restriction on the specific web app to ensure that only traffic tagged with this specific subnet ID is allowed. You can configure the access restriction by using different tools, depending on your preference.

With the Azure portal, you follow four steps to create and configure the setup of App Service and Application Gateway. If you have existing resources, you can skip the first steps.

1. Create an App Service instance by using one of the quickstarts in the App Service documentation. One example is the [.NET Core quickstart](#).
2. Create an application gateway by using the [portal quickstart](#), but skip the section about adding back-end targets.
3. Configure [App Service as a back end in Application Gateway](#), but skip the section about restricting access.
4. Create the [access restriction by using service endpoints](#).

You can now access App Service through Application Gateway. If you try to access App Service directly, you should receive a 403 HTTP error that says the web app is blocking your access.

Error 403 - Forbidden

The web app you have attempted to reach has blocked your access.

Considerations for an internal App Service Environment

An internal App Service Environment isn't exposed to the internet. Traffic between the instance and an application gateway is already isolated to the virtual network. To configure an internal App Service Environment and integrate it with an application gateway by using the Azure portal, see the [how-to guide](#).

If you want to ensure that only traffic from the Application Gateway subnet is reaching the App Service Environment, you can configure a network security group (NSG) that affects all web apps in the App Service Environment. For the NSG, you can specify the subnet IP range and optionally the ports (80/443).

To isolate traffic to an individual web app, you need to use IP-based access restrictions, because service endpoints don't work with an App Service Environment. The IP address should be the private IP of the application gateway.

Considerations for an external App Service Environment

An external App Service Environment has a public-facing load balancer like multitenant App Service apps. Service endpoints don't work for an App Service Environment. With App Service Environment you can use IP-based access restrictions by using the public IP address of the application gateway. To create an external App Service Environment by using the Azure portal, you can follow [this quickstart](#).

You can also [add private endpoints to apps hosted on an external App Service Environment](#).

Considerations for a Kudu/SCM site

The SCM site, also known as Kudu, is an admin site that exists for every web app. It isn't possible to use reverse proxy for the SCM site. You most likely also want to lock it down to individual IP addresses or a specific subnet.

If you want to use the same access restrictions as the main site, you can inherit the settings by using the following command:

Azure CLI

```
az webapp config access-restriction set --resource-group myRG --name myWebApp --use-same-restrictions-for-scm-site
```

If you want to add individual access restrictions for the SCM site, you can use the `--scm-site` flag:

Azure CLI

```
az webapp config access-restriction add --resource-group myRG --name myWebApp --scm-site --rule-name KudoAccess --priority 200 --ip-address 208.130.0.0/16
```

Considerations for using the default domain

Configuring Application Gateway to override the host name and use the default domain of App Service (typically `azurewebsites.net`) is the easiest way to configure the integration. It doesn't require configuring a custom domain and certificate in App Service.

[This article](#) discusses the general considerations for overriding the original host name. In App Service, there are two scenarios where you need to pay attention with this configuration.

Authentication

When you use [the authentication feature](#) in App Service (also known as Easy Auth), your app typically redirects to the sign-in page. Because App Service doesn't know the

original host name of the request, the redirect is done on the default domain name and usually results in an error.

To work around the default redirect, you can configure authentication to inspect a forwarded header and adapt the redirect domain to the original domain. Application Gateway uses a header called `X-Original-Host`. By using [file-based configuration](#) to configure authentication, you can configure App Service to adapt to the original host name. Add this configuration to your configuration file:

JSON

```
{  
  ...  
  "httpSettings": {  
    "forwardProxy": {  
      "convention": "Custom",  
      "customHostHeaderName": "X-Original-Host"  
    }  
  }  
  ...  
}
```

Session affinity

In multiple-instance deployments, [session affinity](#) ensures that client requests are routed to the same instance for the life of the session. Session affinity can be configured to adapt the cookie domain to the incoming header from reverse proxy. By configuring [session affinity proxy](#) to true, session affinity looks for `X-Original-Host` or `X-Forwarded-Host` and adapt the cookie domain to the domain found in this header. As a recommended practice when enabling session affinity proxy, you should configure your access restrictions on the site to ensure that traffic is coming from your reverse proxy.

You can also configure `clientAffinityProxyEnabled` by using the following command:

Azure CLI

```
az resource update --resource-group myRG --name myWebApp --resource-type  
"Microsoft.Web/sites" --set properties.clientAffinityProxyEnabled=true
```

Next steps

For more information on App Service Environments, see the [App Service Environment documentation](#).

To further secure your web app, you can find information about Azure Web Application Firewall on Application Gateway in the [Azure Web Application Firewall documentation](#).

To deploy a secure, resilient site with a custom domain on App Service by using either Azure Front Door or Application Gateway, see [this tutorial](#).

Feedback

Was this page helpful?



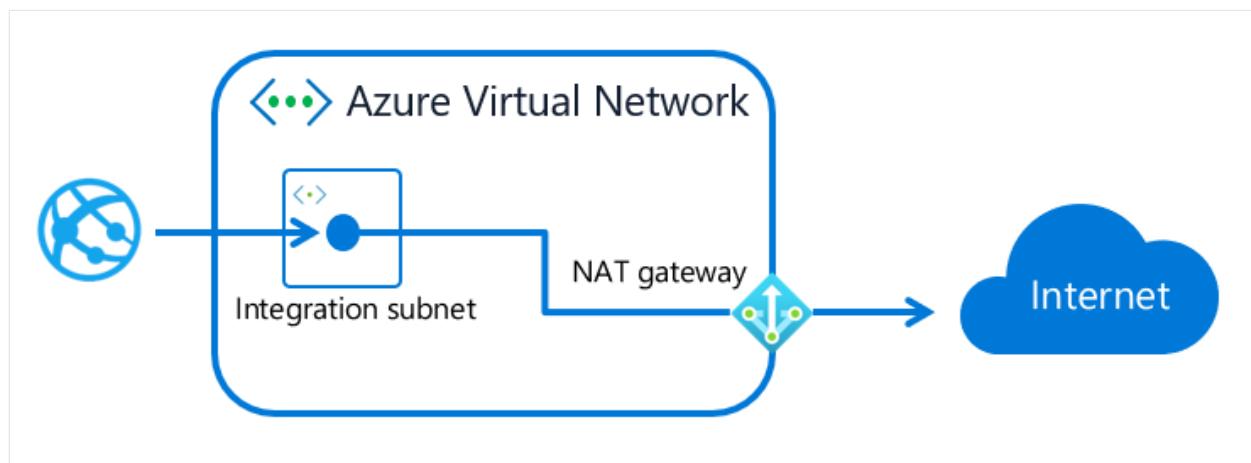
[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

Azure NAT Gateway integration

Article • 10/05/2023

Azure NAT Gateway is a fully managed, highly resilient service that can be associated with one or more subnets. It ensures that all outbound internet-facing traffic is routed through a network address translation (NAT) gateway. With Azure App Service, there are two important scenarios where you can use a NAT gateway.

The NAT gateway gives you a static, predictable public IP address for outbound internet-facing traffic. It also significantly increases the available [source network address translation \(SNAT\) ports](#) in scenarios where you have a high number of concurrent connections to the same public address/port combination.



Here are important considerations about Azure NAT Gateway integration:

- Using a NAT gateway with App Service is dependent on virtual network integration, so it requires a supported pricing tier in an App Service plan.
- When you're using a NAT gateway together with App Service, all traffic to Azure Storage must use private endpoints or service endpoints.
- You can't use a NAT gateway together with App Service Environment v1 or v2.

For more information and pricing, see the [Azure NAT Gateway overview](#).

Configure NAT gateway integration

To configure NAT gateway integration with App Service, first complete the following tasks:

- Configure regional virtual network integration with your app, as described in [Integrate your app with an Azure virtual network](#).

- Ensure that **Route All** is enabled for your virtual network integration, so routes in your virtual network affect the internet-bound traffic.
- Provision a NAT gateway with a public IP address and associate it with the subnet for virtual network integration.

Then, set up Azure NAT Gateway through the Azure portal:

1. In the Azure portal, go to **App Service > Networking**. In the **Outbound Traffic** section, select **Virtual network integration**. Ensure that your app is integrated with a subnet and that **Route All** is enabled.

VNet Details	
VNet NAME	integration-vnet
LOCATION	West Europe

VNet Address Space	
Start Address	10.42.0.0
End Address	10.42.255.255

Subnet Details	
Subnet NAME	webapp-integration-subnet

2. On the Azure portal menu or from the home page, select **Create a resource**. The **New** window appears.
3. Search for **NAT gateway** and select it from the list of results.
4. Fill in the **Basics** information and choose the region where your app is located.

Create network address translation (NAT) gateway

Basics Outbound IP Subnet Tags Review + create

Azure NAT gateway can be used to translate outbound flows from a virtual network to the public internet.
[Learn more about NAT gateways.](#)

Project details

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

Subscription *	Azure Subscription
Resource group *	vnet-integration-setup Create new

Instance details

NAT gateway name *	nat-gateway
Region *	(Europe) West Europe
Availability zone ⓘ	None
Idle timeout (minutes) * ⓘ	4

4-120

5. On the **Outbound IP** tab, create a public IP address or select an existing one.

Create network address translation (NAT) gateway

Basics **Outbound IP** Subnet Tags Review + create

Configure which public IP addresses and public IP prefixes to use. Each outbound IP address provides 64,000 SNAT ports for the NAT gateway resource to use. You can add up to 16 outbound IP addresses.

Note: While you do not have to complete this step to create a NAT gateway, the NAT gateway will not be functional and any subnet with this NAT gateway will not have outbound connectivity until you have added at least one public IP address or public IP prefix. You can also add and reconfigure which IP addresses are included after creating the NAT gateway.

Public IP addresses	0 selected Create a new public IP address						
<div style="border: 1px solid #ccc; padding: 10px; width: 300px;"> <p>Add a public IP address</p> <table border="0"> <tr> <td>Name *</td> <td>nat-public-ip</td> </tr> <tr> <td>SKU</td> <td><input type="radio"/> Basic <input checked="" type="radio"/> Standard</td> </tr> <tr> <td>Assignment</td> <td><input type="radio"/> Dynamic <input checked="" type="radio"/> Static</td> </tr> </table> <p style="text-align: center;">OK Cancel</p> </div>		Name *	nat-public-ip	SKU	<input type="radio"/> Basic <input checked="" type="radio"/> Standard	Assignment	<input type="radio"/> Dynamic <input checked="" type="radio"/> Static
Name *	nat-public-ip						
SKU	<input type="radio"/> Basic <input checked="" type="radio"/> Standard						
Assignment	<input type="radio"/> Dynamic <input checked="" type="radio"/> Static						

6. On the **Subnet** tab, select the subnet that you use for virtual network integration.

Create network address translation (NAT) gateway

Basics Outbound IP **Subnet** Tags Review + create

To use the NAT gateway, at least one subnet must be selected. You can add and remove subnets after creating the NAT gateway.

Virtual network ⓘ

integration-vnet

[Create new](#)

ⓘ Subnets that have any of the following resources are not shown because they are not compatible:

- A load balancer with a Basic SKU
- A public IP address with a Basic SKU
- An IPv6 address space
- An existing NAT gateway

Subnet name

Subnet address range

webapp-integration-subnet

10.42.0.0/24

[Manage subnets >](#)

- Fill in tags if needed, and then select **Create**. After the NAT gateway is provisioned, select **Go to resource group**, and then select the new NAT gateway. The **Outbound IP** pane shows the public IP address that your app will use for outbound internet-facing traffic.

nat-gateway | Outbound IP

NAT gateway

Search (Ctrl+ /)

«

Save

Discard

Refresh

Overview

View and configure which public IP addresses and public IP prefixes will be used for outbound connectivity. At least one is required for outbound connectivity.

Activity log

Tags

Public IP addresses [Change](#)

Name

IP address

DNS name

[nat-public-ip](#)

51.144.156.57

Settings

Outbound IP

Public IP prefixes [Change](#)

Subnets

No public IP prefixes

Configuration

Properties

If you prefer to use the Azure CLI to configure your environment, these are the important commands. As a prerequisite, create an app with virtual network integration configured.

- Ensure that **Route All** is configured for your virtual network integration:

Azure CLI

```
az webapp config set --resource-group [myResourceGroup] --name [myWebApp] --vnet-route-all-enabled
```

2. Create a public IP address and a NAT gateway:

Azure CLI

```
az network public-ip create --resource-group [myResourceGroup] --name myPublicIP --sku standard --allocation static  
az network nat gateway create --resource-group [myResourceGroup] --name myNATgateway --public-ip-addresses myPublicIP --idle-timeout 10
```

3. Associate the NAT gateway with the subnet for virtual network integration:

Azure CLI

```
az network vnet subnet update --resource-group [myResourceGroup] --vnet-name [myVnet] --name [myIntegrationSubnet] --nat-gateway myNATgateway
```

Scale a NAT gateway

You can use the same NAT gateway across multiple subnets in the same virtual network. That approach allows you to use a NAT gateway across multiple apps and App Service plans.

Azure NAT Gateway supports both public IP addresses and public IP prefixes. A NAT gateway can support up to 16 IP addresses across individual IP addresses and prefixes. Each IP address allocates 64,512 ports (SNAT ports), which allows up to 1 million available ports. Learn more in [Azure NAT Gateway resource](#).

Next steps

For more information on Azure NAT Gateway, see the [Azure NAT Gateway documentation](#).

For more information on virtual network integration, see the [documentation about virtual network integration](#).

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

Controlling Azure App Service traffic with Azure Traffic Manager

Article • 03/31/2020

ⓘ Note

This article provides summary information for Microsoft Azure Traffic Manager as it relates to Azure App Service. More information about Azure Traffic Manager itself can be found by visiting the links at the end of this article.

Introduction

You can use Azure Traffic Manager to control how requests from web clients are distributed to apps in Azure App Service. When App Service endpoints are added to an Azure Traffic Manager profile, Azure Traffic Manager keeps track of the status of your App Service apps (running, stopped, or deleted) so that it can decide which of those endpoints should receive traffic.

Routing methods

Azure Traffic Manager uses four different routing methods. These methods are described in the following list as they pertain to Azure App Service.

- **Priority:** use a primary app for all traffic, and provide backups in case the primary or the backup apps are unavailable.
- **Weighted:** distribute traffic across a set of apps, either evenly or according to weights, which you define.
- **Performance:** when you have apps in different geographic locations, use the "closest" app in terms of the lowest network latency.
- **Geographic:** direct users to specific apps based on which geographic location their DNS query originates from.

For more information, see [Traffic Manager routing methods](#).

App Service and Traffic Manager Profiles

To configure the control of App Service app traffic, you create a profile in Azure Traffic Manager that uses one of the four load balancing methods described previously, and

then add the endpoints (in this case, App Service) for which you want to control traffic to the profile. Your app status (running, stopped, or deleted) is regularly communicated to the profile so that Azure Traffic Manager can direct traffic accordingly.

When using Azure Traffic Manager with Azure, keep in mind the following points:

- For app only deployments within the same region, App Service already provides failover and round-robin functionality without regard to app mode.
- For deployments in the same region that use App Service in conjunction with another Azure cloud service, you can combine both types of endpoints to enable hybrid scenarios.
- You can only specify one App Service endpoint per region in a profile. When you select an app as an endpoint for one region, the remaining apps in that region become unavailable for selection for that profile.
- The App Service endpoints that you specify in an Azure Traffic Manager profile appears under the **Domain Names** section on the Configure page for the app in the profile, but is not configurable there.
- After you add an app to a profile, the **Site URL** on the Dashboard of the app's portal page displays the custom domain URL of the app if you have set one up. Otherwise, it displays the Traffic Manager profile URL (for example, `contoso.trafficmanager.net`). Both the direct domain name of the app and the Traffic Manager URL are visible on the app's Configure page under the **Domain Names** section.
- Your custom domain names work as expected, but in addition to adding them to your apps, you must also configure your DNS map to point to the Traffic Manager URL. For information on how to set up a custom domain for an App Service app, see [Configure a custom domain name in Azure App Service with Traffic Manager integration](#).
- You can only add apps that are in standard or premium mode to an Azure Traffic Manager profile.
- Adding an app to a Traffic Manager profile causes the app to be restarted.

Next Steps

For a conceptual and technical overview of Azure Traffic Manager, see [Traffic Manager Overview](#).

Azure App Service Hybrid Connections

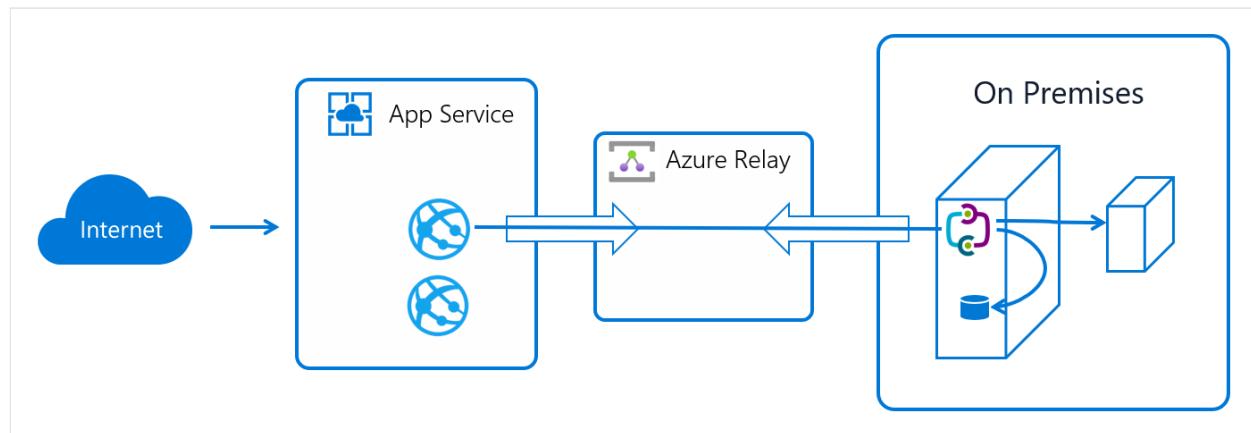
Article • 01/03/2024

Hybrid Connections is both a service in Azure and a feature in Azure App Service. As a service, it has uses and capabilities beyond those that are used in App Service. To learn more about Hybrid Connections and their usage outside App Service, see [Azure Relay Hybrid Connections](#).

Within App Service, Hybrid Connections can be used to access application resources in any network that can make outbound calls to Azure over port 443. Hybrid Connections provides access from your app to a TCP endpoint and doesn't enable a new way to access your app. As used in App Service, each Hybrid Connection correlates to a single TCP host and port combination. This feature enables your apps to access resources on any OS, provided it's a TCP endpoint. The Hybrid Connections feature doesn't know or care what the application protocol is, or what you are accessing. It simply provides network access.

How it works

Hybrid Connections requires a relay agent to be deployed where it can reach both the desired endpoint and Azure. The relay agent, Hybrid Connection Manager (HCM), calls out to Azure Relay over port 443. From the web app site, the App Service infrastructure also connects to Azure Relay on your application's behalf. Through the joined connections, your app is able to access the desired endpoint. The connection uses TLS 1.2 for security and shared access signature (SAS) keys for authentication and authorization.



When your app makes a DNS request that matches a configured Hybrid Connection endpoint, the outbound TCP traffic is redirected through the Hybrid Connection.

Note

This means that you should try to always use a DNS name for your Hybrid Connection. Some client software does not do a DNS lookup if the endpoint uses an IP address instead.

App Service Hybrid Connection benefits

There are many benefits to the Hybrid Connections capability, including:

- Apps can access on-premises systems and services securely.
- The feature doesn't require an internet-accessible endpoint.
- It's quick and easy to set up. No gateways required.
- Each Hybrid Connection matches to a single host:port combination, helpful for security.
- It normally doesn't require firewall holes. The connections are all outbound over standard web ports.
- Because the feature is network level, it's agnostic to the language used by your app and the technology used by the endpoint.
- It can be used to provide access in multiple networks from a single app.
- Supported in GA for Windows apps and Linux apps. It isn't supported for Windows custom containers.

Things you can't do with Hybrid Connections

Things you can't do with Hybrid Connections include:

- Mount a drive.
- Use UDP.
- Access TCP-based services that use dynamic ports, such as FTP Passive Mode or Extended Passive Mode.
- Support LDAP, because it can require UDP.
- Support Active Directory, because you can't domain join an App Service worker.

Add and Create Hybrid Connections in your app

To create a Hybrid Connection, go to the [Azure portal](#) and select your app. Select **Networking > Configure your Hybrid Connection endpoints**. Here you can see the

Hybrid Connections that are configured for your app.

The screenshot shows the 'Hybrid connections' page for an app named 'hcportalapp'. At the top, there's a refresh button and a link to 'Learn more'. Below that, it says 'App Service plan (pricing tier): hcportalplan (PremiumV3)' and 'Location: West Europe'. A circular progress bar indicates 'Connections used: 0' out of a 'Connections quota: 220'. There are links to 'Download connection manager' and 'Add hybrid connection'. The main table below shows no results.

Name	Status	Endpoint	Namespace
No results			

To add a new Hybrid Connection, select **[+] Add hybrid connection**. You see a list of the Hybrid Connections that you already created. To add one or more of them to your app, select the ones you want, and then select **Add selected Hybrid Connection**.

The screenshot shows the 'Add hybrid connection' page for the same app. It has buttons for 'Create new hybrid connection' and 'Add selected hybrid connection'. Below is a table with one entry:

Name	Host	Port	Namespace	Location
myhybrid	localhost	80	mylistener	West Europe

If you want to create a new Hybrid Connection, select **Create new hybrid connection**. Specify the:

- Hybrid Connection name.
- Endpoint hostname.
- Endpoint port.
- Service Bus namespace you want to use.

Create new hybrid connection X

Hybrid connection Name * ⓘ
mysql-hybridconnection ✓

Endpoint Host * ⓘ
mysqldev ✓

Endpoint Port * ⓘ
3306 ✓

Servicebus namespace * ⓘ
 Create new Select existing

Location *
West Europe ▼

Name *
hybridcon-sbns ✓

OK

Every Hybrid Connection is tied to a Service Bus namespace, and each Service Bus namespace is in an Azure region. It's important to try to use a Service Bus namespace in the same region as your app, to avoid network induced latency.

If you want to remove your Hybrid Connection from your app, right-click it and select **Disconnect**.

When a Hybrid Connection is added to your app, you can see details on it simply by selecting it.

Create a Hybrid Connection in the Azure Relay portal

In addition to the portal experience from within your app, you can create Hybrid Connections from within the Azure Relay portal. For a Hybrid Connection to be used by App Service, it must:

- Require client authorization.
- Have a metadata item and named endpoint that contains a host:port combination as the value.

Hybrid Connections and App Service plans

App Service Hybrid Connections are only available in Basic, Standard, Premium, and Isolated pricing SKUs. Hybrid Connections aren't available for function apps in Consumption plans. There are limits tied to the pricing plan.

[Expand table](#)

Pricing plan	Number of Hybrid Connections usable in the plan
Basic	5 per plan
Standard	25 per plan
Premium (v1-v3)	220 per app

Pricing plan	Number of Hybrid Connections usable in the plan
Isolated (v1-v2)	220 per app

The App Service plan UI shows you how many Hybrid Connections are being used and by what apps.

The screenshot shows two side-by-side views of the Azure portal. On the left is the 'Hybrid connect...' blade for the 'hcportalplan' app service plan, displaying a table of hybrid connections. One connection, 'mysql-hybridconnection', is selected, showing its details in the table. On the right is the 'Properties' blade for the same hybrid connection, providing detailed configuration information such as host, port, and connection string.

Name	Endpoint
mysql-hybridconnection	3306 : mysqldev

Properties

HYBRID CONNECTION NAME: mysql-hybridconnection

ENDPOINT HOST: mysqldev

ENDPOINT PORT: 3306

SERVICE BUS NAMESPACE: hybridcon-sbns

NAMESPACE LOCATION: West Europe

NUMBER OF CONNECTED SITES: 1

GATEWAY CONNECTION STRING: Endpoint=sb://hybridcon-sbns.servicebus.windows.net/;SharedAccessKeyName=defaultListener;Shared...

Select the Hybrid Connection to see details. You can see all the information that you saw at the app view. You can also see how many other apps in the same plan are using that Hybrid Connection.

There's a limit on the number of Hybrid Connection endpoints that can be used in an App Service plan. Each Hybrid Connection used, however, can be used across any number of apps in that plan. For example, a single Hybrid Connection that is used in five separate apps in an App Service plan counts as one Hybrid Connection.

Pricing

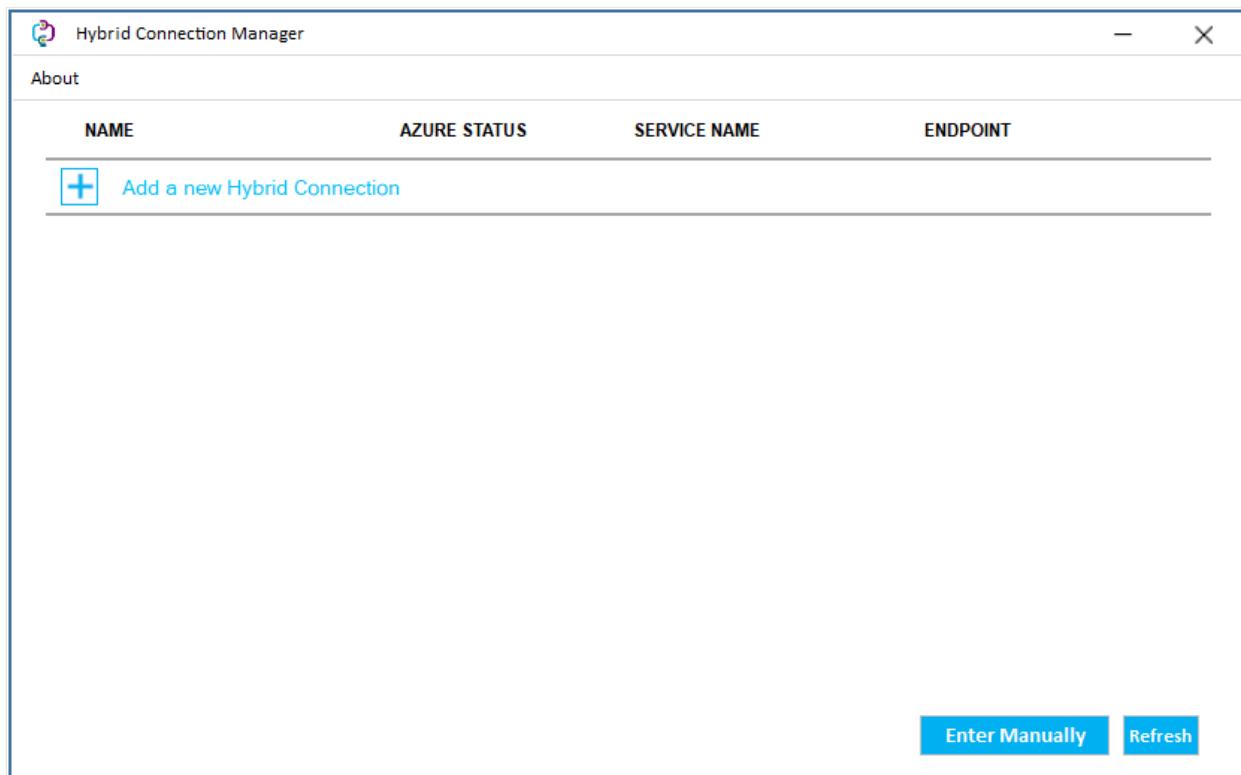
In addition to there being an App Service plan SKU requirement, there's an extra cost to using Hybrid Connections. There's a charge for each listener used by a Hybrid Connection. The listener is the Hybrid Connection Manager. If you had five Hybrid Connections supported by two Hybrid Connection Managers, that would be 10 listeners. For more information, see [Service Bus pricing](#).

Hybrid Connection Manager

The Hybrid Connections feature requires a relay agent in the network that hosts your Hybrid Connection endpoint. That relay agent is called the Hybrid Connection Manager (HCM). To download HCM, from your app in the [Azure portal](#), select **Networking > Configure your Hybrid Connection endpoints**.

This tool runs on Windows Server 2012 and later. The HCM runs as a service and connects outbound to Azure Relay on port 443.

After installing HCM, you can run HybridConnectionManagerUi.exe to use the UI for the tool. This file is in the Hybrid Connection Manager installation directory. In Windows 10, you can also just search for *Hybrid Connection Manager UI* in your search box.



When you start the HCM UI, the first thing you see is a table that lists all the Hybrid Connections that are configured with this instance of the HCM. If you want to make any changes, first authenticate with Azure.

To add one or more Hybrid Connections to your HCM:

1. Start the HCM UI.

2. Select Add a new Hybrid Connection.

NAME	REGION	SERVICE NAME	ENDPOINT
myhybrid	westeurope	mylistener	localhost:80
mysql-hybridconnection	westeurope	hybridcon-sbns	mysqldev:3306

Save Cancel Enter Manually
Subscription: Test Environment Enter Manually Refresh

3. Sign in with your Azure account to get your Hybrid Connections available with your subscriptions. The HCM doesn't continue to use your Azure account beyond this step.

4. Choose a subscription.

5. Select the Hybrid Connections that you want the HCM to relay.

NAME	AZURE STATUS	SERVICE NAME	ENDPOINT
mysql-hybridconnection	Connected	hybridcon-sbns	mysqldev:3306

Add a new Hybrid Connection Enter Manually Refresh

6. Select Save.

You can now see the Hybrid Connections you added. You can also select the configured Hybrid Connection to see details.

The screenshot shows the 'Hybrid Connection Manager' window. At the top, there's a header bar with the title 'Hybrid Connection Manager'. Below it, a table lists hybrid connections with columns: NAME, AZURE STATUS, SERVICE NAME, and ENDPOINT. One row is selected, showing 'mysql_hybridconnection' with 'Connected' status, 'hybridcon_chrc' service name, and 'mysqldev:3306' endpoint. A blue '+' button is at the top left of the table area, labeled 'Add a new Hybrid Connection'. To the right of the table, a modal window titled 'Hybrid Connection Details' displays various configuration settings for the selected connection. These include Name (mysql-hybridconnection), Namespace (hybridcon-sbns), Endpoint (mysqldev:3306), Status (Connected), Service Bus Endpoint (hybridcon-sbns.servicebus.windows.net), Azure IP Address (104.46.32.56), Azure Ports (80, 443), Created On (05-05-2021 10:51:28), and Last Updated (05-05-2021 10:51:38). A 'Close' button is at the bottom right of the modal.

To support the Hybrid Connections it's configured with, HCM requires:

- TCP access to Azure over port 443.
- TCP access to the Hybrid Connection endpoint.
- The ability to do DNS look-ups on the endpoint host and the Service Bus namespace. In other words, the hostname in the Azure relay connection should be resolvable from the machine hosting the HCM.

ⓘ Note

Azure Relay relies on Web Sockets for connectivity. This capability is only available on Windows Server 2012 or later. Because of that, HCM is not supported on anything earlier than Windows Server 2012.

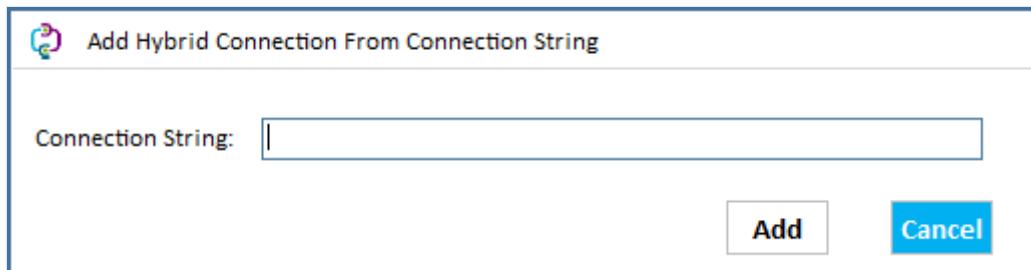
Redundancy

Each HCM can support multiple Hybrid Connections. Also, any given Hybrid Connection can be supported by multiple HCMs. The default behavior is to route traffic across the configured HCMs for any given endpoint. If you want high availability on your Hybrid Connections from your network, run multiple HCMs on separate machines. The load

distribution algorithm used by the Relay service to distribute traffic to the HCMs is random assignment.

Manually add a Hybrid Connection

To enable someone outside your subscription to host an HCM instance for a given Hybrid Connection, share the gateway connection string for the Hybrid Connection with them. You can see the gateway connection string in the Hybrid Connection properties in the [Azure portal](#). To use that string, select **Enter Manually** in the HCM, and paste in the gateway connection string.



Upgrade

There are periodic updates to the Hybrid Connection Manager to fix issues or provide improvements. When upgrades are released, a popup shows up in the HCM UI. Applying the upgrade applies the changes and restarts the HCM.

Adding a Hybrid Connection to your app programmatically

There's Azure CLI support for Hybrid Connections. The commands provided operate at both the app and the App Service plan level. The app level commands are:

```
Azure CLI

az webapp hybrid-connection

Group
  az webapp hybrid-connection : Methods that list, add and remove hybrid-connections from webapps.
    This command group is in preview. It may be changed/removed in a future release.
  Commands:
    add    : Add a hybrid-connection to a webapp.
    list   : List the hybrid-connections on a webapp.
    remove : Remove a hybrid-connection from a webapp.
```

The App Service plan commands enable you to set which key a given hybrid-connection uses. There are two keys set on each Hybrid Connection, a primary and a secondary. You can choose to use the primary or secondary key with the below commands. This option enables you to switch keys for when you want to periodically regenerate your keys.

Azure CLI

```
az appservice hybrid-connection --help

Group
  az appservice hybrid-connection : A method that sets the key a hybrid-
  connection uses.
    This command group is in preview. It may be changed/removed in a
    future release.
  Commands:
    set-key : Set the key that all apps in an appservice plan use to connect
    to the hybrid-
      connections in that appservice plan.
```

Secure your Hybrid Connections

An existing Hybrid Connection can be added to other App Service Web Apps by any user who has sufficient permissions on the underlying Azure Service Bus Relay. This means if you must prevent others from reusing that same Hybrid Connection (for example when the target resource is a service that doesn't have any other security measures in place to prevent unauthorized access), you must lock down access to the Azure Service Bus Relay.

Anyone with `Reader` access to the Relay is able to *see* the Hybrid Connection when attempting to add it to their Web App in the Azure portal, but they can't *add* it as they lack the permissions to retrieve the connection string that is used to establish the relay connection. In order to successfully add the Hybrid Connection, they must have the `listKeys` permission

(`Microsoft.Relay/namespaces/hybridConnections/authorizationRules/listKeys/action`). The `Contributor` role or any other role that includes this permission on the Relay allows users to use the Hybrid Connection and add it to their own Web Apps.

Manage your Hybrid Connections

If you need to change the endpoint host or port for a Hybrid Connection, follow the following steps:

1. Remove the Hybrid Connection from the Hybrid Connection Manager on the local machine by selecting the connection and selecting **Remove** at the top left of the Hybrid Connection Details window.
2. Disconnect the Hybrid Connection from your App Service by navigating to **Hybrid Connections** in the App Service **Networking** page.
3. Navigate to the Relay for the endpoint you need to update and select **Hybrid Connections** under **Entities** in the left-hand navigation menu.
4. Select the Hybrid Connection you want to update and select **Properties** under **Settings** in the left-hand navigation menu.
5. Make your changes and hit **Save changes** at the top.
6. Return to the **Hybrid Connections** settings for your App Service and add the Hybrid Connection again. Ensure the endpoint is updated as intended. If you don't see the Hybrid Connection in the list, refresh in 5-10 minutes.
7. Return to the Hybrid Connection Manager on the local machine and add the connection again.

Troubleshooting

The status of "Connected" means that at least one HCM is configured with that Hybrid Connection, and is able to reach Azure. If the status for your Hybrid Connection doesn't say **Connected**, your Hybrid Connection isn't configured on any HCM that has access to Azure. When your HCM shows **Not Connected**, there are a few things to check:

- Does your host have outbound access to Azure on port 443? You can test from your HCM host using the PowerShell command *Test-NetConnection Destination -Port*
- Is your HCM potentially in a bad state? Try restarting the 'Azure Hybrid Connection Manager Service' local service.
- Do you have conflicting software installed? Hybrid Connection Manager can't coexist with Biztalk Hybrid Connection Manager or Service Bus for Windows Server. When you install the HCM, any versions of these packages should be removed first.
- Do you have a firewall between your HCM host and Azure? If so, you need to allow outbound access to both the Service Bus endpoint URL AND the Service Bus gateways that service your Hybrid Connection.
 - You can find the Service Bus endpoint URL in the Hybrid Connection Manager UI.

The screenshot shows the 'Hybrid Connection Manager' interface. At the top, there's a header with a logo and the title 'Hybrid Connection Manager'. Below it is a table with columns: NAME, AZURE STATUS, SERVICE NAME, and ENDPOINT. A row for 'mysql_hybridconnection' is selected, showing 'Connected' in the Azure Status column. The SERVICE NAME is 'hybridcon_sbns' and the ENDPOINT is 'mysqldev:3306'. Below the table is a modal window titled 'Hybrid Connection Details' for the selected connection. It contains fields for Name (mysql-hybridconnection), Namespace (hybridcon-sbns), Endpoint (mysqldev:3306), and Status (Connected). The 'Service Bus Endpoint' field is highlighted with a red box and contains the value 'hybridcon-sbns.servicebus.windows.net'. Other fields in the modal include Azure IP Address (104.46.32.56), Azure Ports (80, 443), Created On (05-05-2021 10:51:28), and Last Updated (05-05-2021 10:51:38). A 'Close' button is at the bottom right of the modal.

- The Service Bus gateways are the resources that accept the request into the Hybrid Connection and pass it through the Azure Relay. You need to allowlist all 128 of the gateways. The gateways are in the format of *G#-prod-[stamp]-sb.servicebus.windows.net* where "#" is a number between 0 and 127 and "stamp" is the name of the instance within your Azure data center where your Service Bus endpoint exists.
- If you can use a wildcard, you can allowlist "**.servicebus.windows.net*".
- If you can't use a wildcard, you must allowlist all 128 gateways.

You can find out the stamp using "nslookup" on the Service Bus endpoint URL.

```
C:\Users\jamesche>nslookup jimsns.servicebus.windows.net
Server: FIOS_Quantum_Gateway.fios-router.home
Address: 192.168.1.1

Non-authoritative answer:
Name: ns-sb2-prod-sn3-010.cloudapp.net
Address: 40.84.185.67
Aliases: jimsns.servicebus.windows.net
```

In this example, the stamp is "sn3-010". To allowlist the Service Bus gateways, you need the following entries:

G0-prod-sn3-010-sb.servicebus.windows.net
G1-prod-sn3-010-sb.servicebus.windows.net

G2-prod-sn3-010-sb.servicebus.windows.net
G3-prod-sn3-010-sb.servicebus.windows.net
...
G126-prod-sn3-010-sb.servicebus.windows.net
G127-prod-sn3-010-sb.servicebus.windows.net

If your status says **Connected** but your app can't reach your endpoint then:

- Make sure you're using a DNS name in your Hybrid Connection. If you use an IP address, then the required client DNS lookup might not happen. If the client running in your web app doesn't do a DNS lookup, then the Hybrid Connection doesn't work.
- Check that the DNS name used in your Hybrid Connection can resolve from the HCM host. Check the resolution using *nslookup EndpointDNSname* where EndpointDNSname is an exact match to what is used in your Hybrid Connection definition.
- Test access from your HCM host to your endpoint using the PowerShell command *Test-NetConnection EndpointDNSname -P Port* If you can't reach the endpoint from your HCM host then check firewalls between the two hosts including any host-based firewalls on the destination host.
- If you're using App Service on Linux, make sure you're not using "localhost" as your endpoint host. Instead, use your machine name if you're trying to create a connection with a resource on your local machine.

In App Service, the **tcpping** command-line tool can be invoked from the Advanced Tools (Kudu) console. This tool can tell you if you have access to a TCP endpoint, but it doesn't tell you if you have access to a Hybrid Connection endpoint. When you use the tool in the console against a Hybrid Connection endpoint, you're only confirming that it uses a host:port combination.

If you have a command-line client for your endpoint, you can test connectivity from the app console. For example, you can test access to web server endpoints by using curl.

How-to: build a real-time collaborative whiteboard using Azure Web PubSub and deploy it to Azure App Service

Article • 12/11/2024

A new class of applications is reimagining what modern work could be. While [Microsoft Word](#) brings editors together, [Figma](#) gathers up designers on the same creative endeavor. This class of applications builds on a user experience that makes us feel connected with our remote collaborators. From a technical point of view, user's activities need to be synchronized across users' screens at a low latency.

Important

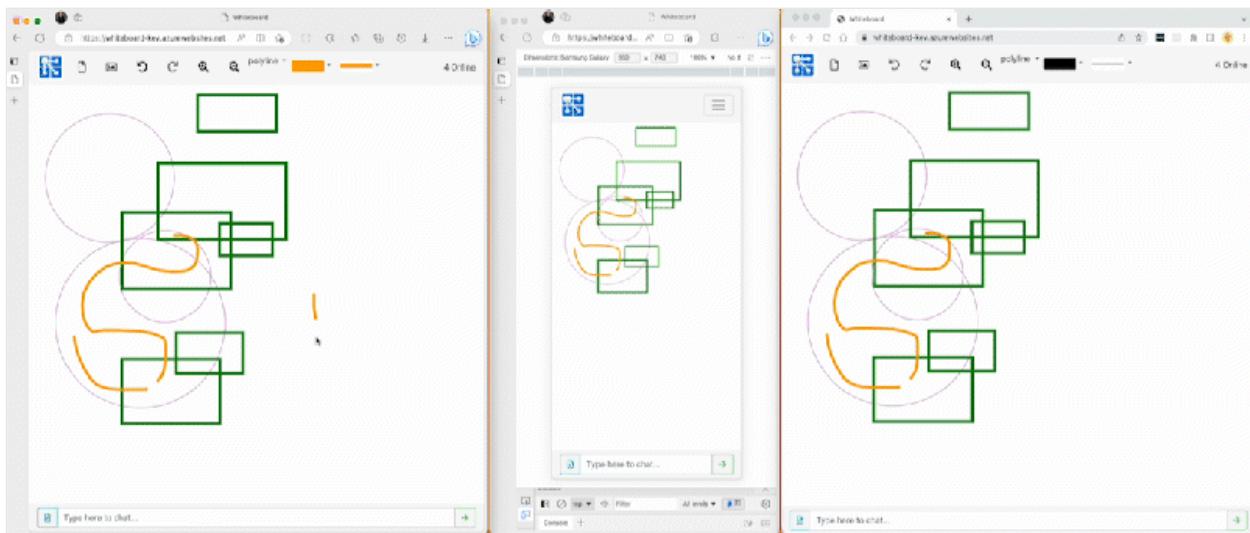
Raw connection strings appear in this article for demonstration purposes only.

A connection string includes the authorization information required for your application to access Azure Web PubSub service. The access key inside the connection string is similar to a root password for your service. In production environments, always protect your access keys. Use Azure Key Vault to manage and rotate your keys securely and [secure your connection with WebPubSubServiceClient](#).

Avoid distributing access keys to other users, hard-coding them, or saving them anywhere in plain text that is accessible to others. Rotate your keys if you believe they may have been compromised.

Overview

In this how-to guide, we take a cloud-native approach and use Azure services to build a real-time collaborative whiteboard and we deploy the project as a Web App to Azure App Service. The whiteboard app is accessible in the browser and allows anyone can draw on the same canvas.

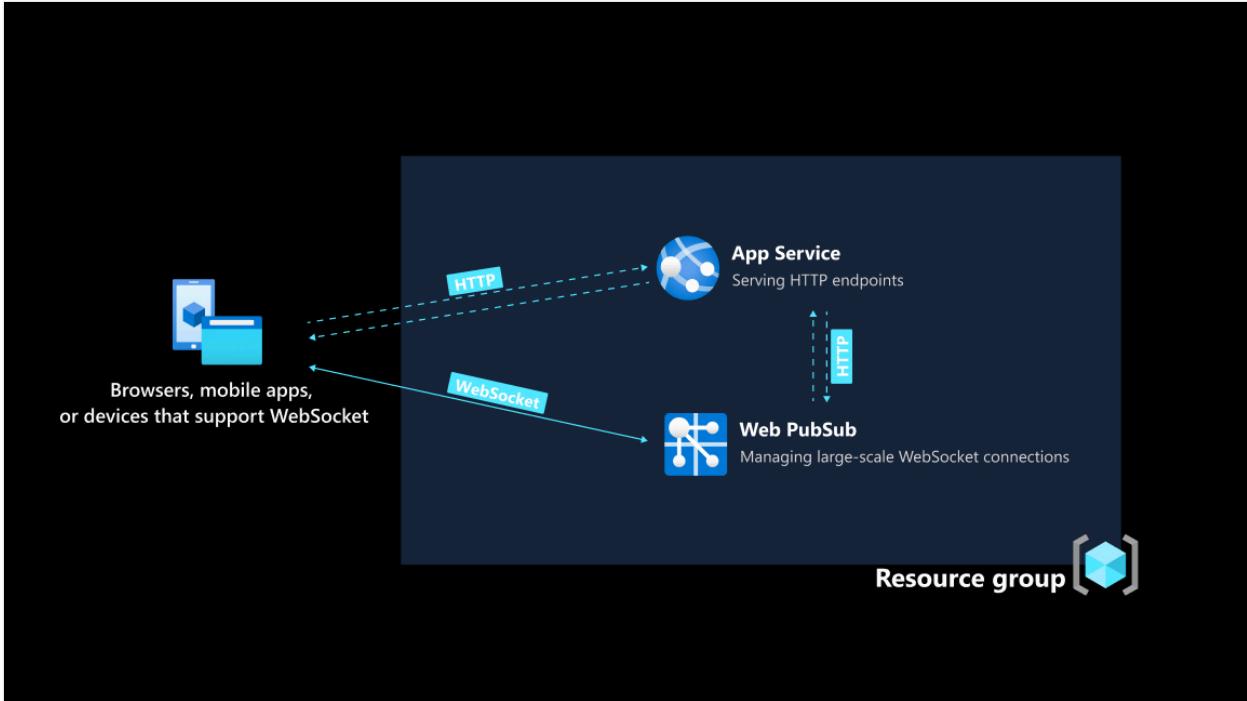


[Check out live whiteboard demo](#)

Architecture

[Expand table](#)

Azure service name	Purpose	Benefits
Azure App Service	Provides the hosting environment for the backend application, which is built with Express ↗	Fully managed environment for application backends, with no need to worry about infrastructure where the code runs
Azure Web PubSub	Provides low-latency, bi-directional data exchange channel between the backend application and clients	Drastically reduces server load by freeing server from managing persistent WebSocket connections and scales to 100 K concurrent client connections with just one resource



Prerequisites

You can find detailed explanation of the [data flow](#) at the end of this how-to guide as we're going to focus on building and deploying the whiteboard app first.

In order to follow the step-by-step guide, you need

- ✓ An [Azure](#) account. If you don't have an Azure subscription, create an [Azure free account](#) before you begin.
- ✓ [Azure CLI](#) (version 2.29.0 or higher) or [Azure Cloud Shell](#) to manage Azure resources.

Create Azure resources using Azure CLI

1. Sign in

1. Sign in to Azure CLI by running the following command.

```
Azure CLI
```

```
az login
```

2. Create a resource group on Azure.

```
Azure CLI
```

```
az group create \
--location "westus" \
--name "whiteboard-group"
```

2. Create a Web App resource

1. Create a free App Service plan.

Azure CLI

```
az appservice plan create \
--resource-group "whiteboard-group" \
--name "demo" \
--sku FREE
--is-linux
```

2. Create a Web App resource

Azure CLI

```
az webapp create \
--resource-group "whiteboard-group" \
--name "whiteboard-app" \
--plan "demo" \
--runtime "NODE:18-lts"
```

3. Create a Web PubSub resource

1. Create a Web PubSub resource.

Azure CLI

```
az webpubsub create \
--name "whiteboard-app" \
--resource-group "whiteboard-group" \
--location "westus" \
--sku Free_F1
```

2. Show and store the value of `primaryConnectionString` somewhere for later use.

Raw connection strings appear in this article for demonstration purposes only. In production environments, always protect your access keys. Use Azure Key Vault to manage and rotate your keys securely and [secure your connection with WebPubSubServiceClient](#).

Azure CLI

```
az webpubsub key show \
--name "whiteboard-app" \
--resource-group "whiteboard-group"
```

Get the application code

Run the following command to get a copy of the application code. You can find detailed explanation of the [data flow](#) at the end of this how-to guide.

Bash

```
git clone https://github.com/Azure/awps-webapp-sample.git
```

Deploy the application to App Service

1. App Service supports many deployment workflows. For this guide, we're going to deploy a ZIP package. Run the following commands to prepare the ZIP.

Bash

```
npm install
npm run build
zip -r app.zip *
```

2. Use the following command to deploy it to Azure App Service.

Azure CLI

```
az webapp deployment source config-zip \
--resource-group "whiteboard-group" \
--name "whiteboard-app" \
--src app.zip
```

3. Set Azure Web PubSub connection string in the application settings. Use the value of `primaryConnectionString` you stored from an earlier step.

Azure CLI

```
az webapp config appsettings set \
--resource-group "whiteboard-group" \
```

```
--name "whiteboard-app" \
--setting Web_PubSub_ConnectionString=""
```

Configure upstream server to handle events coming from Web PubSub

Whenever a client sends a message to Web PubSub service, the service sends an HTTP request to an endpoint you specify. This mechanism is what your backend server uses to further process messages, for example, if you can persist messages in a database of choice.

As is with HTTP requests, Web PubSub service needs to know where to locate your application server. Since the backend application is now deployed to App Service, we get a publicly accessible domain name for it.

1. Show and store the value of `name` somewhere.

Azure CLI

```
az webapp config hostname list \
--resource-group "whiteboard-group"
--webapp-name "whiteboard-app"
```

2. The endpoint we decided to expose on the backend server is `/eventhandler` and the `hub` name for whiteboard app `"sample_draw"`

Azure CLI

```
az webpubsub hub create \
--resource-group "whiteboard-group" \
--name "whiteboard-app" \
--hub-name "sample_draw" \
--event-handler url-template="https://<Replace with the hostname of
your Web App resource>/eventhandler" user-event-pattern="*" system-
event="connected" system-event="disconnected"
```

ⓘ Important

`url-template` has three parts: protocol + hostname + path, which in our case is `https://<The hostname of your Web App resource>/eventhandler`.

View the whiteboard app in a browser

Now head over to your browser and visit your deployed Web App. It's recommended to have multiple browser tabs open so that you can experience the real-time collaborative aspect of the app. Or better, share the link with a colleague or friend.

Data flow

Overview

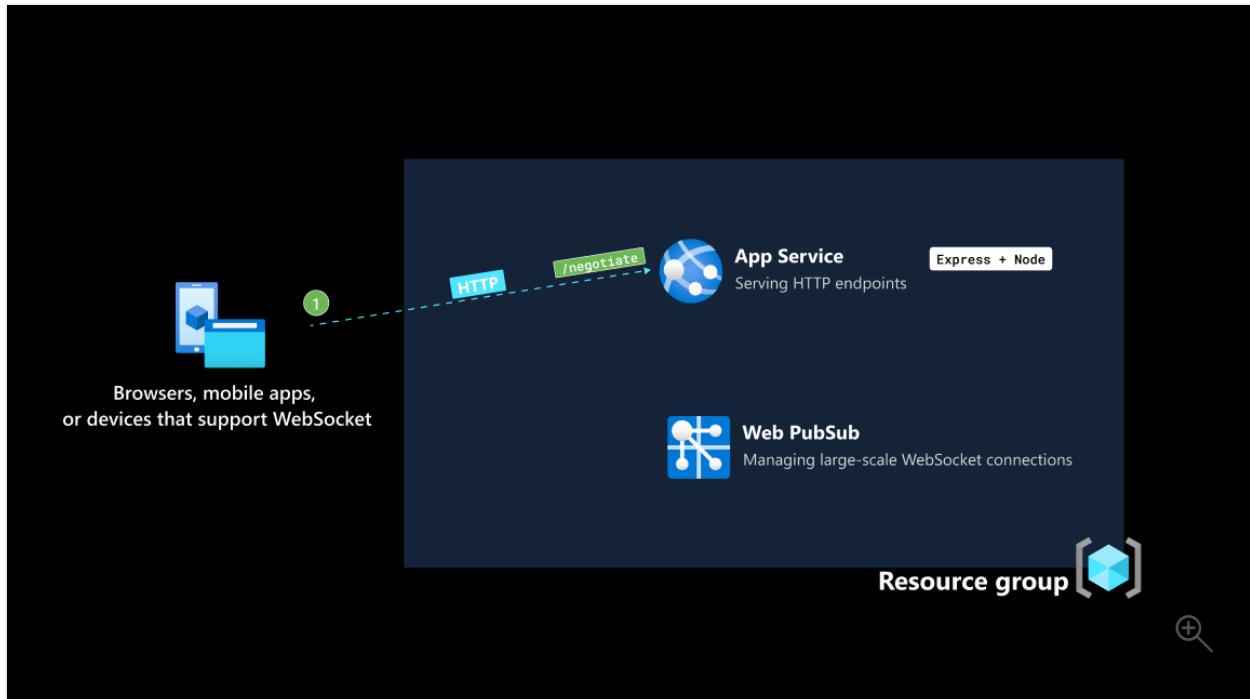
The data flow section dives deeper into how the whiteboard app is built. The whiteboard app has two transport methods.

- HTTP service written as an Express app and hosted on App Service.
- WebSocket connections managed by Azure Web PubSub.

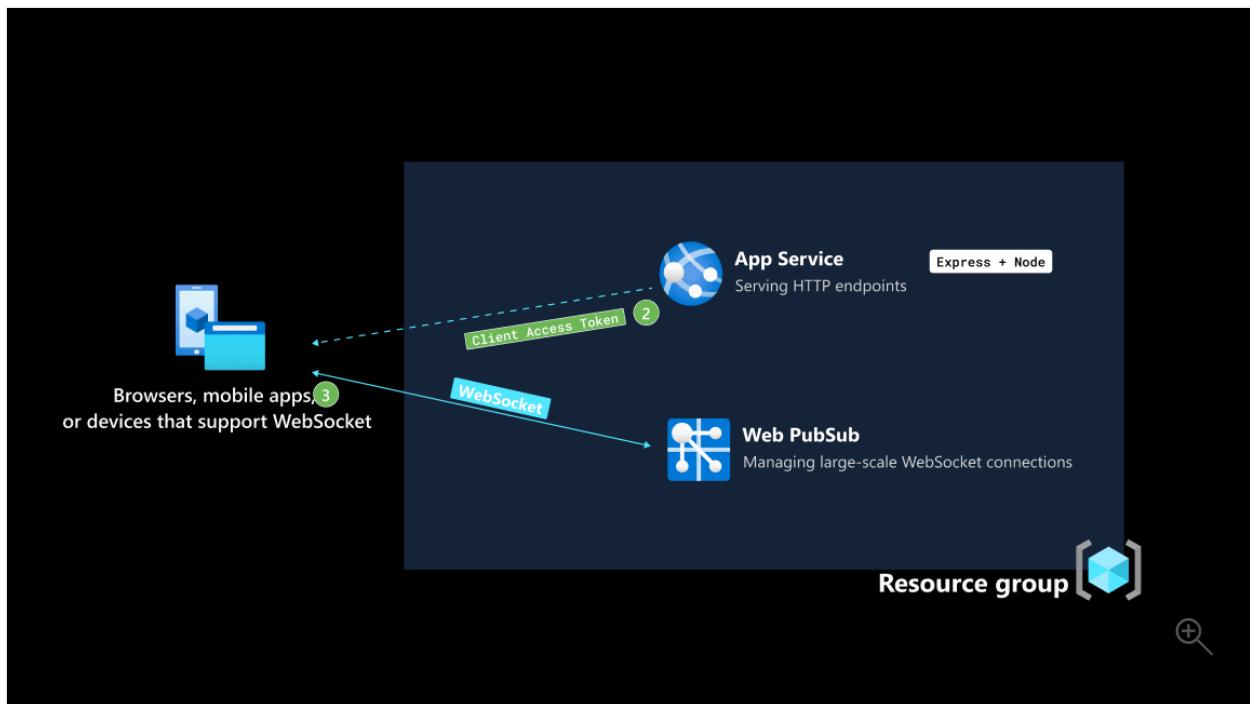
By using Azure Web PubSub to manage WebSocket connections, the load on the Web App is reduced. Apart from authenticating the client and serving images, the Web App isn't involved synchronizing drawing activities. A client's drawing activities are directly sent to Web PubSub and broadcasted to all clients in a group.

At any point in time, there maybe more than one client drawing. If the Web App were to manage WebSocket connections on its own, it needed to broadcast every drawing activity to all other clients. The huge traffic and processing are a large burden to the server.

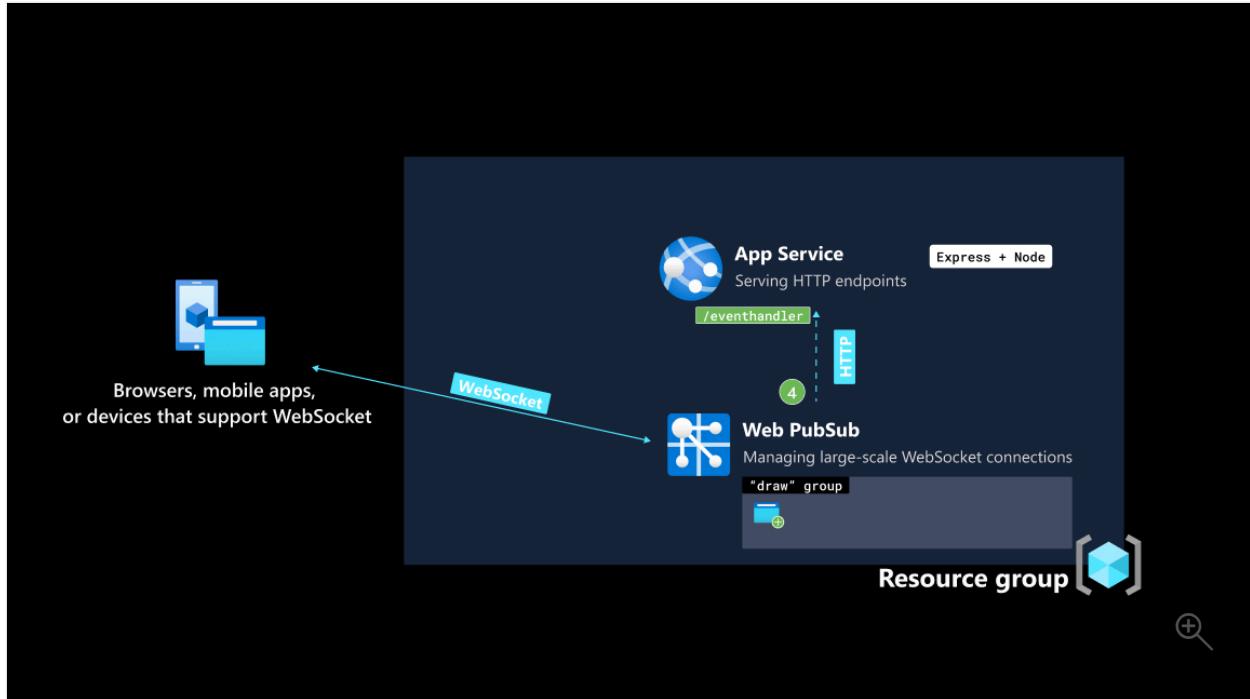
The client, built with [Vue](#), makes an HTTP request for a Client Access Token to an endpoint `/negotiate`. The backend application is an [Express app](#) and hosted as a Web App using Azure App Service.



When the backend application successfully [returns the Client Access Token](#) to the connecting client, the client uses it to establish a WebSocket connection with Azure Web PubSub.



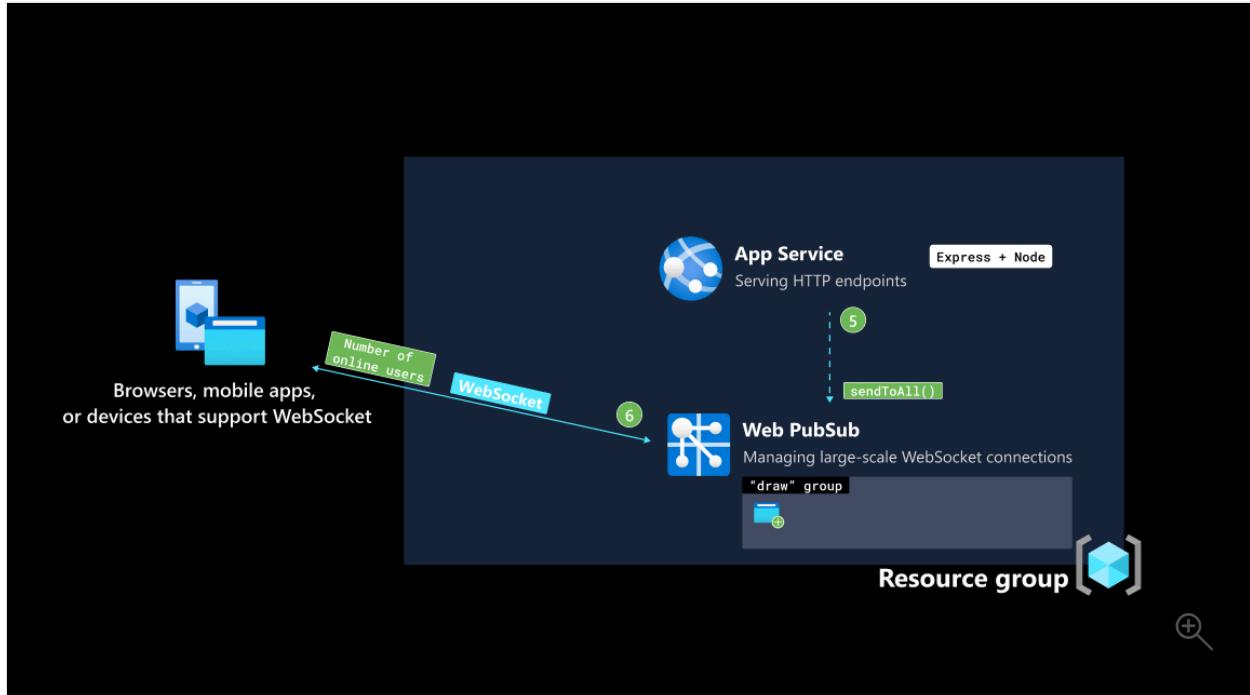
If the handshake with Azure Web PubSub is successful, the client is added to a group named `draw`, effectively subscribing to messages published to this group. Also, the client is given the permission to send messages to the [draw group](#).



ⓘ Note

To keep this how-to guide focused, all connecting clients are added to the same group named `draw` and is given the permission to send messages to this group. To manage client connections at a granular level, see the full references of the APIs provided by Azure Web PubSub.

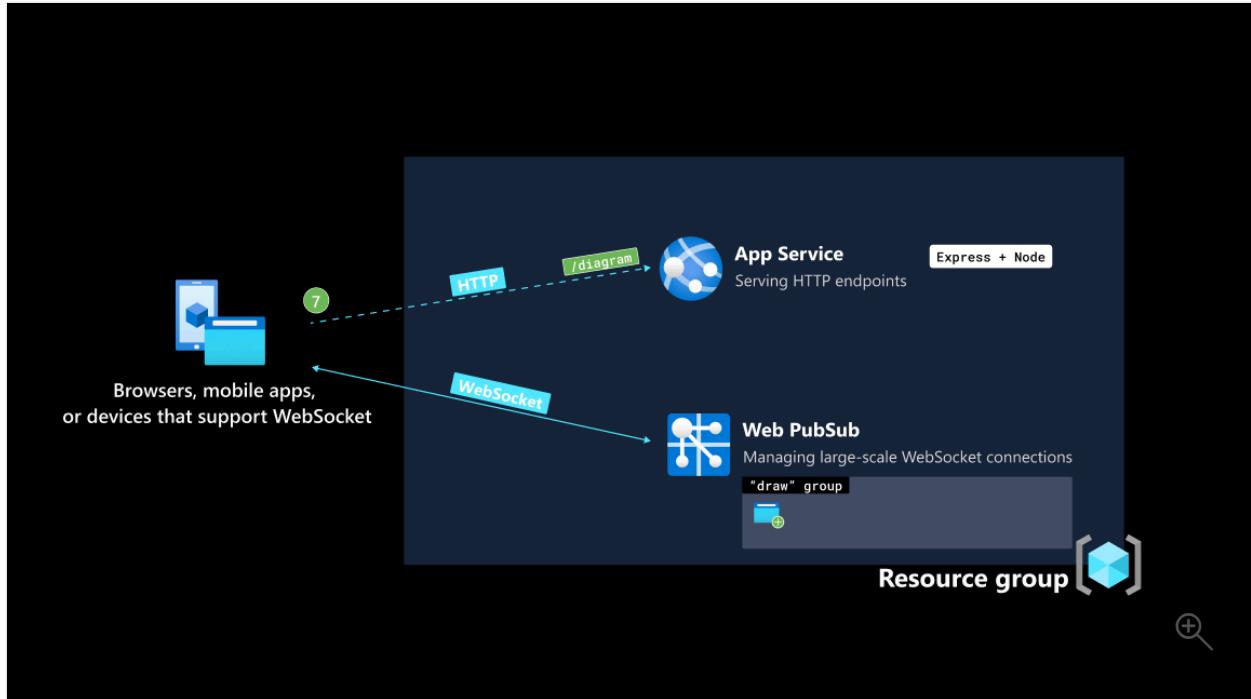
Azure Web PubSub notifies the backend application that a client has connected. The backend application handles the `onConnected` event by calling the `sendToAll()`, with a payload of the latest number of connected clients.



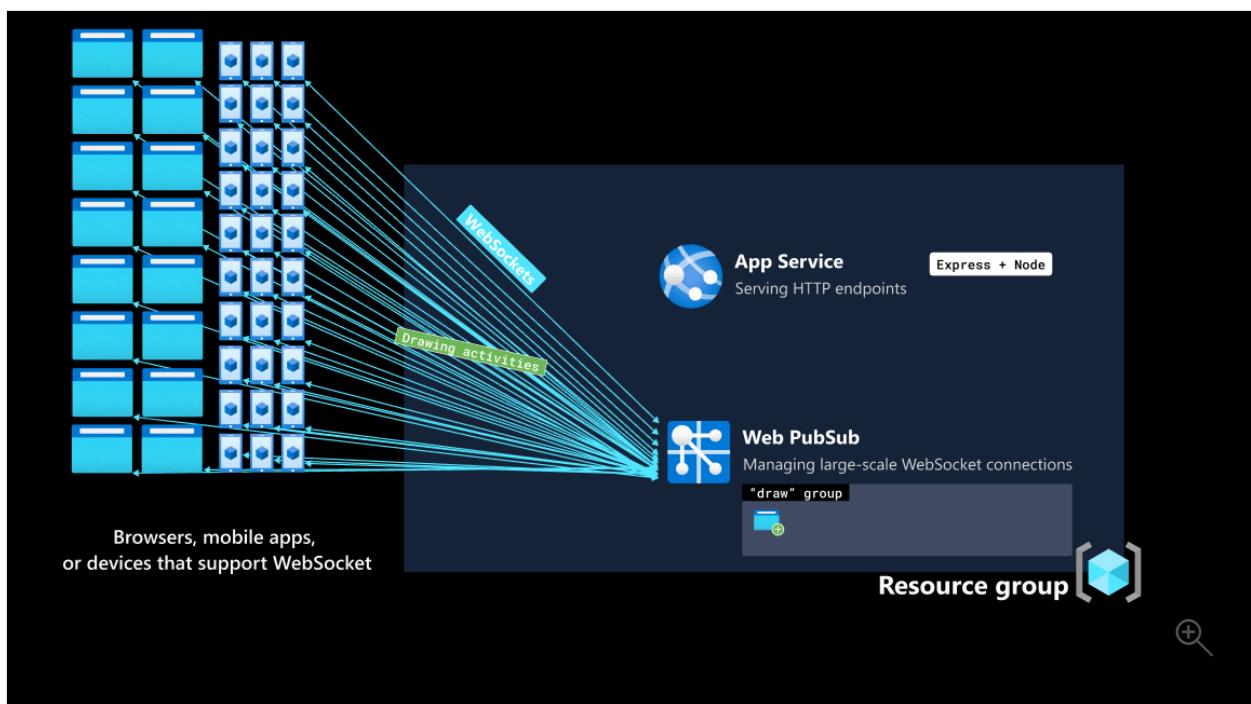
① Note

It is important to note that if there are a large number of online users in the `draw` group, with a **single** network call from the backend application, all the online users will be notified that a new user has just joined. This drastically reduces the complexity and load of the backend application.

As soon as a client establishes a persistent connection with Web PubSub, it makes an HTTP request to the backend application to fetch the latest shape and background data at [/diagram](#). An HTTP service hosted on App Service can be combined with Web PubSub. App Service takes care serving HTTP endpoints, while Web PubSub takes care of managing WebSocket connections.



Now that the clients and backend application have two ways to exchange data. One is the conventional HTTP request-response cycle and the other is the persistent, bi-directional channel through Web PubSub. The drawing actions, which originate from one user and need to be broadcasted to all users as soon as it takes place, are delivered through Web PubSub. It doesn't require involvement of the backend application.



Clean up resources

Although the application uses only the free tiers of both services, it's best practice to delete resources if you no longer need them. You can delete the resource group along

with the resources in it using following command,

Azure CLI

```
az group delete  
--name "whiteboard-group"
```

Next steps

[Check out more demos built with Web PubSub](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

Azure Policy built-in definitions for Azure App Service

Article • 10/22/2024

This page is an index of [Azure Policy](#) built-in policy definitions for Azure App Service. For additional Azure Policy built-ins for other services, see [Azure Policy built-in definitions](#).

The name of each built-in policy definition links to the policy definition in the Azure portal. Use the link in the **Version** column to view the source on the [Azure Policy GitHub repo](#).

Azure App Service

 Expand table

Name (Azure portal)	Description	Effect(s)	Version (GitHub)
[Preview]: App Service Plans should be Zone Redundant	App Service Plans can be configured to be Zone Redundant or not. When the 'zoneRedundant' property is set to 'false' for an App Service Plan, it is not configured for Zone Redundancy. This policy identifies and enforces the Zone Redundancy configuration for App Service Plans.	Audit, Deny, Disabled	1.0.0-preview
App Service app slots should be injected into a virtual network	Injecting App Service Apps in a virtual network unlocks advanced App Service networking and security features and provides you with greater control over your network security configuration. Learn more at: https://docs.microsoft.com/azure/app-service/web-sites-integrate-with-vnet .	Audit, Deny, Disabled	1.0.0
App Service app slots should disable public network access	Disabling public network access improves security by ensuring that the App Service is not exposed on the public internet. Creating private endpoints can limit exposure of an App Service. Learn more at: https://aka.ms/app-service-private-endpoint .	Audit, Disabled, Deny	1.0.0
App Service app slots should enable configuration routing to Azure Virtual Network	By default, app configuration such as pulling container images and mounting content storage will not be routed through the regional virtual network integration. Using the API to set routing options to true enables configuration traffic through the Azure Virtual Network. These settings allow features like network security groups and user defined routes to be used, and service endpoints to be private. For more information, visit this article .	Audit, Deny, Disabled	1.0.0

Name (Azure portal)	Description	Effect(s)	Version (GitHub)
	https://aka.ms/appservice-vnet-configuration-routing .		
App Service app slots should enable outbound non-RFC 1918 traffic to Azure Virtual Network	By default, if one uses regional Azure Virtual Network (VNET) integration, the app only routes RFC1918 traffic into that respective virtual network. Using the API to set 'vnetRouteAllEnabled' to true enables all outbound traffic into the Azure Virtual Network. This setting allows features like network security groups and user defined routes to be used for all outbound traffic from the App Service app.	Audit, Deny, Disabled	1.0.0 ↗
App Service app slots should have Client Certificates (Incoming client certificates) enabled	Client certificates allow for the app to request a certificate for incoming requests. Only clients that have a valid certificate will be able to reach the app. This policy applies to apps with Http version set to 1.1.	AuditIfNotExists, Disabled	1.0.0 ↗
App Service app slots should have local authentication methods disabled for FTP deployments	Disabling local authentication methods for FTP deployments improves security by ensuring that App Service slots exclusively require Microsoft Entra identities for authentication. Learn more at: https://aka.ms/app-service-disable-basic-auth .	AuditIfNotExists, Disabled	1.0.3 ↗
App Service app slots should have local authentication methods disabled for SCM site deployments	Disabling local authentication methods for SCM sites improves security by ensuring that App Service slots exclusively require Microsoft Entra identities for authentication. Learn more at: https://aka.ms/app-service-disable-basic-auth .	AuditIfNotExists, Disabled	1.0.4 ↗
App Service app slots should have remote debugging turned off	Remote debugging requires inbound ports to be opened on an App Service app. Remote debugging should be turned off.	AuditIfNotExists, Disabled	1.0.1 ↗
App Service app slots should have resource logs enabled	Audit enabling of resource logs on the app. This enables you to recreate activity trails for investigation purposes if a security incident occurs or your network is compromised.	AuditIfNotExists, Disabled	1.0.0 ↗
App Service app slots should not have CORS configured to allow every resource to access your apps	Cross-Origin Resource Sharing (CORS) should not allow all domains to access your app. Allow only required domains to interact with your app.	AuditIfNotExists, Disabled	1.0.0 ↗
App Service app slots should only be accessible over HTTPS	Use of HTTPS ensures server/service authentication and protects data in transit from network layer eavesdropping attacks.	Audit, Disabled, Deny	2.0.0 ↗
App Service app slots should require FTPS only	Enable FTPS enforcement for enhanced security.	AuditIfNotExists, Disabled	1.0.0 ↗

Name (Azure portal)	Description	Effect(s)	Version (GitHub)
App Service app slots should use an Azure file share for its content directory	The content directory of an app should be located on an Azure file share. The storage account information for the file share must be provided before any publishing activity. To learn more about using Azure Files for hosting app service content refer to https://go.microsoft.com/fwlink/?linkid=2151594 .	Audit, Disabled	1.0.0
App Service app slots should use latest 'HTTP Version'	Periodically, newer versions are released for HTTP either due to security flaws or to include additional functionality. Using the latest HTTP version for web apps to take advantage of security fixes, if any, and/or new functionalities of the newer version.	AuditIfNotExists, Disabled	1.0.0
App Service app slots should use managed identity	Use a managed identity for enhanced authentication security	AuditIfNotExists, Disabled	1.0.0
App Service app slots should use the latest TLS version	Periodically, newer versions are released for TLS either due to security flaws, include additional functionality, and enhance speed. Upgrade to the latest TLS version for App Service apps to take advantage of security fixes, if any, and/or new functionalities of the latest version.	AuditIfNotExists, Disabled	1.1.0
App Service app slots that use Java should use a specified 'Java version'	Periodically, newer versions are released for Java software either due to security flaws or to include additional functionality. Using the latest Java version for App Service apps is recommended in order to take advantage of security fixes, if any, and/or new functionalities of the latest version. This policy only applies to Linux apps. This policy requires you to specify a Java version that meets your requirements.	AuditIfNotExists, Disabled	1.0.0
App Service app slots that use PHP should use a specified 'PHP version'	Periodically, newer versions are released for PHP software either due to security flaws or to include additional functionality. Using the latest PHP version for App Service apps is recommended in order to take advantage of security fixes, if any, and/or new functionalities of the latest version. This policy only applies to Linux apps. This policy requires you to specify a PHP version that meets your requirements.	AuditIfNotExists, Disabled	1.0.0
App Service app slots that use Python should use a specified 'Python version'	Periodically, newer versions are released for Python software either due to security flaws or to include additional functionality. Using the latest Python version for App Service apps is recommended in order to take advantage of security fixes, if any, and/or new functionalities of the latest version. This policy only applies to	AuditIfNotExists, Disabled	1.0.0

Name (Azure portal)	Description	Effect(s)	Version (GitHub)
	Linux apps. This policy requires you to specify a Python version that meets your requirements.		
App Service apps should be injected into a virtual network ↗	Injecting App Service Apps in a virtual network unlocks advanced App Service networking and security features and provides you with greater control over your network security configuration. Learn more at: https://docs.microsoft.com/azure/app-service/web-sites-integrate-with-vnet .	Audit, Deny, Disabled	3.0.0 ↗
App Service apps should disable public network access ↗	Disabling public network access improves security by ensuring that the App Service is not exposed on the public internet. Creating private endpoints can limit exposure of an App Service. Learn more at: https://aka.ms/app-service-private-endpoint .	Audit, Disabled, Deny	1.1.0 ↗
App Service apps should enable configuration routing to Azure Virtual Network ↗	By default, app configuration such as pulling container images and mounting content storage will not be routed through the regional virtual network integration. Using the API to set routing options to true enables configuration traffic through the Azure Virtual Network. These settings allow features like network security groups and user defined routes to be used, and service endpoints to be private. For more information, visit https://aka.ms/appservice-vnet-configuration-routing .	Audit, Deny, Disabled	1.0.0 ↗
App Service apps should enable outbound non-RFC 1918 traffic to Azure Virtual Network ↗	By default, if one uses regional Azure Virtual Network (VNET) integration, the app only routes RFC1918 traffic into that respective virtual network. Using the API to set 'vnetRouteAllEnabled' to true enables all outbound traffic into the Azure Virtual Network. This setting allows features like network security groups and user defined routes to be used for all outbound traffic from the App Service app.	Audit, Deny, Disabled	1.0.0 ↗
App Service apps should have authentication enabled ↗	Azure App Service Authentication is a feature that can prevent anonymous HTTP requests from reaching the web app, or authenticate those that have tokens before they reach the web app.	AuditIfNotExists, Disabled	2.0.1 ↗
App Service apps should have Client Certificates (Incoming client certificates) enabled ↗	Client certificates allow for the app to request a certificate for incoming requests. Only clients that have a valid certificate will be able to reach the app. This policy applies to apps with Http version set to 1.1.	AuditIfNotExists, Disabled	1.0.0 ↗

Name (Azure portal)	Description	Effect(s)	Version (GitHub)
App Service apps should have local authentication methods disabled for FTP deployments ↗	Disabling local authentication methods for FTP deployments improves security by ensuring that App Services exclusively require Microsoft Entra identities for authentication. Learn more at: https://aka.ms/app-service-disable-basic-auth ↗.	AuditIfNotExists, Disabled	1.0.3 ↗
App Service apps should have local authentication methods disabled for SCM site deployments ↗	Disabling local authentication methods for SCM sites improves security by ensuring that App Services exclusively require Microsoft Entra identities for authentication. Learn more at: https://aka.ms/app-service-disable-basic-auth ↗.	AuditIfNotExists, Disabled	1.0.3 ↗
App Service apps should have remote debugging turned off ↗	Remote debugging requires inbound ports to be opened on an App Service app. Remote debugging should be turned off.	AuditIfNotExists, Disabled	2.0.0 ↗
App Service apps should have resource logs enabled ↗	Audit enabling of resource logs on the app. This enables you to recreate activity trails for investigation purposes if a security incident occurs or your network is compromised.	AuditIfNotExists, Disabled	2.0.1 ↗
App Service apps should not have CORS configured to allow every resource to access your apps ↗	Cross-Origin Resource Sharing (CORS) should not allow all domains to access your app. Allow only required domains to interact with your app.	AuditIfNotExists, Disabled	2.0.0 ↗
App Service apps should only be accessible over HTTPS ↗	Use of HTTPS ensures server/service authentication and protects data in transit from network layer eavesdropping attacks.	Audit, Disabled, Deny	4.0.0 ↗
App Service apps should require FTPS only ↗	Enable FTPS enforcement for enhanced security.	AuditIfNotExists, Disabled	3.0.0 ↗
App Service apps should use a SKU that supports private link ↗	With supported SKUs, Azure Private Link lets you connect your virtual network to Azure services without a public IP address at the source or destination. The Private Link platform handles the connectivity between the consumer and services over the Azure backbone network. By mapping private endpoints to apps, you can reduce data leakage risks. Learn more about private links at: https://aka.ms/private-link ↗.	Audit, Deny, Disabled	4.1.0 ↗
App Service apps should use a virtual network service endpoint ↗	Use virtual network service endpoints to restrict access to your app from selected subnets from an Azure virtual network. To learn more about App Service service endpoints, visit https://aka.ms/appservice-vnet-service-endpoint ↗.	AuditIfNotExists, Disabled	2.0.1 ↗

Name (Azure portal)	Description	Effect(s)	Version (GitHub)
App Service apps should use an Azure file share for its content directory ↗	The content directory of an app should be located on an Azure file share. The storage account information for the file share must be provided before any publishing activity. To learn more about using Azure Files for hosting app service content refer to https://go.microsoft.com/fwlink/?linkid=2151594 .	Audit, Disabled	3.0.0 ↗
App Service apps should use latest 'HTTP Version' ↗	Periodically, newer versions are released for HTTP either due to security flaws or to include additional functionality. Using the latest HTTP version for web apps to take advantage of security fixes, if any, and/or new functionalities of the newer version.	AuditIfNotExists, Disabled	4.0.0 ↗
App Service apps should use managed identity ↗	Use a managed identity for enhanced authentication security	AuditIfNotExists, Disabled	3.0.0 ↗
App Service apps should use private link ↗	Azure Private Link lets you connect your virtual networks to Azure services without a public IP address at the source or destination. The Private Link platform handles the connectivity between the consumer and services over the Azure backbone network. By mapping private endpoints to App Service, you can reduce data leakage risks. Learn more about private links at: https://aka.ms/private-link .	AuditIfNotExists, Disabled	1.0.1 ↗
App Service apps should use the latest TLS version ↗	Periodically, newer versions are released for TLS either due to security flaws, include additional functionality, and enhance speed. Upgrade to the latest TLS version for App Service apps to take advantage of security fixes, if any, and/or new functionalities of the latest version.	AuditIfNotExists, Disabled	2.1.0 ↗
App Service apps that use Java should use a specified 'Java version' ↗	Periodically, newer versions are released for Java software either due to security flaws or to include additional functionality. Using the latest Java version for App Service apps is recommended in order to take advantage of security fixes, if any, and/or new functionalities of the latest version. This policy only applies to Linux apps. This policy requires you to specify a Java version that meets your requirements.	AuditIfNotExists, Disabled	3.1.0 ↗
App Service apps that use PHP should use a specified 'PHP version' ↗	Periodically, newer versions are released for PHP software either due to security flaws or to include additional functionality. Using the latest PHP version for App Service apps is recommended in order to take advantage of security fixes, if any, and/or new functionalities of the latest version. This policy only applies to	AuditIfNotExists, Disabled	3.2.0 ↗

Name (Azure portal)	Description	Effect(s)	Version (GitHub)
	Linux apps. This policy requires you to specify a PHP version that meets your requirements.		
App Service apps that use Python should use a specified 'Python version' ↗	Periodically, newer versions are released for Python software either due to security flaws or to include additional functionality. Using the latest Python version for App Service apps is recommended in order to take advantage of security fixes, if any, and/or new functionalities of the latest version. This policy only applies to Linux apps. This policy requires you to specify a Python version that meets your requirements.	AuditIfNotExists, Disabled	4.1.0 ↗
App Service Environment apps should not be reachable over public internet ↗	To ensure apps deployed in an App Service Environment are not accessible over public internet, one should deploy App Service Environment with an IP address in virtual network. To set the IP address to a virtual network IP, the App Service Environment must be deployed with an internal load balancer.	Audit, Deny, Disabled	3.0.0 ↗
App Service Environment should be configured with strongest TLS Cipher suites ↗	The two most minimal and strongest cipher suites required for App Service Environment to function correctly are : TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384 and TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256.	Audit, Disabled	1.0.0 ↗
App Service Environment should be provisioned with latest versions ↗	Only allow App Service Environment version 2 or version 3 to be provisioned. Older versions of App Service Environment require manual management of Azure resources and have greater scaling limitations.	Audit, Deny, Disabled	1.0.0 ↗
App Service Environment should have internal encryption enabled ↗	Setting InternalEncryption to true encrypts the pagefile, worker disks, and internal network traffic between the front ends and workers in an App Service Environment. To learn more, refer to https://docs.microsoft.com/azure/app-service/environment/app-service-app-service-environment-custom-settings#enable-internal-encryption .	Audit, Disabled	1.0.1 ↗
App Service Environment should have TLS 1.0 and 1.1 disabled ↗	TLS 1.0 and 1.1 are out-of-date protocols that do not support modern cryptographic algorithms. Disabling inbound TLS 1.0 and 1.1 traffic helps secure apps in an App Service Environment.	Audit, Deny, Disabled	2.0.1 ↗
Configure App Service app slots to disable local authentication for FTP deployments ↗	Disabling local authentication methods for FTP deployments improves security by ensuring that App Service slots exclusively require Microsoft Entra identities for authentication.	DeployIfNotExists, Disabled	1.0.3 ↗

Name (Azure portal)	Description	Effect(s)	Version (GitHub)
	Learn more at: https://aka.ms/app-service-disable-basic-auth .		
Configure App Service app slots to disable local authentication for SCM sites ↗	Disabling local authentication methods for SCM sites improves security by ensuring that App Service slots exclusively require Microsoft Entra identities for authentication. Learn more at: https://aka.ms/app-service-disable-basic-auth .	DeployIfNotExists, Disabled	1.0.3 ↗
Configure App Service app slots to disable public network access ↗	Disable public network access for your App Services so that it is not accessible over the public internet. This can reduce data leakage risks. Learn more at: https://aka.ms/app-service-private-endpoint .	Modify, Disabled	1.1.0 ↗
Configure App Service app slots to only be accessible over HTTPS ↗	Use of HTTPS ensures server/service authentication and protects data in transit from network layer eavesdropping attacks.	Modify, Disabled	2.0.0 ↗
Configure App Service app slots to turn off remote debugging ↗	Remote debugging requires inbound ports to be opened on an App Service app. Remote debugging should be turned off.	DeployIfNotExists, Disabled	1.1.0 ↗
Configure App Service app slots to use the latest TLS version ↗	Periodically, newer versions are released for TLS either due to security flaws, include additional functionality, and enhance speed. Upgrade to the latest TLS version for App Service apps to take advantage of security fixes, if any, and/or new functionalities of the latest version.	DeployIfNotExists, Disabled	1.2.0 ↗
Configure App Service apps to disable local authentication for FTP deployments ↗	Disabling local authentication methods for FTP deployments improves security by ensuring that App Services exclusively require Microsoft Entra identities for authentication. Learn more at: https://aka.ms/app-service-disable-basic-auth .	DeployIfNotExists, Disabled	1.0.3 ↗
Configure App Service apps to disable local authentication for SCM sites ↗	Disabling local authentication methods for SCM sites improves security by ensuring that App Services exclusively require Microsoft Entra identities for authentication. Learn more at: https://aka.ms/app-service-disable-basic-auth .	DeployIfNotExists, Disabled	1.0.3 ↗
Configure App Service apps to disable public network access ↗	Disable public network access for your App Services so that it is not accessible over the public internet. This can reduce data leakage risks. Learn more at: https://aka.ms/app-service-private-endpoint .	Modify, Disabled	1.1.0 ↗
Configure App Service apps to only be accessible over HTTPS ↗	Use of HTTPS ensures server/service authentication and protects data in transit from	Modify, Disabled	2.0.0 ↗

Name (Azure portal)	Description	Effect(s)	Version (GitHub)
	network layer eavesdropping attacks.		
Configure App Service apps to turn off remote debugging ↗	Remote debugging requires inbound ports to be opened on an App Service app. Remote debugging should be turned off.	DeployIfExists, Disabled	1.0.0 ↗
Configure App Service apps to use the latest TLS version ↗	Periodically, newer versions are released for TLS either due to security flaws, include additional functionality, and enhance speed. Upgrade to the latest TLS version for App Service apps to take advantage of security fixes, if any, and/or new functionalities of the latest version.	DeployIfExists, Disabled	1.1.0 ↗
Configure Function app slots to disable public network access ↗	Disable public network access for your Function apps so that it is not accessible over the public internet. This can reduce data leakage risks. Learn more at: https://aka.ms/app-service-private-endpoint .	Modify, Disabled	1.1.0 ↗
Configure Function app slots to only be accessible over HTTPS ↗	Use of HTTPS ensures server/service authentication and protects data in transit from network layer eavesdropping attacks.	Modify, Disabled	2.0.0 ↗
Configure Function app slots to turn off remote debugging ↗	Remote debugging requires inbound ports to be opened on a Function app. Remote debugging should be turned off.	DeployIfExists, Disabled	1.1.0 ↗
Configure Function app slots to use the latest TLS version ↗	Periodically, newer versions are released for TLS either due to security flaws, include additional functionality, and enhance speed. Upgrade to the latest TLS version for Function apps to take advantage of security fixes, if any, and/or new functionalities of the latest version.	DeployIfExists, Disabled	1.2.0 ↗
Configure Function apps to disable public network access ↗	Disable public network access for your Function apps so that it is not accessible over the public internet. This can reduce data leakage risks. Learn more at: https://aka.ms/app-service-private-endpoint .	Modify, Disabled	1.1.0 ↗
Configure Function apps to only be accessible over HTTPS ↗	Use of HTTPS ensures server/service authentication and protects data in transit from network layer eavesdropping attacks.	Modify, Disabled	2.0.0 ↗
Configure Function apps to turn off remote debugging ↗	Remote debugging requires inbound ports to be opened on Function apps. Remote debugging should be turned off.	DeployIfExists, Disabled	1.0.0 ↗
Configure Function apps to use the latest TLS version ↗	Periodically, newer versions are released for TLS either due to security flaws, include additional functionality, and enhance speed. Upgrade to the latest TLS version for Function apps to take advantage of security fixes, if any, and/or new functionalities of the latest version.	DeployIfExists, Disabled	1.1.0 ↗

Name	Description	Effect(s)	Version
(Azure portal)			(GitHub)
Enable logging by category group for App Service (microsoft.web/sites) to Log Analytics ↗	Resource logs should be enabled to track activities and events that take place on your resources and give you visibility and insights into any changes that occur. This policy deploys a diagnostic setting using a category group to route logs to a Log Analytics workspace for App Service (microsoft.web/sites).	DeployIfNotExists, AuditIfNotExists, Disabled	1.0.0 ↗
Enable logging by category group for App Service Environments (microsoft.web/hostingenvironments) to Event Hub ↗	Resource logs should be enabled to track activities and events that take place on your resources and give you visibility and insights into any changes that occur. This policy deploys a diagnostic setting using a category group to route logs to an Event Hub for App Service Environments (microsoft.web/hostingenvironments).	DeployIfNotExists, AuditIfNotExists, Disabled	1.0.0 ↗
Enable logging by category group for App Service Environments (microsoft.web/hostingenvironments) to Log Analytics ↗	Resource logs should be enabled to track activities and events that take place on your resources and give you visibility and insights into any changes that occur. This policy deploys a diagnostic setting using a category group to route logs to a Log Analytics workspace for App Service Environments (microsoft.web/hostingenvironments).	DeployIfNotExists, AuditIfNotExists, Disabled	1.0.0 ↗
Enable logging by category group for App Service Environments (microsoft.web/hostingenvironments) to Storage ↗	Resource logs should be enabled to track activities and events that take place on your resources and give you visibility and insights into any changes that occur. This policy deploys a diagnostic setting using a category group to route logs to a Storage Account for App Service Environments (microsoft.web/hostingenvironments).	DeployIfNotExists, AuditIfNotExists, Disabled	1.0.0 ↗
Enable logging by category group for Function App (microsoft.web/sites) to Log Analytics ↗	Resource logs should be enabled to track activities and events that take place on your resources and give you visibility and insights into any changes that occur. This policy deploys a diagnostic setting using a category group to route logs to a Log Analytics workspace for Function App (microsoft.web/sites).	DeployIfNotExists, AuditIfNotExists, Disabled	1.0.0 ↗
Function app slots should disable public network access ↗	Disabling public network access improves security by ensuring that the Function app is not exposed on the public internet. Creating private endpoints can limit exposure of a Function App. Learn more at: https://aka.ms/app-service-private-endpoint ↗ .	Audit, Disabled, Deny	1.0.0 ↗

Name	Description	Effect(s)	Version
(Azure portal)			(GitHub)
Function app slots should have Client Certificates (Incoming client certificates) enabled ↗	Client certificates allow for the app to request a certificate for incoming requests. Only clients that have a valid certificate will be able to reach the app. This policy applies to apps with Http version set to 1.1.	AuditIfNotExists, Disabled	1.0.0 ↗
Function app slots should have remote debugging turned off ↗	Remote debugging requires inbound ports to be opened on Function apps. Remote debugging should be turned off.	AuditIfNotExists, Disabled	1.0.0 ↗
Function app slots should not have CORS configured to allow every resource to access your apps ↗	Cross-Origin Resource Sharing (CORS) should not allow all domains to access your Function app. Allow only required domains to interact with your Function app.	AuditIfNotExists, Disabled	1.0.0 ↗
Function app slots should only be accessible over HTTPS ↗	Use of HTTPS ensures server/service authentication and protects data in transit from network layer eavesdropping attacks.	Audit, Disabled, Deny	2.0.0 ↗
Function app slots should require FTPS only ↗	Enable FTPS enforcement for enhanced security.	AuditIfNotExists, Disabled	1.0.0 ↗
Function app slots should use an Azure file share for its content directory ↗	The content directory of a Function app should be located on an Azure file share. The storage account information for the file share must be provided before any publishing activity. To learn more about using Azure Files for hosting app service content refer to https://go.microsoft.com/fwlink/?linkid=2151594 ↗.	Audit, Disabled	1.0.0 ↗
Function app slots should use latest 'HTTP Version' ↗	Periodically, newer versions are released for HTTP either due to security flaws or to include additional functionality. Using the latest HTTP version for web apps to take advantage of security fixes, if any, and/or new functionalities of the newer version.	AuditIfNotExists, Disabled	1.0.0 ↗
Function app slots should use the latest TLS version ↗	Periodically, newer versions are released for TLS either due to security flaws, include additional functionality, and enhance speed. Upgrade to the latest TLS version for Function apps to take advantage of security fixes, if any, and/or new functionalities of the latest version.	AuditIfNotExists, Disabled	1.1.0 ↗
Function app slots that use Java should use a specified 'Java version' ↗	Periodically, newer versions are released for Java software either due to security flaws or to include additional functionality. Using the latest Java version for Function apps is recommended in order to take advantage of security fixes, if any, and/or new functionalities of the latest version. This policy only applies to Linux apps. This policy requires you to specify a Java version that meets your requirements.	AuditIfNotExists, Disabled	1.0.0 ↗

Name (Azure portal)	Description	Effect(s)	Version (GitHub)
Function app slots that use Python should use a specified 'Python version'	Periodically, newer versions are released for Python software either due to security flaws or to include additional functionality. Using the latest Python version for Function apps is recommended in order to take advantage of security fixes, if any, and/or new functionalities of the latest version. This policy only applies to Linux apps. This policy requires you to specify a Python version that meets your requirements.	AuditIfNotExists, Disabled	1.0.0
Function apps should disable public network access	Disabling public network access improves security by ensuring that the Function app is not exposed on the public internet. Creating private endpoints can limit exposure of a Function App. Learn more at: https://aka.ms/app-service-private-endpoint .	Audit, Disabled, Deny	1.0.0
Function apps should have authentication enabled	Azure App Service Authentication is a feature that can prevent anonymous HTTP requests from reaching the Function app, or authenticate those that have tokens before they reach the Function app.	AuditIfNotExists, Disabled	3.0.0
Function apps should have Client Certificates (Incoming client certificates) enabled	Client certificates allow for the app to request a certificate for incoming requests. Only clients that have a valid certificate will be able to reach the app. This policy applies to apps with Http version set to 1.1.	AuditIfNotExists, Disabled	1.0.0
Function apps should have remote debugging turned off	Remote debugging requires inbound ports to be opened on Function apps. Remote debugging should be turned off.	AuditIfNotExists, Disabled	2.0.0
Function apps should not have CORS configured to allow every resource to access your apps	Cross-Origin Resource Sharing (CORS) should not allow all domains to access your Function app. Allow only required domains to interact with your Function app.	AuditIfNotExists, Disabled	2.0.0
Function apps should only be accessible over HTTPS	Use of HTTPS ensures server/service authentication and protects data in transit from network layer eavesdropping attacks.	Audit, Disabled, Deny	5.0.0
Function apps should require FTPS only	Enable FTPS enforcement for enhanced security.	AuditIfNotExists, Disabled	3.0.0
Function apps should use an Azure file share for its content directory	The content directory of a Function app should be located on an Azure file share. The storage account information for the file share must be provided before any publishing activity. To learn more about using Azure Files for hosting app service content refer to https://go.microsoft.com/fwlink/?linkid=2151594 .	Audit, Disabled	3.0.0

Name (Azure portal)	Description	Effect(s)	Version (GitHub)
Function apps should use latest 'HTTP Version'	Periodically, newer versions are released for HTTP either due to security flaws or to include additional functionality. Using the latest HTTP version for web apps to take advantage of security fixes, if any, and/or new functionalities of the newer version.	AuditIfNotExists, Disabled	4.0.0
Function apps should use managed identity	Use a managed identity for enhanced authentication security	AuditIfNotExists, Disabled	3.0.0
Function apps should use the latest TLS version	Periodically, newer versions are released for TLS either due to security flaws, include additional functionality, and enhance speed. Upgrade to the latest TLS version for Function apps to take advantage of security fixes, if any, and/or new functionalities of the latest version.	AuditIfNotExists, Disabled	2.1.0
Function apps that use Java should use a specified 'Java version'	Periodically, newer versions are released for Java software either due to security flaws or to include additional functionality. Using the latest Java version for Function apps is recommended in order to take advantage of security fixes, if any, and/or new functionalities of the latest version. This policy only applies to Linux apps. This policy requires you to specify a Java version that meets your requirements.	AuditIfNotExists, Disabled	3.1.0
Function apps that use Python should use a specified 'Python version'	Periodically, newer versions are released for Python software either due to security flaws or to include additional functionality. Using the latest Python version for Function apps is recommended in order to take advantage of security fixes, if any, and/or new functionalities of the latest version. This policy only applies to Linux apps. This policy requires you to specify a Python version that meets your requirements.	AuditIfNotExists, Disabled	4.1.0

Release notes

October 2024

- TLS 1.3 is now supported in App Service apps and slots. The following policies have been updated to enforce setting the minimum TLS version to 1.3:
 - "App Service apps should use the latest TLS version"
 - "App Service app slots should use the latest TLS version"
 - "Configure App Service apps to use the latest TLS version"
 - "Configure App Service app slots to use the latest TLS version"
 - "Function apps should use the latest TLS version"
 - "Configure Function apps to use the latest TLS version"

- "Function app slots should use the latest TLS version"
- "Configure Function app slots to use the latest TLS version"

April 2023

- **App Service apps that use Java should use the latest 'Java version'**
 - Rename of policy to "App Service apps that use Java should use a specified 'Java version'"
 - Update policy so that it requires a version specification before assignment
- **App Service apps that use Python should use the latest 'Python version'**
 - Rename of policy to "App Service apps that use Python should use a specified 'Python version'"
 - Update policy so that it requires a version specification before assignment
- **Function apps that use Java should use the latest 'Java version'**
 - Rename of policy to "Function apps that use Java should use a specified 'Java version'"
 - Update policy so that it requires a version specification before assignment
- **Function apps that use Python should use the latest 'Python version'**
 - Rename of policy to "Function apps that use Python should use a specified 'Python version'"
 - Update policy so that it requires a version specification before assignment
- **App Service apps that use PHP should use the latest 'PHP version'**
 - Rename of policy to "App Service apps that use PHP should use a specified 'PHP version'"
 - Update policy so that it requires a version specification before assignment
- **App Service app slots that use Python should use a specified 'Python version'**
 - New policy created
- **Function app slots that use Python should use a specified 'Python version'**
 - New policy created
- **App Service app slots that use PHP should use a specified 'PHP version'**
 - New policy created
- **App Service app slots that use Java should use a specified 'Java version'**
 - New policy created
- **Function app slots that use Java should use a specified 'Java version'**
 - New policy created

November 2022

- Deprecation of policy **App Service apps should enable outbound non-RFC 1918 traffic to Azure Virtual Network**
 - Replaced by a policy with the same display name based on the site property to support *Deny* effect
- Deprecation of policy **App Service app slots should enable outbound non-RFC 1918 traffic to Azure Virtual Network**
 - Replaced by a policy with the same display name based on the site property to support *Deny* effect
- **App Service apps should enable outbound non-RFC 1918 traffic to Azure Virtual Network**
 - New policy created
- **App Service app slots should enable outbound non-RFC 1918 traffic to Azure Virtual Network**
 - New policy created

- App Service apps should enable configuration routing to Azure Virtual Network
 - New policy created
- App Service app slots should enable configuration routing to Azure Virtual Network
 - New policy created

October 2022

- Function app slots should have remote debugging turned off
 - New policy created
- App Service app slots should have remote debugging turned off
 - New policy created
- Function app slots should use latest 'HTTP Version'
 - New policy created
- Function app slots should use the latest TLS version
 - New policy created
- App Service app slots should use the latest TLS version
 - New policy created
- App Service app slots should have resource logs enabled
 - New policy created
- App Service app slots should enable outbound non-RFC 1918 traffic to Azure Virtual Network
 - New policy created
- App Service app slots should use managed identity
 - New policy created
- App Service app slots should use latest 'HTTP Version'
 - New policy created
- Deprecation of policy **Configure App Services to disable public network access**
 - Replaced by "Configure App Service apps to disable public network access"
- Deprecation of policy **App Services should disable public network access**
 - Replaced by "App Service apps should disable public network access" to support *Deny* effect
- App Service apps should disable public network access
 - New policy created
- App Service app slots should disable public network access
 - New policy created
- Configure App Service apps to disable public network access
 - New policy created
- Configure App Service app slots to disable public network access
 - New policy created
- Function apps should disable public network access
 - New policy created
- Function app slots should disable public network access
 - New policy created
- Configure Function apps to disable public network access
 - New policy created
- Configure Function app slots to disable public network access
 - New policy created
- Configure App Service app slots to turn off remote debugging

- New policy created
- **Configure Function app slots to turn off remote debugging**
 - New policy created
- **Configure App Service app slots to use the latest TLS version**
 - New policy created
- **Configure Function app slots to use the latest TLS version**
 - New policy created
- **App Service apps should use latest 'HTTP Version'**
 - Update scope to include Windows apps
- **Function apps should use latest 'HTTP Version'**
 - Update scope to include Windows apps
- **App Service Environment apps should not be reachable over public internet**
 - Modify policy definition to remove check on API version

September 2022

- **App Service apps should be injected into a virtual network**
 - Update scope of policy to remove slots
 - Creation of "App Service app slots should be injected into a virtual network" to monitor slots
- **App Service app slots should be injected into a virtual network**
 - New policy created
- **Function apps should have 'Client Certificates (Incoming client certificates)' enabled**
 - Update scope of policy to remove slots
 - Creation of "Function app slots should have 'Client Certificates (Incoming client certificates)' enabled" to monitor slots
- **Function app slots should have 'Client Certificates (Incoming client certificates)' enabled**
 - New policy created
- **Function apps should use an Azure file share for its content directory**
 - Update scope of policy to remove slots
 - Creation of "Function app slots should use an Azure file share for its content directory" to monitor slots
- **Function app slots should use an Azure file share for its content directory**
 - New policy created
- **App Service apps should have 'Client Certificates (Incoming client certificates)' enabled**
 - Update scope of policy to remove slots
 - Creation of "App Service app slots should have 'Client Certificates (Incoming client certificates)' enabled" to monitor slots
- **App Service app slots should have 'Client Certificates (Incoming client certificates)' enabled**
 - New policy created
- **App Service apps should use an Azure file share for its content directory**
 - Update scope of policy to remove slots
 - Creation of "App Service app slots should use an Azure file share for its content directory" to monitor slots
- **App Service app slots should use an Azure file share for its content directory**
 - New policy created

- Function app slots should require FTPS only
 - New policy created
- App Service app slots should require FTPS only
 - New policy created
- Function app slots should not have CORS configured to allow every resource to access your apps
 - New policy created
- App Service app slots should not have CORS configured to allow every resource to access your app
 - New policy created
- Function apps should only be accessible over HTTPS
 - Update scope of policy to remove slots
 - Creation of "Function app slots should only be accessible over HTTPS" to monitor slots
 - Add "Deny" effect
 - Creation of "Configure Function apps to only be accessible over HTTPS" for enforcement of policy
- Function app slots should only be accessible over HTTPS
 - New policy created
- Configure Function apps to only be accessible over HTTPS
 - New policy created
- Configure Function app slots to only be accessible over HTTPS
 - New policy created
- App Service apps should use a SKU that supports private link
 - Update list of supported SKUs of policy to include the Workflow Standard tier for Logic Apps
- Configure App Service apps to use the latest TLS version
 - New policy created
- Configure Function apps to use the latest TLS version
 - New policy created
- Configure App Service apps to turn off remote debugging
 - New policy created
- Configure Function apps to turn off remote debugging
 - New policy created

August 2022

- App Service apps should only be accessible over HTTPS
 - Update scope of policy to remove slots
 - Creation of "App Service app slots should only be accessible over HTTPS" to monitor slots
 - Add "Deny" effect
 - Creation of "Configure App Service apps to only be accessible over HTTPS" for enforcement of policy
- App Service app slots should only be accessible over HTTPS
 - New policy created
- Configure App Service apps to only be accessible over HTTPS
 - New policy created
- Configure App Service app slots to only be accessible over HTTPS

- New policy created

July 2022

- Deprecation of the following policies:
 - Ensure API app has 'Client Certificates (Incoming client certificates)' set to 'On'
 - Ensure that 'Python version' is the latest, if used as a part of the API app
 - CORS should not allow every resource to access your API App
 - Managed identity should be used in your API App
 - Remote debugging should be turned off for API Apps
 - Ensure that 'PHP version' is the latest, if used as a part of the API app
 - API apps should use an Azure file share for its content directory
 - FTPS only should be required in your API App
 - Ensure that 'Java version' is the latest, if used as a part of the API app
 - Ensure that 'HTTP Version' is the latest, if used to run the API app
 - Latest TLS version should be used in your API App
 - Authentication should be enabled on your API app
- Function apps should have 'Client Certificates (Incoming client certificates)' enabled
 - Update scope of policy to include slots
 - Update scope of policy to exclude Logic apps
- Ensure WEB app has 'Client Certificates (Incoming client certificates)' set to 'On'
 - Rename of policy to "App Service apps should have 'Client Certificates (Incoming client certificates)' enabled"
 - Update scope of policy to include slots
 - Update scope of policy to include all app types except Function apps
- Ensure that 'Python version' is the latest, if used as a part of the Web app
 - Rename of policy to "App Service apps that use Python should use the latest 'Python version'"
 - Update scope of policy to include all app types except Function apps
- Ensure that 'Python version' is the latest, if used as a part of the Function app
 - Rename of policy to "Function apps that use Python should use the latest 'Python version'"
 - Update scope of policy to exclude Logic apps
- CORS should not allow every resource to access your Web Applications
 - Rename of policy to "App Service apps should not have CORS configured to allow every resource to access your apps"
 - Update scope of policy to include all app types except Function apps
- CORS should not allow every resource to access your Function Apps
 - Rename of policy to "Function apps should not have CORS configured to allow every resource to access your apps"
 - Update scope of policy to exclude Logic apps
- Managed identity should be used in your Function App
 - Rename of policy to "Function apps should use managed identity"
 - Update scope of policy to exclude Logic apps
- Managed identity should be used in your Web App
 - Rename of policy to "App Service apps should use managed identity"
 - Update scope of policy to include all app types except Function apps

- **Remote debugging should be turned off for Function Apps**
 - Rename of policy to "Function apps should have remote debugging turned off"
 - Update scope of policy to exclude Logic apps
- **Remote debugging should be turned off for Web Applications**
 - Rename of policy to "App Service apps should have remote debugging turned off"
 - Update scope of policy to include all app types except Function apps
- **Ensure that 'PHP version' is the latest, if used as a part of the WEB app**
 - Rename of policy to "App Service apps that use PHP should use the latest 'PHP version'"
 - Update scope of policy to include all app types except Function apps
- **App Service slots should have local authentication methods disabled for SCM site deployment**
 - Rename of policy to "App Service app slots should have local authentication methods disabled for SCM site deployments"
- **App Service should have local authentication methods disabled for SCM site deployments**
 - Rename of policy to "App Service apps should have local authentication methods disabled for SCM site deployments"
- **App Service slots should have local authentication methods disabled for FTP deployments**
 - Rename of policy to "App Service app slots should have local authentication methods disabled for FTP deployments"
- **App Service should have local authentication methods disabled for FTP deployments**
 - Rename of policy to "App Service apps should have local authentication methods disabled for FTP deployments"
- **Function apps should use an Azure file share for its content directory**
 - Update scope of policy to include slots
 - Update scope of policy to exclude Logic apps
- **Web apps should use an Azure file share for its content directory**
 - Rename of policy to "App Service apps should use an Azure file share for its content directory"
 - Update scope of policy to include slots
 - Update scope of policy to include all app types except Function apps
- **FTPS only should be required in your Function App**
 - Rename of policy to "Function apps should require FTPS only"
 - Update scope of policy to exclude Logic apps
- **FTPS should be required in your Web App**
 - Rename of policy to "App Service apps should require FTPS only"
 - Update scope of policy to include all app types except Function apps
- **Ensure that 'Java version' is the latest, if used as a part of the Function app**
 - Rename of policy to "Function apps that use Java should use the latest 'Java version'"
 - Update scope of policy to exclude Logic apps
- **Ensure that 'Java version' is the latest, if used as a part of the Web app**
 - Rename of policy to "App Service apps that use Java should use the latest 'Java version'"
 - Update scope of policy to include all app types except Function apps
- **App Service should use private link**
 - Rename of policy to "App Service apps should use private link"
- **Configure App Services to use private DNS zones**
 - Rename of policy to "Configure App Service apps to use private DNS zones"

- **App Service Apps should be injected into a virtual network**
 - Rename of policy to "App Service apps should be injected into a virtual network"
 - Update scope of policy to include slots
- **Ensure that 'HTTP Version' is the latest, if used to run the Web app**
 - Rename of policy to "App Service apps should use latest 'HTTP Version'"
 - Update scope of policy to include all app types except Function apps
- **Ensure that 'HTTP Version' is the latest, if used to run the Function app**
 - Rename of policy to "Function apps should use latest 'HTTP Version'"
 - Update scope of policy to exclude Logic apps
- **Latest TLS version should be used in your Web App**
 - Rename of policy to "App Service apps should use the latest TLS version"
 - Update scope of policy to include all app types except Function apps
- **Latest TLS version should be used in your Function App**
 - Rename of policy to "Function apps should use the latest TLS version"
 - Update scope of policy to exclude Logic apps
- **App Service Environment should disable TLS 1.0 and 1.1**
 - Rename of policy to "App Service Environment should have TLS 1.0 and 1.1 disabled"
- **Resource logs in App Services should be enabled**
 - Rename of policy to "App Service apps should have resource logs enabled"
- **Authentication should be enabled on your web app**
 - Rename of policy to "App Service apps should have authentication enabled"
- **Authentication should be enabled on your Function app**
 - Rename of policy to "Function apps should have authentication enabled"
 - Update scope of policy to exclude Logic apps
- **App Service Environment should enable internal encryption**
 - Rename of policy to "App Service Environment should have internal encryption enabled"
- **Function apps should only be accessible over HTTPS**
 - Update scope of policy to exclude Logic apps
- **App Service should use a virtual network service endpoint**
 - Rename of policy to "App Service apps should use a virtual network service endpoint"
 - Update scope of policy to include all app types except Function apps

June 2022

- Deprecation of policy **API App should only be accessible over HTTPS**
- **Web Application should only be accessible over HTTPS**
 - Rename of policy to "App Service apps should only be accessible over HTTPS"
 - Update scope of policy to include all app types except Function apps
 - Update scope of policy to include slots
- **Function apps should only be accessible over HTTPS**
 - Update scope of policy to include slots
- **App Service apps should use a SKU that supports private link**
 - Update logic of policy to include checks on App Service plan tier or name so that the policy supports Terraform deployments
 - Update list of supported SKUs of policy to include the Basic and Standard tiers

Next steps

- See the built-ins on the [Azure Policy GitHub repo](#).
 - Review the [Azure Policy definition structure](#).
 - Review [Understanding policy effects](#).
-

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

Azure App Service plan overview

Article • 09/04/2024

ⓘ Note

Starting June 1, 2024, all newly created App Service apps will have the option to generate a unique default hostname using the naming convention <app-name>-<random-hash>. <region>.azurewebsites.net. Existing app names will remain unchanged.

Example: myapp-ds27dh7271aah175.westus-01.azurewebsites.net

For further details, refer to [Unique Default Hostname for App Service Resource ↗](#).

An app service always runs in an *App Service plan*. In addition, [Azure Functions](#) also has the option of running in an *App Service plan*. An App Service plan defines a set of compute resources for a web app to run.

When you create an App Service plan in a certain region (for example, West Europe), a set of compute resources is created for that plan in that region. Whatever apps you put into this App Service plan run on these compute resources as defined by your App Service plan. Each App Service plan defines:

- Operating System (Windows, Linux)
- Region (West US, East US, and so on)
- Number of VM instances
- Size of VM instances (Small, Medium, Large)
- Pricing tier (Free, Shared, Basic, Standard, Premium, PremiumV2, PremiumV3, Isolated, IsolatedV2)

The *pricing tier* of an App Service plan determines what App Service features you get and how much you pay for the plan. The pricing tiers available to your App Service plan depend on the operating system selected at creation time. These are the categories of pricing tiers:

- **Shared compute:** Free and Shared, the two base tiers, run an app on the same Azure VM as other App Service apps, including apps of other customers. These tiers allocate CPU quotas to each app that runs on the shared resources, and the resources cannot scale out. These tiers are intended to be used only for development and testing purposes.

- **Dedicated compute:** The **Basic**, **Standard**, **Premium**, **PremiumV2**, and **PremiumV3** tiers run apps on dedicated Azure VMs. Only apps in the same App Service plan share the same compute resources. The higher the tier, the more VM instances that are available to you for scale-out.
- **Isolated:** The **Isolated** and **IsolatedV2** tiers run dedicated Azure VMs on dedicated Azure virtual networks. They provide network isolation on top of compute isolation to your apps. They provide the maximum scale-out capabilities.

Each tier also provides a specific subset of App Service features. These features include custom domains and TLS/SSL certificates, autoscaling, deployment slots, backups, Traffic Manager integration, and more. The higher the tier, the more features that are available. To find out which features are supported in each pricing tier, see [App Service plan details](#).

PremiumV3 pricing tier

The **PremiumV3** pricing tier guarantees machines with faster processors (minimum 195 [ACU](#) per virtual CPU), SSD storage, memory-optimized options and quadruple memory-to-core ratio compared to **Standard** tier. **PremiumV3** also supports higher scale via increased instance count while still providing all the advanced capabilities found in **Standard** tier. All features available in the existing **PremiumV2** tier are included in **PremiumV3**.

Multiple VM sizes are available for this tier, including 4-to-1 and 8-to-1 memory-to-core ratios:

- P0v3 (1 vCPU, 4 GiB of memory)
- P1v3 (2 vCPU, 8 GiB of memory)
- P1mv3 (2 vCPU, 16 GiB of memory)
- P2v3 (4 vCPU, 16 GiB of memory)
- P2mv3 (4 vCPU, 32 GiB of memory)
- P3v3 (8 vCPU, 32 GiB of memory)
- P3mv3 (8 vCPU, 64 GiB of memory)
- P4mv3 (16 vCPU, 128 GiB of memory)
- P5mv3 (32 vCPU, 256 GiB of memory)

For **PremiumV3** pricing information, see [App Service Pricing](#).

To get started with the new **PremiumV3** pricing tier, see [Configure PremiumV3 tier for App Service](#).

How does my app run and scale?

In the **Free** and **Shared** tiers, an app receives CPU minutes on a shared VM instance and cannot scale out. In other tiers, an app runs and scales as follows.

When you create an app in App Service, it's part of an App Service plan. When the app runs, it runs on all the VM instances configured in the App Service plan. If multiple apps are in the same App Service plan, they all share the same VM instances. If you have multiple deployment slots for an app, all deployment slots also run on the same VM instances. If you enable diagnostic logs, perform backups, or run [WebJobs](#), they also use CPU cycles and memory on these VM instances.

In this way, the App Service plan is the scale unit of the App Service apps. If the plan is configured to run five VM instances, then all apps in the plan run on all five instances. If the plan is configured for autoscaling, then all apps in the plan are scaled out together based on the autoscale settings.

For information on scaling out an app, see [Scale instance count manually or automatically](#).

How much does my App Service plan cost?

This section describes how App Service apps are billed. For detailed, region-specific pricing information, see [App Service Pricing](#).

Except for the **Free** tier, an App Service plan carries a charge on the compute resources it uses.

- In the **Shared** tier, each app receives a quota of CPU minutes, so *each app* is charged for the CPU quota.
- In the dedicated compute tiers (**Basic**, **Standard**, **Premium**, **PremiumV2**, **PremiumV3**), the App Service plan defines the number of VM instances the apps are scaled to, so *each VM instance* in the App Service plan is charged. These VM instances are charged the same regardless of how many apps are running on them. To avoid unexpected charges, see [Clean up an App Service plan](#).
- In the **Isolated** and **IsolatedV2** tiers, the App Service Environment defines the number of isolated workers that run your apps, and *each worker* is charged. In addition, in the **Isolated** tier there's a flat Stamp Fee for running the App Service Environment itself.

You don't get charged for using the App Service features that are available to you (configuring custom domains, TLS/SSL certificates, deployment slots, backups, etc.). The

exceptions are:

- App Service Domains - You pay when you purchase one in Azure and when you renew it each year.
- App Service Certificates - You pay when you purchase one in Azure and when you renew it each year.
- IP-based TLS connections - There's an hourly charge for each IP-based TLS connection, but some **Standard** tiers or above give you one IP-based TLS connection for free. SNI-based TLS connections are free.

ⓘ Note

If you integrate App Service with another Azure service, you may need to consider charges from these other services. For example, if you use Azure Traffic Manager to scale your app geographically, Azure Traffic Manager also charges you based on your usage. To estimate your cross-services cost in Azure, see [Pricing calculator](#).

Want to optimize and save on your cloud spending?

Azure services cost money. Azure Cost Management helps you set budgets and configure alerts to keep spending under control. Analyze, manage, and optimize your Azure costs with Cost Management. To learn more, see the [quickstart on analyzing your costs](#).

What if my app needs more capabilities or features?

Your App Service plan can be scaled up and down at any time. It's as simple as changing the pricing tier of the plan. You can choose a lower pricing tier at first and scale up later when you need more App Service features.

For example, you can start testing your web app in a **Free** App Service plan and pay nothing. When you add your [custom DNS name](#) to the web app, just scale your plan up to **Shared** tier. Later, when you want to [create a TLS binding](#), scale your plan up to **Basic** tier. When you want to have [staging environments](#), scale up to **Standard** tier. When you need more cores, memory, or storage, scale up to a bigger VM size in the same tier.

The same works in reverse. When you feel you no longer need the capabilities or features of a higher tier, you can scale down to a lower tier, which saves you money.

For information on scaling up the App Service plan, see [Scale up an app in Azure](#).

If your app is in the same App Service plan with other apps, you may want to improve the app's performance by isolating the compute resources. You can do this by moving the app into a separate App Service plan. For more information, see [Move an app to another App Service plan](#).

Should I put an app in a new plan or an existing plan?

Since you pay for the computing resources your App Service plan allocates (see [How much does my App Service plan cost?](#)), you can potentially save money by putting multiple apps into one App Service plan. You can continue to add apps to an existing plan as long as the plan has enough resources to handle the load. However, keep in mind that apps in the same App Service plan all share the same compute resources. To determine whether the new app has the necessary resources, you need to understand the capacity of the existing App Service plan, and the expected load for the new app. Overloading an App Service plan can potentially cause downtime for your new and existing apps.

Isolate your app into a new App Service plan when:

- The app is resource-intensive. The number may actually be lower depending on how resource intensive the hosted applications are. However, for general guidance, refer to the table below:

[+] [Expand table](#)

App Service Plan SKU	Maximum Apps
B1, S1, P1v2, I1v1	8
B2, S2, P2v2, I2v1	16
B3, S3, P3v2, I3v1	32
P0v3	8
P1v3, I1v2	16
P2v3, I2v2, P1mv3	32
P3v3, I3v2, P2mv3	64
I4v2, I5v2, I6v2	Maximum density bound by vCPU usage
P3mv3, P4mv3, P5mv3	Maximum density bound by vCPU usage

- You want to scale the app independently from the other apps in the existing plan.
- The app needs resources in a different geographical region.

This way you can allocate a new set of resources for your app and gain greater control of your apps.

Note

An active slot is also classified as an active app because it too is competing for resources on the same App Service Plan.

Next step

[Manage an App Service plan](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

Manage an App Service plan in Azure

Article • 09/09/2024

ⓘ Note

Starting June 1, 2024, all newly created App Service apps will have the option to generate a unique default hostname using the naming convention <app-name>-<random-hash>. <region>.azurewebsites.net . Existing app names will remain unchanged.

Example: myapp-ds27dh7271aah175.westus-01.azurewebsites.net

For further details, refer to [Unique Default Hostname for App Service Resource ↗](#).

An [Azure App Service plan](#) provides the resources that an App Service app needs to run. This guide shows how to manage an App Service plan.

Create an App Service plan

ⓘ Tip

If you want to create a plan in an App Service Environment, you can select it in the **Region** and follow the rest of the steps as described below.

You can create an empty App Service plan, or you can create a plan as part of app creation.

1. To start creating an App Service plan, go to [Create App Service Plan ↗](#) on the Azure portal.

Create App Service Plan

X

[Basics](#) [Tags](#) [Review + create](#)

App Service plans give you the flexibility to allocate specific apps to a given set of resources and further optimize your Azure resource utilization. This way, if you want to save money on your testing environment you can share a plan across multiple apps. [Learn more ↗](#)

Project Details

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

Subscription * ⓘ

Resource Group * ⓘ

(New) Resource group

[Create new](#)

App Service Plan details

Name *

Enter a name for your App Service Plan

Operating System *

Linux Windows

Region *

East US

Pricing Tier

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

Pricing plan

Premium V3 P1V3 (195 minimum ACU/vCPU, 8 GB memory, 2 vCPU)

[Explore pricing plans](#)

Zone redundancy

An App Service plan can be deployed as a zone redundant service in the regions that support it. This is a deployment time only decision. You can't make an App Service plan zone redundant after it has been deployed. [Learn more ↗](#)

Zone redundancy

- Enabled:** Your App Service plan and the apps in it will be zone redundant. The minimum App Service plan instance count will be three.
- Disabled:** Your App Service Plan and the apps in it will not be zone redundant. The minimum App Service plan instance count will be one.

[Review + create](#)

[< Previous](#)

[Next : Tags >](#)

2. Configure the **Project Details** section before configuring the App Service plan.
3. In the **App Service Plan details** section, name the App Service plan, and then select the **Operating System** and **Region**. The region specifies where your App Service plan is created.
4. When creating a plan, you can select the pricing tier of the new plan. In **Pricing Tier**, select a **Pricing plan** or select **Explore pricing plans** to view additional details.
5. In the **Zone redundancy** section, select whether the App Service plan zone redundancy should be enabled or disabled.
6. Select **Review + create** to create the App Service plan.

ⓘ Important

When you create a new App Service plan in an existing resource group, certain conditions with existing apps can trigger these errors:

- The pricing tier is not allowed in this resource group
- <SKU_NAME> workers are not available in resource group
<RESOURCE_GROUP_NAME>

This can happen due to incompatibilities with pricing tiers, regions, operating systems, availability zones, existing function apps, or existing web apps. If one of these errors occurs, create your App Service plan in a **new** resource group.

Move an app to another App Service plan

You can move an app to another App Service plan, as long as the source plan and the target plan are in the *same resource group and geographical region and of the same OS type*. Any change in type, such as Windows to Linux or any type that's different from the originating type, isn't supported.

ⓘ Note

Azure deploys each new App Service plan into a deployment unit, internally called a *webspace*. Each region can have many webspaces, but your app can only move between plans that are created in the same webspace. An App Service Environment can have multiple webspaces, but your app can only move between plans that are created in the same webspace.

You can't specify the webspace you want when creating a plan, but it's possible to ensure that a plan is created in the same webspace as an existing plan. In brief, all plans created with the same resource group, region combination, and operating system are deployed into the same webspace. For example, if you created a plan in resource group A and region B, then any plan you subsequently create in resource group A and region B is deployed into the same webspace. Note that plans can't move webspaces after they're created, so you can't move a plan into "the same webspace" as another plan by moving it to another resource group.

1. In the [Azure portal](#), search for and select **App services** and select the app that you want to move.

2. From the left menu, under App Service Plan, select Change App Service plan.

The screenshot shows the Azure portal interface for managing an app service. On the left, there's a sidebar with various options like App Service plan, Quotas, and Change App Service plan (which is highlighted with a red box). The main area is titled 'Change App Service plan' and contains sections for 'CURRENT PLAN DETAILS' (showing 'appservice79') and 'DESTINATION PLAN DETAILS'. In the destination details, there's a note about resource group and geographical restrictions, followed by fields for 'App Service plan*' (with a dropdown showing '(New) ASP-phpstarter-mangesh-82f1' and a 'Create new' button), 'Resource group' (set to 'Default-Web-SouthCentralUS'), 'Region' (set to 'South Central US'), and 'Pricing Tier' (set to 'Free (F1)'). A large red box highlights the 'OK' button at the bottom right of the dialog.

3. In the **App Service plan** dropdown, select an existing plan to move the app to. The dropdown shows only plans that are in the same resource group and geographical region as the current App Service plan. If no such plan exists, it lets you create a plan by default. You can also create a new plan manually by selecting **Create new**.
4. If you create a plan, you can select the pricing tier of the new plan. In **Pricing Tier**, select the existing tier to change it.

Important

If you're moving an app from a higher-tiered plan to a lower-tiered plan, such as from D1 to F1, the app might lose certain capabilities in the target plan. For

example, if your app uses TLS/SSL certificates, you might see this error message:

```
Cannot update the site with hostname '<app_name>' because its current  
TLS/SSL configuration 'SNI based SSL enabled' is not allowed in the  
target compute mode. Allowed TLS/SSL configuration is 'Disabled'.
```

5. When finished, select OK.

Move an app to a different region

The region in which your app runs is the region of the App Service plan it's in. However, you can't change an App Service plan's region. If you want to run your app in a different region, one alternative is app cloning. Cloning makes a copy of your app in a new or existing App Service plan in any region.

You can find **Clone App** in the **Development Tools** section of the menu.

ⓘ Important

Cloning has some limitations. You can read about them in [Azure App Service App cloning](#).

Scale an App Service plan

To scale up an App Service plan's pricing tier, see [Scale up an app in Azure](#).

To scale out an app's instance count, see [Scale instance count manually or automatically](#).

Delete an App Service plan

To avoid unexpected charges, when you delete the last app in an App Service plan, App Service also deletes the plan by default. If you choose to keep the plan instead, you should change the plan to the **Free** tier so you're not charged.

ⓘ Important

App Service plans that have no apps associated with them still incur charges because they continue to reserve the configured VM instances.

Next step

[Scale up an app in Azure](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

Configure Premium V3 tier for Azure App Service

Article • 08/29/2023

The new Premium V3 pricing tier gives you faster processors, SSD storage, memory-optimized options, and quadruple the memory-to-core ratio of the existing pricing tiers (double the Premium V2 tier). With the performance advantage, you could save money by running your apps on fewer instances. In this article, you learn how to create an app in Premium V3 tier or scale up an app to Premium V3 tier.

Prerequisites

To scale-up an app to Premium V3, you need to have an Azure App Service app that runs in a pricing tier lower than Premium V3, and the app must be running in an App Service deployment that supports Premium V3. Additionally the App Service deployment must support the desired SKU within Premium V3.

Premium V3 availability

The Premium V3 tier is available for both native and custom containers, including both Windows containers and Linux containers.

Premium V3 as well as specific Premium V3 SKUs are available in some Azure regions and availability in additional regions is being added continually. To see if a specific PremiumV3 offering is available in your region, run the following Azure CLI command in the [Azure Cloud Shell](#) (substitute *P1v3* with the desired SKU):

Azure CLI

```
az appservice list-locations --sku P1V3
```

Create an app in Premium V3 tier

The pricing tier of an App Service app is defined in the [App Service plan](#) that it runs on. You can create an App Service plan by itself or create it as part of app creation.

When configuring the new App Service plan in the [Azure portal](#), select **Pricing plan** and pick one of the **Premium V3** tiers.

To see all the Premium V3 options, select **Explore pricing plans**, then select one of the Premium V3 plans and select **Select**.

Hardware Features

Name	ACU/vCPU	vCPU	Memory (GB)	Remote Storage (GB)	Scale (instances)
▼ Popular options					
<input type="checkbox"/> Free F1	60 minutes/day...	N/A	1	1	N/A
<input type="checkbox"/> Basic B1	100	1	1.75	10	3
<input type="checkbox"/> Premium v3 P0V3	195*	1	4	250	30
<input type="checkbox"/> Premium v3 P1V3	195	2	8	250	30
<input type="checkbox"/> Premium v3 P2V3	195	4	16	250	30
<input type="checkbox"/> Premium v3 P3V3	195	8	32	250	30
<input checked="" type="checkbox"/> Premium v3 P1mv3	195*	2	16	250	30
<input type="checkbox"/> Premium v3 P2mv3	195*	4	32	250	30
<input type="checkbox"/> Premium v3 P3mv3	195*	8	64	250	30
<input type="checkbox"/> Premium v3 P4mv3	195*	16	128	250	30
<input type="checkbox"/> Premium v3 P5mv3	195*	32	256	250	30
<input type="checkbox"/> Isolated v2 I1V2	195	2	8	1000	N/A
<input type="checkbox"/> Isolated v2 I2V2	195	4	16	1000	N/A
<input type="checkbox"/> Isolated v2 I3V2	195	8	32	1000	N/A
▼ Dev/Test (For less demanding workloads)					

Select *ACU/vCPU is an approximation of the SKU's relative performance. [Learn more about pricing ↗](#)

ⓘ Important

If you don't see **P0V3**, **P1V3**, **P2V3**, **P3V3**, **P1mv3**, **P2mv3**, **P3mv3**, **P4mv3**, and **P5mv3** as options, or if some options are greyed out, then either **Premium V3** or an individual SKU within **Premium V3** isn't available in the underlying App Service deployment that contains the App Service plan. See [Scale up from an unsupported resource group and region combination](#) for more details.

Scale up an existing app to Premium V3 tier

Before scaling an existing app to Premium V3 tier, make sure that both Premium V3 as well as the specific SKU within Premium V3 are available. For information, see [PremiumV3 availability](#). If it's not available, see [Scale up from an unsupported resource group and region combination](#).

Depending on your hosting environment, scaling up may require extra steps.

In the [Azure portal](#), open your App Service app page.

In the left navigation of your App Service app page, select **Scale up (App Service plan)**.

The screenshot shows the Azure App Service blade for 'my-demo-app'. On the left, there's a sidebar with various settings like Identity, Backups, Custom domains, TLS/SSL settings, Networking, Scale up (App Service plan) (which is highlighted with a red box), Scale out (App Service plan), and WebJobs. The main panel has a message about .NET Framework 4.8 deployment. Below that, it shows Resource group, Status, Location, Subscription, and Subscription ID.

Select one of the Premium V3 plans and select **Select**.

The screenshot shows the 'Scale up (App Service plan)' configuration page. It has tabs for Hardware (selected) and Features. A table lists various SKU options:

Name	ACU/vCPU	vCPU	Memory (GB)	Remote Storage (GB)	Scale (instances)
Free F1	60 minutes/day...	N/A	1	1	N/A
Basic B1	100	1	1.75	10	3
Premium v3 P0V3	195*	1	4	250	30
Premium v3 P1V3	195	2	8	250	30
Premium v3 P2V3	195	4	16	250	30
Premium v3 P3V3	195	8	32	250	30
Premium v3 P1mv3	195*	2	16	250	30
Premium v3 P2mv3	195*	4	32	250	30
Premium v3 P3mv3	195*	8	64	250	30
Premium v3 P4mv3	195*	16	128	250	30
Premium v3 P5mv3	195*	32	256	250	30
Isolated v2 I1V2	195	2	8	1000	N/A
Isolated v2 I2V2	195	4	16	1000	N/A
Isolated v2 I3V2	195	8	32	1000	N/A

A dropdown menu at the bottom shows 'Popular options' and 'Dev/Test (For less demanding workloads)'. At the bottom, there's a 'Select' button and a note about ACU/vCPU being an approximation.

If your operation finishes successfully, your app's overview page shows that it's now in a Premium V3 tier.

^ Essentials	
Resource group (change) myResourceGroup	URL https://my-demo-app.azurewebsites.net
Status Running	App Service Plan myAppServicePlan (P1v3: 1)
Location West Central US	FTP/deployment username my-demo-app\deployuser
Subscription (change) MySubscription	FTP hostname ftp://waws-prod-cy4-009.ftp.azurewebsites.wind...
Subscription ID 00000000-0000-0000-0000-000000000000	FTPS hostname https://waws-prod-cy4-009.ftp.azurewebsites.wind...
Tags (change) Click here to add tags	

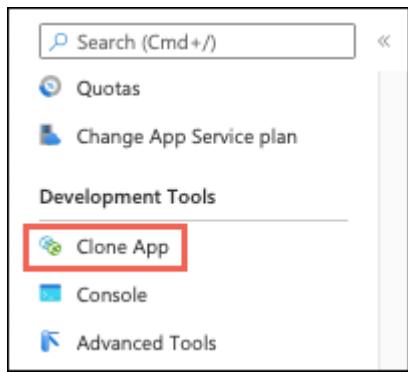
If you get an error

Some App Service plans can't scale up to the Premium V3 tier, or to a newer SKU within Premium V3, if the underlying App Service deployment doesn't support the requested Premium V3 SKU. See [Scale up from an unsupported resource group and region combination](#) for more details.

Scale up from an unsupported resource group and region combination

If your app runs in an App Service deployment where Premium V3 isn't available, or if your app runs in a region that currently does not support Premium V3, you need to re-deploy your app to take advantage of Premium V3. Alternatively newer Premium V3 SKUs may not be available, in which case you also need to re-deploy your app to take advantage of newer SKUs within Premium V3. You have two options:

- Create an app in a new resource group and with a new App Service plan. When creating the App Service plan, select the desired Premium V3 tier. This step ensures that the App Service plan is deployed into a deployment unit that supports Premium V3 as well as the specific SKU within Premium V3. Then, redeploy your application code into the newly created app. Even if you scale the new App Service plan down to a lower tier to save costs, you can always scale back up to Premium V3 and the desired SKU within Premium V3 because the deployment unit supports it.



In the **Clone app** page, you can create an App Service plan using Premium V3 in the region you want, and specify the app settings and configuration that you want to clone.

Automate with scripts

You can automate app creation in the Premium V3 tier with scripts, using the [Azure CLI](#) or [Azure PowerShell](#).

Azure CLI

The following command creates an App Service plan in *P1V3*. You can run it in the Cloud Shell. The options for `--sku` are *P0V3*, *P1V3*, *P2V3*, *P3V3*, *P1mV3*, *P2mV3*, *P3mV3*, *P4mV3*, and *P5mV3*.

```
Azure CLI

az appservice plan create \
    --resource-group <resource_group_name> \
    --name <app_service_plan_name> \
    --sku P1V3
```

Azure PowerShell

Note

We recommend that you use the Azure Az PowerShell module to interact with Azure. See [Install Azure PowerShell](#) to get started. To learn how to migrate to the Az PowerShell module, see [Migrate Azure PowerShell from AzureRM to Az](#).

The following command creates an App Service plan in *P1V3*. The options for `-WorkerSize` are *Small*, *Medium*, and *Large*.

PowerShell

```
New-AzAppServicePlan -ResourceGroupName <resource_group_name>`  
    -Name <app_service_plan_name>`  
    -Location <region_name>`  
    -Tier "Premium V3" `  
    -WorkerSize "Small"
```

More resources

- [Scale up an app in Azure](#)
- [Scale instance count manually or automatically](#)
- [Tutorial: Run a load test to identify performance bottlenecks in a web app](#)

Back up and restore your app in Azure App Service

Article • 09/09/2024

In [Azure App Service](#), you can easily restore app backups. You can also make on-demand custom backups or configure scheduled custom backups. You can restore a backup by overwriting an existing app or by restoring to a new app or slot. This article shows you how to restore a backup and make custom backups.

Back up and restore is supported in the **Basic**, **Standard**, **Premium**, and **Isolated** tiers. For the **Basic** tier, only the production slot can be backed up and restored. For more information about scaling your App Service plan to use a higher tier, see [Scale up an app in Azure](#).

Note

For App Service Environments:

- Automatic backups can be restored to a target app within the App Service Environment itself, not in another App Service Environment.
- Custom backups can be restored to a target app in another App Service Environment, such as from App Service Environment v2 to App Service Environment v3.
- Backups can be restored to a target app of the same OS platform as the source app.

Back up and restore vs. disaster recovery

 Expand table

Platform	Back up and restore guidance	Disaster recovery guidance
App Service web apps (Free and Shared pricing tiers)	If you have web applications deployed to the Free or Shared tier and require access to back up and restore capabilities for these web apps, scale up to the Basic tier or higher.	Bring App Service resources back online in a different Azure region during a regional disaster. Starting March 31, 2025, App Service applications won't be placed in disaster recovery mode during a disaster in an Azure

Platform	Back up and restore guidance	Disaster recovery guidance
		<p>region as explained in the recover from region-wide failure article. We recommend that you implement commonly used disaster recovery techniques to prevent downtime and data loss during a regional disaster.</p>
App Service web apps (Basic, Standard, and Premium pricing tiers)	<p>In Azure App Service, You can make on-demand custom backups or utilize automatic backups. You can restore a backup by overwriting an existing app or by restoring to a new app or slot.</p> <p>Refer to Back up and restore your app in Azure App Service for more info.</p>	<p>Current guidance regarding how to bring App Service resources back online in a different Azure region during a regional disaster is available at Recover from region-wide failure - Azure App Service.</p> <p>Starting March 31, 2025, Azure App Service web applications will no longer be placed in disaster recovery mode during a disaster in an Azure region as explained in the recover from region-wide failure article. We encourage you to implement commonly used disaster recovery techniques to prevent loss of functionality or data for your web apps if there's a regional disaster.</p>
App Service Environment (V2 and V3)	<p>In Azure App Service Environment, You can make on-demand custom backups or use automatic backups. Automatic backups can be restored to a target app within the same App Service Environment, not in another App Service Environment. Custom backups can be restored to a target app in another App Service Environment (such as from a V2 App Service Environment to a V3 App Service Environment). Backups can be restored to a target app of the same OS platform as the source app.</p> <p>Refer to Back up and restore your app in Azure App Service for more details.</p>	<p>We encourage you to implement disaster recovery guidance for web apps deployed to App Service Environment using commonly used disaster recovery techniques.</p>
Azure Functions: Dedicated plan	<p>When you run your function app in a Dedicated (App Service) plan, required function app content is maintained using</p>	<p>Current guidance regarding how to bring function app resources in a Dedicated plan back online in a</p>

Platform	Back up and restore guidance	Disaster recovery guidance
	<p>built-in storage. In a Dedicated plan, you can make on-demand custom backups or use automatic backups. You can restore a backup by overwriting an existing app or by restoring to a new app or slot.</p> <p>For more information, see Back up and restore your app in Azure App Service.</p> <p>Azure Files isn't used by a Dedicated plan, but if you have misconfigured your app with an Azure Files connection, backup isn't supported.</p>	<p>different Azure region during a regional disaster is available at Recover from region-wide failure - Azure App Service.</p> <p>Starting March 31, 2025, App Service applications won't be placed in disaster recovery mode during a disaster in an Azure region as explained in the recover from region-wide failure article. You should instead plan for reliability in your function apps.</p> <p>You can also refer to commonly used disaster recovery techniques for function apps in a Dedicated plan.</p>
Azure Functions: Flex Consumption, Consumption, and Premium plans	<p>Function apps that run in a Flex Consumption plan, in a Consumption plan, or in a Premium plan can't use custom or automatic backup functionality in App Service. In these dynamic scale plans, function app content is maintained in Azure Storage. Use Azure Storage redundancy options to ensure your storage account meets its availability and durability targets during an outage.</p> <p>You can also download your existing function app project as a .zip file from the Azure portal.</p>	<p>We strongly encourage you to plan for reliability in your function apps.</p>

Automatic vs. custom backups

There are two types of backups in App Service. Automatic backups are created for your app regularly as long as it's in a supported pricing tier. Custom backups require initial configuration and can be made on-demand or on a schedule. The following table shows the differences between the two types.

[] [Expand table](#)

Feature	Automatic backups	Custom backups
Pricing tiers	Basic, Standard, Premium, Isolated.	Basic, Standard, Premium, Isolated.
Configuration required	No.	Yes.
Backup size	30 GB.	10 GB, 4 GB of which can be the linked database.
Linked database	Not backed up.	The following linked databases can be backed up: SQL Database , Azure Database for MySQL , Azure Database for PostgreSQL , MySQL in-app .
Storage account required	No.	Yes.
Backup frequency	Hourly, not configurable.	Configurable.
Retention	30 days, not configurable. - Days 1-3: hourly backups retained. - Days 4-14: every third hourly backup retained. - Days 15-30: every sixth hourly backup retained.	0-30 days or indefinite.
Downloadable	No.	Yes, as Azure Storage blobs.
Partial backups	Not supported.	Supported.
Backups over a virtual network	Not supported.	Supported.

Restore a backup

Note

App Service stops the target app or target slot while restoring a backup. To minimize downtime for a production app, restore the backup to a [deployment slot](#) first, then [swap](#) into production.

1. In your app management page in the [Azure portal](#), in the left menu, select **Backups**. The **Backups** page lists all the automatic and custom backups for your app and the status of each.

my-demo-app | Backups

Oldest backup: 09/04/2022 Automatic backup: Every 1 hour

Type: All Status: All Time range: None Add filter Reset

Showing 10 of 206 results

Backup time ↓	Status ↑	Type ↑	Restore
09/05/2022, 14:52:19	✓ Succeeded	Automatic	
09/05/2022, 14:47:15	✓ Succeeded	Custom	
09/05/2022, 14:36:31	✗ Failed	Custom	
09/05/2022, 14:20:20	✗ Failed	Custom	
09/05/2022, 13:52:19	✓ Succeeded	Automatic	
09/05/2022, 12:52:19	✓ Succeeded	Automatic	
09/05/2022, 11:52:19	✓ Succeeded	Automatic	

2. Select the automatic backup or custom backup to restore by selecting its **Restore** link.

Backup time ↓	Status ↑	Type ↑	Restore
09/05/2022, 14:52:19	✓ Succeeded	Automatic	

3. The **Backup details** section is automatically populated for you.

4. Specify the restore destination in **Choose a destination**. To restore to a new app, select **Create new** under the **App Service** box. To restore to a new **deployment slot**, select **Create new** under the **Deployment slot** box.

If you choose an existing slot, all existing data in its file system is erased and overwritten. The production slot has the same name as the app name.

5. You can choose to restore your site configuration under **Advanced options**.

6. Select **Restore**.

Create a custom backup

1. On your app management page in the [Azure portal](#), in the left menu, select **Backups**.

Backup time ↓	Status ↑	Type ↑	Restore
09/05/2022, 14:52:19	✓ Succeeded	Automatic	⟳
09/05/2022, 14:47:15	✓ Succeeded	Custom	⟳
09/05/2022, 14:36:31	✗ Failed	Custom	⟳
09/05/2022, 14:20:20	✗ Failed	Custom	⟳
09/05/2022, 13:52:19	✓ Succeeded	Automatic	⟳
09/05/2022, 12:52:19	✓ Succeeded	Automatic	⟳
09/05/2022, 11:52:19	✓ Succeeded	Automatic	⟳

2. At the top of the **Backups** page, select **Configure custom backups**.
3. In **Storage account**, select an existing storage account (in the same subscription) or select **Create new**. Do the same thing in **Container**.

To back up the linked databases, select **Next: Advanced > Include database**, and select the databases to backup.

(!) Note

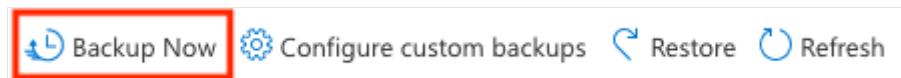
For a supported database to appear in this list, its connection string must exist in the **Connection strings** section of the **Configuration** page for your app.

In-app MySQL databases are always backed up without any configuration. If you make settings for in-app MySQL databases manually, such as adding connection strings, the backups might not work correctly.

4. Select **Configure**.

Once the storage account and container is configured, you can initiate an on-demand backup at any time. On-demand backups are retained indefinitely.

5. At the top of the **Backups** page, select **Backup Now**.



The custom backup is displayed in the list with a progress indicator. If it fails with an error, you can select the line item to see the error message.

Configure custom scheduled backups

1. On the [Configure custom backups](#) page, select **Set schedule**.
2. Configure the backup schedule as desired and then select **Configure**.

Back up and restore a linked database

Custom backups can include linked databases (except when the backup is configured over Azure Virtual Network). To make sure your backup includes a linked database, do the following:

1. Make sure the linked database is [supported](#).
2. Create a connection string that points to your database. A database is considered "linked" to your app when there's a valid connection string for it in your app's configuration.
3. Follow the steps in [Create a custom backup](#) to select the linked database in the **Advanced** tab.

To restore a database that's included in a custom backup:

1. Follow the steps in [Restore a backup](#).
2. In **Advanced options**, select **Include database**.

For troubleshooting information, see [Why is my linked database not backed up?](#).

Back up and restore over Azure Virtual Network

With [custom backups](#), you can back up your app's files and configuration data to a firewall-protected storage account if the following requirements are fulfilled:

- The app is [integrated with a virtual network](#), or the app is in a v3 [App Service Environment](#).
- The storage account [allows access from the virtual network](#) that the app is integrated with, or that the v3 App Service Environment is created with.

To back up and restore over Azure Virtual Network:

1. When configuring [custom backups](#), select **Backup/restore over virtual network integration**.

2. Save your settings by selecting **Configure**.

If you don't see the checkbox, or if the checkbox is disabled, verify that your resources fulfill the requirements.

Once the configuration is saved, any manual backup, scheduled backup, or restore is made through the virtual network. If you make changes to the app, the virtual network, or the storage account that prevent the app from accessing the storage account through the virtual network, the backup or restore operations fail.

Configure partial backups

Partial backups are supported for custom backups (but not for automatic 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.

To exclude folders and files from being stored in your future backups, create a `_backup.filter` file in the [%HOME%\site\wwwroot folder](#) of your app. Specify the list of files and folders you want to exclude in this file.

Tip

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, say you want to filter out the highlighted folder and files.

	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 the root `%HOME%`. List one directory or file per line. The content of the file should be:

```
\site\wwwroot\Images\brand.png
\site\wwwroot\Images\2014
\site\wwwroot\Images\2013
```

Upload the `_backup.filter` file to the `D:\home\site\wwwroot\` directory of your site by using [FTP](#) or any other method. If you want, you can create the file directly by using Kudu [DebugConsole](#) and insert the content there.

Run backups the same way you would normally do it: [custom on-demand](#) or [custom scheduled](#). Any files and folders that are specified in `_backup.filter` are excluded from the future backups.

Note

`_backup.filter` changes the way a restore works. Without `_backup.filter`, restoring a backup deletes all existing files in the app and replaces them with the files in the backup. With `_backup.filter`, any content in the app's file system that's included in `_backup.filter` is left as is (not deleted).

How backups are stored

After you make 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 through 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 that's 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.

Error messages

The **Backups** page shows you the status of each backup. To get log details regarding a failed backup, select the line item in the list. Use the following table to help you troubleshoot your backup. If the failure isn't documented in the table, open a support ticket.

 Expand table

Error	Fix
Storage access failed.	Delete the 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 the database connection string.
Cannot resolve {0}. {1} (CannotResolveStorageAccount)	Delete the backup schedule and reconfigure it.

Error	Fix
Login failed for user '{0}'.	Update the database connection string.
Create Database copy of {0} ({1}) threw an exception. Could not create Database copy.	Use an administrative user in the connection string.
The server principal "<name>" is not able to access the database "master" under the current security context. Cannot open database "master" requested by the login. The login failed. Login failed for user '<name>'.	Use an administrative user in the connection string.
A network-related or instance-specific error occurred while establishing a connection to SQL Server. The server was not found or was not accessible. Verify that the instance name is correct and that SQL Server is configured to allow remote connections. (provider: Named Pipes Provider, error: 40 - Could not open a connection to SQL Server).	Ensure 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.	Ensure 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. Specify SSL options and retry when trying to connect.	SSL connectivity to Azure Database for MySQL and Azure Database for PostgreSQL isn't supported for database backups. Use the native backup feature in the respective database instead.

Automate with scripts

You can automate backup management with scripts by using [Azure CLI](#) or [Azure PowerShell](#).

For samples, see:

- [Azure CLI samples](#).
- [Azure PowerShell samples](#).

Frequently asked questions

- [Are the backups incremental updates or complete backups?](#)

- Does Azure Functions support automatic backups?
- What's included in an automatic backup?
- What's included in a custom backup?
- Why is my linked database not backed up?
- What happens if the backup size exceeds the allowable maximum?
- Can I use a storage account that has security features enabled?
- How do I restore to an app in a different subscription?
- How do I restore to an app in the same subscription but in a different region?
- Where are the automatic backups stored?
- How do I stop an automatic backup?

Are the backups incremental updates or complete backups?

Each backup is a complete offline copy of your app, not an incremental update.

Does Azure Functions support automatic backups?

Automatic backups are available for Azure Functions in [dedicated \(App Service\) Basic](#), [Standard](#), and [Premium](#) tiers. Automatic backups aren't supported for function apps in the [Consumption](#) or [Elastic Premium](#) pricing tiers.

What's included in an automatic backup?

The following table shows which content is backed up in an automatic backup:

[] [Expand table](#)

Content	Restored?
Windows apps: All app content under the <code>%HOME%</code> directory. Linux apps: All app content under the <code>/home</code> directory. Custom containers (Windows and Linux): Content in persistent storage .	Yes
Content of the run-from-ZIP package	No
Content from any custom-mounted Azure storage , such as from an Azure Files share	No

The following table shows which app configurations are restored when you choose to restore app configurations:

[] [Expand table](#)

Settings	Restored?
Native log settings, including the Azure Storage account and container settings	Yes
Application Insights configuration	Yes
Health check	Yes
Network features, such as private endpoints , hybrid connections , and virtual network integration	No
Authentication	No
Managed identities	No
Custom domains	No
TLS/SSL	No
Scale out	No
Diagnostics with Azure Monitor	No
Alerts and Metrics	No
Backup	No
Associated deployment slots	No
Any linked database that custom backup supports	No

What's included in a custom backup?

A custom backup (on-demand backup or scheduled backup) includes all content and configuration that's included in an [automatic backup](#), plus any linked database, up to the allowable maximum size.

When [backing up over Azure Virtual Network](#), you can't [back up the linked database](#).

Why is my linked database not backed up?

Linked databases are backed up only for custom backups, up to the allowable maximum size. If the maximum backup size (10 GB) or the maximum database size (4 GB) is exceeded, your backup fails. Here are a few common reasons why your linked database isn't backed up:

- Backup of [TLS-enabled Azure Database for MySQL](#) isn't supported. If a backup is configured, you get backup failures.

- Backup of [TLS-enabled Azure Database for PostgreSQL](#) isn't supported. If a backup is configured, you get backup failures.
- In-app MySQL databases are automatically backed up without any configuration. If you make manual settings for in-app MySQL databases, such as adding connection strings, the backups might not work correctly.

What happens if the backup size exceeds the allowable maximum?

Automatic backups can't be restored if the backup size exceeds the maximum size. Similarly, custom backups fail if the maximum backup size or the maximum database size is exceeded. To reduce your storage size, consider moving files like logs, images, audio, and videos to Azure Storage, for example.

Can I use a storage account that has security features enabled?

You can back up to a firewall-protected storage account if it's part of the same virtual network topology as your app. See [Back up and restore over Azure Virtual Network](#).

How do I restore to an app in a different subscription?

1. Make a custom backup to an Azure Storage container.
2. [Download the backup ZIP file](#) to your local machine.
3. On the **Backups** page for your target app, select **Restore** in the top menu.
4. In **Backup details**, select **Storage** in **Source**.
5. Select the preferred storage account.
6. In **Zip file**, select **Upload file**.
7. In **Name**, select **Browse** and select the downloaded ZIP file.
8. Configure the rest of the sections as described in [Restore a backup](#).

How do I restore to an app in the same subscription but in a different region?

The steps are the same as in [How do I restore to an app in a different subscription?](#).

Where are the automatic backups stored?

Automatic backups are stored in the same datacenter as the App Service. They shouldn't be relied upon as your disaster recovery plan.

How do I stop an automatic backup?

You can't stop automatic backups. The automatic backup is stored on the platform and has no effect on the underlying app instance or its storage.

Next step

[Azure Blob Storage documentation](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

Azure App Service App Cloning Using PowerShell

Article • 11/15/2021

ⓘ Note

We recommend that you use the Azure Az PowerShell module to interact with Azure. See [Install Azure PowerShell](#) to get started. To learn how to migrate to the Az PowerShell module, see [Migrate Azure PowerShell from AzureRM to Az](#).

With the release of Microsoft Azure PowerShell version 1.1.0, a new option has been added to `New-AzWebApp` that lets you clone an existing App Service app to a newly created app in a different region or in the same region. This option enables customers to deploy a number of apps across different regions quickly and easily.

App cloning is supported for Standard, Premium, Premium V2, and Isolated app service plans. The new feature uses the same limitations as App Service Backup feature, see [Back up an app in Azure App Service](#).

Cloning an existing app

Scenario: An existing app in South Central US region, and you want to clone the contents to a new app in North Central US region. It can be accomplished by using the Azure Resource Manager version of the PowerShell cmdlet to create a new app with the `-SourceWebApp` option.

Knowing the resource group name that contains the source app, you can use the following PowerShell command to get the source app's information (in this case named `source-webapp`):

PowerShell

```
$srcapp = Get-AzWebApp -ResourceGroupName SourceAzureResourceGroup -Name source-webapp
```

To create a new App Service Plan, you can use `New-AzAppServicePlan` command as in the following example

PowerShell

```
New-AzAppServicePlan -Location "North Central US" -ResourceGroupName  
DestinationAzureResourceGroup -Name DestinationAppServicePlan -Tier Standard
```

Using the `New-AzWebApp` command, you can create the new app in the North Central US region, and tie it to an existing App Service Plan. Moreover, you can use the same resource group as the source app, or define a new resource group, as shown in the following command:

PowerShell

```
$destapp = New-AzWebApp -ResourceGroupName DestinationAzureResourceGroup -  
Name dest-webapp -Location "North Central US" -AppServicePlan  
DestinationAppServicePlan -SourceWebApp $srcapp
```

To clone an existing app including all associated deployment slots, you need to use the `IncludeSourceWebAppSlots` parameter. Note that the `IncludeSourceWebAppSlots` parameter is only supported for cloning an entire app including all of its slots. The following PowerShell command demonstrates the use of that parameter with the `New-AzWebApp` command:

PowerShell

```
$destapp = New-AzWebApp -ResourceGroupName DestinationAzureResourceGroup -  
Name dest-webapp -Location "North Central US" -AppServicePlan  
DestinationAppServicePlan -SourceWebApp $srcapp -IncludeSourceWebAppSlots
```

To clone an existing app within the same region, you need to create a new resource group and a new app service plan in the same region, and then use the following PowerShell command to clone the app:

PowerShell

```
$destapp = New-AzWebApp -ResourceGroupName NewAzureResourceGroup -Name dest-  
webapp -Location "South Central US" -AppServicePlan NewAppServicePlan -  
SourceWebApp $srcapp
```

Cloning an existing App to an App Service Environment

Scenario: An existing app in South Central US region, and you want to clone the contents to a new app to an existing App Service Environment (ASE).

Knowing the resource group name that contains the source app, you can use the following PowerShell command to get the source app's information (in this case named `source-webapp`):

PowerShell

```
$srcapp = Get-AzWebApp -ResourceGroupName SourceAzureResourceGroup -Name source-webapp
```

Knowing the ASE's name, and the resource group name that the ASE belongs to, you can create the new app in the existing ASE, as shown in the following command:

PowerShell

```
$destapp = New-AzWebApp -ResourceGroupName DestinationAzureResourceGroup -Name dest-webapp -Location "North Central US" -AppServicePlan DestinationAppServicePlan -ASEName DestinationASE -ASEResourceGroupName DestinationASEResourceGroupName -SourceWebApp $srcapp
```

The `Location` parameter is required due to legacy reason, but it is ignored when you create the app in an ASE.

Cloning an existing App Slot

Scenario: You want to clone an existing deployment slot of an app to either a new app or a new slot. The new app can be in the same region as the original app slot or in a different region.

Knowing the resource group name that contains the source app, you can use the following PowerShell command to get the source app slot's information (in this case named `source-appslot`) tied to `source-app`:

PowerShell

```
$srcappslot = Get-AzWebAppSlot -ResourceGroupName SourceAzureResourceGroup -Name source-app -Slot source-appslot
```

The following command demonstrates creating a clone of the source app to a new app:

PowerShell

```
$destapp = New-AzWebApp -ResourceGroupName DestinationAzureResourceGroup -Name dest-app -Location "North Central US" -AppServicePlan DestinationAppServicePlan -SourceWebApp $srcappslot
```

Configuring Traffic Manager while cloning an app

Creating multi-region apps and configuring Azure Traffic Manager to route traffic to all these apps, is an important scenario to ensure that customers' apps are highly available. When cloning an existing app, you have the option to connect both apps to either a new traffic manager profile or an existing one. Only Azure Resource Manager version of Traffic Manager is supported.

Creating a new Traffic Manager profile while cloning an app

Scenario: You want to clone an app to another region, while configuring an Azure Resource Manager traffic manager profile that includes both apps. The following command demonstrates creating a clone of the source app to a new app while configuring a new Traffic Manager profile:

PowerShell

```
$destapp = New-AzWebApp -ResourceGroupName DestinationAzureResourceGroup -  
Name dest-webapp -Location "South Central US" -AppServicePlan  
DestinationAppServicePlan -SourceWebApp $srcapp -TrafficManagerProfileName  
newTrafficManagerProfile
```

Adding new cloned app to an existing Traffic Manager profile

Scenario: You already have an Azure Resource Manager traffic manager profile and want to add both apps as endpoints. To do so, you first need to assemble the existing traffic manager profile ID. You need the subscription ID, the resource group name, and the existing traffic manager profile name.

PowerShell

```
$TMProfileID = "/subscriptions/<Your subscription ID goes  
here>/resourceGroups/<Your resource group name goes  
here>/providers/Microsoft.TrafficManagerProfiles/ExistingTrafficManagerProfi  
leName"
```

After having the traffic manger ID, the following command demonstrates creating a clone of the source app to a new app while adding them to an existing Traffic Manager profile:

PowerShell

```
$destapp = New-AzWebApp -ResourceGroupName <Resource group name> -Name dest-webapp -Location "South Central US" -AppServicePlan DestinationAppServicePlan -SourceWebApp $srcapp -TrafficManagerProfileId $TMProfileID
```

ⓘ Note

If you are receiving an error that states, "SSL validation on the traffic manager hostname is failing" then we suggest you use -IgnoreCustomHostNames attribute in the powershell cmdlet while performing the clone operation or else use the portal.

Current Restrictions

Here are the known restrictions of app cloning:

- Auto scale settings are not cloned
- Backup schedule settings are not cloned
- VNET settings are not cloned
- App Insights are not automatically set up on the destination app
- Easy Auth settings are not cloned
- Kudu Extension are not cloned
- TiP rules are not cloned
- Database content is not cloned
- Outbound IP Addresses changes if cloning to a different scale unit
- Not available for Linux Apps
- Managed Identities are not cloned
- Not available for Function Apps

References

- [App Service Cloning](#)
- [Back up an app in Azure App Service](#)
- [Azure Resource Manager support for Azure Traffic Manager Preview](#)
- [Introduction to App Service Environment](#)

- Using Azure PowerShell with Azure Resource Manager

Restore Deleted App Service App Using PowerShell

Article • 01/06/2025

If you happened to accidentally delete your app in Azure App Service, you can restore it using the commands from the [Az PowerShell module](#).

ⓘ Note

- Deleted apps are purged from the system 30 days after the initial deletion.
After an app is purged, it can't be recovered.
- Undelete functionality isn't supported for function apps hosted on the Consumption plan or Elastic Premium plan.

Re-register App Service resource provider

Some customers might come across an issue where retrieving the list of deleted apps fails. To resolve the issue, run the following command:

PowerShell

```
Register-AzResourceProvider -ProviderNamespace "Microsoft.Web"
```

List deleted apps

To get the collection of deleted apps, you can use `Get-AzDeletedWebApp`.

For details on a specific deleted app you can use:

PowerShell

```
Get-AzDeletedWebApp -Name <your_deleted_app> -Location  
<your_deleted_app_location>
```

The detailed information includes:

- DeletedSitelId:** Unique identifier for the app, used for scenarios where multiple apps with the same name have been deleted

- **SubscriptionID**: Subscription containing the deleted resource
- **Location**: Location of the original app
- **ResourceGroupName**: Name of the original resource group
- **Name**: Name of the original app.
- **Slot**: the name of the slot.
- **Deletion Time**: When was the app deleted

Restore deleted app

! Note

- `Restore-AzDeletedWebApp` isn't supported for function apps hosted on the Consumption plan or Elastic Premium plan.
- The `Restore-AzDeletedWebApp` cmdlet restores a deleted web app. The web app specified by `TargetResourceGroupName`, `TargetName`, and `TargetSlot` will be overwritten with the contents and settings of the deleted web app. If the target parameters are not specified, they will automatically be filled with the deleted web app's resource group, name, and slot. If the target web app does not exist, it will automatically be created in the app service plan specified by `TargetAppServicePlanName`.
- By default `Restore-AzDeletedWebApp` will restore both your app configuration as well any content. If you want to only restore content, you use the `-RestoreContentOnly` flag with this commandlet.
- Custom domains, bindings, or certs that you import to your app won't be restored. You'll need to re-add them after your app is restored.

After identifying the app you want to restore, you can restore it using `Restore-AzDeletedWebApp`, as shown in the following examples.

You can find the full commandlet reference here: [Restore-AzDeletedWebApp](#) .

Restore to the original app name:

PowerShell

```
Restore-AzDeletedWebApp -TargetResourceGroupName <my_rg> -Name <my_app> -  
TargetAppServicePlanName <my_asp>
```

Restore to a different app name:

PowerShell

```
Restore-AzDeletedWebApp -ResourceGroupName <original_rg> -Name  
<original_app> -TargetResourceGroupName <target_rg> -TargetName <target_app>  
-TargetAppServicePlanName <target_asp>
```

Restore a slot to target app:

PowerShell

```
Restore-AzDeletedWebApp -TargetResourceGroupName <my_rg> -Name <my_app> -  
TargetAppServicePlanName <my_asp> -Slot <original_slot>
```

① Note

Deployment slots are not restored as part of your app. If you need to restore a staging slot, use the `-Slot <slot-name>` flag. The commandlet is restoring original slot to the target app's production slot. By default `Restore-AzDeletedWebApp` will restore both your app configuration as well any content to target app. If you want to only restore content, you use the `-RestoreContentOnly` flag with this commandlet.

Restore only site content to the target app

PowerShell

```
Restore-AzDeletedWebApp -TargetResourceGroupName <my_rg> -Name <my_app> -  
TargetAppServicePlanName <my_asp> -RestoreContentOnly
```

Restore used for scenarios where multiple apps with the same name have been deleted with `-DeletedSiteId`

PowerShell

```
Restore-AzDeletedWebApp -ResourceGroupName <original_rg> -Name  
<original_app> -DeletedId /subscriptions/aaaa0a0a-bb1b-cc2c-dd3d-  
eeeeee4e4e4e/providers/Microsoft.Web/locations/location/deletedSites/1234 -
```

```
TargetAppServicePlanName <my_asp>
```

The inputs for command are:

- **Target Resource Group:** Target resource group where the app is to be restored
- **TargetName:** Target app for the deleted app to be restored to
- **TargetAppServicePlanName:** App Service plan linked to the app
- **Name:** Name for the app, should be globally unique.
- **ResourceGroupName:** Original resource group for the deleted app, you can get it from `Get-AzDeletedWebApp -Name <your_deleted_app> -Location <your_deleted_app_location>`
- **Slot:** Slot for the deleted app
- **RestoreContentOnly:** By default `Restore-AzDeletedWebApp` restores both your app configuration as well any content. If you want to only restore content, you can use the `-RestoreContentOnly` flag with this commandlet.

ⓘ Note

If the app was hosted on and then deleted from an App Service Environment, it can be restored only if the corresponding App Service Environment still exists.

Restore deleted function app

If the function app was hosted on a **Dedicated app service plan**, it can be restored, as long as it was using the default App Service storage.

1. Fetch the DeletedSiteId of the app version you want to restore, using `Get-AzDeletedWebApp` cmdlet:

PowerShell

```
Get-AzDeletedWebApp -ResourceGroupName <RGofDeletedApp> -Name <NameofApp>
```

2. Create a new function app in a Dedicated plan. Refer to the instructions for [how to create an app in the portal](#).
3. Restore to the newly created function app using this cmdlet:

PowerShell

```
Restore-AzDeletedWebApp -ResourceGroupName <RGofnewapp> -Name <newApp> -  
deletedId
```

```
"/subscriptions/xxxx/providers/Microsoft.Web/locations/xxxx/deletedSites/xxx  
x"
```

Currently there's no support for Undelete (Restore-AzDeletedWebApp) Function app that's hosted in a Consumption plan or Elastic premium plan since the content resides on Azure Files in a Storage account. If you haven't 'hard' deleted that Azure Files storage account, or if the account exists and file shares haven't been deleted, then you may use the steps as workaround:

1. Create a new function app in a Consumption or Premium plan. Refer the instructions for [how to create an app in the portal](#).
2. Set the following [app settings](#) to refer to the old storage account , which contains the content from the previous app.

 [Expand table](#)

App Setting	Suggested value
AzureWebJobsStorage	Connection String for the storage account used by the deleted app.
WEBSITE_CONTENTAZUREFILECONNECTIONSTRING	Connection String for the storage account used by the deleted app.
WEBSITE_CONTENTSHARE	File share on storage account used by the deleted app.

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

Relocate Azure App Services to another region

Article • 08/22/2024

This article describes how to move App Service resources to a different Azure region.

There are various reasons why you may want to move your existing Azure resources from one region to another. You may want to:

- Take advantage of a new Azure region.
- Deploy features or services available in specific regions only.
- Meet internal policy and governance requirements.
- Align with company mergers and acquisitions
- Meet capacity planning requirements.

App Service resources are region-specific and can't be moved across regions. You must create a copy of your existing App Service resources in the target region, then relocate your content over to the new app. If your source app uses a custom domain, you can [migrate it to the new app in the target region](#) after completion of the relocation.

To make copying your app easier, you can [backup and restore individual App Service app](#) into an App Service plan in another region.

Prerequisites

- Make sure that the App Service app is in the Azure region from which you want to move.
- Make sure that the target region supports App Service and any related service, whose resources you want to move.
- Validate that sufficient permission exist to deploy App Service resources to the target subscription and region.
- Validate if any Azure policy is assigned with a region restriction.
- Consider any operating costs, as Compute resource prices can vary from region to region. To estimate your possible costs, see [Pricing calculator](#).

Prepare

Identify all the App Service resources that you're currently using. For example:

- App Service apps

- App Service plans
- Deployment slots
- Custom domains purchased in Azure
- TLS/SSL certificates
- Azure Virtual Network integration
- Hybrid connections.
- Managed identities
- Backup settings

Certain resources, such as imported certificates or hybrid connections, contain integration with other Azure services. For information on how to move those resources across regions, see the [documentation for the respective services](#).

Plan

This section is a planning checklist in the following areas:

- State, Storage and downstream dependencies
- Certificates
- Configuration
- VNet Connectivity / Custom Names / DNS
- Identities
- Service Endpoints

State, storage, and downstream dependencies

- **Determine whether your App Service App is stateful or stateless.** Although we recommend that App Service Apps are stateless and the files on the `%HOME%\site` drive should be only those that are required to run the deployed application with any temporary files, it's still possible to store runtime application state on the `%HOME%\site` virtual drive. If your application writes state on the app shared storage path, make sure to plan how you're going to manage that state during a resource move.

Tip

You can use Kudu to, along with portal access, to provide a file access API (Virtual File System (VFS)) that can read/write files under the `%HOME%\site` directory. For more information, see [Kudu Wiki](#).

- **Check for internal caching and state** in application code.
- **Disable session affinity setting.** Where possible, we recommend that you disable the session affinity setting. Disabling session affinity improves load balancing for a horizontal scale-out. Any internal state may impact the planning for cutting over a workload - particularly if zero down time is a requirement. Where possible, it may be beneficial to refactor out any application state to make the application stateless in preparation for the move.
- **Analyze database connection strings.** Database connection strings can be found in the App Settings. However, they may also be hard coded or managed in config files that are shipped with the application. Analyze and plan for data migration/replication as part of the higher level planning to move the workload. For chatty or Latency Critical Applications it isn't performant for the application in the target region to reach back to data sources in the source region.
- **Analyze external caching (for example Redis).** Application caches should be deployed as close as possible to the application. Analyze how caches are populated, expiry/eviction policies and any impact a cold cache may have on the first users to access the workload after cut-over.
- **Analyze and plan for API (or application) dependencies** Cross-region communication is significantly less performant if the app in the target region reaches back to dependencies that are still in the source region. We recommend that you relocate all downstream dependencies as part of the workload relocation. However, *on-premises* resources are the exception, in particular those resources that are geographically closer to the target region (as may be the case for repatriation scenarios).

Azure Container Registry can be a downstream (runtime) dependency for App Service that's configured to run against Custom Container Images. It makes more sense for the Container Registry to be in the same region as the App itself. Consider uploading the required images to a new ACR in the target get region. Otherwise, consider using the [geo-replication feature](#) if you plan on keeping the images in the source region.

- **Analyze and plan for regional services.** Application Insights and Log Analytics data are regional services. Consider the creation of new Application Insights and Log Analytics storage in the target region. For App Insights, a new resource also impacts the connection string that must be updated as part of the change in App Configuration.

Certificates

App Service certificate resources can be moved to a new resource group or subscription but not across regions. Certificates that can be exported can also be imported into the app or into Key Vault in the new region. This export and import process is equivalent to a move between regions.

There are different types of certificates that need to be taken into consideration as you plan your service relocation:

[+] Expand table

Certificate type	Exportable	Comments
App Service managed	No	Recreate these certificates in the new region.
Azure Key Vault managed	Yes	These certificates can be exported from Key Vault and then imported into Key Vault in the new region.
Private key (self-managed)	Yes	Certificates you acquired outside of Azure can be exported from App Service and then imported either into the new app or into Key Vault in the new region.
Public key	No	Your app might have certificates with only a public key and no secret, which are used to access other secured endpoints. Obtain the required public key certificate files and import them into the app in the new region.

Some further points to consider:

- App Assigned Addresses, where an App Service app's SSL connection is bound to a specific app designated IP, can be used for allow-listing calls from third party networks into App Service. For example, a network / IT admin may want to lock down outbound calls from an on-premises network or VNet to use a static, well-known address. As such, if the App Assigned Address feature is in use, upstream firewall rules - such as internal, external, or third parties - for the callers into the app should be checked and informed of the new address. Firewall rules can be internal, external or third parties, such as partners or well-known customers.
- Consider any upstream Network Virtual Appliance (NVA) or Reverse Proxy. The NVA config may need to change if you're rewriting the host header or and/or SSL terminating.

Note

App Service Environment is the only App Service offering allows downstream calls to downstream application dependencies over SSL, where the SSL relies on self-signed/PKI with built with [non-standard Root CA certificates](#). The multitenant service doesn't provide access for customers to upload to the trusted certificate store.

App Service Environment today doesn't allow SSL certificate purchase, only Bring Your Own certificates. IP-SSL isn't possible (and doesn't make sense), but SNI is. Internal App Service Environment would not be associated with a public domain and therefore the SSL certs used must be provided by the customer and are therefore transportable, for example certs for internal use generated using PKI. App Service Environment v3 in external mode shares the same features as the regular multitenant App Service.

Configuration

- You can capture a snapshot of the existing app settings and connection strings from the Azure portal. Expand **Settings > Environment variables**, select **Advanced edit** under either **App settings** or **Connections strings** and save the JSON output that contains the existing settings or connections. You need to recreate these settings in the new region, but the values themselves are likely to change as a result of subsequent region changes in the connected services.
- Existing [Key Vault references](#) can't be exported across an Azure geographical boundary. You must recreate any required references in the new region.
- Your app configuration might be managed by [Azure App Configuration](#) or from some other central (downstream) database dependency. Review any App Configuration store or similar stores for environment and region-specific settings that might require modifications.
- Make sure to check any disk file configuration, which may or may not be overridden by application settings.

VNet Connectivity / Custom Names / DNS

- App Service Environment is a VNet-Injected single tenant service. App Service Environment networking differs from the multitenant App Service, which requires one or both "Private Endpoints" or "Regional VNet integration". Other options that

may be in play include the legacy P2S VPN based VNet integration and Hybrid Connections (an Azure Relay service).

ⓘ Note

ASEv3 Networking is simplified - the Azure Management traffic and the App Service Environments own downstream dependencies are not visible to the customer Virtual Network, greatly simplifying the configuration required where the customer is using a force-tunnel for all traffic, or sending a subset of outbound traffic, through a Network Virtual Appliance/Firewall.

Hybrid Connections (Azure Relay) are regional. If Hybrid Connections are used and although an Azure Relay Namespace can be moved to another region, it would be simpler to redeploy the Hybrid Connection (ensure the Hybrid connection is setup in the new region on deploy of the target resources) and re-link it to the Hybrid Connection Manager. The Hybrid Connection Manager location should be carefully considered.

- **Follow the strategy for a warm standby region.** Ensure that core networking and connectivity, Hub network, domain controllers, DNS, VPN or Express Route, etc., are present and tested prior to the resource relocation.
- **Validate any upstream or downstream network ACLs and configuration.** For example, consider an external downstream service that allowlists only your App traffic. A relocation to a new Application Plan for a multitenant App Service would then also be a change in outbound IP addresses.
- In most cases, it's best to **ensure that the target region VNets have unique address space.** A unique address space facilitates VNet connectivity if it's required, for example, to replicate data. Therefore, in these scenarios there's an implicit requirement to change:
 - Private DNS
 - Any hard coded or external configuration that references resources by IP address (without a hostname)
 - Network ACLs including Network Security Groups and Firewall configuration (consider the impact to any on-premises NVAs here too)
 - Any routing rules, User Defined Route Tables

Also, make sure to check configuration including region specific IP Ranges / Service Tags if carrying forward existing DevOps deployment resources.

- Fewer changes are required for customer-deployed private DNS that is configured to forward to Azure for Azure domains and Azure DNS Private Zones. However, as Private Endpoints are based on a resource FQDN and this is often also the resource name (which can be expected to be different in the target region), remember to **cross check configuration to ensure that FQDNs referenced in configuration are updated accordingly.**
- **Recreate Private Endpoints, if used, in the target region.** The same applies for Regional VNet integration.
- DNS for App Service Environment is typically managed via the customers private custom DNS solution (there is a manual settings override available on a per app basic). App Service Environment provides a load balancer for ingress/egress, while App Service itself filters on Host headers. Therefore, multiple custom names can be pointed towards the same App Service Environment ingress endpoint. App Service Environment doesn't require domain validation.

 **Note**

Kudu endpoint for App Service Environment v3 is only available at `{resourcename}.scm.{asename}.appserviceenvironment.net`. For more information on App Service Environment v3 DNS and Networking etc see [App Service Environment networking](#).

For App Service Environment, the customer owns the routing and therefore the resources used for the cut-over. Wherever access is enabled to the App Service Environment externally - typically via a Layer 7 NVA or Reverse Proxy - Traffic Manager, or Azure Front Door/Other L7 Global Load Balancing Service can be used.

- For the public multitenant version of the service, a default name `{resourcename}.azurewebsites.net` is provisioned for the data plane endpoints, along with a default name for the Kudu (SCM) endpoint. As the service provides a public endpoint by default, the binding must be verified to prove domain ownership. However, once the binding is in place, re-verification isn't required, nor is it required for public DNS records to point at the App Service endpoint.
- If you use a custom domain, [bind it preemptively to the target app](#). Verify and [enable the domain in the target app](#).

Identities

- You need to recreate any system assigned managed identities along with your app in the new target region. Typically, an automatically created Microsoft Entra ID app, used by EasyAuth, defaults to the app resource name.
- User-assigned managed identities also can't be moved across regions. To keep user-assigned managed identities in the same resource group with your app, you must recreate them in the new region. For more information, see [Relocate managed identities for Azure resources to another region](#).
- Grant the managed identities the same permissions in your relocated services as the original identities that they're replacing, including Group memberships.
- **Plan for relocating the Identity Provider (IDP) to the target region.** Although Microsoft Entra ID is a global service, some solutions rely on a local (or downstream on premises) IDP.
- **Update any resources to the App Service that may rely on Kudu FTP credentials.**

Service endpoints

The virtual network service endpoints for Azure App Service restrict access to a specified virtual network. The endpoints can also restrict access to a list of IPv4 (internet protocol version 4) address ranges. Any user connecting to the Event Hubs from outside those sources is denied access. If Service endpoints were configured in the source region for the Event Hubs resource, the same would need to be done in the target one.

For a successful recreation of the Azure App Service to the target region, the VNet and Subnet must be created beforehand. In case the move of these two resources is being carried out with the Azure Resource Mover tool, the service endpoints won't be configured automatically. Hence, they need to be configured manually, which can be done through the [Azure portal](#), the [Azure CLI](#), or [Azure PowerShell](#).

Relocate

To relocate App Service resources, you can use either Azure portal or Infrastructure as Code (IaC).

Relocate using Azure portal

The greatest advantage of using Azure portal to relocate is its simplicity. The app, plan, and contents, as well as many settings are cloned into the new App Service resource and plan.

Keep in mind that for App Service Environment (Isolated) tiers, you need to redeploy the entire App Service Environment in another region first, and then you can start redeploying the individual plans in the new App Service Environment in the new region.

To relocate your App Service resources to a new region using Azure portal:

1. [Create a back up of the source app.](#)
2. [Create an app in a new App Service plan, in the target region.](#)
3. [Restore the back up in the target app](#)
4. If you use a custom domain, [bind it preemptively to the target app](#) with `asuid`.
and [enable the domain in the target app](#).
5. Configure everything else in your target app to be the same as the source app and verify your configuration.
6. When you're ready for the custom domain to point to the target app, [remap the domain name](#).

Relocate using IaC

Use IaC when an existing Continuous Integration and Continuous Delivery/Deployment(CI/CD) pipeline exists, or can be created. With an CI/CD pipeline in place, your App Service resource can be created in the target region by means of a deployment action or a Kudu zip deployment.

SLA requirements should determine how much additional effort is required. For example: Is this a redeploy with limited downtime, or is it a near real time cut-over required with minimal to no downtime?

The inclusion of external, global traffic routing edge services, such as Traffic Manager, or Azure Front Door help to facilitate cut-over for external users and applications.

💡 Tip

It's possible to use Traffic Manager (ATM) when failing over App Services behind private endpoints. Although the private endpoints are not reachable by Traffic Manager Probes - if all endpoints are degraded, then ATM allows routing. For more information, see [Controlling Azure App Service traffic with Azure Traffic Manager](#).

Validate

Once the relocation is completed, test and validate Azure App Service with the recommended guidelines:

- Once the Azure App Service is relocated to the target region, run a smoke and integration test. You can manually test or run a test through a script. Make sure to validate that all configurations and dependent resources are properly linked and that all configured data are accessible.
- Validate all Azure App Service components and integration.
- Perform integration testing on the target region deployment, including all formal regression testing. Integration testing should align with the usual Rhythm of Business deployment and test processes applicable to the workload.
- In some scenarios, particularly where the relocation includes updates, changes to the applications or Azure Resources, or a change in usage profile, use load testing to validate that the new workload is fit for purpose. Load testing is also an opportunity to validate operations and monitoring coverage. For example, use load testing to validate that the required infrastructure and application logs are being generated correctly. Load tests should be measured against established workload performance baselines.

💡 Tip

An App Service relocation is also an opportunity to re-assess Availability and Disaster Recovery planning. App Service and App Service Environment (App Service Environment v3) supports [availability zones](#) and it's recommended that configure with an availability zone configuration. Keep in mind the prerequisites for deployment, pricing, and limitations and factor these into the resource move planning. For more information on availability zones and App Service, see [Reliability in Azure App Service](#).

Clean up

Delete the source app and App Service plan. [An App Service plan in the non-free tier carries a charge, even if no app is running in it.](#)

Next steps

[Azure App Service App Cloning Using PowerShell](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

Move Azure resources to a new resource group or subscription

Article • 01/24/2025

This article explains how to move Azure resources between resource groups within the same subscription or across different subscriptions. If the move involves different subscriptions, both subscriptions must be part of the same Microsoft Entra ID tenant. You can use tools like the [Azure portal](#), [Azure PowerShell](#), [Azure CLI](#), the [REST API](#), or [Python](#) to move the resources.

During the move operation, both the source and target resource groups are locked, meaning that you can't create, delete, or update resources within these resource groups while the move is in progress. However, existing resources remain fully operational. For example, if you move a virtual machine from one resource group to another, it can't be deleted and its properties (such as its size) can't be modified during the move. Despite this, the virtual machine continues to operate normally, and services relying on it don't experience any additional downtime. The lock can last up to four hours, most moves are completed faster, and the lock is removed accordingly.

Only top-level (parent) resources should be specified in the move request. Child resources move automatically with their parent but can't be moved independently. For example, a parent resource like `Microsoft.Compute/virtualMachines` can be moved, and its child resource such as `Microsoft.Compute/virtualMachines/extensions` moves with it. However, the child resource can't be moved on its own.

While moving a resource preserves its dependencies with child resources, dependencies with other resources can break and might need to be configured again. Moving a resource only changes its associated resource group and doesn't alter the physical region of the resource.

ⓘ Note

Azure resources can't be moved if a read-only lock exists on the source, destination resource group, or subscription.

Changed resource ID

When you move a resource, you change its resource ID. The standard format for a resource ID is

`/subscriptions/{subscriptionId}/resourceGroups/{resourceGroupName}/providers/{resourceProviderNamespace}/{resourceType}/{resourceName}`. When you move a resource to a new resource group or subscription, you change one or more values in that path.

If you use the resource ID anywhere, you need to change that value. For example, if you have a [custom dashboard](#) in the portal that references a resource ID, you need to update that value. Look for any scripts or templates that need to be updated for the new resource ID.

Checklist before moving resources

Some important steps precede moving a resource. You can avoid errors if you verify these conditions.

1. The source and destination subscriptions must be active. If you have trouble enabling an account that's been disabled, [create an Azure support request](#). Select **Subscription Management** for the issue type.
2. The source and destination subscriptions must exist within the same [Microsoft Entra tenant](#). Use the Azure CLI or PowerShell to check that both subscriptions have the same tenant ID.



The screenshot shows the Azure CLI interface with a command history window. The title bar says "Azure CLI". The command history window has the following content:

```
Azure CLI
az account show --subscription <your-source-subscription> --query tenantId
az account show --subscription <your-destination-subscription> --query tenantId
```

If the tenant IDs for the source and destination subscriptions don't match, use the following methods to reconcile them:

- [Transfer billing ownership of an Azure subscription](#).
 - [Associate or add an Azure subscription to your Microsoft Entra tenant](#).
3. If you're attempting to move resources to or from a Cloud Solution Provider (CSP) partner, see [Transfer Azure subscriptions between subscribers and CSPs](#) for more information.
 4. The resources you want to move must support the move operation. See [Azure resource types for move operations](#) for a list of which resources support move

operations.

5. Since some services have specific limitations or requirements when moving resources, check the following move guidance before moving resources within these services:

- If you're using Azure Stack Hub, you can't move resources between groups.
- [Azure App Services](#)
- [Azure DevOps Services](#)
- [Classic deployment model](#) for classic compute, storage, virtual networks, and cloud services
- [Cloud Services \(extended support\)](#)
- [Networking](#)
- [Azure Recovery Services](#)
- [Virtual machines](#)
- See [Move subscriptions](#) to move an Azure subscription to a new management group.

6. The destination subscription must be registered for the resource provider of the resource being moved. If not, you receive an error stating that the **subscription is not registered for a resource type**. You might see this error when moving a resource to a new subscription, but that subscription has never been used with that resource type.

Azure CLI

To get the registration status:

Azure CLI

```
az account set -s <destination-subscription-name-or-id>
az provider list --query "[].{Provider:namespace,
Status:registrationState}" --out table
```

To register a resource provider:

Azure CLI

```
az provider register --namespace Microsoft.Batch
```

7. Check the subscription quota for the subscription to which you're moving resources before starting a move operation. Verify if you can request an increase in

a quota that would cause a destination subscription to exceed its limit. See [Azure subscription and service limits, quotas, and constraints](#) for detailed guidance about limits and how to request an increase.

8. The account moving the resources must have at least the following permissions:

- At the source resource group:
`Microsoft.Resources/subscriptions/resourceGroups/moveResources/action`
- At the destination resource group:
`Microsoft.Resources/subscriptions/resourceGroups/write`

9. If you move a resource with an active Azure role assignment (or its child resource with this same assignment), the role assignment doesn't move and becomes orphaned. You must create the role assignment again after the move. Eventually, the orphaned role assignment is automatically removed, but it's recommended to remove the role assignment before the move.

See [List Azure role assignments](#) and [Assign Azure roles](#) to learn more about how to manage role assignments.

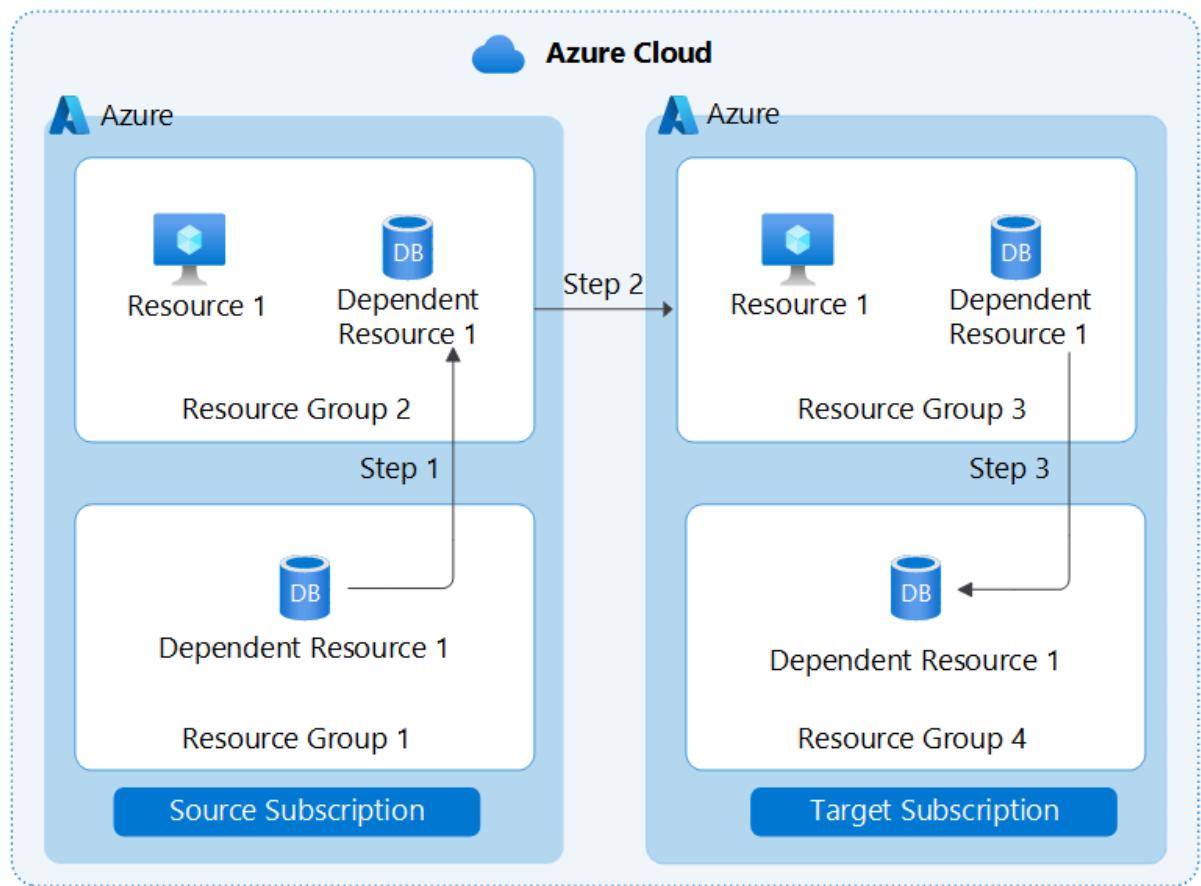
10. **For a move across subscriptions, the resource and its dependent resources must be located in the same resource group and they must be moved together.** For example, a virtual machine with managed disks would require the virtual machine and managed disks move together, along with other dependent resources.

If you're moving a resource to a new subscription, check if the resource has any dependent resources and if they're located in the same resource group. If the resources aren't in the same resource group, check if they can be combined into the same one. If so, use one move operation across resource groups to consolidate all the resources into the same resource group.

Continue to [Scenario for move across subscriptions](#) to learn more.

Scenario for move across subscriptions

Moving resources from one subscription to another is a three-step process. Only one dependent resource is depicted to illustrate these steps:



- Step 1: If dependent resources are distributed across different resource groups, first move them into one resource group.
- Step 2: Move the resource and dependent resources together from the source subscription to the target subscription.
- Step 3: Optionally, redistribute the dependent resources to different resource groups within the target subscription.

Move resources

Use the Azure portal

1. To move resources, select the resource group that contains those resources.
2. Select the resources that you want to move. To move all of the resources, select the checkbox at the top of list. Or, select resources individually.

The screenshot shows the Microsoft Azure portal interface for a resource group named 'sourceGroup'. The left sidebar contains navigation links like Overview, Activity log, Access control (IAM), Tags, Resource visualizer, Events, Settings, Cost Management, Monitoring, and Automation. The main area is titled 'Overview' under 'Essentials'. It shows a list of resources with a filter applied. The filter bar at the top has two entries: 'Type equals all' and 'Location equals all'. Below these, a dropdown menu shows a list of filters, with 'Name ↑' selected. A red box highlights the 'Name' entry in this dropdown. The list of resources includes 'name' and 'storepttx6pwgm7f7w'.

3. Select the **Move** button.

The screenshot shows the Microsoft Azure portal interface for the same 'sourceGroup'. The left sidebar and main overview area are similar to the previous screenshot. However, the ellipsis menu on the right is open, showing various actions: Delete resource group, Refresh, Export to CSV, Open query, Assign tags, Move, Delete, and Export template. The 'Move' option is selected, and a dropdown menu appears below it, listing 'Move to another resource group', 'Move to another subscription', and 'Move to another region'. A red box highlights these three options.

This button gives you three options:

- Move to a new resource group.
- Move to a new subscription.
- Move to a new region. To change regions, see [Move resources across regions \(from resource group\) with Azure Resource Mover](#).

4. Select if you're moving the resources to a new resource group or subscription.

5. The source resource group sets automatically. Specify the destination resource group. If you're moving to a new subscription, specify this also. Select **Next**.

Move resources

sourceGroup

Subscription

: Test 4

Resource group

sourceGroup

Target

Subscription

: Test 4

Resource group *

(New) destinationGroup

Create new

1. The portal validates that the resources can be moved. Wait for validation to complete.

Home > sourceGroup >

Move resources

sourceGroup

1 Source + target

2 Resources to move

3 Review

Checking whether these resources can be moved. This might take a few minutes.

+ Add resources

X Remove from the move list

Name	Type	Resource type	Validation status
exampleVM1	Virtual machine	microsoft.compute/virtualmachines	Pending validation
exampleVM1-ip	Public IP address	microsoft.network/publicipaddresses	Pending validation
exampleVM1-nsg	Network security group	microsoft.network/networksecuritygroups	Pending validation
examplevm1920	Network interface	microsoft.network/networkinterfaces	Pending validation
exampleVM1_OsDisk_1_38427735e90a427l	Disk	microsoft.compute/disks	Pending validation
sourceGroup-vnet	Virtual network	microsoft.network/virtualnetworks	Pending validation

1. When validation completes successfully, select **Next**.

2. Acknowledge that you need to update tools and scripts for these resources. To start moving the resources, select **Move**.

Home > sourceGroup >

Move resources

sourceGroup

1 Source + target 2 Resources to move 3 Review

Selection summary

Source subscription	Documentation Testing 1
Source resource group	sourceGroup
Target subscription	Documentation Testing 1
Target resource group	destinationGroup
Number of resources to move	6

I understand that tools and scripts associated with moved resources will not work until I update them to use new resource IDs

Previous

Move

1. The Azure portal notifies you when the move has completed.

Notifications

X

[More events in the activity log →](#)

[Dismiss all](#) 



Moving resources complete

X

Successfully moved 6 resources from resource group 'sourceGroup' in subscription 'Documentation Testing 1' to resource group 'destinationGroup' in subscription 'Documentation Testing 1'

[Feedback](#)

[Related events](#)

Use the Azure CLI

Validate

To test your move scenario without actually moving resources in real time, use the [az resource invoke-action](#) command. Use this command only when you need to model the results without following through. To run this operation, you need the resource ID of the source resource group, target resource group, and each resource that's moving.

Use `\\"` to escape double quotes in the body of the request.

Azure CLI

```
az resource invoke-action --action validateMoveResources \
--ids "/subscriptions/{subscription-id}/resourceGroups/{source-rg}" \
--request-body "{ \"resources\": [\"/subscriptions/{subscription-
id}/resourceGroups/{source-rg}/providers/{resource-provider}/{resource-
type}/{resource-name}\", \"/subscriptions/{subscription-
id}/resourceGroups/{source-rg}/providers/{resource-provider}/{resource-
type}/{resource-name}\", \"/subscriptions/{subscription-
id}/resourceGroups/{source-rg}/providers/{resource-provider}/{resource-
type}/{resource-name}\"] }, \"targetResourceGroup\": \"/subscriptions/{subscription-
id}/resourceGroups/{destination-rg}\" }"
```

If validation passes, you see:

Azure CLI

```
{} Finished ..
```

If validation fails, you see an error message explaining why the resources won't move.

Move

To move existing resources to another resource group or subscription, use the [az resource move](#) command. In the `--ids` parameter, provide a space-separated list of the resource IDs to move.

The following commands show how to move several resources to a new resource group. They work with the Azure CLI in a Bash terminal or in an Azure PowerShell console. Provide the `--destination-subscription-id` parameter to move resources to a new subscription.

Azure CLI

```
webapp=$(az resource show -g OldRG -n ExampleSite --resource-type
"Microsoft.Web/sites" --query id --output tsv)
plan=$(az resource show -g OldRG -n ExamplePlan --resource-type
"Microsoft.Web/serverfarms" --query id --output tsv)
az resource move --destination-group newgroup --ids $webapp $plan
```

Use Azure PowerShell

Validate

To test your move scenario without actually moving resources in real time, use the [Invoke-AzResourceAction](#) command in Azure PowerShell. Use this command only when you need to model the results without following through.

Azure PowerShell

```
$sourceName = "sourceRG"
$destinationName = "destinationRG"
$resourcesToMove = @("app1", "app2")

$sourceResourceGroup = Get-AzResourceGroup -Name $sourceName
$destinationResourceGroup = Get-AzResourceGroup -Name $destinationName

$resources = Get-AzResource -ResourceGroupName $sourceName | Where-Object {
    $_.Name -in $resourcesToMove }

Invoke-AzResourceAction -Action validateMoveResources ` 
    -ResourceId $sourceResourceGroup.ResourceId ` 
    -Parameters @{
        resources = $resources.ResourceId; # Wrap in an @() array if providing a
        single resource ID string.
        targetResourceGroup = $destinationResourceGroup.ResourceId
    }
```

An output won't display if the validation succeeds. However, an error message explaining the why the resources won't move will appear if the validation fails.

Move

To move existing resources to another resource group or subscription, use the [Move-AzResource](#) command. The following example shows how to move several resources to a new resource group.

Azure PowerShell

```
$sourceName = "sourceRG"
$destinationName = "destinationRG"
$resourcesToMove = @("app1", "app2")

$resources = Get-AzResource -ResourceGroupName $sourceName | Where-Object {
    $_.Name -in $resourcesToMove }
```

```
Move-AzResource -DestinationResourceGroupName $destinationName -ResourceId  
$resources.ResourceId
```

To move to a new subscription, include a value for the `DestinationSubscriptionId` parameter.

Use Python

Validate

To test your move scenario without actually moving resources in real time, use the `ResourceManagementClient.resources.begin_validate_move_resources` method. Use this method only when you need to model the results without following through.

Python

```
import os
from azure.identity import AzureCliCredential
from azure.mgmt.resource import ResourceManagementClient

credential = AzureCliCredential()
subscription_id = os.environ["AZURE_SUBSCRIPTION_ID"]

resource_client = ResourceManagementClient(credential, subscription_id)

source_name = "sourceRG"
destination_name = "destinationRG"
resources_to_move = ["app1", "app2"]

destination_resource_group =
resource_client.resource_groups.get(destination_name)

resources = [
    resource for resource in
resource_client.resources.list_by_resource_group(source_name)
    if resource.name in resources_to_move
]

resource_ids = [resource.id for resource in resources]

validate_move_resources_result =
resource_client.resources.begin_validate_move_resources(
    source_name,
    {
        "resources": resource_ids,
        "target_resource_group": destination_resource_group.id
    }
).result()
```

```
print("Validate move resources result:  
{}".format(validate_move_resources_result))
```

An output won't display if the validation succeeds. However, an error message explaining the why the resources won't move will appear if the validation fails.

Move

To move existing resources to another resource group or subscription, use the [ResourceManagementClient.resources.begin_move_resources](#) method in Python. The following example shows how to move several resources to a new resource group.

Python

```
import os  
from azure.identity import AzureCliCredential  
from azure.mgmt.resource import ResourceManagementClient  
  
credential = AzureCliCredential()  
subscription_id = os.environ["AZURE_SUBSCRIPTION_ID"]  
  
resource_client = ResourceManagementClient(credential, subscription_id)  
  
source_name = "sourceRG"  
destination_name = "destinationRG"  
resources_to_move = ["app1", "app2"]  
  
destination_resource_group =  
resource_client.resource_groups.get(destination_name)  
  
resources = [  
    resource for resource in  
    resource_client.resources.list_by_resource_group(source_name)  
    if resource.name in resources_to_move  
]  
  
resource_ids = [resource.id for resource in resources]  
  
resource_client.resources.begin_move_resources(  
    source_name,  
    {  
        "resources": resource_ids,  
        "target_resource_group": destination_resource_group.id  
    }  
)
```

Use REST API

Validate

The [validate move operation](#) lets you test your move scenario without actually moving resources. Use this operation to check if the move can succeed. Validation is automatically called when you send a move request. Use this operation only when you need to model the results without following through. To run this operation, you need the:

- Name of the source resource group
- Resource ID of the target resource group
- Resource ID of each resource to move
- The [access token](#) for your account

Send the following request:

HTTP

```
POST https://management.azure.com/subscriptions/<subscription-id>/resourceGroups/<source-group>/validateMoveResources?api-version=2019-05-10
Authorization: Bearer <access-token>
Content-type: application/json
```

With a request body:

JSON

```
{
  "resources": ["<resource-id-1>", "<resource-id-2>"],
  "targetResourceGroup": "/subscriptions/<subscription-id>/resourceGroups/<target-group>"
}
```

If the request is formatted correctly, the operation returns:

HTTP

```
Response Code: 202
cache-control: no-cache
pragma: no-cache
expires: -1
location: https://management.azure.com/subscriptions/<subscription-id>/operationresults/<operation-id>?api-version=2018-02-01
retry-after: 15
...
```

A 202 status code indicates that the validation request was accepted, but it hasn't yet determined if the move operation will succeed. The `location` value contains a URL that you use to check the status of the long-running operation.

To check the status, send the following request:

HTTP

```
GET <location-url>
Authorization: Bearer <access-token>
```

You will continue to receive a 202 status code while the operation runs. Wait the number of seconds indicated in the `retry-after` value before trying again. You will receive a 204 status code if the move validation succeeds. If the move validation fails, you receive an error message that resembles:

JSON

```
{"error": {"code": "ResourceMoveProviderValidationFailed", "message": "<message>"}...}
```

Move

To move existing resources to another resource group or subscription, use the [Move resources](#) operation.

HTTP

```
POST https://management.azure.com/subscriptions/{source-subscription-id}/resourcegroups/{source-resource-group-name}/moveResources?api-version={api-version}
```

Specify the target resource group and resources to move in the body of the request.

JSON

```
{
  "resources": ["<resource-id-1>", "<resource-id-2>"],
  "targetResourceGroup": "/subscriptions/<subscription-id>/resourceGroups/<target-group>"
}
```

Frequently asked questions

My resource move operation that usually takes a few minutes has been running for almost an hour. Is something wrong?

Moving a resource is a complex operation with different phases. It can involve more than just the resource provider of the resource you're trying to move. Azure Resource Manager allows a move operation four hours to finish because of the dependencies between resource providers. This duration gives them time to recover from transient issues. If your move request is within the four-hour period, the operation keeps trying to complete and might succeed. The source and destination resource groups are locked during this time to avoid consistency issues.

Why is my resource group locked for four hours during resource move?

- Move operations are allowed four hours to complete. The source and destination resource groups are locked during this time to prevent them from being modified.
- There are two phases in a move request. Resources move during the first phase, and resource providers that depend on the resources being moved are notified during the second phase. A resource group can be locked for all four hours when a resource provider fails either phase. Resource Manager will initiate any failed steps during the span of the move operation.
- Resource Manager unlocks both resource groups if a resource won't move within four hours. Resources that have moved successfully are in the destination resource group. Resources that failed to move remain in the source resource group.

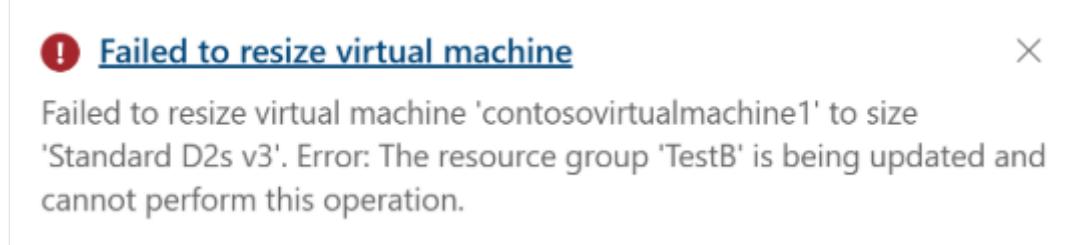
What are the implications of the source and destination resource groups being locked during the resource move?

The lock prevents you from deleting either resource group. Secondly, the lock prevents you from creating a new resource, deleting a resource, or updating a resource's properties within each resource group (e.g., changing a virtual machine's size).

The following image shows an error message from the Azure portal when a user tries to delete a resource group that's part of an ongoing move:



In the image below, the virtual machine resource belongs to a resource group ("TestB") that's currently undergoing a move operation. When a user attempts to update a virtual machine's property (such as its size), the Azure portal returns an error message. This occurs because the resource group is locked during the move, which protects its resources from being modified.



Additionally, neither the source nor the destination resource group can participate in other move operations simultaneously during a resource move. For example, if resources are moving from Resource Group A to Resource Group B, neither Group A nor Group B can be involved in another move operation at the same time (like moving resources to or from Resource Group C). This restriction prevents multiple conflicting operations from locking resource groups during the move process.

What does the error code "MissingMoveDependentResources" mean?

When you move a resource, its dependent resources must exist in the destination resource group or subscription, or be included in the move request. You get the **MissingMoveDependentResources** error code when a dependent resource doesn't meet this requirement. The error message provides details about the dependent resource that needs to be included in the move request.

For example, moving a virtual machine could require moving seven resource types with three different resource providers. Those resource providers and types are:

- Microsoft.Compute
 - virtualMachines
 - disks
- Microsoft.Network
 - networkInterfaces
 - publicIPAddresses
 - networkSecurityGroups
 - virtualNetworks
- Microsoft.Storage
 - storageAccounts

Another common example involves moving a virtual network where you might have to move several other resources associated with that virtual network. The move request could require moving public IP addresses, route tables, virtual network gateways, network security groups, and other resources. A virtual network gateway should be in the same resource group as its virtual network since they can't move separately.

What does the error code "RequestDisallowedByPolicy" mean?

Resource Manager validates your move request before attempting a move. This validation includes checking policies defined for the resources involved in the move. For example, if you're attempting to move a key vault but your organization has a policy to deny a key vault being created in the target resource group, the validation fails and the move is blocked. The returned error code is **RequestDisallowedByPolicy**.

For more information about policies, see [What is Azure Policy?](#).

Why can't I move some resources in Azure?

Not all Azure resources allow move operations at this time.

How many resources can I move in one operation?

Separate large moves into different move operations when possible. Resource Manager immediately returns an error when there are more than 800 resources in a single operation. However, moving less than 800 resources can also fail by timing out.

What is the meaning of the error that a resource isn't in a "succeeded" state?

When you get an error message indicating that a resource won't move because it isn't in a **Succeeded** state, it might be because a dependent resource is blocking the move. Typically, the error code is **MoveCannotProceedWithResourcesNotInSucceededState**.

If the source or target resource group contains a virtual network, the states of all resources that depend on that virtual network are checked during the move; this includes resources that directly and indirectly depend on the network. The move is blocked if any resources have failed. For example, if a virtual machine using a virtual network has failed, the move is blocked. It's also blocked even when the virtual machine isn't one of the resources being moved and isn't in one of the resource groups during the move.

Two solutions for this obstacle are moving your resources to a resource group that doesn't have a virtual network or [contacting support](#).

Can I move a resource group to a different subscription?

No, you can't move a resource group to a new subscription. But, you can move all resources in a resource group to a resource group in another subscription. Settings such as tags, role assignments, and policies don't transfer automatically from the original resource group to the destination resource group. You need to apply these settings manually to the new resource group.

Next steps

Continue to reference [Move operation support for resources](#) to verify which Azure resources support move operations.

 **Note:** The author created this article with assistance from AI. [Learn more](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

Plan and manage costs for Azure App Service

Article • 03/22/2023

This article describes how you plan for and manage costs for Azure App Service. First, you use the Azure pricing calculator to help plan for App Service costs before you add any resources for the service to estimate costs. Next, as you add Azure resources, review the estimated costs. After you've started using App Service resources, use [Cost Management](#) features to set budgets and monitor costs. You can also review forecasted costs and identify spending trends to identify areas where you might want to act. Costs for Azure App Service are only a portion of the monthly costs in your Azure bill. Although this article explains how to plan for and manage costs for App Service, you're billed for all Azure services and resources used in your Azure subscription, including the third-party services.

Understand the full billing model for Azure App Service

Azure App Service runs on Azure infrastructure that accrues costs when you deploy new resources. It's important to understand that there could be other additional infrastructure costs that might accrue.

How you're charged for Azure App Service

When you create or use App Service resources, you're charged for the following meters:

- You're charged an hourly rate based on the pricing tier of your App Service plan, prorated to the second.
- The charge is applied to each scaled-out instance in your plan, based on the amount of time that the VM instance is allocated.

Other cost resources for App Service are (see [App Service pricing](#) for details):

- [App Service domains](#) Your subscription is charged for the domain registration on a yearly basis, if you enable automatic renewal.
- [App Service certificates](#) One-time charge at the time of purchase. If you have multiple subdomains to secure, you can reduce cost by purchasing one wildcard certificate instead of multiple standard certificates.

- **IP-based SSL binding** The binding is configured on a certificate at the app level. Costs are accrued for each binding. For **Standard** tier and above, the first IP-based binding is not charged.

At the end of your billing cycle, the charges for each VM instance. Your bill or invoice shows a section for all App Service costs. There's a separate line item for each meter.

Other costs that might accrue with Azure App Service

Depending on which feature you use in App Service, the following cost-accruing resources may be created:

- **Isolated tier** A [Virtual Network](#) is required for an App Service environment and is charged separately.
- **Backup** A [Storage account](#) is required to make backups and is charged separately.
- **Diagnostic logs** You can select [Storage account](#) as the logging option, or integrate with [Azure Log Analytics](#). These services are charged separately.
- **App Service certificates** Certificates you purchase in Azure must be maintained in [Azure Key Vault](#), which is charged separately.

Costs that might accrue after resource deletion

When you delete all apps in an App Service plan, the plan continues to accrue charges based on its configured pricing tier and number of instances. To avoid unwanted charges, delete the plan or scale it down to **Free** tier.

After you delete Azure App Service resources, resources from related Azure services might continue to exist. They continue to accrue costs until you delete them. For example:

- The Virtual Network that you created for an **Isolated** tier App Service plan
- Storage accounts you created to store backups or diagnostic logs
- Key Vault you created to store App Service certificates
- Log Analytic namespaces you created to ship diagnostic logs
- [Instance or stamp reservations](#) for App Service that haven't expired yet

Using Azure Prepayment with Azure App Service

You can pay for Azure App Service charges with your Azure Prepayment credit. However, you can't use Azure Prepayment credit to pay for charges for third-party products and services, including those from the Azure Marketplace.

Estimate costs

An easy way to estimate and optimize your App Service cost beforehand is by using the [Azure pricing calculator](#).

To use the pricing calculator, click **App Service** in the **Products** tab. Then, scroll down to work with the calculator. The following screenshot is an example and doesn't reflect current pricing.

The screenshot shows the Azure Pricing Calculator interface. At the top, it displays "Your Estimate" with four icons for edit, delete, save, and refresh. Below this, it shows the selected product as "App Service" with a description "Premium V3 Tier; 1 P1V3 (2 Core(s), 8 GB RAM, 250 ...)" and a breakdown of costs: "Upfront: \$0.00" and "Monthly: \$244.55".

The main configuration area for "App Service" includes dropdowns for "REGION: West US", "OPERATING SYSTEM: Windows", and "TIER: Premium V3".

Under "Premium V3", there's a section for "INSTANCE: P1V3: 2 Cores(s), 8 GB RAM, 250 GB Storage, \$0.335" with a dropdown menu. It shows 1 instance running for 730 hours, resulting in a total monthly cost of \$244.55.

The "Savings Options" section indicates savings up to 42% on pay-as-you-go prices with 1 year or 3 year reserved instances. It shows the current selection as "Pay as you go". Other options include "1 year reserved (~25% discount)" and "3 year reserved (~40% discount)". The monthly cost remains at \$244.55.

The "SSL Connections" section shows "Upfront cost: \$0.00" and "Monthly cost: \$244.55".

The "Support" section shows "SUPPORT: Included" with a cost of \$0.00.

The "Programs and Offers" section shows "LICENSING PROGRAM: Microsoft Online Services Agreement" and a checkbox for "SHOW DEV/TEST PRICING".

At the bottom, it lists "Estimated upfront cost: \$0.00" and "Estimated monthly cost: \$244.55".

Review estimated costs in the Azure portal

When you create an App Service app or an App Service plan, you can see the estimated costs.

To create an app and view the estimated price:

1. On the create page, scroll down to **App Service plan**, and click **Create new**.
2. Specify a name and click **OK**.
3. Next to **Sku and size**, click **Change size**.
4. Review the estimated price shown in the summary. The following screenshot is an example and doesn't reflect current pricing.

The screenshot shows the 'Spec Picker' dialog with the title 'Spec Picker' at the top right. Below it is a section titled 'Recommended pricing tiers' containing six cards arranged in two rows of three. Each card provides details for a specific SKU:

SKU	Total ACU	Memory	Compute Equivalent	Estimated Price
P1V2	210 total ACU	3.5 GB memory	Dv2-Series compute equivalent	81.03 USD/Month (Estimated)
P2V2	420 total ACU	7 GB memory	Dv2-Series compute equivalent	161.33 USD/Month (Estimated)
P3V2	840 total ACU	14 GB memory	Dv2-Series compute equivalent	322.66 USD/Month (Estimated)
P1V3	195 minimum ACU/vCPU	8 GB memory	2 vCPU	93.44 USD/Month (Estimated)
P2V3	195 minimum ACU/vCPU	16 GB memory	4 vCPU	186.88 USD/Month (Estimated)
P3V3	195 minimum ACU/vCPU	32 GB memory	8 vCPU	373.76 USD/Month (Estimated)

Below the cards is a link 'See additional options'. The dialog is divided into sections: 'Included features' (with icons for custom domains/SSL and auto scale) and 'Included hardware' (with icons for Azure Compute Units (ACU) and Memory).

If your Azure subscription has a spending limit, Azure prevents you from spending over your credit amount. As you create and use Azure resources, your credits are used. When you reach your credit limit, the resources that you deployed are disabled for the rest of that billing period. You can't change your credit limit, but you can remove it. For more information about spending limits, see [Azure spending limit](#).

Optimize costs

At a basic level, App Service apps are charged by the App Service plan that hosts them. The costs associated with your App Service deployment depend on a few main factors:

- **Pricing tier** Otherwise known as the SKU of the App Service plan. Higher tiers provide more CPU cores, memory, storage, or features, or combinations of them.
- **Instance count** dedicated tiers (Basic and above) can be scaled out, and each scaled out instance accrues costs.

- **Stamp fee** In the Isolated tier, a flat fee is accrued on your App Service environment, regardless of how many apps or worker instances are hosted.

An App Service plan can host more than one app. Depending on your deployment, you could save costs hosting more apps on one App Service plans (i.e. hosting your apps on fewer App Service plans).

For details, see [App Service plan overview](#)

Non-production workloads

To test App Service or your solution while accruing low or minimal cost, you can begin by using the two entry-level pricing tiers, **Free** and **Shared**, which are hosted on shared instances. To test your app on dedicated instances with better performance, you can upgrade to **Basic** tier, which supports both Windows and Linux apps.

ⓘ Note

Azure Dev/Test Pricing To test pre-production workloads that require higher tiers (all tiers except for **Isolated**), Visual Studio subscribers can also take advantage of the [Azure Dev/Test Pricing](#), which offers significant discounts.

Both the **Free** and **Shared** tier, as well as the Azure Dev/Test Pricing discounts, don't carry a financially backed SLA.

Production workloads

Production workloads come with the recommendation of the dedicated **Standard** pricing tier or above. While the price goes up for higher tiers, it also gives you more memory and storage and higher-performing hardware, giving you higher app density per compute instance. That translates to lower instance count for the same number of apps, and therefore lower cost. In fact, **Premium V3** (the highest non-**Isolated** tier) is the most cost effective way to serve your app at scale. To add to the savings, you can get deep discounts on [Premium V3 reservations](#).

ⓘ Note

Premium V3 supports both Windows containers and Linux containers.

Once you choose the pricing tier you want, you should minimize the idle instances. In a scale-out deployment, you can waste money on underutilized compute instances. You

should [configure autoscaling](#), available in **Standard** tier and above. By creating scale-out schedules, as well as metric-based scale-out rules, you only pay for the instances you really need at any given time.

Azure Reservations

If you plan to utilize a known minimum number of compute instances for one year or more, you should take advantage of **Premium V3** tier and drive down the instance cost drastically by reserving those instances in 1-year or 3-year increments. The monthly cost savings can be as much as 55% per instance. Two types of reservations are possible:

- **Windows (or platform agnostic)** Can apply to Windows or Linux instances in your subscription.
- **Linux specific** Applies only to Linux instances in your subscription.

The reserved instance pricing applies to the applicable instances in your subscription, up to the number of instances that you reserve. The reserved instances are a billing matter and are not tied to specific compute instances. If you run fewer instances than you reserve at any point during the reservation period, you still pay for the reserved instances. If you run more instances than you reserve at any point during the reservation period, you pay the normal accrued cost for the additional instances.

The **Isolated** tier (App Service environment) also supports 1-year and 3-year reservations at reduced pricing. For more information, see [How reservation discounts apply to Azure App Service](#).

Monitor costs

As you use Azure resources with App Service, you incur costs. Azure resource usage unit costs vary by time intervals (seconds, minutes, hours, and days). As soon as App Service use starts, costs are incurred and you can see the costs in [cost analysis](#).

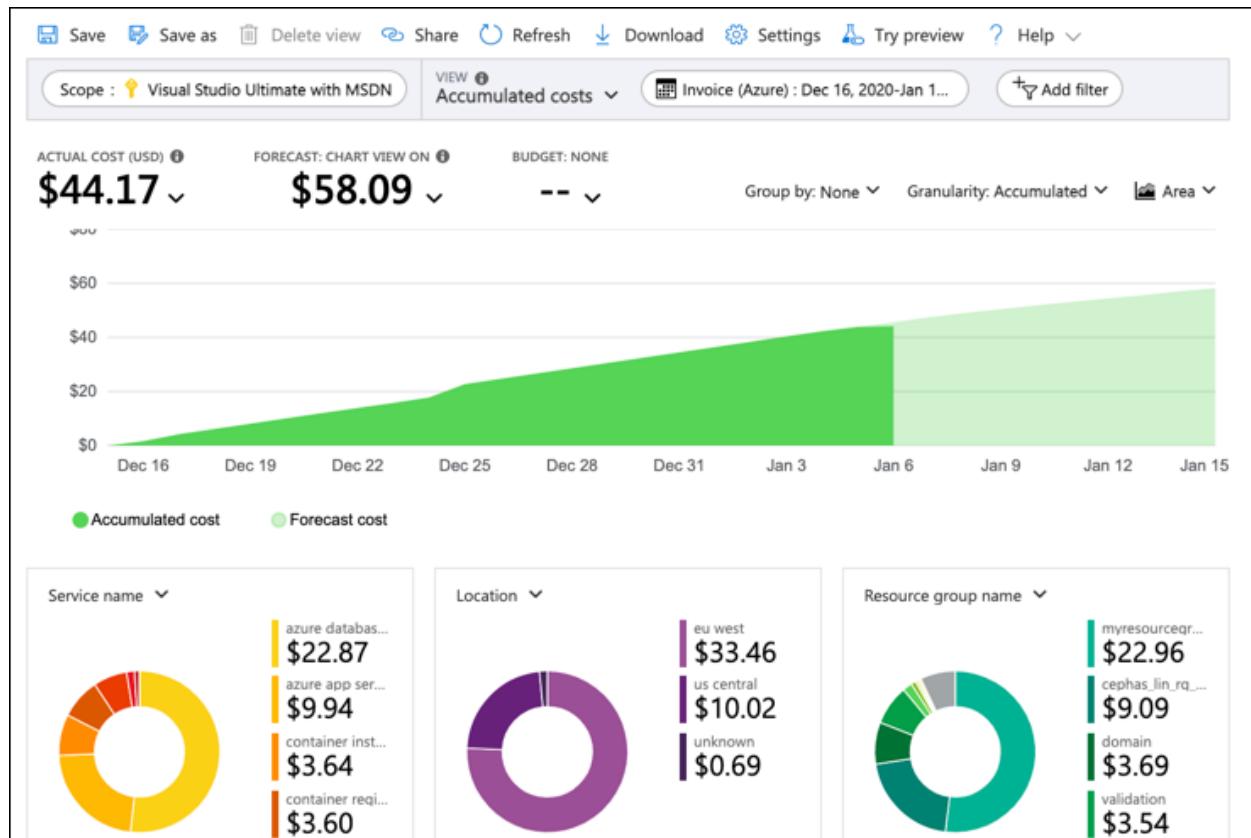
When you use cost analysis, you view App Service costs in graphs and tables for different time intervals. Some examples are by day, current and prior month, and year. You also view costs against budgets and forecasted costs. Switching to longer views over time can help you identify spending trends. And you see where overspending might have occurred. If you've created budgets, you can also easily see where they're exceeded.

To view App Service costs in cost analysis:

1. Sign in to the Azure portal.

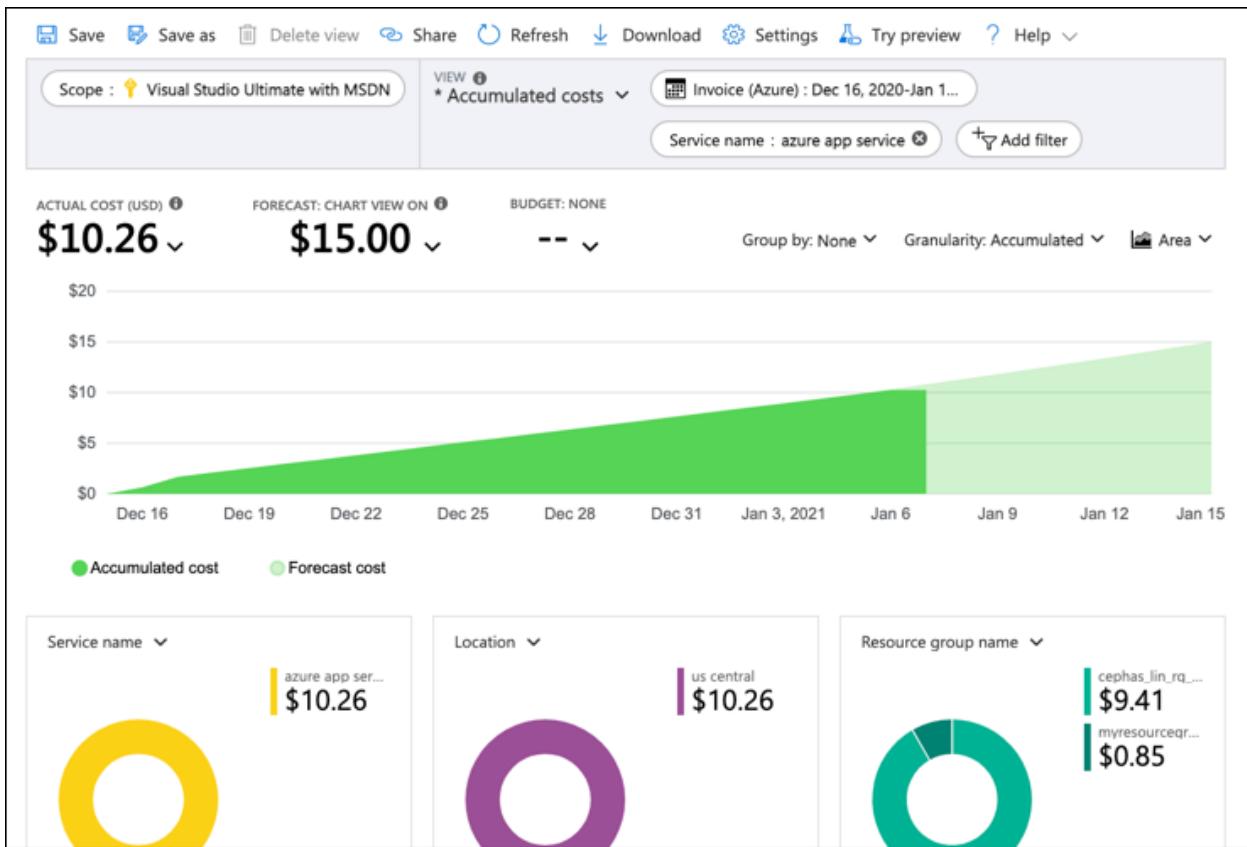
2. Open the scope in the Azure portal and select **Cost analysis** in the menu. For example, go to **Subscriptions**, select a subscription from the list, and then select **Cost analysis** in the menu. Select **Scope** to switch to a different scope in cost analysis.
3. By default, cost for services are shown in the first donut chart. Select the area in the chart labeled App Service.

Actual monthly costs are shown when you initially open cost analysis. Here's an example showing all monthly usage costs.



To narrow costs for a single service, like App Service, select **Add filter** and then select **Service name**. Then, select **App Service**.

Here's an example showing costs for just App Service.



In the preceding example, you see the current cost for the service. Costs by Azure regions (locations) and App Service costs by resource group are also shown. From here, you can explore costs on your own.

Create budgets

You can create [budgets](#) to manage costs and create [alerts](#) that automatically notify stakeholders of spending anomalies and overspending risks. Alerts are based on spending compared to budget and cost thresholds. Budgets and alerts are created for Azure subscriptions and resource groups, so they're useful as part of an overall cost monitoring strategy.

Budgets can be created with filters for specific resources or services in Azure if you want more granularity present in your monitoring. Filters help ensure that you don't accidentally create new resources that cost you extra money. For more information about the filter options available when you create a budget, see [Group and filter options](#).

Export cost data

You can also [export your cost data](#) to a storage account. This is helpful when you need or others to do more data analysis for costs. For example, a finance team can analyze the data using Excel or Power BI. You can export your costs on a daily, weekly, or

monthly schedule and set a custom date range. Exporting cost data is the recommended way to retrieve cost datasets.

Next steps

- Learn more on how pricing works with Azure Storage. See [App Service pricing](#).
- Learn [how to optimize your cloud investment with Azure Cost Management](#).
- Learn more about managing costs with [cost analysis](#).
- Learn about how to [prevent unexpected costs](#).
- Take the [Cost Management](#) guided learning course.

Configure a custom container for Azure App Service

Article • 01/23/2025

This article shows you how to configure a custom container to run on Azure App Service.

This guide provides key concepts and instructions for containerization of Linux apps in App Service. If you're new to Azure App Service, follow the [custom container quickstart](#) and [tutorial](#) first. For sidecar containers (preview), see [Tutorial: Configure a sidecar container for custom container in Azure App Service \(preview\)](#).

Change the Docker image of a custom container

To change an existing custom container from the current Docker image to a new image, use the following command:

Azure CLI

```
az webapp config container set --name <app-name> --resource-group <group-name> --docker-custom-image-name <docker-hub-repo>/<image>
```

Use an image from a private registry

To use an image from a private registry, such as Azure Container Registry, run the following command:

Azure CLI

```
az webapp config container set --name <app-name> --resource-group <group-name> --docker-custom-image-name <image-name> --docker-registry-server-url <private-repo-url> --docker-registry-server-user <username> --docker-registry-server-password <password>
```

For `<username>` and `<password>`, supply the sign-in credentials for your private registry account.

Use managed identity to pull image from Azure Container Registry

Use the following steps to configure your web app to pull from Azure Container Registry (ACR) using managed identity. The steps use system-assigned managed identity, but you can use user-assigned managed identity as well.

1. Enable the system-assigned managed identity for the web app by using the `az webapp identity assign` command:

Azure CLI

```
az webapp identity assign --resource-group <group-name> --name <app-name> --query principalId --output tsv
```

Replace `<app-name>` with the name you used in the previous step. The output of the command (filtered by the `--query` and `--output` arguments) is the service principal ID of the assigned identity.

2. Get the resource ID of your Azure Container Registry:

Azure CLI

```
az acr show --resource-group <group-name> --name <registry-name> --query id --output tsv
```

Replace `<registry-name>` with the name of your registry. The output of the command (filtered by the `--query` and `--output` arguments) is the resource ID of the Azure Container Registry.

3. Grant the managed identity permission to access the container registry:

Azure CLI

```
az role assignment create --assignee <principal-id> --scope <registry-resource-id> --role "AcrPull"
```

Replace the following values:

- `<principal-id>` with the service principal ID from the `az webapp identity assign` command
- `<registry-resource-id>` with the ID of your container registry from the `az acr show` command

For more information about these permissions, see [What is Azure role-based access control](#).

4. Configure your app to use the managed identity to pull from Azure Container Registry.

Azure CLI

```
az webapp config set --resource-group <group-name> --name <app-name> --generic-configurations '{"acrUseManagedIdentityCreds": true}'
```

Replace the following values:

- <app-name> with the name of your web app.

Tip

If you are using PowerShell console to run the commands, you need to escape the strings in the `--generic-configurations` argument in this and the next step. For example: `--generic-configurations`

```
'\\"acrUseManagedIdentityCreds\\": true'
```

5. (Optional) If your app uses a [user-assigned managed identity](#), make sure the identity is configured on the web app and then set the `acrUserManagedIdentityID` property to specify its client ID:

Azure CLI

```
az identity show --resource-group <group-name> --name <identity-name> -query clientId --output tsv
```

Replace the `<identity-name>` of your user-assigned managed identity and use the output `<client-id>` to configure the user-assigned managed identity ID.

Azure CLI

```
az webapp config set --resource-group <group-name> --name <app-name> --generic-configurations '{"acrUserManagedIdentityID": "<client-id>"}'
```

You're all set, and the web app now uses managed identity to pull from Azure Container Registry.

Use an image from a network protected registry

To connect and pull from a registry inside a virtual network or on-premises, your app must integrate with a virtual network (VNET). VNET integration is also needed for Azure Container Registry with private endpoint. When your network and DNS resolution is configured, you enable the routing of the image pull through the virtual network by configuring the `vnetImagePullEnabled` site setting:

Azure CLI

```
az resource update --resource-group <group-name> --name <app-name> --  
resource-type "Microsoft.Web/sites" --set properties.vnetImagePullEnabled  
[true|false]
```

I don't see the updated container

If you change your Docker container settings to point to a new container, it might take a few minutes before the app serves HTTP requests from the new container. While the new container is being pulled and started, App Service continues to serve requests from the old container. Only when the new container is started and ready to receive requests does App Service start sending requests to it.

How container images are stored

The first time you run a custom Docker image in App Service, App Service does a `docker pull` and pulls all image layers. These layers are stored on disk, like if you were using Docker on-premises. Each time the app restarts, App Service does a `docker pull`, but only pulls layers that have changed. If there are no changes, App Service uses existing layers on the local disk.

If the app changes compute instances for any reason, such as scaling up and down the pricing tiers, App Service must pull down all layers again. The same is true if you scale out to add more instances. There are also rare cases where the app instances might change without a scale operation.

Configure port number

By default, App Service assumes your custom container is listening on port 80. If your container listens to a different port, set the `WEBSITES_PORT` app setting in your App Service app. You can set it via the [Cloud Shell ↗](#). In Bash:

Azure CLI

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

In PowerShell:

Azure PowerShell

```
Set-AzWebApp -ResourceGroupName <group-name> -Name <app-name> -AppSettings @{"WEBSITES_PORT"="8000"}
```

App Service currently allows your container to expose only one port for HTTP requests.

Configure environment variables

Your custom container might use environment variables that need to be supplied externally. You can pass them in via the [Cloud Shell ↗](#). In Bash:

Azure CLI

```
az webapp config appsettings set --resource-group <group-name> --name <app-name> --settings DB_HOST="myownserver.mysql.database.azure.com"
```

In PowerShell:

Azure PowerShell

```
Set-AzWebApp -ResourceGroupName <group-name> -Name <app-name> -AppSettings @{"DB_HOST"="myownserver.mysql.database.azure.com"}
```

When your app runs, the App Service app settings are injected into the process as environment variables automatically. You can verify container environment variables with the URL `https://<app-name>.scm.azurewebsites.net/Env`.

If your app uses images from a private registry or from Docker Hub, credentials for accessing the repository are saved in environment variables:

`DOCKER_REGISTRY_SERVER_URL`, `DOCKER_REGISTRY_SERVER_USERNAME` and

`DOCKER_REGISTRY_SERVER_PASSWORD`. Because of security risks, none of these reserved variable names are exposed to the application.

This method works both for single-container apps or multi-container apps, where the environment variables are specified in the `docker-compose.yml` file.

Use persistent shared storage

You can use the `/home` directory in your custom container file system to persist files across restarts and share them across instances. The `/home` directory is provided to enable your custom container to access persistent storage. Saving data within `/home` contributes to the [storage space quota](#) included with your App Service Plan.

When persistent storage is disabled, then writes to the `/home` directory aren't persisted across app restarts or across multiple instances. When persistent storage is enabled, all writes to the `/home` directory are persisted and can be accessed by all instances of a scaled-out app. Additionally, any contents inside the `/home` directory of the container are overwritten by any existing files already present on the persistent storage when the container starts.

The only exception is the `/home/LogFiles` directory, which is used to store the container and application logs. This folder always persists upon app restarts if [application logging is enabled](#) with the **File System** option, independently of the persistent storage being enabled or disabled. In other words, enabling or disabling the persistent storage doesn't affect the application logging behavior.

It's recommended to write data to `/home` or a [mounted Azure storage path](#). Data written outside these paths isn't persistent during restarts and is saved to platform-managed host disk space separate from the App Service Plans file storage quota.

By default, persistent storage is *disabled* on Linux custom containers. To enable it, set the `WEBSITES_ENABLE_APP_SERVICE_STORAGE` app setting value to `true` via the [Cloud Shell](#).

In Bash:

Azure CLI

```
az webapp config appsettings set --resource-group <group-name> --name <app-name> --settings WEBSITES_ENABLE_APP_SERVICE_STORAGE=true
```

In PowerShell:

Azure PowerShell

```
Set-AzWebApp -ResourceGroupName <group-name> -Name <app-name> -AppSettings  
@{"WEBSITES_ENABLE_APP_SERVICE_STORAGE"=true}
```

ⓘ Note

You can also [configure your own persistent storage](#).

Detect HTTPS session

App Service terminates TLS/SSL at the front ends. That means that TLS/SSL requests never get to your app. You don't need to, and shouldn't implement any support for TLS/SSL into your app.

The front ends are located inside Azure data centers. If you use TLS/SSL with your app, your traffic across the Internet is always safely encrypted.

Enable SSH

Secure Shell (SSH) is commonly used to execute administrative commands remotely from a command-line terminal. In order to enable the Azure portal SSH console feature with custom containers, the following steps are required:

1. Create a standard `sshd_config` file with the following example contents and place it on the application project root directory:

```
Port          2222
ListenAddress 0.0.0.0
LoginGraceTime 180
X11Forwarding yes
Ciphers aes128-cbc,3des-cbc,aes256-cbc,aes128-ctr,aes192-ctr,aes256-ctr
MACs hmac-sha1,hmac-sha1-96
StrictModes   yes
SyslogFacility DAEMON
PasswordAuthentication yes
PermitEmptyPasswords no
PermitRootLogin yes
Subsystem sftp internal-sftp
```

ⓘ Note

This file configures OpenSSH and must include the following items in order to comply with the Azure portal SSH feature:

- Port must be set to 2222.
- Ciphers must include at least one item in this list: aes128-cbc,3des-cbc,aes256-cbc .
- MACs must include at least one item in this list: hmac-sha1,hmac-sha1-96 .

2. Create an entrypoint script with the name `entrypoint.sh` (or change any existing entrypoint file) and add the command to start the SSH service, along with the application startup command. The following example demonstrates starting a Python application. Replace the last command according to the project language/stack:

Debian

Bash

```
#!/bin/sh
set -e
service ssh start
exec gunicorn -w 4 -b 0.0.0.0:8000 app:app
```

3. Add to the Dockerfile the following instructions according to the base image distribution. These instructions copy the new files, install OpenSSH server, set proper permissions and configure the custom entrypoint, and expose the ports required by the application and SSH server, respectively:

Debian

Dockerfile

```
COPY entrypoint.sh .

# Start and enable SSH
RUN apt-get update \
    && apt-get install -y --no-install-recommends dialog \
    && apt-get install -y --no-install-recommends openssh-server \
    && echo "root:Docker!" | chpasswd \
    && chmod u+x ./entrypoint.sh
COPY sshd_config /etc/ssh/

EXPOSE 8000 2222
```

```
ENTRYPOINT [ "./entrypoint.sh" ]
```

ⓘ Note

The root password must be exactly Docker! as it's used by App Service to let you access the SSH session with the container. This configuration doesn't allow external connections to the container. Port 2222 of the container is accessible only within the bridge network of a private virtual network and isn't accessible to an attacker on the internet.

4. Rebuild and push the Docker image to the registry, and then test the Web App SSH feature on Azure portal.

Further troubleshooting information is available at the Azure App Service blog: [Enabling SSH on Linux Web App for Containers ↗](#)

Access diagnostic logs

You can access the console logs generated from inside the container.

First, turn on container logging by running the following command:

Azure CLI

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

Replace `<app-name>` and `<resource-group-name>` with the names appropriate for your web app.

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

Azure CLI

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

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

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

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

Configure multi-container apps

ⓘ Note

Sidecar containers (preview) will succeed multi-container apps in App Service. To get started, see [Tutorial: Configure a sidecar container for custom container in Azure App Service \(preview\)](#).

- Use persistent storage in Docker Compose
- Preview limitations
- Docker Compose options

Use persistent storage in Docker Compose

Multi-container apps like WordPress need persistent storage to function properly. To enable it, your Docker Compose configuration must point to a storage location *outside* your container. Storage locations inside your container don't persist changes beyond app restart.

Enable persistent storage by setting the `WEBSITES_ENABLE_APP_SERVICE_STORAGE` app setting, using the `az webapp config appsettings set` command in [Cloud Shell](#).

Azure CLI

```
az webapp config appsettings set --resource-group <group-name> --name <app-name> --settings WEBSITES_ENABLE_APP_SERVICE_STORAGE=TRUE
```

In your `docker-compose.yml` file, map the `volumes` option to `WEBAPP_STORAGE_HOME`.

`WEBAPP_STORAGE_HOME` is an environment variable in App Service that is mapped to persistent storage for your app. For example:

YAML

```
wordpress:  
  image: <image name:tag>  
  volumes:  
    - "${WEBAPP_STORAGE_HOME}/site/wwwroot:/var/www/html"
```

```
- "${WEBAPP_STORAGE_HOME}/phpmyadmin:/var/www/phpmyadmin"  
- "${WEBAPP_STORAGE_HOME}/LogFiles:/var/log"
```

Preview limitations

Multi-container is currently in preview. The following App Service platform features aren't supported:

- Authentication / Authorization
- Managed Identities
- CORS
- Virtual network integration isn't supported for Docker Compose scenarios
- Docker Compose on Azure App Services currently has a limit of 4,000 characters at this time.

Docker Compose options

The following lists show supported and unsupported Docker Compose configuration options:

Supported options

- command
- entrypoint
- environment
- image
- ports
- restart
- services
- volumes ([mapping to Azure Storage is unsupported](#))

Unsupported options

- build (not allowed)
- [depends_on](#) (ignored)
- networks (ignored)
- secrets (ignored)
- ports other than 80 and 8080 (ignored)
- default environment variables like `$variable` and `${variable}` unlike in docker

Syntax Limitations

- "version x.x" always needs to be the first YAML statement in the file
- ports section must use quoted numbers
- image > volume section must be quoted and can't have permissions definitions
- volumes section must not have an empty curly brace after the volume name

Note

Any other options not explicitly called out are ignored in Public Preview.

robots933456 in logs

You may see the following message in the container logs:

```
2019-04-08T14:07:56.641002476Z "-" - - [08/Apr/2019:14:07:56 +0000] "GET /robots933456.txt HTTP/1.1" 404 415 "-" "-"
```

You can safely ignore this message. `/robots933456.txt` is a dummy URL path that App Service uses to check if the container is capable of serving requests. A 404 response simply indicates that the path doesn't exist, but it lets App Service know that the container is healthy and ready to respond to requests.

Next steps

[Tutorial: Migrate custom software to Azure App Service using a custom container](#)

[Tutorial: Configure a sidecar container for custom container in Azure App Service \(preview\)](#)

Or, see more resources:

- [Environment variables and app settings reference](#)
- [Load certificate in Windows/Linux containers](#)

Feedback

Was this page helpful?

 Yes

 No

Provide product feedback  | Get help at Microsoft Q&A

Continuous deployment with custom containers in Azure App Service

Article • 04/04/2024

In this tutorial, you configure continuous deployment for a custom container image from managed [Azure Container Registry](#) repositories or [Docker Hub](#).

1. Go to Deployment Center

In the [Azure portal](#), navigate to the management page for your App Service app.

From the left menu, click **Deployment Center > Settings**.

2. Choose deployment source

Choose the deployment source depends on your scenario:

- **Container registry** sets up CI/CD between your container registry and App Service.
- The **GitHub Actions** option is for you if you maintain the source code for your container image in GitHub. Triggered by new commits to your GitHub repository, the deploy action can run `docker build` and `docker push` directly to your container registry, then update your App Service app to run the new image. For more information, see [How CI/CD works with GitHub Actions](#).
- To set up CI/CD with **Azure Pipelines**, see [Deploy an Azure Web App Container from Azure Pipelines](#).

ⓘ Note

For a Docker Compose app, select **Container Registry**.

If you choose GitHub Actions, **click Authorize** and follow the authorization prompts. If you've already authorized with GitHub before, you can deploy from a different user's repository by clicking **Change Account**.

Once you authorize your Azure account with GitHub, **select the Organization, Repository, and Branch** to deploy from.

3. Configure registry settings

 **Note**

Sidecar containers (preview) will succeed multi-container (Docker Compose) apps in App Service. To get started, see [Tutorial: Configure a sidecar container for custom container in Azure App Service \(preview\)](#).

To deploy a multi-container (Docker Compose) app, **select Docker Compose** in **Container Type**.

If you don't see the **Container Type** dropdown, scroll back up to **Source** and **select Container Registry**.

In **Registry source**, **select** where your container registry is. If it's neither Azure Container Registry nor Docker Hub, **select Private Registry**.

 **Note**

If your multi-container (Docker Compose) app uses more than one private image, make sure the private images are in the same private registry and accessible with the same user credentials. If your multi-container app only uses public images, **select Docker Hub**, even if some images are not in Docker Hub.

Follow the next steps by selecting the tab that matches your choice.

Azure Container Registry

The **Registry** dropdown displays the registries in the same subscription as your app. **Select** the registry you want.

 **Note**

- If want to use Managed Identities to lock down ACR access follow this guide:
 - [How to use system-assigned Managed Identities with App Service and Azure Container Registry](#)
 - [How to use user-assigned Managed Identities with App Service and Azure Container Registry](#)
- To deploy from a registry in a different subscription, **select Private Registry** in **Registry source** instead.

Follow the next step depending on the **Container Type**:

- For **Docker Compose**, select the registry for your private images. Click **Choose file** to upload your [Docker Compose file](#), or just **paste** the content of your Docker Compose file into **Config**.
- For **Single Container**, select the **Image** and **Tag** to deploy. If you want, **type** the start up command in **Startup File**.

App Service appends the string in **Startup File** to [the end of the docker run command \(as the \[COMMAND\] \[ARG...\]\) segment](#) when starting your container.

4. Enable CI/CD

App Service supports CI/CD integration with Azure Container Registry and Docker Hub. To enable it, **select On** in **Continuous deployment**.

ⓘ Note

If you select **GitHub Actions** in **Source**, you don't get this option because CI/CD is handled by GitHub Actions directly. Instead, you see a **Workflow Configuration** section, where you can **click Preview file** to inspect the workflow file. Azure commits this file into your selected GitHub source repository to handle build and deploy tasks. For more information, see [How CI/CD works with GitHub Actions](#).

When you enable this option, App Service adds a webhook to your repository in Azure Container Registry or Docker Hub. Your repository posts to this webhook whenever your selected image is updated with `docker push`. The webhook causes your App Service app to restart and run `docker pull` to get the updated image.

ⓘ Note

To ensure the proper functioning of the webhook, it's essential to enable the **Basic Auth Publishing Credentials** option within your Web App. Failure to do so may result in a 401 unauthorized error for the webhook. To verify whether **Basic Auth Publishing Credentials** is enabled, follow these steps:

- Navigate to your Web App's **Configuration > General Settings**.
- Look for the **Platform Setting** section, where you will find the **Basic Auth Publishing Credentials** option.

For other private registries, you can post to the webhook manually or as a step in a CI/CD pipeline. In **Webhook URL**, click the **Copy** button to get the webhook URL.

ⓘ Note

Support for multi-container (Docker Compose) apps is limited:

- For Azure Container Registry, App Service creates a webhook in the selected registry with the registry as the scope. A `docker push` to any repository in the registry (including the ones not referenced by your Docker Compose file) triggers an app restart. You may want to [modify the webhook](#) to a narrower scope.
- Docker Hub doesn't support webhooks at the registry level. You must **add** the webhooks manually to the images specified in your Docker Compose file.

5. Save your settings

Click **Save**.

How CI/CD works with GitHub Actions

If you choose **GitHub Actions** in **Source** (see [Choose deployment source](#)), App Service sets up CI/CD in the following ways:

- Deposits a GitHub Actions workflow file into your GitHub repository to handle build and deploy tasks to App Service.
- Adds the credentials for your private registry as GitHub secrets. The generated workflow file runs the [Azure/docker-login](#) action to sign in with your private registry, then runs `docker push` to deploy to it.
- Adds the publishing profile for your app as a GitHub secret. The generated workflow file uses this secret to authenticate with App Service, then runs the [Azure/webapps-deploy](#) action to configure the updated image, which triggers an app restart to pull in the updated image.
- Captures information from the [workflow run logs](#) and displays it in the **Logs** tab in your app's **Deployment Center**.

You can customize the GitHub Actions build provider in the following ways:

- Customize the workflow file after it's generated in your GitHub repository. For more information, see [Workflow syntax for GitHub Actions](#). Just make sure that

the workflow ends with the [Azure/webapps-deploy](#) action to trigger an app restart.

- If the selected branch is protected, you can still preview the workflow file without saving the configuration, then add it and the required GitHub secrets into your repository manually. This method doesn't give you the log integration with the Azure portal.
- Instead of a publishing profile, deploy using a [service principal](#) in Microsoft Entra ID.

Authenticate with a service principal

This optional configuration replaces the default authentication with publishing profiles in the generated workflow file.

Generate a service principal with the `az ad sp create-for-rbac` command in the [Azure CLI](#). In the following example, replace `<subscription-id>`, `<group-name>`, and `<app-name>` with your own values. **Save** the entire JSON output for the next step, including the top-level `{}`.

Azure CLI

```
az ad sp create-for-rbac --name "myAppDeployAuth" --role contributor \
    --scopes /subscriptions/<subscription-
    id>/resourceGroups/<group-name>/providers/Microsoft.Web/sites/<app-name> \
    --json-auth
```

ⓘ Important

For security, grant the minimum required access to the service principal. The scope in the previous example is limited to the specific App Service app and not the entire resource group.

In [GitHub](#), **browse** to your repository, then **select Settings > Secrets > Add a new secret**. Paste the entire JSON output from the Azure CLI command into the secret's value field. **Give** the secret a name like `AZURE_CREDENTIALS`.

In the workflow file generated by the **Deployment Center**, **revise** the `azure/webapps-deploy` step with code like the following example:

YAML

```
- name: Sign in to Azure
# Use the GitHub secret you added
- uses: azure/login@v1
  with:
    creds: ${{ secrets.AZURE_CREDENTIALS }}
- name: Deploy to Azure Web App
# Remove publish-profile
- uses: azure/webapps-deploy@v2
  with:
    app-name: '<app-name>'
    slot-name: 'production'
    images: '<registry-server>/${{
secrets.AzureAppService_ContainerUsername_... }}/<image>:${{ github.sha }}'
      - name: Sign out of Azure
        run: |
          az logout
```

Automate with CLI

To configure the container registry and the Docker image, [run az webapp config container set](#).

Azure Container Registry

Azure CLI

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

To configure a multi-container (Docker Compose) app, [prepare a Docker Compose file locally](#), then [run az webapp config container set](#) with the `--multicontainer-config-file` parameter. If your Docker Compose file contains private images, [add `--docker-registry-server-*`](#) parameters as shown in the previous example.

Azure CLI

```
az webapp config container set --resource-group <group-name> --name <app-
name> --multicontainer-config-file <docker-compose-file>
```

To configure CI/CD from the container registry to your app, [run az webapp deployment container config](#) with the `--enable-cd` parameter. The command outputs the webhook

URL, but you must create the webhook in your registry manually in a separate step. The following example enables CI/CD on your app, then uses the webhook URL in the output to create the webhook in Azure Container Registry.

Azure CLI

```
ci_cd_url=$(az webapp deployment container config --name <app-name> --resource-group <group-name> --enable-cd true --query CI_CD_URL --output tsv)

az acr webhook create --name <webhook-name> --registry <registry-name> --resource-group <group-name> --actions push --uri $ci_cd_url --scope '<image>:<tag>'
```

More resources

- [Azure Container Registry](#)
- [Create a .NET Core web app in App Service on Linux](#)
- [Quickstart: Run a custom container on App Service](#)
- [App Service on Linux FAQ](#)
- [Configure custom containers](#)
- [Actions workflows to deploy to Azure](#)

Deploy a custom container to App Service using GitHub Actions

Article • 01/22/2025

[GitHub Actions](#) gives you the flexibility to build an automated software development workflow. With the [Azure Web Deploy action](#), you can automate your workflow to deploy custom containers to [App Service](#) using GitHub Actions.

A workflow is defined by a YAML (.yml) file in the `/.github/workflows/` path in your repository. This definition contains the various steps and parameters that are in the workflow.

For an Azure App Service container workflow, the file has three sections:

[] [Expand table](#)

Section	Tasks
Authentication	1. Retrieve a service principal or publish profile. 2. Create a GitHub secret.
Build	1. Create the environment. 2. Build the container image.
Deploy	1. Deploy the container image.

Prerequisites

- An Azure account with an active subscription. [Create an account for free](#)
- A GitHub account. If you don't have one, sign up for [free](#). You need to have code in a GitHub repository to deploy to Azure App Service.
- A working container registry and Azure App Service app for containers. This example uses Azure Container Registry. Make sure to complete the full deployment to Azure App Service for containers. Unlike regular web apps, web apps for containers don't have a default landing page. Publish the container to have a working example.
 - [Learn how to create a containerized Node.js application using Docker, push the container image to a registry, and then deploy the image to Azure App Service](#)

Generate deployment credentials

The recommended way to authenticate with Azure App Services for GitHub Actions is with a publish profile. You can also authenticate with a service principal or Open ID Connect but the process requires more steps.

Save your publish profile credential or service principal as a [GitHub secret](#) to authenticate with Azure. You'll access the secret within your workflow.

Publish profile

A publish profile is an app-level credential. Set up your publish profile as a GitHub secret.

1. Go to your app service in the Azure portal.
2. On the **Overview** page, select **Get Publish profile**.

ⓘ Note

As of October 2020, Linux web apps will need the app setting `WEBSITE_WEBDEPLOY_USE_SCM` set to `true` before downloading the file. This requirement will be removed in the future. See [Configure an App Service app in the Azure portal](#), to learn how to configure common web app settings.

3. Save the downloaded file. You'll use the contents of the file to create a GitHub secret.

Configure the GitHub secret for authentication

Publish profile

In [GitHub](#), browse your repository. Select **Settings > Security > Secrets and variables > Actions > New repository secret**.

To use [app-level credentials](#), paste the contents of the downloaded publish profile file into the secret's value field. Name the secret `AZURE_WEBAPP_PUBLISH_PROFILE`.

When you configure your GitHub workflow, you use the `AZURE_WEBAPP_PUBLISH_PROFILE` in the deploy Azure Web App action. For example:

YAML

```
- uses: azure/webapps-deploy@v2
  with:
    publish-profile: ${{ secrets.AZURE_WEBAPP_PUBLISH_PROFILE }}
```

Configure GitHub secrets for your registry

Define secrets to use with the Docker Login action. The example in this document uses Azure Container Registry for the container registry.

1. Go to your container in the Azure portal or Docker and copy the username and password. You can find the Azure Container Registry username and password in the Azure portal under **Settings > Access keys** for your registry.
2. Define a new secret for the registry username named `REGISTRY_USERNAME`.
3. Define a new secret for the registry password named `REGISTRY_PASSWORD`.

Build the Container image

The following example show part of the workflow that builds a Node.js Docker image. Use [Docker Login](#) to log into a private container registry. This example uses Azure Container Registry but the same action works for other registries.

YAML

```
name: Linux Container Node Workflow

on: [push]

jobs:
  build:
    runs-on: ubuntu-latest

    steps:
      - uses: actions/checkout@v2
      - uses: azure/docker-login@v1
        with:
          login-server: mycontainer.azurecr.io
          username: ${{ secrets.REGISTRY_USERNAME }}
          password: ${{ secrets.REGISTRY_PASSWORD }}
      - run:
          docker build . -t mycontainer.azurecr.io/myapp:${{ github.sha }}
          docker push mycontainer.azurecr.io/myapp:${{ github.sha }}
```

You can also use [Docker sign-in](#) to log into multiple container registries at the same time. This example includes two new GitHub secrets for authentication with docker.io. The example assumes that there's a Dockerfile at the root level of the registry.

```
yml

name: Linux Container Node Workflow

on: [push]

jobs:
  build:
    runs-on: ubuntu-latest

    steps:
      - uses: actions/checkout@v2
      - uses: azure/docker-login@v1
        with:
          login-server: mycontainer.azurecr.io
          username: ${{ secrets.REGISTRY_USERNAME }}
          password: ${{ secrets.REGISTRY_PASSWORD }}
      - uses: azure/docker-login@v1
        with:
          login-server: index.docker.io
          username: ${{ secrets.DOCKERIO_USERNAME }}
          password: ${{ secrets.DOCKERIO_PASSWORD }}
      - run:
          docker build . -t mycontainer.azurecr.io/myapp:${{ github.sha }}
          docker push mycontainer.azurecr.io/myapp:${{ github.sha }}
```

Deploy to an App Service container

To deploy your image to a custom container in App Service, use the `azure/webapps-deploy@v2` action. This action has seven parameters:

[+] Expand table

Parameter	Explanation
<code>app-name</code>	(Required) Name of the App Service app
<code>publish-profile</code>	(Optional) Applies to Web Apps(Windows and Linux) and Web App Containers(linux). Multi container scenario not supported. Publish profile (*.publishsettings) file contents with Web Deploy secrets
<code>slot-name</code>	(Optional) Enter an existing Slot other than the Production slot
<code>package</code>	(Optional) Applies to Web App only: Path to package or folder. *.zip, *.war, *.jar

Parameter	Explanation
	or a folder to deploy
images	(Required) Applies to Web App Containers only: Specify the fully qualified container image(s) name. For example, 'myregistry.azurecr.io/nginx:latest' or 'python:3.7.2-alpine/'. For a multi-container app, multiple container image names can be provided (multi-line separated)
configuration-file	(Optional) Applies to Web App Containers only: Path of the Docker-Compose file. Should be a fully qualified path or relative to the default working directory. Required for multi-container apps.
startup-command	(Optional) Enter the start-up command. For ex. dotnet run or dotnet filename.dll

Publish profile

YAML

```

name: Linux Container Node Workflow

on: [push]

jobs:
  build:
    runs-on: ubuntu-latest

    steps:
      - uses: actions/checkout@v2

      - uses: azure/docker-login@v1
        with:
          login-server: mycontainer.azurecr.io
          username: ${{ secrets.REGISTRY_USERNAME }}
          password: ${{ secrets.REGISTRY_PASSWORD }}

      - run:
          docker build . -t mycontainer.azurecr.io/myapp:${{ github.sha }}
          docker push mycontainer.azurecr.io/myapp:${{ github.sha }}

      - uses: azure/webapps-deploy@v2
        with:
          app-name: 'myapp'
          publish-profile: ${{ secrets.AZURE_WEBAPP_PUBLISH_PROFILE }}
          images: 'mycontainer.azurecr.io/myapp:${{ github.sha }}'
```

Next steps

You can find our set of Actions grouped into different repositories on GitHub, each one containing documentation and examples to help you use GitHub for CI/CD and deploy your apps to Azure.

- [Actions workflows to deploy to Azure ↗](#)
 - [Azure sign-in ↗](#)
 - [Azure WebApp ↗](#)
 - [Docker sign-in/out ↗](#)
 - [Events that trigger workflows ↗](#)
 - [K8s deploy ↗](#)
 - [Starter Workflows ↗](#)
-

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

Tutorial: Configure a sidecar container for custom container in Azure App Service

Article • 11/19/2024

In this tutorial, you add an OpenTelemetry collector as a sidecar container to a Linux custom container app in Azure App Service. For bring-your-own-code Linux apps, see [Tutorial: Configure a sidecar container for a Linux app in Azure App Service](#).

In Azure App Service, you can add up to nine sidecar containers for each sidecar-enabled custom container app. Sidecar containers let you deploy extra services and features to your container application without making them tightly coupled to your main application container. For example, you can add monitoring, logging, configuration, and networking services as sidecar containers. An OpenTelemetry collector sidecar is one such monitoring example.

For more information about side container in App Service, see:

- [Introducing Sidecars for Azure App Service for Linux: Now Generally Available ↗](#)
- [Announcing the general availability of sidecar extensibility in Azure App Service ↗](#)

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

1. Set up the needed resources

First you create the resources that the tutorial uses. They're used for this particular scenario and aren't required for sidecar containers in general.

1. In the [Azure Cloud Shell ↗](#), run the following commands:

Azure CLI

```
git clone https://github.com/Azure-Samples/app-service-sidecar-tutorial-prereqs
cd app-service-sidecar-tutorial-prereqs
azd env new my-sidecar-env
azd provision
```

2. When prompted, supply the subscription and region you want. For example:

- Subscription: Your subscription.
- Region: *(Europe) West Europe*.

When deployment completes, you should see the following output:

```
APPLICATIONINSIGHTS_CONNECTION_STRING =  
InstrumentationKey=...;IngestionEndpoint=...;LiveEndpoint=...  
  
Open resource group in the portal:  
https://portal.azure.com/#@/resource/subscriptions/.../resourceGroups/..
```

3. Open the resource group link in a browser tab. You'll need to use the connection string later.

ⓘ Note

`azd provision` uses the included templates to create the following Azure resources:

- A resource group called *my-sidecar-env_group*.
- A container registry with two images deployed:
 - An Nginx image with the OpenTelemetry module.
 - An OpenTelemetry collector image, configured to export to Azure Monitor.
- A log analytics workspace
- An Application Insights component

2. Create a sidecar-enabled app

1. In the resource group's management page, select **Create**.
2. Search for *web app*, then select the down arrow on **Create** and select **Web App**.

Showing 1 to 20 of 646 results for 'web app'. [Clear search](#)

[Tile view](#)

Category	Description	Provider	Actions
Web App	Enjoy secure and flexible development, deployment, and scaling options for your web app	Microsoft Azure Service	Create <input checked="" type="checkbox"/> Web App
Web App + Database	Enjoy secure and flexible development, deployment, and scaling options for your web app plus a database. Choice for optimizing your web application	Microsoft App Service	Create <input type="checkbox"/>
Static Web App	Enjoy secure and flexible development, deployment, and scaling options for your web app	Microsoft Azure Service	Create <input type="checkbox"/>
Web App for Containers	Detailed description of Web App for Containers	DVWA	Detailed description of DVWA
Damn Vulnerable Web App	Detailed description of Damn Vulnerable Web App	Certify The Web - Cloud Managed App License	Free trial Is Marketplace helpful?

3. Configure the Basics panel as follows:

- **Name:** A unique name
- **Publish:** Container
- **Operating System:** Linux
- **Region:** Same region as the one you chose with `azd provision`
- **Linux Plan:** A new App Service plan

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

Project Details

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

Subscription * ⓘ

Resource Group * ⓘ [Create new](#)

Instance Details

Name *  .azurewebsites.net

Publish * Code Container Static Web App

Operating System * Linux Windows

Region * 

 Not finding your App Service Plan? Try a different region or select your App Service Environment.

Pricing plans

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

Linux Plan (West Europe) * ⓘ 
[Create new](#)

[Review + create](#)

[< Previous](#)

[Next : Container >](#)

4. Select Container. Configure the Container panel as follows:

- Sidecar support: Enabled
- Image Source: Azure Container Registry
- Registry: The registry created by `azd provision`
- Image: nginx
- Tag: latest
- Port: 80

Select your preferred source for container images. You can change these settings and other dependencies after creating the app. [Learn more](#)

Sidecar support (preview)

 Enabled
 Disabled

Image Source *

 Quickstart
 Azure Container Registry
 Docker Hub or other registries

Name *

main

Azure container registry options

Registry *

acrptbwpiel3ejsq

Image *

nginx

Tag *

latest

Port *

80

Startup Command ⓘ

Example: /bin/bash; -c; echo hello; sleep 10000

Review + create

< Previous

Next : Networking >

! Note

These settings are configured differently in sidecar-enabled apps. For more information, see [Differences for sidecar-enabled apps](#).

5. Select **Review + create**, then select **Create**.
6. Once the deployment completes, select **Go to resource**.
7. In a new browser tab, navigate to `https://<app-name>.azurewebsites.net` and see the default Nginx page.

3. Add a sidecar container

In this section, you add a sidecar container to your custom container app.

1. In the app's management page, from the left menu, select **Deployment Center**.

The deployment center shows you all the containers in the app. Right now, it only has the main container.

2. Select **Add** and configure the new container as follows:

- **Name:** *otel-collector*
- **Image source:** Azure Container Registry
- **Registry:** The registry created by `azd provision`
- **Image:** *otel-collector*
- **Tag:** *latest*

3. Select **Apply**.

Add container

X

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

Name *

otel-collector

Type

Sidecar

Image source *

Azure Container Registry

Docker Hub or other registries

Authentication *

Admin Credentials

Managed Identity

Registry *

acr2eb2qepliunts

Image *

otel-collector

Image tag *

latest

Port *

4317

Startup command

Example: /bin/bash; -c; echo hello; sleep 10000

Volume mounts

Apply

Discard

You should now see two containers in the deployment center. The main container is marked **Main**, and the sidecar container is marked **Sidecar**. Each app must have one main container but can have multiple sidecar containers.

4. Configure environment variables

For the sample scenario, the otel-collector sidecar is configured to export the OpenTelemetry data to Azure Monitor, but it needs the connection string as an environment variable (see the [OpenTelemetry configuration file for the otel-collector image](#)).

You configure environment variables for the containers like any App Service app, by configuring [app settings](#). The app settings are accessible to all the containers in the app.

1. In the app's management page, from the left menu, select **Environment variables**.
2. Add an app setting by selecting **Add** and configure it as follows:

- **Name:** `APPLICATIONINSIGHTS_CONNECTION_STRING`
- **Value:** The connection string in the output of `azd provision`. If you lost the Cloud Shell session, you can also find it in the **Overview** page of the Application Insight resource, under **Connection String**.

3. Select **Apply**, then **Apply**, then **Confirm**.

The screenshot shows the 'Application settings' tab selected in the Azure App Service management interface. A red box highlights the 'Save' button at the top. Below it, a message states 'Custom Error pages requires a premium App Service Plan.' The 'Application settings' section includes a 'New application setting' button (also highlighted with a red box), a 'Filter application settings' input field, and a table listing one setting: 'APPLICATIONINSIGHTS_CONNECTION_STRING' with value 'InstrumentationKey=xxxxxxxx-xxxx-xxxx'. The 'Connection strings' section below is shown as empty.

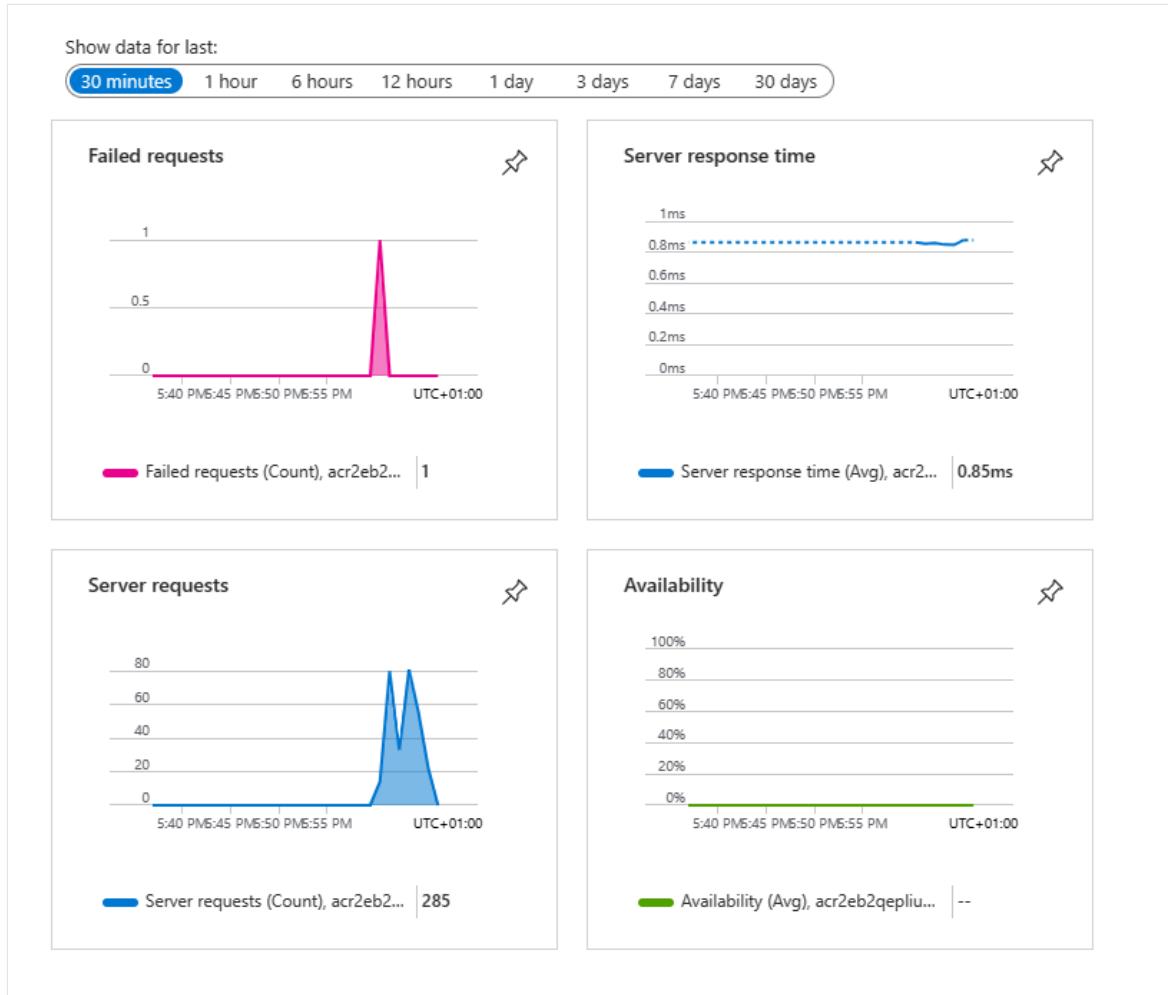
ⓘ Note

Certain app settings don't apply to sidecar-enabled apps. For more information, see [Differences for sidecar-enabled apps](#)

5. Verify in Application Insights

The otel-collector sidecar should export data to Application Insights now.

1. Back in the browser tab for `https://<app-name>.azurewebsites.net`, refresh the page a few times to generate some web requests.
2. Go back to the resource group overview page, then select the Application Insights resource. You should now see some data in the default charts.



ⓘ Note

In this very common monitoring scenario, Application Insights is just one of the OpenTelemetry targets you can use, such as Jaeger, Prometheus, and Zipkin.

Clean up resources

When you no longer need the environment, you can delete the resource group, App service, and all related resources. Just run this command in the Cloud Shell, in the cloned repository:

Azure CLI

```
azd down
```

How do sidecar containers handle internal communication?

Sidecar containers share the same network host as the main container, so the main container (and other sidecar containers) can reach any port on the sidecar with `localhost:<port>`. This is exactly how the Nginx container sends data to the sidecar (see the [OpenTelemetry module configuration for the sample Nginx image ↗](#)).

In the **Edit container** dialog, the **Port** box isn't currently used by App Service. You can use it as part of the sidecar metadata, such as to indicate which port the sidecar is listening to.

Differences for sidecar-enabled apps

You configure sidecar-enabled apps differently than apps that aren't sidecar-enabled. Specifically, you don't configure the main container and sidecars with app settings, but directly in the resource properties. These app settings don't apply for sidecar-enabled apps:

- Registry authentication settings: `DOCKER_REGISTRY_SERVER_URL`,
`DOCKER_REGISTRY_SERVER_USERNAME` and `DOCKER_REGISTRY_SERVER_PASSWORD`.
- Container port: `WEBSITES_PORT`

More resources

- [Configure custom container](#)
- [Deploy custom containers with GitHub Actions](#)
- [OpenTelemetry ↗](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

Migrate custom software to Azure App Service using a custom container

Article • 09/15/2024

Azure App Service uses the Docker container technology to host both built-in images and custom images. To see a list of built-in images, run the Azure CLI command, '[az webapp list-runtimes --os linux](#)'. If those images don't satisfy your needs, you can build and deploy a custom image.

Note

Your container should target the x86-64 architecture. ARM64 is not supported.

In this tutorial, you learn how to:

- ✓ Push a custom Docker image to Azure Container Registry.
- ✓ Deploy the custom image to App Service.
- ✓ Configure environment variables.
- ✓ Pull the image into App Service by using a managed identity.
- ✓ Access diagnostic logs.
- ✓ Enable CI/CD from Azure Container Registry to App Service.
- ✓ Connect to the container by using SSH.

Completing this tutorial incurs a small charge in your Azure account for the container registry and can incur more costs if you host the container for longer than a month.

Set up your initial environment

This tutorial requires version 2.0.80 or later of the Azure CLI. If you're using Azure Cloud Shell, the latest version is already installed.

- Have an Azure account with an active subscription. [Create an account for free](#).
- Use the Bash environment in [Azure Cloud Shell](#). For more information, see [Quickstart for Bash in Azure Cloud Shell](#).

 [Launch Cloud Shell](#) 

- If you prefer to run CLI reference commands locally, [install](#) the Azure CLI. If you're running on Windows or macOS, consider running Azure CLI in a Docker container.

For more information, see [How to run the Azure CLI in a Docker container](#).

- 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 other sign-in options, see [Sign in with the Azure CLI](#).
- When you're prompted, install the Azure CLI extension 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](#).
- Install [Docker](#), which you use to build Docker images. Installing Docker might require a computer restart.

After installing Docker, open a terminal window and verify that the docker is installed:

Bash

```
docker --version
```

Clone or download the sample app

You can obtain the sample for this tutorial via git clone or download.

Clone with Git

Clone the sample repository:

terminal

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

Ensure that you include the `--config core.autocrlf=input` argument to guarantee proper line endings in files that are used inside the Linux container.

Then navigate to the folder:

terminal

```
cd docker-django-webapp-linux
```

Download from GitHub

Instead of using git clone, you can visit <https://github.com/Azure-Samples/docker-django-webapp-linux> and select **Code > Local > Download ZIP**.

Unpack the ZIP file into a folder named *docker-django-webapp-linux*.

Then open a terminal window in the *docker-django-webapp-linux* folder.

(Optional) Examine the Docker file

This is the file in the sample that's named *Dockerfile*. It describes the Docker image and contains configuration instructions.

```
Dockerfile

FROM tiangolo/uwsgi-nginx-flask:python3.6

RUN mkdir /code
WORKDIR /code
ADD requirements.txt /code/
RUN pip install -r requirements.txt --no-cache-dir
ADD . /code/

# ssh
ENV SSH_PASSWD "root:Docker!"
RUN apt-get update \
    && apt-get install -y --no-install-recommends dialog \
    && apt-get update \
    && apt-get install -y --no-install-recommends openssh-server \
    && echo "$SSH_PASSWD" | chpasswd

COPY sshd_config /etc/ssh/
COPY init.sh /usr/local/bin/

RUN chmod u+x /usr/local/bin/init.sh
EXPOSE 8000 2222

#CMD ["python", "/code/manage.py", "runserver", "0.0.0.0:8000"]
ENTRYPOINT ["init.sh"]
```

- The first group of commands installs the app's requirements in the environment.
- The second group of commands creates an [SSH](#) server to provide improved-security communication between the container and the host.
- The last line, `ENTRYPOINT ["init.sh"]`, invokes `init.sh` to start the SSH service and Python server.

Build and test the image locally

ⓘ Note

Docker Hub has [quotas on the number of anonymous pulls per IP and the number of authenticated pulls per free user](#).[↗] If you notice your pulls from Docker Hub are being limited, try `docker login` if you're not already logged in.

1. Run the following command to build the image:

```
Bash
```

```
docker build --tag appsvc-tutorial-custom-image .
```

2. Test that the build works by running the Docker container locally:

```
Bash
```

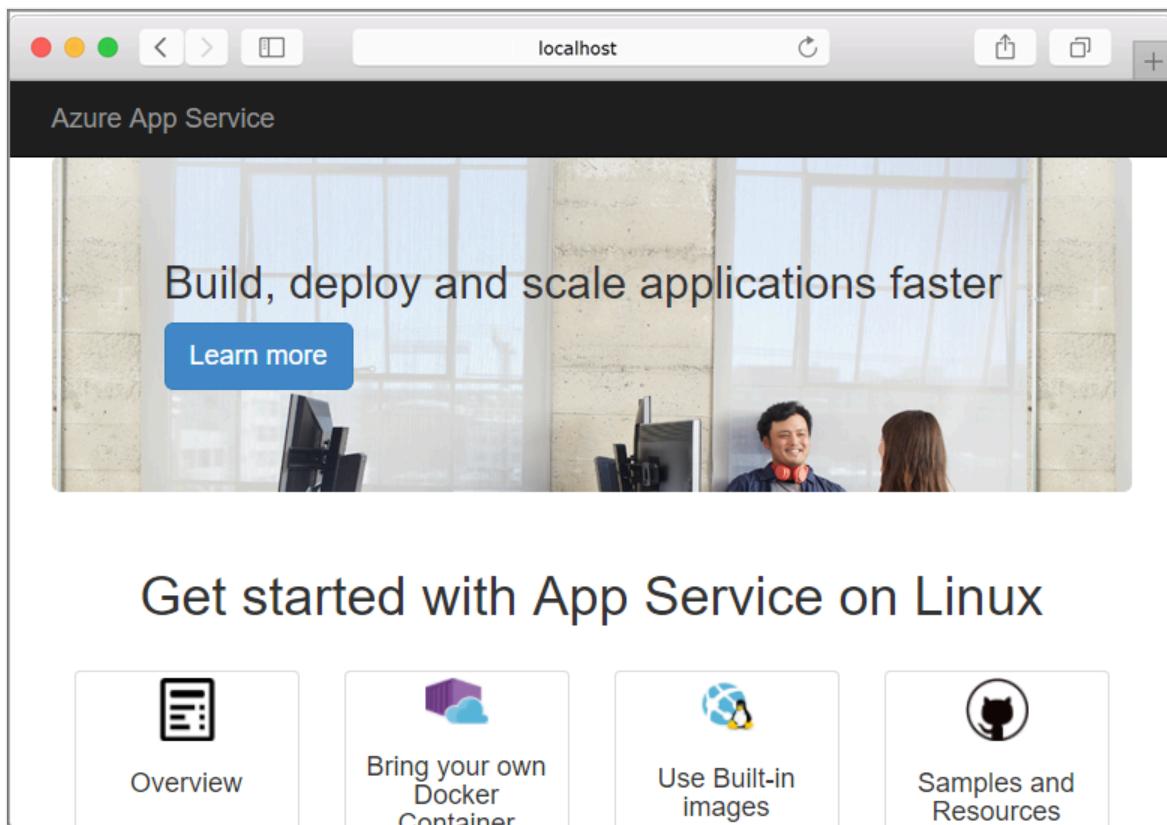
```
docker run -it -p 8000:8000 appsvc-tutorial-custom-image
```

This [docker run](#)[↗] command specifies the port with the `-p` argument and includes the name of the image. `-it` lets you stop it with **Ctrl+C**.

💡 Tip

If you're running on Windows and see the error *standard_init_linux.go:211: exec user process caused "no such file or directory"*, the `init.sh` file contains CRLF line endings instead of the expected LF endings. This error happens if you used Git to clone the sample repository but omitted the `--config core.autocrlf=input` parameter. In this case, clone the repository again with the `--config` argument. You might also see the error if you edited `init.sh` and saved it with CRLF endings. In this case, save the file again with LF endings only.

3. Browse to `http://localhost:8000` to verify that the web app and container are functioning correctly.



I. Create a user-assigned managed identity

App Service can use either a default managed identity or a user-assigned managed identity to authenticate with a container registry. In this tutorial, you'll use a user-assigned managed identity.

Azure CLI

1. Run the `az group create` command to create a resource group:

```
Azure CLI
az group create --name msdocs-custom-container-tutorial --location westeurope
```

You can change the `--location` value to specify a region near you.

2. Create a managed identity in the resource group:

Azure CLI

```
az identity create --name myID --resource-group msdocs-custom-container-tutorial
```

II. Create a container registry

Azure CLI

1. Create a container registry by using the following [az acr create](#) command.

Replace `<registry-name>` with a unique name for your registry. The name must contain only letters and numbers, and must be unique across all of Azure.

Azure CLI

```
az acr create --name <registry-name> --resource-group msdocs-custom-container-tutorial --sku Basic --admin-enabled true
```

The `--admin-enabled` parameter lets you push images to the registry using administrative credentials.

2. Retrieve the administrative credentials by running the [az credential acr show](#) command:

Azure CLI

```
az acr credential show --resource-group msdocs-custom-container-tutorial --name <registry-name>
```

The JSON output of this command provides two passwords along with the registry's user name.

III. Push the sample image to Azure Container Registry

In this section, you push the image to Azure Container Registry, which will be used by App Service later.

1. From the local terminal where you built the sample image, use the `docker login` command to sign in to the container registry:

```
Bash
```

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

Replace `<registry-name>` and `<registry-username>` with values from the previous steps. When prompted, type in one of the passwords from the previous section.

You use the same registry name in all the remaining steps of this section.

2. When the sign-in is successful, tag your local Docker image to the registry:

```
Bash
```

```
docker tag appsvc-tutorial-custom-image <registry-name>.azurecr.io/appsvc-tutorial-custom-image:latest
```

3. Use the `docker push` command to push the image to the registry:

```
Bash
```

```
docker push <registry-name>.azurecr.io/appsvc-tutorial-custom-image:latest
```

Uploading the image the first time might take a few minutes because it includes the base image. Subsequent uploads are typically faster.

While you're waiting, you can complete the steps in the next section to configure App Service to deploy from the registry.

IV. Authorize the managed identity for your registry

The managed identity you created doesn't have authorization to pull from the container registry yet. In this step, you enable the authorization.

Azure CLI

1. Retrieve the principal ID for the managed identity:

Azure CLI

```
principalId=$(az identity show --resource-group msdocs-custom-container-tutorial --name myID --query principalId --output tsv)
```

2. Retrieve the resource ID for the container registry:

Azure CLI

```
registryId=$(az acr show --resource-group msdocs-custom-container-tutorial --name <registry-name> --query id --output tsv)
```

3. Grant the managed identity permission to access the container registry:

Azure CLI

```
az role assignment create --assignee $principalId --scope $registryId --role "AcrPull"
```

For more information about these permissions, see [What is Azure role-based access control?](#).

V. Create the web app

Azure CLI

1. Create an App Service plan using the `az appservice plan create` command:

Azure CLI

```
az appservice plan create --name myAppServicePlan --resource-group msdocs-custom-container-tutorial --is-linux
```

An App Service plan corresponds to the virtual machine that hosts the web app. By default, the previous command uses an inexpensive [B1 pricing tier](#) that's free for the first month. You can specify the tier by using the `--sku` parameter.

2. Create the web app with the `az webapp create` command:

Azure CLI

```
az webapp create --resource-group msdocs-custom-container-tutorial  
--plan myAppServicePlan --name <app-name> --deployment-container-  
image-name <registry-name>.azurecr.io/appsvc-tutorial-custom-  
image:latest
```

Replace `<app-name>` with a name for the web app. The name must be unique across all of Azure. Also replace `<registry-name>` with the name of your registry from the previous section.

💡 Tip

You can retrieve the web app's container settings at any time with the command `az webapp config container show --name <app-name> --resource-group msdocs-custom-container-tutorial`. The image is specified in the property `DOCKER_CUSTOM_IMAGE_NAME`. When the web app is deployed through Azure DevOps or Azure Resource Manager templates, the image can also appear in a property named `LinuxFxVersion`. Both properties serve the same purpose. If both are present in the web app's configuration, `LinuxFxVersion` takes precedence.

VI. Configure the web app

In this step, you configure the web app as follows:

- Configure the app to send requests to port 8000. The sample container is listening on port 8000 for web requests.
- Tell your app to use the managed identity to pull images from your container registry.
- Configure continuous deployment from the container registry (every image push to the registry will trigger your app to pull the new image). This part isn't needed for your web app to pull from your container registry, but it can let your web app know when a new image is pushed to the registry. Without it, you must manually trigger an image pull by restarting the web app.

Azure CLI

1. Use `az webapp config appsettings set` to set the `WEBSITES_PORT` environment variable as expected by the app code:

Azure CLI

```
az webapp config appsettings set --resource-group msdocs-custom-container-tutorial --name <app-name> --settings WEBSITES_PORT=8000
```

Replace `<app-name>` with the name you used in the previous step.

2. Enable the user-assigned managed identity in the web app with the `az webapp identity assign` command:

Azure CLI

```
id=$(az identity show --resource-group msdocs-custom-container-tutorial --name myID --query id --output tsv)
az webapp identity assign --resource-group msdocs-custom-container-tutorial --name <app-name> --identities $id
```

Replace `<app-name>` with the name you used in the previous step.

3. Configure your app to pull from Azure Container Registry by using managed identities.

Azure CLI

```
appConfig=$(az webapp config show --resource-group msdocs-custom-container-tutorial --name <app-name> --query id --output tsv)
az resource update --ids $appConfig --set
properties.acrUseManagedIdentityCreds=True
```

Replace `<app-name>` with the name you used in the previous step.

4. Set the client ID your web app uses to pull from Azure Container Registry. This step isn't needed if you use the system-assigned managed identity.

Azure CLI

```
clientId=$(az identity show --resource-group msdocs-custom-container-tutorial --name myID --query clientId --output tsv)
az resource update --ids $appConfig --set
properties.AcrUserManagedIdentityID=$clientId
```

5. Enable CI/CD in App Service.

Azure CLI

```
cicdUrl=$(az webapp deployment container config --enable-cd true --name <app-name> --resource-group msdocs-custom-container-tutorial --query CI_CD_URL --output tsv)
```

`CI_CD_URL` is a URL that App Service generates for you. Your registry should use this URL to notify App Service that an image push occurred. It doesn't actually create the webhook for you.

6. Create a webhook in your container registry using the `CI_CD_URL` you got from the last step.

Azure CLI

```
az acr webhook create --name appserviceCD --registry <registry-name> --uri $cicdUrl --actions push --scope appsvc-tutorial-custom-image:latest
```

7. To test if your webhook is configured properly, ping the webhook and see if you get a 200 OK response.

Azure CLI

```
eventId=$(az acr webhook ping --name appserviceCD --registry <registry-name> --query id --output tsv)
az acr webhook list-events --name appserviceCD --registry <registry-name> --query "[?id=='$eventId'].eventResponseMessage"
```

💡 Tip

To see all information about all webhook events, remove the `--query` parameter.

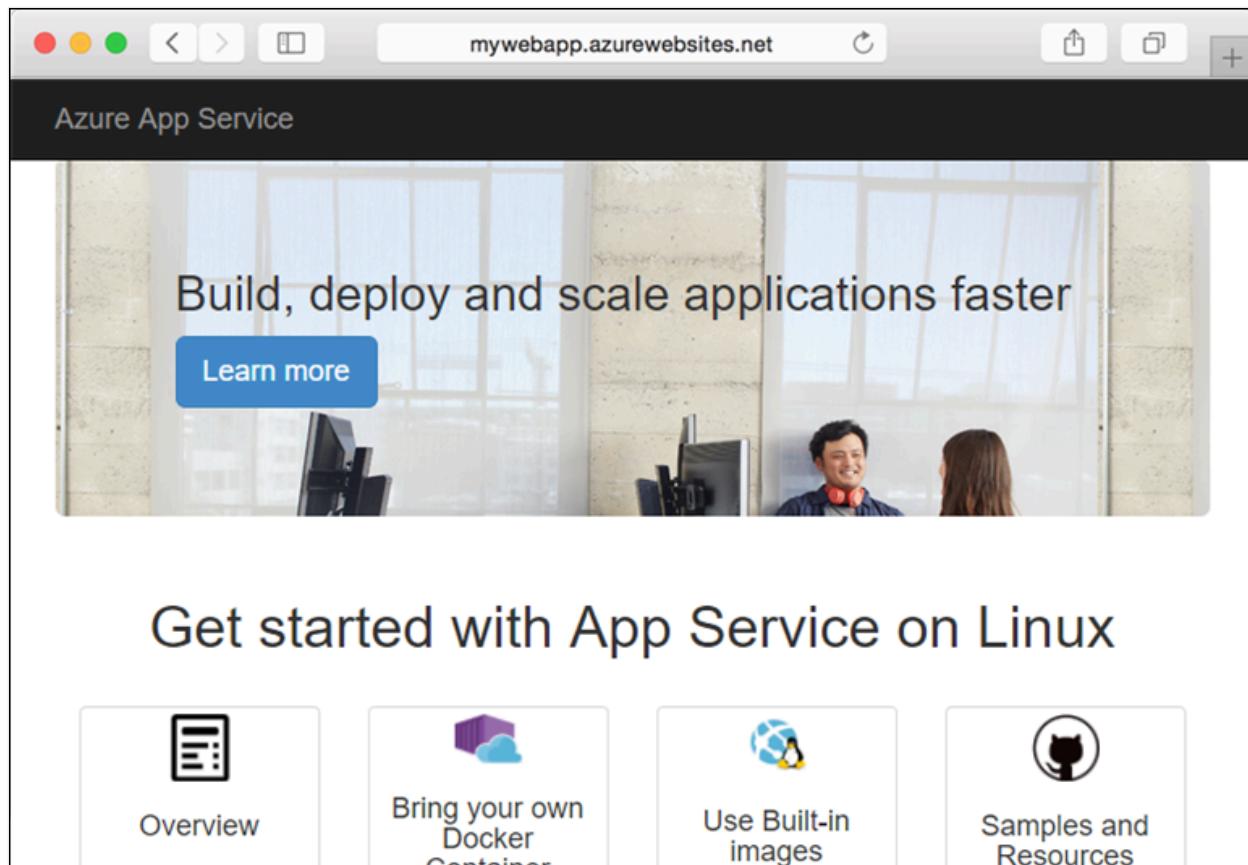
If you're streaming the container log, you should see a `Starting container for site` message after the webhook ping because the webhook triggers the app to restart.

VII. Browse to the web app

Azure CLI

To test the app, browse to `https://<app-name>.azurewebsites.net`. Replace `<app-name>` with the name of your web app.

The first time you attempt to access the app, it might take some time for the app to respond because App Service must pull the entire image from the registry. If the browser times out, just refresh the page. Once the initial image is pulled, subsequent tests will run much faster.



VIII. Access diagnostic logs

Azure CLI

While you're waiting for the App Service to pull in the image, it's helpful to see exactly what App Service is doing by streaming the container logs to your terminal.

1. Turn on container logging:

Azure CLI

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

2. Enable the log stream:

```
Azure CLI
```

```
az webapp log tail --name <app-name> --resource-group msdocs-custom-container-tutorial
```

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

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

3. To stop log streaming at any time, select **Ctrl+C**.

IX. Modify the app code and redeploy

In this section, you make a change to the web app code, rebuild the image, and then push it to your container registry. App Service then automatically pulls the updated image from the registry to update the running web app.

1. In your local *docker-django-webapp-linux* folder, open the file *app/templates/app/index.html*.

2. Change the first HTML element to match the following code.

```
HTML
```

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

3. Save your changes.

4. Change to the *docker-django-webapp-linux* folder and rebuild the image:

```
Bash
```

```
docker build --tag appsvc-tutorial-custom-image .
```

5. Update the image's tag to `latest`:

```
Bash
```

```
docker tag appsvc-tutorial-custom-image <registry-name>.azurecr.io/appsvc-tutorial-custom-image:latest
```

Replace `<registry-name>` with the name of your registry.

6. Push the image to the registry:

```
Bash
```

```
docker push <registry-name>.azurecr.io/appsvc-tutorial-custom-image:latest
```

7. When the image push is complete, the webhook notifies App Service about the push, and App Service tries to pull in the updated image. Wait a few minutes, and then verify that the update has been deployed by browsing to <https://<app-name>.azurewebsites.net>.

X. Connect to the container using SSH

SSH enables improved-security communication between a container and a client. To enable an SSH connection to your container, you must configure your custom image for it. When the container is running, you can open an SSH connection.

Configure the container for SSH

The sample app used in this tutorial already has the necessary configuration in the *Dockerfile*, which installs the SSH server and also sets the sign-in credentials. This section is informational only. To connect to the container, skip to the next section.

```
Dockerfile
```

```
ENV SSH_PASSWD "root:Docker!"  
RUN apt-get update \  
    && apt-get install -y --no-install-recommends dialog \  
    && apt-get update \  
    && apt-get install -y --no-install-recommends openssh-server \  
    && echo "$SSH_PASSWD" | chpasswd
```

ⓘ Note

This configuration doesn't allow external connections to the container. SSH is available only through the Kudu/SCM Site. The Kudu/SCM site is authenticated with your Azure account. `root:Docker!` should not be altered when you use SSH. SCM/KUDU will use your Azure portal credentials. Changing this value will result in an error when you use SSH.

The *Dockerfile* also copies the `sshd_config` file to the `/etc/ssh/` folder and exposes port 2222 on the container:

Dockerfile

```
COPY sshd_config /etc/ssh/  
# ...  
EXPOSE 8000 2222
```

Port 2222 is an internal port accessible only by containers within the bridge network of a private virtual network.

Finally, the entry script, `init.sh`, starts the SSH server.

Bash

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

Open the SSH connection to the container

Azure CLI

1. Browse to <https://<app-name>.scm.azurewebsites.net/webssh/host> and sign in with your Azure account. Replace `<app-name>` with the name of your web app.
2. When you sign in, you're redirected to an informational page for the web app. Select **SSH** at the top of the page to open the shell and use commands.

For example, you can examine the processes running within the app by using the `top` command.

XI. Clean up resources

Azure CLI

The resources you created in this article might incur ongoing costs. To clean up the resources, you only need to delete the resource group that contains them:

Azure CLI

```
az group delete --name msdocs-custom-container-tutorial
```

Next steps

What you learned:

- ✓ Push a custom Docker image to Azure Container Registry.
- ✓ Deploy the custom image to App Service.
- ✓ Configure environment variables.
- ✓ Pull the image into App Service by using a managed identity.
- ✓ Access diagnostic logs.
- ✓ Enable CI/CD from Azure Container Registry to App Service.
- ✓ Connect to the container by using SSH.

In the next tutorial, you learn how to provide security for your app with a custom domain and certificate.

[Provide security with custom domain and certificate](#)

Or, check out other resources:

[Configure a custom container](#)

[Tutorial: Configure a sidecar container for custom container in Azure App Service \(preview\)](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

Deploy a custom container to App Service using Azure Pipelines

Article • 06/11/2024

Azure DevOps enables you to host, build, plan, and test your code with complimentary workflows. Using Azure Pipelines as one of these workflows allows you to deploy your application with CI/CD that works with any platform and cloud. A pipeline is defined as a YAML file in the root directory of your repository.

In this article, we use Azure Pipelines to deploy a Windows container application to App Service from a Git repository in Azure DevOps. It assumes you already have a .NET application with a supporting dockerfile in Azure DevOps.

Prerequisites

- An Azure account with an active subscription. [Create an account for free](#).
- An Azure DevOps organization. [Create one for free](#).
- A working Windows app with Dockerfile hosted on [Azure Repos](#).

Add a Service Connection

Before you create your pipeline, you should first create your Service Connection since you'll be asked to choose and verify your connection when creating your template. A Service Connection allows you to connect to your registry of choice (ACR or Docker Hub) when using the task templates. When adding a new service connection, choose the Docker Registry option. The following form asks you to choose Docker Hub or Azure Container Registry along with pertaining information. To follow along with this tutorial, use Azure Container Registry. You can create a new Service Connection following the directions [here](#).

Secure your secrets

Since we're using sensitive information that you don't want others to access, we use variables to protect our information. Create a variable by following the directions [here](#).

To add a Variable, you click the **Variables** button next to the Save button in the top-right of the editing view for your pipeline. Select the **New Variable** button and enter your information. Add the variables below with your own secrets appropriate from each resource.

- vmlImageName: 'windows-latest'
- imageRepository: 'your-image-repo-name'
- dockerfilePath: '\$(Build.SourcesDirectory)/path/to/Dockerfile'
- dockerRegistryServiceConnection: 'your-service-connection-number'

Create a new pipeline

Once your repository is created with your .NET application and supporting dockerfile, you can create your pipeline following these steps.

1. Navigate to **Pipelines** on the left menu bar and click on the **Create pipeline** button
2. On the next screen, select **Azure Repos Git** as your repository option and select the repository where your code is
3. Under the Configure tab, choose the **Starter Pipeline** option
4. Under the next Review tab, click the **Save** button

Build and push image to Azure container registry

After your pipeline is created and saved, you'll need to edit the pipeline to run the steps for building the container, pushing to a registry, and deploying the image to App Service. To start, navigate to the **Pipelines** menu, choose your pipeline that you created and click the **Edit** button.

First, you need to add the docker task so you can build the image. Add the following code and replace the Dockerfile: app/Dockerfile with the path to your Dockerfile.

```

YAML

trigger:
- main

pool:
  vmImage: 'windows-latest'

variables:
  vmImageName: 'windows-latest'
  imageRepository: 'your-image-repo-name'
  dockerfilePath: '$(Build.SourcesDirectory)/path/to/Dockerfile'
  dockerRegistryServiceConnection: 'your-service-connection-number'

- stage: Build
  displayName: Build and push stage
  jobs:
    - job: Build

```

```
displayName: Build job
pool:
  vmImage: $(vmImageName)
steps:
- task: Docker@2
  displayName: Build and push an image to container registry
  inputs:
    command: buildAndPush
    repository: $(imageRepository)
    dockerfile: $(dockerfilePath)
    containerRegistry: $(dockerRegistryServiceConnection)
    tags: |
      $(tag)
```

Add the App Service deploy task

Next, you need to set up the deploy task. This requires your subscription name, application name, and container registry. Add a new stage to the yaml file by pasting the code below.

YAML

```
- stage: Deploy
  displayName: Deploy to App Service
  jobs:
  - job: Deploy
    displayName: Deploy
    pool:
      vmImage: $(vmImageName)
    steps:
```

Next, navigate to the **Show assistant** tab in the upper right hand corner and find the **Azure App Service deploy** task and fill out the following form:

- Connection type: Azure Resource Manager
- Azure subscription: your-subscription-name
- App Service type: Web App for Containers (Windows)
- App Service name: your-app-name
- Registry or Namespace: your-azure-container-registry-namespace
- Image: your-azure-container-registry-image-name

Once you have those filled out, click the **Add** button to add the task below:

YAML

```
- task: AzureRmWebAppDeployment@4
  inputs:
```

```
ConnectionType: 'AzureRM'
azureSubscription: 'my-subscription-name'
appType: 'webAppHyperVContainer'
WebAppName: 'my-app-name'
DockerNamespace: 'myregsigtry.azurecr.io'
DockerRepository: 'dotnetframework:12'
```

After you've added the task the pipeline is ready to run. Click the **Validate and save** button and run the pipeline. The pipeline goes through the steps to build and push the Windows container image to Azure Container Registry and deploy the image to App Service.

Below is the example of the full yaml file:

YAML

```
trigger:
- main

pool:
  vmImage: 'windows-latest'

variables:
  vmImageName: 'windows-latest'
  imageRepository: 'your-image-repo-name'
  dockerfilePath: '$(Build.SourcesDirectory)/path/to/Dockerfile'
  dockerRegistryServiceConnection: 'your-service-connection-number'

- stage: Build
  displayName: Build and push stage
  jobs:
    - job: Build
      displayName: Build job
      pool:
        vmImage: $(vmImageName)
      steps:
        - task: Docker@2
          displayName: Build and push an image to container registry
          inputs:
            command: buildAndPush
            repository: $(imageRepository)
            dockerfile: $(dockerfilePath)
            containerRegistry: $(dockerRegistryServiceConnection)
            tags: |
              $(tag)

    - stage: Deploy
      displayName: Deploy to App Service
      jobs:
        - job: Deploy
          displayName: Deploy
          pool:
```

```
vmImage: $(vmImageName)
steps:
- task: AzureRmWebAppDeployment@4
  inputs:
    ConnectionType: 'AzureRM'
    azureSubscription: 'my-subscription-name'
    appType: 'webAppHyperVContainer'
    WebAppName: 'my-app-name'
    DockerNamespace: 'myregsitry.azurecr.io'
    DockerRepository: 'dotnetframework:12'
```

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#)

Deploy an application that uses OpenAI on Azure App Service

Article • 01/31/2025

You can use Azure App Service to work with popular AI frameworks like LangChain and Semantic Kernel connected to OpenAI for creating intelligent apps. In the following tutorial, we're adding an Azure OpenAI service using Semantic Kernel to a .NET 8 Blazor web application.

Prerequisites

- An [Azure OpenAI resource](#) or an [OpenAI account](#).
- A .NET 8 Blazor Web App. Create the application with a template [here](#).

Setup Blazor web app

For this Blazor web application, we're building off the Blazor [template](#) and creating a new razor page that can send and receive requests to an Azure OpenAI OR OpenAI service using Semantic Kernel.

1. Right click on the **Pages** folder found under the **Components** folder and add a new item named *OpenAI.razor*
2. Add the following code to the **OpenAI.razor* file and click **Save**

C#

```
@page "/openai"
@rendermode InteractiveServer

<PageTitle>OpenAI</PageTitle>

<h3>OpenAI Query</h3>

<input placeholder="Input query" @bind="newQuery" />
<button class="btn btn-primary" @onclick="SemanticKernelClient">Send
Request</button>

<br />

<h4>Server response:</h4> <p>@serverResponse</p>

@code {

    public string? newQuery;
```

```
    public string? serverResponse;  
}
```

Next, we need to add the new page to the navigation so we can navigate to the service.

1. Go to the `NavMenu.razor` file under the `Layout` folder and add the following div in the nav class. Click **Save**

C#

```
<div class="nav-item px-3">  
    <NavLink class="nav-link" href="openai">  
        <span class="bi bi-list-nested-nav-menu" aria-hidden="true"></span>  
    OpenAI  
    </NavLink>  
</div>
```

After the Navigation is updated, we can start preparing to build the OpenAI client to handle our requests.

API keys and endpoints

In order to make calls to OpenAI with your client, you need to first grab the Keys and Endpoint values from Azure OpenAI, or OpenAI and add them as secrets for use in your application. Retrieve and save the values for later use.

For Azure OpenAI, see [this documentation](#) to retrieve the key and endpoint values. If you're planning to use [managed identity](#) to secure your app you'll only need the `deploymentName` and `endpoint` values. Otherwise, you need each of the following:

- `deploymentName`
- `endpoint`
- `apiKey`
- `modelId`

For OpenAI, see this [documentation](#) to retrieve the API keys. For our application, you need the following values:

- `apiKey`
- `modelId`

Since we're deploying to App Service, we can secure these secrets in **Azure Key Vault** for protection. Follow the [Quickstart](#) to set up your Key Vault and add the secrets you saved

from earlier. Next, we can use Key Vault references as app settings in our App Service resource to reference in our application. Follow the instructions in the [documentation](#) to grant your app access to your Key Vault and to set up Key Vault references. Then, go to the portal Environment Variables blade in your resource and add the following app settings:

For Azure OpenAI, use the following settings:

[+] Expand table

Setting name	Value
DEPLOYMENT_NAME	@Microsoft.KeyVault(SecretUri=https://myvault.vault.azure.net/secrets/mysecret/)
ENDPOINT	@Microsoft.KeyVault(SecretUri=https://myvault.vault.azure.net/secrets/mysecret/)
API_KEY	@Microsoft.KeyVault(SecretUri=https://myvault.vault.azure.net/secrets/mysecret/)
MODEL_ID	@Microsoft.KeyVault(SecretUri=https://myvault.vault.azure.net/secrets/mysecret/)

For OpenAI, use the following settings:

[+] Expand table

Setting name	Value
OPENAI_API_KEY	@Microsoft.KeyVault(SecretUri=https://myvault.vault.azure.net/secrets/mysecret/)
OPENAI_MODEL_ID	@Microsoft.KeyVault(SecretUri=https://myvault.vault.azure.net/secrets/mysecret/)

Once your app settings are saved, you can bring them into the code by injecting IConfiguration and referencing the app settings. Add the following code to your *OpenAI.razor* file:

For Azure OpenAI:

C#

```
@inject Microsoft.Extensions.Configuration.IConfiguration _config

@code {

    private async Task SemanticKernelClient()
    {
        string deploymentName = _config["DEPLOYMENT_NAME"];
        string endpoint = _config["ENDPOINT"];
        string apiKey = _config["API_KEY"];
```

```
        string modelId = _config["MODEL_ID"];
    }
```

For OpenAI:

```
C#
```

```
@inject Microsoft.Extensions.Configuration.IConfiguration _config

@code {

    private async Task SemanticKernelClient()
    {
        // OpenAI
        string OpenAIModelId = _config["OPENAI_MODEL_ID"];
        string OpenAIApiKey = _config["OPENAI_API_KEY"];
    }
}
```

Semantic Kernel

Semantic Kernel is an open-source SDK that enables you to easily develop AI agents to work with your existing code. You can use Semantic Kernel with Azure OpenAI and OpenAI models.

To create the OpenAI client, we'll first start by installing Semantic Kernel.

To install Semantic Kernel, browse the NuGet package manager in Visual Studio and install the **Microsoft.SemanticKernel** package. For NuGet Package Manager instructions, see [here](#). For CLI instructions, see [here](#). Once the Semantic Kernel package is installed, you can now initialize the kernel.

Initialize the kernel

To initialize the Kernel, add the following code to the *OpenAI.razor* file.

```
C#
```

```
@code {

    @using Microsoft.SemanticKernel;

    private async Task SemanticKernelClient()
    {
        var builder = Kernel.CreateBuilder();

        var kernel = builder.Build();
```

```
    }  
}
```

Here we're adding the using statement and creating the Kernel in a method that we can use when we send the request to the service.

Add your AI service

Once the Kernel is initialized, we can add our chosen AI service to the kernel. Here we define our model and pass in our key and endpoint information to be consumed by the chosen model. If you plan to use managed identity with Azure OpenAI, add the service using the example in the next section.

For Azure OpenAI, use the following code:

```
C#  
  
var builder = Kernel.CreateBuilder();  
builder.Services.AddAzureOpenAIChatCompletion(  
    deploymentName: deploymentName,  
    endpoint: endpoint,  
    apiKey: apiKey,  
    modelId: modelId  
);  
var kernel = builder.Build();
```

For OpenAI, use the following code:

```
C#  
  
var builder = Kernel.CreateBuilder();  
builder.Services.AddOpenAIChatCompletion(  
    modelId: OpenAIModelId,  
    apiKey: OpenAIApiKey,  
);  
var kernel = builder.Build();
```

Secure your app with managed identity

If you're using Azure OpenAI, it's highly recommended to secure your application using [managed identity](#) to authenticate your app to your Azure OpenAI resource. This enables your application to access the Azure OpenAI resource without needing to manage API

keys. If you're not using Azure OpenAI, your secrets can remain secure using Azure Key Vault outlined above.

Follow the steps below to secure your application with managed identity:

Add the identity package `Azure.Identity`. This package enables using Azure credentials in your app. Install the package using NuGet package manager and add the using statement to the top of the `OpenAI.razor` file.

```
c#  
  
@using Azure.Identity
```

Next, include the default Azure credentials in the chat completions parameters. The `deploymentName` and `endpoint` parameters are still required and should be secured using the Key Vault method covered in the previous section.

```
c#  
  
var kernel = Kernel.CreateBuilder()  
    .AddAzureOpenAIChatCompletion(  
        deploymentName: deploymentName,  
        endpoint: endpoint,  
        credentials: new DefaultAzureCredential()  
    )  
    .Build();
```

Once the credentials are added to the application, you'll then need to enable managed identity in your application and grant access to the resource.

1. In your web app resource, navigate to the **Identity** blade and turn on **System assigned** and click **Save**
2. Once System assigned identity is turned on, it registers the web app with Microsoft Entra ID and the web app can be granted permissions to access protected resources.
3. Go to your Azure OpenAI resource and navigate to the **Access control (IAM)** blade on the left pane.
4. Find the Grant access to this resource card and click on **Add role assignment**
5. Search for the **Cognitive Services OpenAI User** role and click **Next**
6. On the **Members** tab, find **Assign access to** and choose the **Managed identity** option
7. Next, click on **+Select Members** and find your web app
8. Click **Review + assign**

Your web app is now added as a cognitive service OpenAI user and can communicate to your Azure OpenAI resource.

Configure prompt and create semantic function

Now that our chosen OpenAI service client is created with the correct keys we can add a function to handle the prompt. With Semantic Kernel you can handle prompts by the use of a semantic function, which turn the prompt and the prompt configuration settings into a function the Kernel can execute. Learn more on configuring prompts [here](#).

First, we create a variable that holds the user's prompt. Then add a function with execution settings to handle and configure the prompt. Add the following code to the *OpenAI.razor* file:

C#

```
@using Microsoft.SemanticKernel.Connectors.OpenAI

private async Task SemanticKernelClient()
{
    var builder = Kernel.CreateBuilder();
    builder.Services.AddAzureOpenAIChatCompletion(
        deploymentName: deploymentName,
        endpoint: endpoint,
        APIKey: APIKey,
        modelId: modelId
    );
    var kernel = builder.Build();

    var prompt = @"{{$input}} " + newQuery;

    var summarize = kernel.CreateFunctionFromPrompt(prompt,
executionSettings: new OpenAIPromptExecutionSettings { MaxTokens = 100,
Temperature = 0.2 });

}
```

Lastly, we need to invoke the function and return the response. Add the following to the *OpenAI.razor* file:

C#

```
private async Task SemanticKernelClient()
{
```

```

var builder = Kernel.CreateBuilder();
builder.Services.AddAzureOpenAIChatCompletion(
    deploymentName: deploymentName,
    endpoint: endpoint,
    APIKey: APIKey,
    modelId: modelId
);
var kernel = builder.Build();

var prompt = "{$input} " + newQuery;

var summarize = kernel.CreateFunctionFromPrompt(prompt,
executionSettings: new OpenAIPromptExecutionSettings { MaxTokens = 100,
Temperature = 0.2 })

var result = await kernel.InvokeAsync(summarize);

serverResponse = result.ToString();

}

```

Here's the example in its completed form. In this example, use the Azure OpenAI chat completion service OR the OpenAI chat completion service, not both.

C#

```

@page "/openai"
@rendermode InteractiveServer
@inject Microsoft.Extensions.Configuration.IConfiguration _config

<PageTitle>OpenAI</PageTitle>

<h3>OpenAI input query: </h3>
<input class="col-sm-4" @bind="newQuery" />
<button class="btn btn-primary" @onclick="SemanticKernelClient">Send Request</button>

<br />
<br />

<h4>Server response:</h4> <p>@serverResponse</p>

@code {
    @using Microsoft.SemanticKernel;
    @using Microsoft.SemanticKernel.Connectors.OpenAI

    private string? newQuery;
    private string? serverResponse;

    private async Task SemanticKernelClient()
    {

```

```

    // Azure OpenAI
    string deploymentName = _config["DEPLOYMENT_NAME"];
    string endpoint = _config["ENDPOINT"];
    string apiKey = _config["API_KEY"];
    string modelId = _config["MODEL_ID"];

    // OpenAI
    // string OpenAIModelId = _config["OPENAI_DEPLOYMENT_NAME"];
    // string OpenAIApiKey = _config["OPENAI_API_KEY"];
    // Semantic Kernel client
    var builder = Kernel.CreateBuilder();
    // Azure OpenAI
    builder.Services.AddAzureOpenAIChatCompletion(
        deploymentName: deploymentName,
        endpoint: endpoint,
        apiKey: apiKey,
        modelId: modelId
    );
    // OpenAI
    // builder.Services.AddOpenAIChatCompletion(
    //     modelId: OpenAIModelId,
    //     apiKey: OpenAIApiKey
    // );
    var kernel = builder.Build();

    var prompt = @"{{$input}} " + newQuery;

    var summarize = kernel.CreateFunctionFromPrompt(prompt,
executionSettings: new OpenAIPromptExecutionSettings { MaxTokens = 100,
Temperature = 0.2 });

    var result = await kernel.InvokeAsync(summarize);

    serverResponse = result.ToString();

}
}

```

Now save the application and follow the next steps to deploy it to App Service. If you would like to test it locally first at this step, you can swap out the config values at with the literal string values of your OpenAI service. For example: string modelId = 'gpt-4-turbo';

Deploy to App Service

If you have followed the steps above, you're ready to deploy to App Service. If you run into any issues remember that you need to have done the following: grant your app access to your Key Vault, add the app settings with key vault references as your values. App Service resolves the app settings in your application that match what you've added in the portal.

Authentication

Although optional, it's highly recommended that you also add authentication to your web app when using an Azure OpenAI or OpenAI service. This can add a level of security with no other code. Learn how to enable authentication for your web app [here](#).

Once deployed, browse to the web app and navigate to the OpenAI tab. Enter a query to the service and you should see a populated response from the server. The tutorial is now complete and you now know how to use OpenAI services to create intelligent applications.

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

What is WordPress on App Service?

Article • 12/10/2024

WordPress is one of the world's most popular content management systems (CMS), powering over 40% of websites globally. It enables users to create and manage websites with ease, offering flexibility for blogs, e-commerce platforms, portfolios, corporate sites, and more. Its extensive plugin ecosystem and customizable themes make it a versatile choice for developers and content creators alike.

With WordPress on Azure App Service, you can focus on creating content while Azure takes care of the infrastructure, security, and performance needs. App Service provides a streamlined and scalable platform for hosting WordPress websites. By using Azure's powerful infrastructure, including **Azure App Service**, **Azure Database for MySQL**, **Azure CDN**, and **Azure Blob Storage**, you can quickly deploy and manage a secure, high-performance WordPress site.

This solution is designed to meet the needs of both small and large-scale deployments, whether you're running a personal blog, a corporate website, or an e-commerce platform. With features like automated updates, advanced security, and global availability, WordPress on Azure App Service simplifies infrastructure management, so you can focus on your content and audience.

[Quickstart documentation | Create a WordPress site using the Azure portal ↗](#)

How does App Service simplify WordPress?

Azure App Service makes deploying WordPress sites straightforward for both beginners and experienced developers:

- **Automatic updates:** Automatically updates technologies like Linux, PHP, and NGINX to keep your site secure and up to date.
- **Pre-configured setup:** By leveraging powerful Azure services like Azure App Service, Azure Database for MySQL, Azure CDN, and Azure Blob Storage, you get a pre-configured setup optimized for performance and security.
- **Flexible file transfers:** Easily transfer files via FTP for custom content uploads.
- **Custom server configurations:** Adjust NGINX settings using startup scripts.
- **Staging slots:** Safely test changes in isolated environments before deploying them to production.

What enterprise-grade features are available?

Azure App Service provides advanced tools to ensure your WordPress site is secure, reliable, scalable, and performant:

- **Custom domains and SSL certificates:** Set up personalized domains and secure your site with HTTPS.
- **Scalability:** Adjust resources automatically or manually to handle traffic spikes.
- **CI/CD pipelines:** Automate deployments with continuous integration and deployment workflows.
- **Email integration:** Use custom email domains for professional communication to enhance branding and customer interactions.
- **Load testing:** Simulate real-world traffic conditions to optimize your site's performance.
- **Security:** SSL (both free and paid options), VNET, DDoS protection, Web Application Firewall, Active directory integration, IP restriction settings and many more. Additionally, you can integrate with Microsoft Defender for Cloud to receive security scans and insights, further bolstering your site's security. When you deploy your WordPress site, it runs on its own dedicated instance and NOT a shared instance, making it more performant, secure and avoiding noisy neighbors.

What are the best scenarios for WordPress on App Service?

WordPress on Azure App Service supports a wide range of use cases, including:

- **Corporate websites and intranets:** Scalable and secure solutions for public-facing sites or internal portals.
- **E-commerce platforms:** Handle traffic spikes while ensuring secure customer data management.
- **Content management systems (CMS):** Ideal for blogs, portfolios, and headless CMS applications.
- **Community and social networking sites:** Manage high volumes of user-generated content seamlessly.
- **Marketing campaigns:** Deploy high-performance landing pages to support digital marketing initiatives.

What hosting plans are available?

Azure offers flexible hosting plans to accommodate different needs, from testing to enterprise-grade production workloads:

Plan	App Service	Database for MySQL
Free	F1 (60 CPU mins/day, 1-GB RAM, 1-GB storage)	Burstable, B1ms (2-GB RAM, 32-GB storage)
Basic	B1 (1 core, 1.75-GB RAM, 10-GB storage)	Burstable, B1ms (2-GB RAM, 32-GB storage)
Standard	P0V3 (1 core, 4-GB RAM, 250-GB storage)	B2s (4-GB RAM, 128-GB storage)
Premium	P1V3 (2 cores, 8-GB RAM, 250-GB storage)	D2ds_v4 (8-GB RAM, 128-GB storage)

ⓘ Note

Use the [Azure Pricing Calculator](#) to estimate costs for your specific needs.

Why should you choose WordPress on App Service?

WordPress on Azure App Service combines ease of management, powerful performance features, and enterprise-grade security to provide a reliable platform for hosting WordPress websites. Whether you're managing a small blog or a high-traffic business site, Azure App Service offers the tools and scalability to meet your needs.

- **Simplified management:** Easily handle deployments, updates, backups, and configurations through the Azure portal.
- **Performance and scalability:** Meet traffic demands with autoscaling, CDN integration, and caching solutions.
- **Enterprise-level security:** Secure your site with SSL, VNET, Azure Key Vault, and Azure Defender.
- **Seamless integration:** Extend your site's capabilities with services like Azure Front Door, Azure Monitor, and Azure Backup.
- **Global availability:** Deploy your site in Azure's global regions for low latency and enhanced performance.

What are the limitations of WordPress on App Service?

While WordPress on Azure App Service provides robust features and scalability, it has some limitations to consider:

- **Performance on Shared Hosting Plans:** While Free and Basic plans are cost-effective for smaller sites, they may not provide the performance needed for high-traffic websites.
- **No Support for Windows-based Hosting:** WordPress on App Service is Linux-based, and Windows-based hosting is not available.
- **Custom NGINX Configuration Complexity:** Customizing NGINX settings involves creating and managing startup scripts, which may require technical expertise.

By understanding these limitations, users can plan effectively and decide whether this solution aligns with their specific requirements.

Learning resources

Explore more about WordPress on Azure App Service:

- [Create a WordPress site](#)
- [GitHub documentation ↗](#)
- [Sidecar configuration](#)

Next steps

To get started with WordPress on Azure App Service:

- [Explore the Quickstart documentation.](#)
- [Deploy your WordPress site ↗](#).

ⓘ **Note:** The author created this article with assistance from AI. [Learn more](#)

Feedback

Was this page helpful?

 Yes

 No

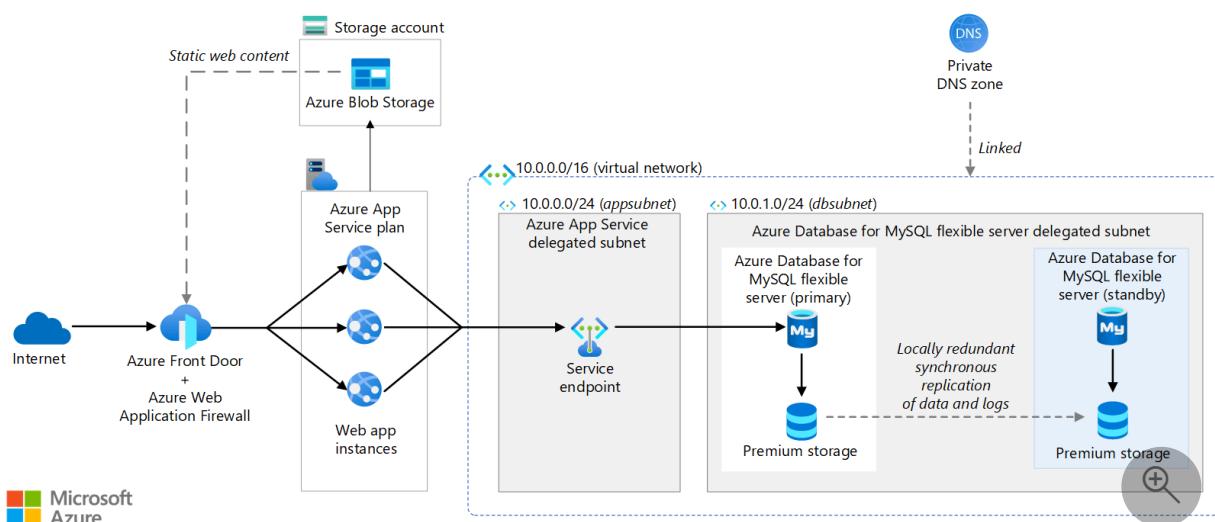
[Provide product feedback ↗](#) | Get help at Microsoft Q&A

WordPress on App Service

Azure Front Door Azure Load Balancer Azure Virtual Network Azure App Service
Azure Database for MySQL

This article describes a solution for small to medium-sized WordPress installations. The solution provides the scalability, reliability, and security of the Azure platform without the need for complex configuration or management. For solutions for larger or storage-intensive installations, see [WordPress hosting options on Azure](#).

Architecture



Download a [Visio file](#) of this architecture.

ⓘ Note

You can extend this solution by implementing tips and recommendations that aren't specific to any particular WordPress hosting method. For general tips for deploying a WordPress installation, see [WordPress on Azure](#).

Dataflow

This scenario covers a scalable installation of WordPress that runs on Azure App Service.

- Users access the front-end website through Azure Front Door with Azure Web Application Firewall enabled.

- Azure Front Door distributes requests across the App Service web apps that WordPress runs on. Azure Front Door retrieves any data that isn't cached from the WordPress web apps.
- The WordPress application uses a service endpoint to access a flexible server instance of Azure Database for MySQL. The WordPress application retrieves dynamic information from the database.
- Locally redundant high availability is enabled for Azure Database for MySQL via a standby server in the same availability zone.
- All static content is hosted in Azure Blob Storage.

Components

- The [WordPress on App Service template](#) is a managed solution for hosting WordPress on App Service. Besides App Service, the solution also uses the other Azure services that are described in this section.
- [App Service](#) provides a framework for building, deploying, and scaling web apps.
- [Azure Front Door](#) is a modern cloud content delivery network. As a distributed network of servers, Azure Front Door efficiently delivers web content to users. Content delivery networks minimize latency by storing cached content on edge servers in point-of-presence locations near end users.
- [Azure Content Delivery Network](#) efficiently delivers web content to users by storing blobs at strategically placed locations. In this solution, you can use Content Delivery Network as an alternative to Azure Front Door.
- [Azure Virtual Network](#) provides a way for deployed resources to communicate with each other, the internet, and on-premises networks. Virtual networks provide isolation and segmentation. They also filter and route traffic and make it possible to establish connections between various locations. In this solution, the two networks are connected via virtual network peering.
- [Azure DDoS Protection](#) provides enhanced DDoS mitigation features. When you combine these features with application-design best practices, they help defend against DDoS attacks. You should enable DDoS Protection on perimeter virtual networks.
- [Network security groups](#) use a list of security rules to allow or deny inbound or outbound network traffic based on source or destination IP address, port, and protocol. In this scenario's subnets, network security group rules restrict traffic flow between the application components.
- [Azure Key Vault](#) stores and controls access to passwords, certificates, and keys.
- [Azure Database for MySQL - flexible server](#) is a relational database service that's based on the open-source MySQL database engine. The flexible server deployment option is a fully managed service that provides granular control and flexibility over

database management functions and configuration settings. In this scenario, Azure Database for MySQL stores WordPress data.

- [Blob Storage](#) provides scalable, optimized object storage. Blob Storage is a good fit for cloud-native workloads, archives, data lakes, high-performance computing, and machine learning.

Alternatives

- You can use [Azure Cache for Redis](#) to host a key-value cache for WordPress performance optimization plug-ins. The cache can be shared among the App Service web apps.
- Instead of Azure Front Door, you can use Content Delivery Network to deliver web content to users.

Scenario details

This example scenario is appropriate for small to medium-sized installations of WordPress.

Potential use cases

- Media events that cause traffic surges
- Blogs that use WordPress as their content management system
- Business or e-commerce websites that use WordPress
- Websites that are built by using other content management systems

Considerations

These considerations implement the pillars of the Azure Well-Architected Framework, which is a set of guiding tenets that can be used to improve the quality of a workload. For more information, see [Microsoft Azure Well-Architected Framework](#).

Reliability

Reliability ensures your application can meet the commitments you make to your customers. For more information, see [Overview of the reliability pillar](#).

Consider the following recommendations when you deploy this solution:

- App Service provides built-in load balancing and health checks. These features help you maintain availability when an App Service web app fails.
- When you use a content delivery network to cache all responses, you gain a small availability benefit. Specifically, when the origin doesn't respond, you can still access content. But caching doesn't provide a complete availability solution.
- You can replicate Blob Storage to a paired region for data redundancy across multiple regions. For more information, see [Azure Storage redundancy](#).
- To increase Azure Database for MySQL availability, enable same-zone high availability. This feature creates a standby server in the same availability zone as the primary server. You need to use the General Purpose or Business Critical compute tier to enable same-zone high availability. For more information, see the [high availability options](#) that apply to your needs.

Security

Security provides assurances against deliberate attacks and the abuse of your valuable data and systems. For more information, see [Overview of the security pillar](#).

Consider the following recommendations when you deploy this solution:

- Use Azure Web Application Firewall on Azure Front Door to help protect the virtual network traffic that flows into the front-end application tier. For more information, see [Azure Web Application Firewall on Azure Front Door](#).
- Don't allow outbound internet traffic to flow from the database tier.
- Don't allow public access to private storage.

For more information about WordPress security, see [General WordPress security and performance tips](#) and [Azure security documentation](#).

Cost optimization

Cost optimization is about looking at ways to reduce unnecessary expenses and improve operational efficiencies. For more information, see [Overview of the cost optimization pillar](#).

Review the following cost considerations when you deploy this solution:

- **Traffic expectations (GB/month).** Your traffic volume is the factor that has the greatest effect on your cost. The amount of traffic that you receive determines the number of App Service instances that you need and the price for outbound data transfer. The traffic volume also directly correlates with the amount of data that's

provided by your content delivery network, where outbound data transfer costs are cheaper.

- **Amount of hosted data.** It's important to consider the amount of data that you host in Blob Storage. Storage pricing is based on used capacity.
- **Write percentage.** Consider how much new data you write to your website and host in Azure Storage. Determine whether the new data is needed. For multi-region deployments, the amount of new data that you write to your website correlates with the amount of data that's mirrored across your regions.
- **Static versus dynamic content.** Monitor your database storage performance and capacity to determine whether a cheaper SKU can support your site. The database stores dynamic content, and the content delivery network caches static content.
- **App Service optimization.** For general tips for optimizing App Service costs, see [Azure App Service and cost optimization](#).

Performance efficiency

Performance efficiency is the ability of your workload to scale to meet the demands placed on it by users in an efficient manner. For more information, see [Performance efficiency pillar overview](#).

This scenario hosts the WordPress front end in App Service. You should enable the autoscale feature to automatically scale the number of App Service instances. You can set an autoscale trigger to respond to customer demand. You can also set a trigger that's based on a defined schedule. For more information, see [Get started with autoscale in Azure](#) and the Azure Well-Architected Framework article [Performance efficiency principles](#).

Contributors

This article is maintained by Microsoft. It was originally written by the following contributors.

Principal author:

- [Vaclav Jirovsky](#) | Cloud Solution Architect

Other contributors:

- Adrian Calinescu | Senior Cloud Solution Architect

To see nonpublic LinkedIn profiles, sign in to LinkedIn.

Next steps

Product documentation:

- [What is Azure Front Door?](#)
- [What is Azure Web Application Firewall?](#)
- [What is Azure Blob Storage?](#)
- [Azure Database for MySQL - Flexible Server](#)
- [What is Azure Virtual Network?](#)
- [About Azure Key Vault](#)
- [Quickstart: Create a WordPress site](#)
- [What is Azure DDoS Protection?](#)

Microsoft training modules:

- [Load balance your web service traffic with Azure Front Door](#)
- [Implement Azure Key Vault](#)
- [Introduction to Azure Virtual Network](#)

Related resources

- [Ten design principles for Azure applications](#)
- [Scalable cloud applications and site reliability engineering](#)

Feedback

Was this page helpful?



Create a WordPress site

Article • 03/28/2024

[WordPress](#) is an open source Content Management System (CMS) used by over 40% of the web to create websites, blogs, and other applications. WordPress can be run on a few different Azure services: [AKS](#), [Virtual Machines](#), [Azure Container Apps](#) and Azure App Service. For a full list of WordPress options on Azure, see [WordPress on Azure Marketplace](#).

In this quickstart, you'll learn how to create and deploy your first [WordPress](#) site to [Azure App Service on Linux](#) with [Azure Database for MySQL - Flexible Server](#) using the [WordPress Azure Marketplace item by App Service](#). This quickstart uses the **Standard** tier for your app and a **Burstable, B2s** tier for your database, and incurs a cost for your Azure Subscription. For pricing, visit [App Service pricing](#), [Azure Database for MySQL pricing](#), [Content Delivery Network pricing](#), and [Azure Blob Storage pricing](#).

To complete this quickstart, you need an Azure account with an active subscription. [Create an account for free](#).

Create WordPress site using Azure portal

1. To start creating the WordPress site, browse to <https://portal.azure.com/#create/WordPress.WordPress>.

Microsoft Azure Search resources, services, and docs (G+)

Home > Marketplace > WordPress on App Service >

Create WordPress on App Service

Basics Advanced Tags Review + create

WordPress optimized for App Service with best practices for security and performance. [Learn More](#)

Project details

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

Subscription * ⓘ

Resource group * ⓘ (New) Resource group

Hosting details

Select the region for your server and provide a name for Web App. Custom domains can be added later.

Region * South Central US

Name * Web App name. .azurewebsites.net

Hosting plans

This hosting plan dictates what resources are available, what features are enabled and how it is priced.

Hosting plan	Standard PremiumV2 App Service, Burstable MySQL database Change plan
--------------	---

WordPress setup

Select the language you want your website to be in. Provide admin email, username and password that you can use to access WordPress admin dashboard.

Site language	<input type="text"/> English (United States)
Admin email *	<input type="text"/>
Admin username * ⓘ	<input type="text"/>
Admin password * ⓘ	<input type="text"/> Password
Confirm password *	<input type="text"/> Confirm password
Enable multisite ⓘ	<input type="checkbox"/>

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

- In the Basics tab, under Project details, make sure the correct subscription is selected. Select Create new resource group and type `myResourceGroup` for the name.

Basics Advanced Tags Review + create

WordPress optimized for App Service with best practices for security and performance. [Learn More ↗](#)

Project details

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

Subscription * ⓘ

Resource group * ⓘ

[Create new](#)

3. Under **Hosting details**, select a **Region** you want to serve your app from, then type a globally unique name for your web app. Under **Hosting plans**, select **Standard**. Select **Change plan** to view features and price comparisons.

Hosting details

Select the region for your server and provide a name for Web App. Custom domains can be added later.

Region *

Name *

.azurewebsites.net

Hosting plans

This hosting plan dictates what resources are available, what features are enabled and how it is priced.

Hosting plan

Standard

PremiumV2 App Service, Burstable MySQL database

[Change plan](#)

4. Under **WordPress setup**, choose your preferred **Site Language**, then type an **Admin Email**, **Admin Username**, and **Admin Password**. The **Admin Email** is used for WordPress administrative sign-in only. Clear the **Enable multisite** checkbox.

WordPress setup

Select the language you want your website to be in. Provide admin email, username and password that you can use to access WordPress admin dashboard.

Site language

Admin email *

Admin username * ⓘ

Admin password * ⓘ

Confirm password *

Enable multisite ⓘ

5. Select the **Advanced** tab. If you're unfamiliar with an [Azure CDN](#), [Azure Front Door](#), or [Blob Storage](#), then clear the checkboxes. For more details on the Content Distribution options, see [WordPress on App Service ↗](#).

Basics Advanced Tags Review + create

This section allows you to opt for advanced features. The recommended features are selected by default. Please go through the descriptions before making changes.

Azure CDN (Recommended)

Azure Content Delivery Network helps in improving performance, availability, and security by using a distributed network of servers that can store cached content in point-of-presence locations, close to end users. [Learn More](#)

Enable Azure CDN [\(i\)](#)

Azure Front Door (Preview)

Azure Front Door (AFD) provides dynamic site acceleration that reduces response times while also allowing content delivery by caching at nearest edge servers for faster media downloads. [Learn More](#)

Enable AFD [\(i\)](#)

AFD Profiles

Select AFD Profile



You can choose to select either Azure CDN or Azure Front Door.

Azure Blob Storage (Recommended)

Azure Blob Storage allows you to store and access images, videos and other files. This effectively reduces the load on your web server thereby improving performance and user experience. [Learn More](#)

Enable Blob Storage [\(i\)](#)

Storage Account [\(i\)](#)

(New) mydemowpsi23bb6e5df3



Virtual Network

Virtual networks are logically isolated from each other in Azure. You can configure their IP address ranges, subnets, route tables, gateways, and security settings, much like a traditional network in your data center. Virtual machines in the same virtual network can access each other by default. [Learn More](#)

Virtual Network [\(i\)](#)

(New) mydemowpsi-21f6b45092-vnet



Note

The WordPress app requires a virtual network with an address space of /23 at minimum.

6. Select the **Review + create** tab. After validation runs, select the **Create** button at the bottom of the page to create the WordPress site.

Create

< Previous

Next >

7. Browse to your site URL and verify the app is running properly. The site may take a few minutes to load. If you receive an error, allow a few more minutes then refresh the browser.

Hello world!

Welcome to WordPress. This is your first post. Edit or delete it, then start writing!

Published January 24, 2022
Categorized as [Uncategorized](#)

Search

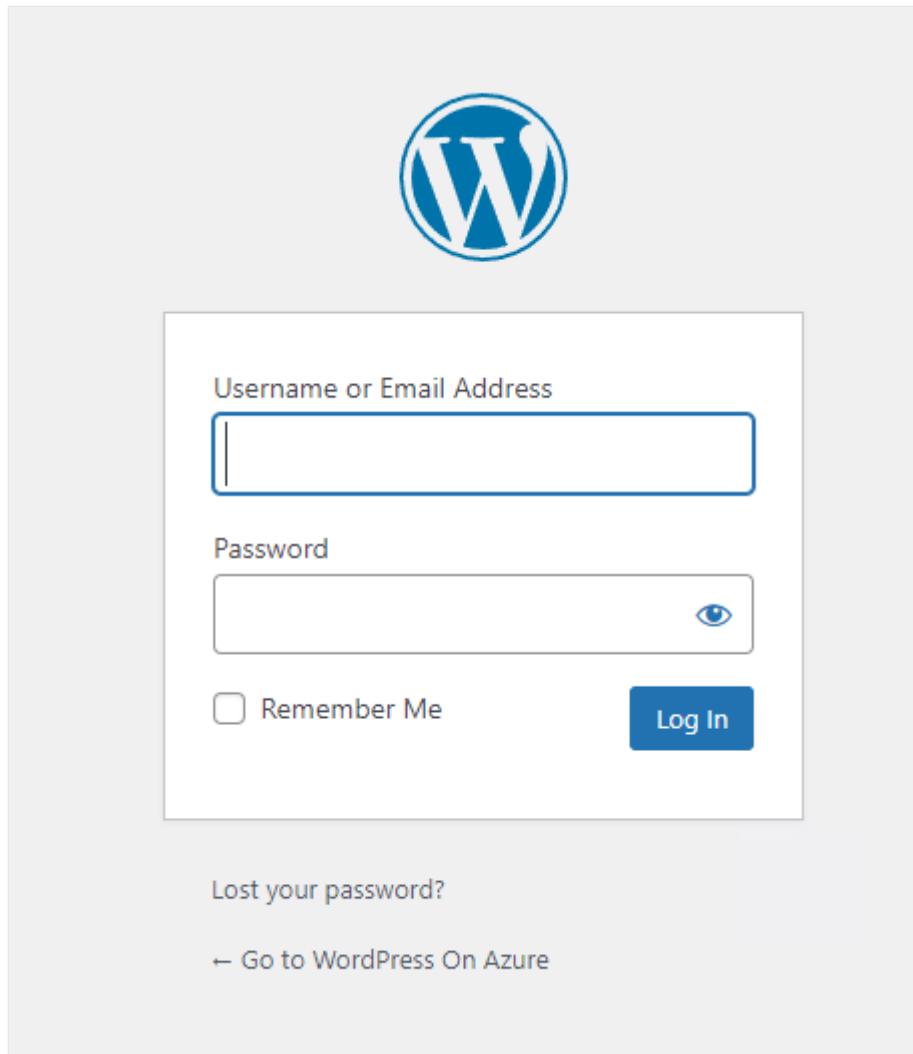
Recent Posts

[Hello world!](#)

Recent Comments

[A WordPress Commenter](#) on [Hello world!](#)

8. To access the WordPress Admin page, browse to `/wp-admin` and use the credentials you created in the [WordPress setup](#) step.



Clean up resources

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

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

The screenshot shows the Azure portal interface for an App Service named 'myWPApp'. On the left, there's a sidebar with links like Overview, Activity log, Access control (IAM), Tags, Diagnose and solve problems, Security, and Events (preview). The main content area has a header with 'Browse', 'Stop', 'Swap', 'Restart', and 'Delete' buttons. Below the header, under the 'Essentials' section, the 'Resource group' field is set to 'myResourceGroup', which is highlighted with a red box. Other details shown include Status: Running, Location: Central US, Subscription: {your subscription name}, and Subscription ID: {your subscription id}. There's also a 'Tags (Edit)' link and a note to 'Click here to add tags'.

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

The screenshot shows the Azure portal interface for a Resource group named 'myResourceGroup'. The sidebar includes links for Overview, Activity log, Access control (IAM), Tags, Resource visualizer, and Events. The main content area has a header with 'Create', 'Edit columns', and a 'Delete resource group' button, which is highlighted with a red box. Under the 'Essentials' section, it shows Subscription (Move) : {your subscription name}, Subscription ID : {your subscription id}, and Tags (Edit) : Click here to add tags. At the bottom, there are tabs for 'Resources' (which is selected) and 'Recommendations (4)'.

Manage the MySQL flexible server, username, or password

- The MySQL Flexible Server is created behind a private [Virtual Network](#) and can't be accessed directly. To access or manage the database, use [phpMyAdmin](#) that's deployed with the WordPress site. You can access [phpMyAdmin](#) by following these steps:
 - Navigate to the URL: <https://<sitename>.azurewebsites.net/phpmyadmin>
 - Login with the flexible server's username and password

- Database username and password of the MySQL Flexible Server are generated automatically. To retrieve these values after the deployment go to Application Settings section of the Configuration page in Azure App Service. The WordPress configuration is modified to use these [Application Settings](#) to connect to the MySQL database.
- To change the MySQL database password, see [Reset admin password](#). Whenever the MySQL database credentials are changed, the [Application Settings](#) need to be updated. The [Application Settings for MySQL database](#) begin with the `DATABASE_` prefix. For more information on updating MySQL passwords, see [WordPress on App Service ↗](#).

Change WordPress admin password

The [Application Settings](#) for WordPress admin credentials are only for deployment purposes. Modifying these values has no effect on the WordPress installation. To change the WordPress admin password, see [resetting your password ↗](#). The [Application Settings for WordPress admin credentials](#) begin with the `WORDPRESS_ADMIN_` prefix. For more information on updating the WordPress admin password, see [Changing WordPress Admin Credentials ↗](#).

Migrate to App Service on Linux

There's a couple approaches when migrating your WordPress app to App Service on Linux. You could use a WP plugin or migrate manually using FTP and a MySQL client. Additional documentation, including [Migrating to App Service ↗](#), can be found at [WordPress - App Service on Linux ↗](#).

Next steps

Congratulations, you've successfully completed this quickstart!

[Secure with custom domain and certificate](#)

[Tutorial: PHP app with MySQL](#)

[Configure PHP app](#)

Migrate WordPress on App Service on Linux

Article • 03/15/2023

This article describes two ways to migrate WordPress from App Service on Windows or external hosting providers to App Service on Linux.

Note

Migrate the content to a test instance, validate all scenarios, and if everything works as expected, swap this instance to the production slot.

You can migrate your site to WordPress on Azure App Service in two ways:

1. WordPress plugin: All-In-One WP Migration
2. Manual process of migration

Migrate WordPress with All-In-One WP Migration plugin

This plugin is popular for migrating sites with ease. This approach is recommended for sites less than 256MB. If it's more, you can either [purchase the premium version](#) of the plugin or [migrate manually](#) using the steps outlined in [manual migration process](#).

By default, the file upload size for WordPress on Linux App Services is limited to 50MB, and it can be increased up to 256MB (Maximum Limit). To change the file upload limit, add the following [Application Settings](#) in the App Service and save it.

Application Setting Name	Default Value	New Value
UPLOAD_MAX_FILESIZE	50M	256M
POST_MAX_SIZE	128M	256M

Important

Install All-In-One Migration plugin on both source and target sites.

Export the data at source site

1. Launch WordPress Admin page.
2. Open All-In-One WP Migration plugin.
3. Click on 'Export' option and specify the export type as file.
4. Download the bundle.

Import the data at destination site

1. Launch WordPress Admin page
2. Open All-In-One WP Migration plugin
3. Click on import option on the destination site, and upload the file downloaded in previous section
4. Empty the caches in W3TC plugin (or any other caches) and validate the content of the site.
 - Click on the **Performance** option given in the left sidebar of the admin panel to open the W3TC plugin.
 - Then click on the **Dashboard** option shown below it.
 - On the dashboard, you will see a button with the label **Empty All Caches**.

Manual migration process

The prerequisite is that the WordPress on Linux Azure App Service must have been created with an appropriate hosting plan from here: [WordPress on Linux App Service ↗](#).

Manually export the data at source site

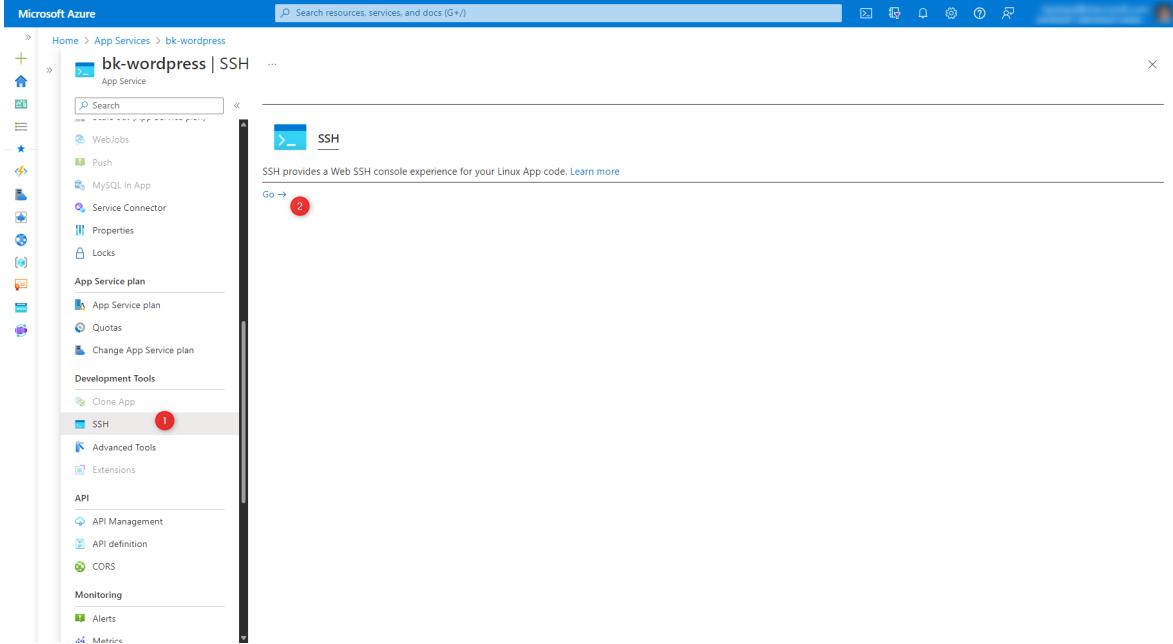
Note

Depending on the size of your content and your internet connection, this operation could take several minutes.

1. Download the **wp-content** folder from the source site. You can use popular FTP tools like [FileZilla ↗](#) to connect to the web server and download the content.
2. Export the contents of the source database into an SQL file. You can perform this task either using MySQL client tools like HeidiSQL, [MySQL workbench ↗](#), [PhpMyAdmin ↗](#) or through command line interface. For more information on exporting the database, refer to the following [documentation ↗](#).

Manually import the data at destination site

1. Create a new Wordpress app using our [WordPress on Linux App Service template](#)
2. Open an SSH session using [WebSSH](#) from the Azure portal.



3. Delete the existing content of `/home/site/wwwroot/wp-content` folder using the following command.

```
Bash  
rm -rf /home/site/wwwroot/wp-content/*
```

4. Upload the new contents of `wp-content` folder using the File Manager. Click on the label that says '**Drag a File/Folder here to upload, or click to select one**'.
5. You can either [use an existing MySQL database](#) or migrate the content to a new Azure MySQL Flexible Server created by App Service on Linux.

ⓘ Note

Azure Database for MySQL - Single Server is on the road to retirement by 16 September 2024. If your existing MySQL database is hosted on Azure Database for MySQL - Single Server, consider migrating to Azure Database for MySQL - Flexible Server using the following steps, or using [Azure Database Migration Service \(DMS\)](#).

6. If you migrate the database, import the SQL file downloaded from the source database into the database of your newly created WordPress site. You can do it via the PhpMyAdmin dashboard available at `<sitename>.azurewebsites.net/phpmyadmin`. If you're unable to one single large SQL file, separate the files into parts and try uploading again. Steps to import the database through phpmyadmin are described [here](#).

7. Launch the Azure Portal and navigate to your **App Service -> Configuration** blade. Update the database name in the **Application Settings** of App Service and save it. This will restart your App and the new changes will get reflected. [Learn more: WordPress Application Settings](#)

Application Setting Name	Update Required?
DATABASE_NAME	Yes, replace with the source (exported) database name
DATABASE_HOST	Not Required
DATABASE_USERNAME	Not Required
DATABASE_PASSWORD	Not Required

The screenshot shows the Azure portal's Application settings blade for an app service. The left sidebar has a 'Configuration' section selected. The main area shows a table of application settings:

Name	Value	Source	Deployment slot
DATABASE_HOST	(Hidden value. Click to show value)	App Service Config	
DATABASE_NAME	(Hidden value. Click to show value)	App Service Config	
DATABASE_PASSWORD	(Hidden value. Click to show value)	App Service Config	
DATABASE_USERNAME	(Hidden value. Click to show value)	App Service Config	
DOCKER_REGISTRY_SERVER_URL	(Hidden value. Click to show value)	App Service Config	
WEBSITES_CONTAINER_START_TIME_LIMIT	(Hidden value. Click to show value)	App Service Config	

Post migration actions

Install recommended plugins

It's an optional step, after the site migration it is recommended to validate that you have the default recommended/equivalent plugins activated and configured accurate as before. If you're prohibited from not configuring them as per your organization governing policies, then you can uninstall the plugins.

- The W3TC plugin should be activated and configured properly to use the local Redis cache server and Azure CDN/Blob Storage (if it was configured to use them originally). For more information on how to configure these, refer to the following documentations:
 - [Local Redis Cache ↗](#)
 - [Azure CDN ↗](#)
 - [Azure Blob Storage ↗](#)
- WP Smush plugin is activated and configured properly for image optimization. See [Image Compression ↗](#) for more information on configuration.

Recommended WordPress settings

The following WordPress settings are recommended. However, when the users migrate their custom sites, is it up to them to decide whether to use these settings or not.

1. Open the WordPress Admin dashboard.
2. Set the permalink structure to 'day and name', as it performs better compared to the plain permalinks that use the format ?p=123.
3. Under the comment settings, enable the option to break comments into pages.
4. Show excerpts instead of the full post in the feed.

Search and replace (paths and domains)

One common issue that users face during migration is that some of the contents of their old site use absolute urls/paths instead of relative ones. To resolve this issue, you can use plugins like [Search and Replace ↗](#) to update the database records.

Configuring custom domain

To configure your site with a custom domain follow the steps described here: Tutorial: [Map existing custom DNS name](#)

Migrating custom domain

When you migrate a live site and its DNS domain name to App Service, that DNS name is already serving live traffic. You can avoid DNS resolution downtime by binding the active DNS name to your app as described in [Migrate an active DNS name](#).

Updating SSL certificates

If your site is configured with SSL certs, then follow [Add and manage TLS/SSL certificates](#) to configure SSL.

Next steps: [At-scale assessment of .NET web apps](#)

Environment variables and app settings in Azure App Service

Article • 10/16/2024

ⓘ Note

Starting June 1, 2024, all newly created App Service apps will have the option to generate a unique default hostname using the naming convention `<app-name>-<random-hash>. <region>.azurewebsites.net`. Existing app names will remain unchanged.

Example: `myapp-ds27dh7271aah175.westus-01.azurewebsites.net`

For further details, refer to [Unique Default Hostname for App Service Resource](#).

In [Azure App Service](#), certain settings are available to the deployment or runtime environment as environment variables. Some of these settings can be customized when you set them manually as [app settings](#). This reference shows the variables you can use or customize.

App environment

The following environment variables are related to the app environment in general.

[Expand table](#)

Setting name	Description	Example
<code>WEBSITE_SITE_NAME</code>	Read-only. App name.	
<code>WEBSITE_RESOURCE_GROUP</code>	Read-only. Azure resource group name that contains the app resource.	
<code>WEBSITE_OWNER_NAME</code>	Read-only. Contains the Azure subscription ID that owns the app, the resource group, and the webspace.	
<code>REGION_NAME</code>	Read-only. Region name of the app.	
<code>WEBSITE_PLATFORM_VERSION</code>	Read-only. App Service platform version.	
<code>HOME</code>	Read-only. Path to the home directory (for example, <code>D:\home</code> for Windows).	
<code>SERVER_PORT</code>	Read-only. The port the app should listen to.	
<code>WEBSITE_WARMUP_PATH</code>	A relative path to ping to warm up the app, beginning with a slash. The default is <code>/</code> , which pings the root path. The specific path can be pinged by an unauthenticated client, such as Azure Traffic Manager, even if App Service authentication is set to reject unauthenticated clients. (NOTE: This app setting doesn't change the path used by AlwaysOn.)	
<code>WEBSITE_COMPUTE_MODE</code>	Read-only. Specifies whether app runs on dedicated (<code>Dedicated</code>) or shared (<code>Shared</code>) VM/s.	

Setting name	Description	Example
<code>WEBSITE_SKU</code>	Read-only. SKU of the app. Possible values are <code>Free</code> , <code>Shared</code> , <code>Basic</code> , and <code>Standard</code> .	
<code>SITE_BITNESS</code>	Read-only. Shows whether the app is 32-bit (<code>x86</code>) or 64-bit (<code>AMD64</code>).	
<code>WEBSITE_HOSTNAME</code>	Read-only. Primary hostname for the app. Custom hostnames aren't accounted for here.	
<code>WEBSITE_VOLUME_TYPE</code>	Read-only. Shows the storage volume type currently in use.	
<code>WEBSITE_NPM_DEFAULT_VERSION</code>	Default npm version the app is using.	
<code>WEBSOCKET_CONCURRENT_REQUEST_LIMIT</code>	Read-only. Limit for websocket's concurrent requests. For <code>Standard</code> tier and above, the value is <code>-1</code> , but there's still a per VM limit based on your VM size (see Cross VM Numerical Limits).	
<code>WEBSITE_PRIVATE_EXTENSIONS</code>	Set to <code>0</code> to disable the use of private site extensions.	
<code>WEBSITE_TIME_ZONE</code>	By default, the time zone for the app is always UTC. You can change it to any of the valid values that are listed in Default Time Zones . If the specified value isn't recognized, UTC is used.	<code>Atlantic Standard Time</code>
<code>WEBSITE_ADD_SITENAME_BINDINGS_IN_APPHOST_CONFIG</code>	After slot swaps, the app may experience unexpected restarts. This is because after a swap, the hostname binding configuration goes out of sync, which by itself doesn't cause restarts. However, certain underlying storage events (such as storage volume failovers) may detect these discrepancies and force all worker processes to restart. To minimize these types of restarts, set the app setting value to <code>1</code> on all slots (default is <code>0</code>). However, don't set this value if you're running a Windows Communication Foundation (WCF) application. For more information, see Troubleshoot swaps	
<code>WEBSITE_PROACTIVE_AUTOHEAL_ENABLED</code>	By default, a VM instance is proactively "autohealed" when it's using more than 90% of allocated memory for more than 30 seconds, or when 80% of the total requests in the last two minutes take longer than 200 seconds. If a VM instance has triggered one of these rules, the recovery process is an overlapping restart of the instance. Set to <code>false</code> to disable this recovery behavior. The default is <code>true</code> . For more information, see Proactive Auto Heal .	
<code>WEBSITE_PROACTIVE_CRASHMONITORING_ENABLED</code>	Whenever the <code>w3wp.exe</code> process on a VM instance of your app crashes due to an unhandled exception for more than three times in 24 hours, a debugger process is attached to the main worker process on that instance, and collects a memory dump when the worker process crashes again. This memory dump is then analyzed and the call stack of the thread that caused the	

Setting name	Description	Example
	crash is logged in your App Service's logs. Set to <code>false</code> to disable this automatic monitoring behavior. The default is <code>true</code> . For more information, see Proactive Crash Monitoring .	
<code>WEBSITE_DAAS_STORAGE_SASURI</code>	During crash monitoring (proactive or manual), the memory dumps are deleted by default. To save the memory dumps to a storage blob container, specify the SAS URI.	
<code>WEBSITE_CRASHMONITORING_ENABLED</code>	Set to <code>true</code> to enable crash monitoring manually. You must also set <code>WEBSITE_DAAS_STORAGE_SASURI</code> and <code>WEBSITE_CRASHMONITORING_SETTINGS</code> . The default is <code>false</code> . This setting has no effect if remote debugging is enabled. Also, if this setting is set to <code>true</code> , proactive crash monitoring is disabled.	
<code>WEBSITE_CRASHMONITORING_SETTINGS</code>	A JSON with the following format: <pre>{"StartTimeUtc": "2020-02-10T08:21", "MaxHours": "<elapsed-hours-from-StartTimeUtc>", "MaxDumpCount": "<max-number-of-crash-dumps>"}.</pre> Required to configure crash monitoring if <code>WEBSITE_CRASHMONITORING_ENABLED</code> is specified. To only log the call stack without saving the crash dump in the storage account, add <code>,"UseStorageAccount": "false"</code> in the JSON.	
<code>REMOTEDEBUGGINGVERSION</code>	Remote debugging version.	
<code>WEBSITE_CONTENTAZUREFILECONNECTIONSTRING</code>	By default, App Service creates a shared storage for you at app creation. To use a custom storage account instead, set to the connection string of your storage account. For functions, see App settings reference for Functions .	<code>DefaultEndpointsProtocol=https;AccountName=<name>;AccountKey=<key></code>
<code>WEBSITE_CONTENTSHARE</code>	When you use specify a custom storage account with <code>WEBSITE_CONTENTAZUREFILECONNECTIONSTRING</code> , App Service creates a file share in that storage account for your app. To use a custom name, set this variable to the name you want. If a file share with the specified name doesn't exist, App Service creates it for you.	<code>myapp123</code>
<code>WEBSITE_SCM_ALWAYS_ON_ENABLED</code>	Read-only. Shows whether Always On is enabled (1) or not (0).	
<code>WEBSITE_SCM_SEPARATE_STATUS</code>	Read-only. Shows whether the Kudu app is running in a separate process (1) or not (0).	
<code>WEBSITE_DNS_ATTEMPTS</code>	Number of times to try name resolve.	
<code>WEBSITE_DNS_TIMEOUT</code>	Number of seconds to wait for name resolve	

Variable prefixes

The following table shows environment variable prefixes that App Service uses for various purposes.

[Expand table](#)

Setting name	Description
APPSETTING_	Signifies that a variable is set by the customer as an app setting in the app configuration. It's injected into a .NET app as an app setting.
MAINSITE_	Signifies a variable is specific to the app itself.
SCMSITE_	Signifies a variable is specific to the Kudu app.
SQLCONNSTR_	Signifies a SQL Server connection string in the app configuration. It's injected into a .NET app as a connection string.
SQLAZURECONNSTR_	Signifies an Azure SQL Database connection string in the app configuration. It's injected into a .NET app as a connection string.
POSTGRESQLCONNSTR_	Signifies a PostgreSQL connection string in the app configuration. It's injected into a .NET app as a connection string.
CUSTOMCONNSTR_	Signifies a custom connection string in the app configuration. It's injected into a .NET app as a connection string.
MYSQLCONNSTR_	Signifies a MySQL Database connection string in the app configuration. It's injected into a .NET app as a connection string.
AZUREFILESTORAGE_	A connection string to a custom share for a custom container in Azure Files.
AZUREBLOBSTORAGE_	A connection string to a custom storage account for a custom container in Azure Blob Storage.
NOTIFICATIONHUBCONNSTR_	Signifies a connection string to a notification hub in Azure Notification Hubs.
SERVICEBUSCONNSTR_	Signifies a connection string to an instance of Azure Service Bus.
EVENTHUBCONNSTR_	Signifies a connection string to an event hub in Azure Event Hubs.
DOCDBCONNSTR_	Signifies a connection string to a database in Azure Cosmos DB.
REDISCACHECONNSTR_	Signifies a connection string to a cache in Azure Cache for Redis.
FILESHARESTORAGE_	Signifies a connection string to a custom file share.

Deployment

The following environment variables are related to app deployment. For variables related to App Service build automation, see [Build automation](#).

[Expand table](#)

Setting name	Description
DEPLOYMENT_BRANCH	For local Git or cloud Git deployment (such as GitHub), set to the branch in Azure you want to deploy to. By default, it's <code>master</code> .
WEBSITE_RUN_FROM_PACKAGE	Set to <code>1</code> to run the app from a local ZIP package, or set to the URL of an external URL to run the app from a remote ZIP package. For more information, see Run your app in Azure App Service directly from a ZIP package .
WEBSITE_USE_ZIP	Deprecated. Use <code>WEBSITE_RUN_FROM_PACKAGE</code> .
WEBSITE_RUN_FROM_ZIP	Deprecated. Use <code>WEBSITE_RUN_FROM_PACKAGE</code> .

Setting name	Description
SCM_MAX_ZIP_PACKAGE_COUNT	Your app keeps 5 of the most recent zip files deployed using zip deploy . You can keep more or less by setting the app setting to a different number.
WEBSITE_WEBDEPLOY_USE_SCM	Set to <code>false</code> for WebDeploy to stop using the Kudu deployment engine. The default is <code>true</code> . To deploy to Linux apps using Visual Studio (WebDeploy/MSDeploy), set it to <code>false</code> .
MSDEPLOY_RENAME_LOCKED_FILES	Set to <code>1</code> to attempt to rename DLLs if they can't be copied during a WebDeploy deployment. This setting isn't applicable if <code>WEBSITE_WEBDEPLOY_USE_SCM</code> is set to <code>false</code> .
WEBSITE_DISABLE_SCM_SEPARATION	By default, the main app and the Kudu app run in different sandboxes. When you stop the app, the Kudu app is still running, and you can continue to use Git deploy and MSDeploy. Each app has its own local files. Turning off this separation (setting to <code>true</code>) is a legacy mode that's no longer fully supported.
WEBSITE_ENABLE_SYNC_UPDATE_SITE	Set to <code>1</code> ensure that REST API calls to update <code>site</code> and <code>siteconfig</code> are completely applied to all instances before returning. The default is <code>1</code> if deploying with an ARM template, to avoid race conditions with subsequent ARM calls.
WEBSITE_START_SCM_ON_SITE_CREATION	In an ARM template deployment, set to <code>1</code> in the ARM template to pre-start the Kudu app as part of app creation.
WEBSITE_START_SCM_WITH_PRELOAD	For Linux apps, set to <code>true</code> to force preloading the Kudu app when Always On is enabled by pinging its URL. The default is <code>false</code> . For Windows apps, the Kudu app is always preloaded.

Build automation

Kudu (Windows)

Kudu build configuration applies to native Windows apps and is used to control the behavior of Git-based (or ZIP-based) deployments.

[Expand table](#)

Setting name	Description	Example
SCM_BUILD_ARGS	Add things at the end of the msbuild command line, such that it overrides any previous parts of the default command line.	To do a clean build: <code>-t:Clean;Compile</code>
SCM_SCRIPT_GENERATOR_ARGS	Kudu uses the <code>azure site deploymentscript</code> command described here to generate a deployment script. It automatically detects the language framework type and determines the parameters to pass to the command. This setting overrides the automatically generated parameters.	To treat your repository as plain content files: <code>--basic -p <folder-to-deploy></code>
SCM_TRACE_LEVEL	Build trace level. The default is <code>1</code> . Set to higher values, up to <code>4</code> , for more tracing.	<code>4</code>
SCM_COMMAND_IDLE_TIMEOUT	Time out in seconds for each command that the build process launches to wait before without producing any output. After that, the command is considered idle and killed. The default is <code>60</code> (one minute). In Azure, there's also a general idle request timeout that disconnects clients after 230 seconds. However, the command will still continue running server-side after that.	
SCM_LOGSTREAM_TIMEOUT	Time-out of inactivity in seconds before stopping log streaming. The default is <code>1800</code> (30 minutes).	
SCM_SITEEXTENSIONS_FEED_URL	URL of the site extensions gallery. The default is https://www.nuget.org/api/v2/ . The URL of the old feed is http://www.siteextensions.net/api/v2/ .	
SCM_USE_LIBGIT2SHARP_REPOSITORY	Set to <code>0</code> to use git.exe instead of libgit2sharp for git operations.	

Setting name	Description	Example
WEBSITE_LOAD_USER_PROFILE	In case of the error <code>The specified user does not have a valid profile.</code> during ASP.NET build automation (such as during Git deployment), set this variable to <code>1</code> to load a full user profile in the build environment. This setting is only applicable when <code>WEBSITE_COMPUTE_MODE</code> is <code>Dedicated</code> .	
WEBSITE_SCM_IDLE_TIMEOUT_IN_MINUTES	Time out in minutes for the SCM (Kudu) site. The default is <code>20</code> .	
SCM_DO_BUILD_DURING_DEPLOYMENT	With ZIP deploy , the deployment engine assumes that a ZIP file is ready to run as-is and doesn't run any build automation. To enable the same build automation as in Git deploy , set to <code>true</code> .	

Language-specific settings

This section shows the configurable runtime settings for each supported language framework. Additional settings are available during [build automation](#) at deployment time.

.NET

[\[+\] Expand table](#)

Setting name	Description
PORT	Read-only. For Linux apps, port that the .NET runtime listens to in the container.
WEBSITE_ROLE_INSTANCE_ID	Read-only. ID of the current instance.
HOME	Read-only. Directory that points to shared storage (<code>/home</code>).
DUMP_DIR	Read-only. Directory for the crash dumps (<code>/home/logs/dumps</code>).
APP_SVC_RUN_FROM_COPY	Linux apps only. By default, the app is run from <code>/home/site/wwwroot</code> , a shared directory for all scaled-out instances. Set this variable to <code>true</code> to copy the app to a local directory in your container and run it from there. When using this option, be sure not to hard-code any reference to <code>/home/site/wwwroot</code> . Instead, use a path relative to <code>/home/site/wwwroot</code> .
MACHINEKEY_Decryption	For Windows native apps or Windows containerized apps, this variable is injected into app environment or container to enable ASP.NET cryptographic routines (see machineKey Element). To override the default <code>decryption</code> value, configure it as an App Service app setting, or set it directly in the <code>machineKey</code> element of the <code>Web.config</code> file.
MACHINEKEY_DecryptionKey	For Windows native apps or Windows containerized apps, this variable is injected into the app environment or container to enable ASP.NET cryptographic routines (see machineKey Element). To override the automatically generated <code>decryptionKey</code> value, configure it as an App Service app setting, or set it directly in the <code>machineKey</code> element of the <code>Web.config</code> file.
MACHINEKEY_Validation	For Windows native apps or Windows containerized apps, this variable is injected into the app environment or container to enable ASP.NET cryptographic routines (see machineKey Element). To override the default <code>validation</code> value, configure it as an App Service app setting, or set it directly in the <code>machineKey</code> element of the <code>Web.config</code> file.
MACHINEKEY_ValidationKey	For Windows native apps or Windows containerized apps, this variable is injected into the app environment or container to enable ASP.NET cryptographic routines (see machineKey Element). To override the automatically generated <code>validationKey</code> value, configure it as an App Service app setting, or set it directly in the <code>machineKey</code> element of the <code>Web.config</code> file.

WordPress

[Expand table](#)

Application Setting	Scope	Value	Max	Description
<code>WEBSITES_ENABLE_APP_SERVICE_STORAGE</code>	Web App	true	-	When set to TRUE, file contents are preserved during restarts.
<code>WP_MEMORY_LIMIT</code>	WordPress	128M	512M	Frontend or general wordpress PHP memory limit (per script). Can't be more than <code>PHP_MEMORY_LIMIT</code>
<code>WP_MAX_MEMORY_LIMIT</code>	WordPress	256M	512M	Admin dashboard PHP memory limit (per script). Generally Admin dashboard/ backend scripts takes lot of memory compared to frontend scripts. Can't be more than <code>PHP_MEMORY_LIMIT</code> .
<code>PHP_MEMORY_LIMIT</code>	PHP	512M	512M	Memory limits for general PHP script. It can only be decreased.
<code>FILE_UPLOADS</code>	PHP	On	-	Can be either On or Off. Note that values are case sensitive. Enables or disables file uploads.
<code>UPLOAD_MAX_FILESIZE</code>	PHP	50M	256M	Max file upload size limit. Can be increased up to 256M.
<code>POST_MAX_SIZE</code>	PHP	128M	256M	Can be increased up to 256M. Generally should be more than <code>UPLOAD_MAX_FILESIZE</code> .
<code>MAX_EXECUTION_TIME</code>	PHP	120	120	Can only be decreased. Please break down the scripts if it is taking more than 120 seconds. Added to avoid bad scripts from slowing the system.
<code>MAX_INPUT_TIME</code>	PHP	120	120	Max time limit for parsing the input requests. Can only be decreased.
<code>MAX_INPUT_VARS</code>	PHP	10000	10000	-
<code>DATABASE_HOST</code>	Database	-	-	Database host used to connect to WordPress.
<code>DATABASE_NAME</code>	Database	-	-	Database name used to connect to WordPress.
<code>DATABASE_USERNAME</code>	Database	-	-	Database username used to connect to WordPress.
<code>DATABASE_PASSWORD</code>	Database	-	-	Database password used to connect to the MySQL database. To change the MySQL database password, see update admin password . Whenever the MySQL database password is changed, the Application Settings also need to be updated.
<code>WORDPRESS_ADMIN_EMAIL</code>	Deployment only	-	-	WordPress admin email.
<code>WORDPRESS_ADMIN_PASSWORD</code>	Deployment only	-	-	WordPress admin password. This is only for deployment purposes. Modifying this value has no effect on the WordPress installation. To change the WordPress admin password, see resetting your password .
<code>WORDPRESS_ADMIN_USER</code>	Deployment only	-	-	WordPress admin username
<code>WORDPRESS_ADMIN_LOCALE_CODE</code>	Deployment only	-	-	Database username used to connect to WordPress.

Domain and DNS

[Expand table](#)

Setting name	Description	Example
<code>WEBSITE_DNS_SERVER</code>	IP address of primary DNS server for outgoing connections (such as to a back-end service). The default DNS server for App Service is Azure DNS, whose IP address is <code>168.63.129.16</code> . If your app uses VNet integration or is in an App Service environment , it inherits the DNS server configuration from the VNet by default.	<code>10.0.0.1</code>
<code>WEBSITE_DNS_ALT_SERVER</code>	IP address of fallback DNS server for outgoing connections. See <code>WEBSITE_DNS_SERVER</code> .	
<code>WEBSITE_ENABLE_DNS_CACHE</code>	Allows successful DNS resolutions to be cached. By Default expired DNS cache entries will be flushed & in addition to the existing cache to be flushed every 4.5 mins.	

TLS/SSL

For more information, see [Use a TLS/SSL certificate in your code in Azure App Service](#).

[Expand table](#)

Setting name	Description
<code>WEBSITE_LOAD_CERTIFICATES</code>	Comma-separate thumbprint values to the certificate you want to load in your code, or <code>*</code> to allow all certificates to be loaded in code. Only certificates added to your app can be loaded.
<code>WEBSITE_PRIVATE_CERTS_PATH</code>	Read-only. Path in a Windows container to the loaded private certificates.
<code>WEBSITE_PUBLIC_CERTS_PATH</code>	Read-only. Path in a Windows container to the loaded public certificates.
<code>WEBSITE_INTERMEDIATE_CERTS_PATH</code>	Read-only. Path in a Windows container to the loaded intermediate certificates.
<code>WEBSITE_ROOT_CERTS_PATH</code>	Read-only. Path in a Windows container to the loaded root certificates.

Deployment slots

For more information on deployment slots, see [Set up staging environments in Azure App Service](#).

[Expand table](#)

Setting name	Description	Example
<code>WEBSITE_OVERRIDE_STICKY_EXTENSION_VERSIONS</code>	By default, the versions for site extensions are specific to each slot. This prevents unanticipated application behavior due to changing extension versions after a swap. If you want the extension versions to swap as well, set to <code>0</code> on <i>all slots</i> .	
<code>WEBSITE_OVERRIDE_PRESERVE_DEFAULT_STICKY_SLOT_SETTINGS</code>	Designates certain settings as sticky or not swappable by default . Default is <code>true</code> . Set this setting to <code>false</code> or <code>0</code> for <i>all deployment slots</i> to make them swappable instead. There's no fine-grain control for specific setting types.	
<code>WEBSITE_SWAP_WARMUP_PING_PATH</code>	Path to ping to warm up the target slot in a swap, beginning with a slash. The default is <code>/</code> , which pings the root path over HTTP.	<code>/statuscheck</code>
<code>WEBSITE_SWAP_WARMUP_PING_STATUSES</code>	Valid HTTP response codes for the warm-up operation during a swap. If the returned status code isn't in the list, the warmup and swap operations are stopped. By default, all response codes are valid.	<code>200,202</code>
<code>WEBSITE_SLOT_NUMBER_OF_TIMEOUTS_BEFORE_RESTART</code>	During a slot swap, maximum number of timeouts after which we force restart the site on a specific VM instance. The default is <code>3</code> .	

Setting name	Description	Example
<code>WEBSITE_SLOT_MAX_NUMBER_OF_TIMEOUTS</code>	During a slot swap, maximum number of timeout requests for a single URL to make before giving up. The default is <code>5</code> .	
<code>WEBSITE_SKIP_ALL_BINDINGS_IN_APPHOST_CONFIG</code>	Set to <code>true</code> or <code>1</code> to skip all bindings in <code>applicationHost.config</code> . The default is <code>false</code> . If your app triggers a restart because <code>applicationHost.config</code> is updated with the swapped hostnames of the slots, set this variable to <code>true</code> to avoid a restart of this kind. If you're running a Windows Communication Foundation (WCF) app, don't set this variable.	

Custom containers

For more information on custom containers, see [Run a custom container in Azure](#).

[Expand table](#)

Setting name	Description	Example
<code>WEBSITES_ENABLE_APP_SERVICE_STORAGE</code>	For Linux custom containers: set to <code>true</code> to enable the <code>/home</code> directory to be shared across scaled instances. The default is <code>false</code> for Linux custom containers. For Windows containers: set to <code>true</code> to enable the <code>c:\home</code> directory to be shared across scaled instances. The default is <code>true</code> for Windows containers.	
<code>WEBSITES_CONTAINER_START_TIME_LIMIT</code>	Amount of time in seconds to wait for the container to complete start-up before restarting the container. Default is <code>230</code> . You can increase it up to the maximum of <code>1800</code> .	
<code>WEBSITES_CONTAINER_STOP_TIME_LIMIT</code>	Amount of time in seconds to wait for the container to terminate gracefully. Default is <code>5</code> . You can increase to a maximum of <code>120</code> .	
<code>DOCKER_REGISTRY_SERVER_URL</code>	URL of the registry server, when running a custom container in App Service. For security, this variable isn't passed on to the container.	<code>https://<server-name>.azuredcr.io</code>
<code>DOCKER_REGISTRY_SERVER_USERNAME</code>	Username to authenticate with the registry server at <code>DOCKER_REGISTRY_SERVER_URL</code> . For security, this variable isn't passed on to the container.	
<code>DOCKER_REGISTRY_SERVER_PASSWORD</code>	Password to authenticate with the registry server at <code>DOCKER_REGISTRY_SERVER_URL</code> . For security, this variable isn't passed on to the container.	
<code>DOCKER_ENABLE_CI</code>	Set to <code>true</code> to enable the continuous deployment for custom containers. The default is <code>false</code> for custom containers.	
<code>WEBSITE_PULL_IMAGE_OVER_VNET</code>	Connect and pull from a registry inside a Virtual Network or on-premises. Your app will need to be connected to a Virtual Network using VNet integration feature. This setting is also needed for Azure Container Registry with Private Endpoint.	
<code>WEBSITES_WEB_CONTAINER_NAME</code>	In a Docker Compose app, only one of the containers can be internet accessible. Set to the name of the container defined in the configuration file to override the default container selection. By default, the internet accessible container is the first container to define port 80 or 8080, or, when no such container is found, the first container defined in the configuration file.	
<code>WEBSITES_PORT</code>	For a custom container, the custom port number on the container for App Service to route requests to. By default, App Service attempts automatic	

Setting name	Description	Example
	port detection of ports 80 and 8080. This setting isn't injected into the container as an environment variable.	
WEBSITE_CPU_CORES_LIMIT	By default, a Windows container runs with all available cores for your chosen pricing tier. To reduce the number of cores, set to the number of desired cores limit. For more information, see Customize the number of compute cores .	
WEBSITE_MEMORY_LIMIT_MB	By default all Windows Containers deployed in Azure App Service have a memory limit configured depending on the App Service Plan SKU. Set to the desired memory limit in MB. The cumulative total of this setting across apps in the same plan must not exceed the amount allowed by the chosen pricing tier. For more information, see Customize container memory .	

Scaling

[Expand table](#)

Setting name	Description
WEBSITE_INSTANCE_ID	Read-only. Unique ID of the current VM instance, when the app is scaled out to multiple instances.
WEBSITE_IIS_SITE_NAME	Deprecated. Use <code>WEBSITE_INSTANCE_ID</code> .
WEBSITE_DISABLE_OVERLAPPED_RECYCLING	Overlapped recycling makes it so that before the current VM instance of an app is shut down, a new VM instance starts. In some cases, it can cause file locking issues. You can try turning it off by setting to <code>1</code> .
WEBSITE_DISABLE_CROSS_STAMP_SCALE	By default, apps are allowed to scale across stamps if they use Azure Files or a Docker container. Set to <code>1</code> or <code>true</code> to disable cross-stamp scaling within the app's region. The default is <code>0</code> . Custom Docker containers that set <code>WEBSITES_ENABLE_APP_SERVICE_STORAGE</code> to <code>true</code> or <code>1</code> can't scale cross-stamps because their content isn't completely encapsulated in the Docker container.

Logging

[Expand table](#)

Setting name	Description	Example
WEBSITE_HTTPLOGGING_ENABLED	Read-only. Shows whether the web server logging for Windows native apps is enabled (<code>1</code>) or not (<code>0</code>).	
WEBSITE_HTTPLOGGING_RETENTION_DAYS	Retention period in days of web server logs, if web server logs are enabled for a Windows native or Linux app.	10
WEBSITE_HTTPLOGGING_CONTAINER_URL	SAS URL of the blob storage container to store web server logs for Windows native apps, if web server logs are enabled. If not set, web server logs are stored in the app's file system (default shared storage).	
DIAGNOSTICS_AZUREBLOBRETENTIONINDAYS	Retention period in days of application logs for Windows native apps, if application logs are enabled.	10
DIAGNOSTICS_AZUREBLOBCONTAINERSASURL	SAS URL of the blob storage container to store application logs for Windows native apps, if application logs are enabled.	
APPSERVICEAPPLOGS_TRACE_LEVEL	Minimum log level to ship to Log Analytics for the AppServiceAppLogs log type.	

Setting name	Description	Example
DIAGNOSTICS_LASTRESORTFILE	The filename to create, or a relative path to the log directory, for logging internal errors for troubleshooting the listener. The default is <code>logging-errors.txt</code> .	
DIAGNOSTICS_LOGGINGSETTINGSFILE	Path to the log settings file, relative to <code>D:\home</code> or <code>/home</code> . The default is <code>site\diagnostics\settings.json</code> .	
DIAGNOSTICS_TEXTTRACELOGDIRECTORY	The log folder, relative to the app root (<code>D:\home\site\wwwroot</code> or <code>/home/site/wwwroot</code>).	<code>..\..\LogFiles\Application</code>
DIAGNOSTICS_TEXTTRACEMAXLOGFILESIZEBYTES	Maximum size of the log file in bytes. The default is <code>131072</code> (128 KB).	
DIAGNOSTICS_TEXTTRACEMAXLOGFOLDERSIZEBYTES	Maximum size of the log folder in bytes. The default is <code>1048576</code> (1 MB).	
DIAGNOSTICS_TEXTTRACEMAXNUMLOGFILES	Maximum number of log files to keep. The default is <code>20</code> .	
DIAGNOSTICS_TEXTTRACETURNOFFPERIOD	Time out in milliseconds to keep application logging enabled. The default is <code>43200000</code> (12 hours).	
WEBSITE_LOG_BUFFERING	By default, log buffering is enabled. Set to <code>0</code> to disable it.	
WEBSITE_ENABLE_PERF_MODE	For native Windows apps, set to <code>TRUE</code> to turn off IIS log entries for successful requests returned within 10 seconds. This is a quick way to do performance benchmarking by removing extended logging.	

Performance counters

The following are 'fake' environment variables that don't exist if you enumerate them, but return their value if you look them up individually. The value is dynamic and can change on every lookup.

[Expand table](#)

Setting name	Description
WEBSITE_COUNTERS_ASPNET	A JSON object containing the ASP.NET perf counters.
WEBSITE_COUNTERS_APP	A JSON object containing sandbox counters.
WEBSITE_COUNTERS_CLR	A JSON object containing CLR counters.
WEBSITE_COUNTERS_ALL	A JSON object containing the combination of the other three variables.

Caching

[Expand table](#)

Setting name	Description
WEBSITE_LOCAL_CACHE_OPTION	Whether local cache is enabled. Available options are: <ul style="list-style-type: none"> - <code>Default</code>: Inherit the stamp-level global setting. - <code>Always</code>: Enable for the app. - <code>OnStorageUnavailability</code> - <code>Disabled</code>: Disabled for the app.
WEBSITE_LOCAL_CACHE_READWRITE_OPTION	Read-write options of the local cache. Available options are: <ul style="list-style-type: none"> - <code>ReadOnly</code>: Cache is read-only. - <code>WriteButDiscardChanges</code>: Allow writes to local cache but discard changes made locally.

Setting name	Description
WEBSITE_LOCAL_CACHE_SIZEINMB	Size of the local cache in MB. Default is 1000 (1 GB).
WEBSITE_LOCALCACHE_READY	Read-only flag indicating if the app using local cache.
WEBSITE_DYNAMIC_CACHE	Due to network file shared nature to allow access for multiple instances, the dynamic cache improves performance by caching the recently accessed files locally on an instance. Cache is invalidated when file is modified. The cache location is %SYSTEMDRIVE%\local\DynamicCache (same %SYSTEMDRIVE%\local quota is applied). To enable full content caching, set to 1, which includes both file content and directory/file metadata (timestamps, size, directory content). To conserve local disk use, set to 2 to cache only directory/file metadata (timestamps, size, directory content). To turn off caching, set to 0. For Windows apps and for Linux apps created with the WordPress template , the default is 1. For all other Linux apps, the default is 0.
WEBSITE_READONLY_APP	When using dynamic cache, you can disable write access to the app root (D:\home\site\wwwroot or /home/site/wwwroot) by setting this variable to 1. Except for the App_Data directory, no exclusive locks are allowed, so that deployments don't get blocked by locked files.

Networking

The following environment variables are related to [hybrid connections](#) and [VNET integration](#).

[Expand table](#)

Setting name	Description
WEBSITE_RELAYS	Read-only. Data needed to configure the Hybrid Connection, including endpoints and service bus data.
WEBSITE_REWRITE_TABLE	Read-only. Used at runtime to do the lookups and rewrite connections appropriately.
WEBSITE_VNET_ROUTE_ALL	By default, if you use regional VNet Integration , your app only routes RFC1918 traffic into your VNet. Set to 1 to route all outbound traffic into your VNet and be subject to the same NSGs and UDRs. The setting lets you access non-RFC1918 endpoints through your VNet, secure all outbound traffic leaving your app, and force tunnel all outbound traffic to a network appliance of your own choosing.
WEBSITE_PRIVATE_IP	Read-only. IP address associated with the app when integrated with a VNet . For Regional VNet Integration, the value is an IP from the address range of the delegated subnet, and for Gateway-required VNet Integration, the value is an IP from the address range of the point-to-site address pool configured on the Virtual Network Gateway. This IP is used by the app to connect to the resources through the VNet. Also, it can change within the described address range.
WEBSITE_PRIVATE_PORTS	Read-only. In VNet integration, shows which ports are useable by the app to communicate with other nodes.
WEBSITE_CONTENTOVERVNET	If you are mounting an Azure File Share on the App Service and the Storage account is restricted to a VNET, ensure to enable this setting with a value of 1.

Key vault references

The following environment variables are related to [key vault references](#).

[Expand table](#)

Setting name	Description
WEBSITE_KEYVAULT_REFERENCES	Read-only. Contains information (including statuses) for all Key Vault references that are currently configured in the app.
WEBSITE_SKIP_CONTENTSHARE_VALIDATION	If you set the shared storage connection of your app (using WEBSITE_CONTENTAZUREFILECONNECTIONSTRING) to a Key Vault reference, the app can't resolve the key vault reference at app creation or update if one of the following conditions is true: - The app accesses the key vault with a system-assigned identity. - The app accesses the key vault with a user-assigned identity, and the key vault is locked with a

Setting name	Description
<code>VNet</code>	To avoid errors at create or update time, set this variable to <code>1</code> .
<code>WEBSITE_DELAY_CERT_DELETION</code>	This env var can be set to <code>1</code> by users in order to ensure that a certificate that a worker process is dependent upon isn't deleted until it exits.

CORS

The following environment variables are related to Cross-Origin Resource Sharing (CORS) configuration.

[Expand table](#)

Setting name	Description
<code>WEBSITE_CORS_ALLOWED_ORIGINS</code>	Read-only. Shows the allowed origins for CORS.
<code>WEBSITE_CORS_SUPPORT_CREDENTIALS</code>	Read-only. Shows whether setting the <code>Access-Control-Allow-Credentials</code> header to <code>true</code> is enabled (<code>True</code>) or not (<code>False</code>).

Authentication & Authorization

The following environment variables are related to [App Service authentication](#).

[Expand table](#)

Setting name	Description
<code>WEBSITE_AUTH_DISABLE_IDENTITY_FLOW</code>	When set to <code>true</code> , disables assigning the thread principal identity in ASP.NET-based web applications (including v1 Function Apps). This is designed to allow developers to protect access to their site with auth, but still have it use a separate sign-in mechanism within their app logic. The default is <code>false</code> .
<code>WEBSITE_AUTH_HIDE_DEPRECATED_SID</code>	<code>true</code> or <code>false</code> . The default value is <code>false</code> . This is a setting for the legacy Azure Mobile Apps integration for Azure App Service. Setting this to <code>true</code> resolves an issue where the SID (security ID) generated for authenticated users might change if the user changes their profile information. Changing this value may result in existing Azure Mobile Apps user IDs changing. Most apps don't need to use this setting.
<code>WEBSITE_AUTH_NONCE_DURATION</code>	A <code>timespan</code> value in the form <code>_hours_:_minutes_:_seconds_</code> . The default value is <code>00:05:00</code> , or 5 minutes. This setting controls the lifetime of the cryptographic nonce generated for all browser-driven logins. If a sign-in fails to complete in the specified time, the sign-in flow will be retried automatically. This application setting is intended for use with the V1 (classic) configuration experience. If using the V2 authentication configuration schema, you should instead use the <code>login.nonce.nonceExpirationInterval</code> configuration value.
<code>WEBSITE_AUTH_PRESERVE_URL_FRAGMENT</code>	When set to <code>true</code> and users select on app links that contain URL fragments, the sign-in process will ensure that the URL fragment part of your URL doesn't get lost in the sign-in redirect process. For more information, see Customize sign-in and sign-out in Azure App Service authentication .
<code>WEBSITE_AUTH_USE_LEGACY CLAIMS</code>	To maintain backward compatibility across upgrades, the authentication module uses the legacy claims mapping of short to long names in the <code>/auth/me</code> API, so certain mappings are excluded (e.g. "roles"). To get the more modern version of the claims mappings, set this variable to <code>False</code> . In the "roles" example, it would be mapped to the long claim name " <code>http://schemas.microsoft.com/ws/2008/06/identity/claims/role</code> ".
<code>WEBSITE_AUTH_DISABLE_WWWAUTHENTICATE</code>	<code>true</code> or <code>false</code> . The default value is <code>false</code> . When set to <code>true</code> , removes the WWW-Authenticate HTTP response header from module-generated HTTP 401 responses. This application setting is intended for use with the V1 (classic) configuration experience. If using the V2 authentication configuration schema, you should instead use the <code>identityProviders.azureActiveDirectory.login.disableWwwAuthenticate</code> configuration value.

Setting name	Description
<code>WEBSITE_AUTH_STATE_DIRECTORY</code>	A local file system directory path where tokens are stored when the file-based token store is enabled. The default value is <code>%HOME%\Data\auth</code> . This application setting is intended for use with the V1 (classic) configuration experience. If using the V2 authentication configuration schema, you should instead use the <code>login.tokenStore.FileSystem.directory</code> configuration value.
<code>WEBSITE_AUTH_TOKEN_CONTAINER_SASURL</code>	A fully qualified blob container URL. Instructs the auth module to store and load all encrypted tokens to the specified blob storage container instead of using the default local file system.
<code>WEBSITE_AUTH_TOKEN_REFRESH_HOURS</code>	Any positive decimal number. The default value is <code>72</code> (hours). This setting controls the amount of time after a session token expires that the <code>/auth/refresh</code> API can be used to refresh it. Refresh attempts after this period will fail and end users will be required to sign-in again. This application setting is intended for use with the V1 (classic) configuration experience. If using the V2 authentication configuration schema, you should instead use the <code>login.tokenStore.tokenRefreshExtensionHours</code> configuration value.
<code>WEBSITE_AUTH_TRACE_LEVEL</code>	Controls the verbosity of authentication traces written to Application Logging . Valid values are <code>Off</code> , <code>Error</code> , <code>Warning</code> , <code>Information</code> , and <code>Verbose</code> . The default value is <code>Verbose</code> .
<code>WEBSITE_AUTH_VALIDATE_NONCE</code>	<code>true</code> or <code>false</code> . The default value is <code>true</code> . This value should never be set to <code>false</code> except when temporarily debugging cryptographic nonce validation failures that occur during interactive logins. This application setting is intended for use with the V1 (classic) configuration experience. If using the V2 authentication configuration schema, you should instead use the <code>login.nonce.validateNonce</code> configuration value.
<code>WEBSITE_AUTH_V2_CONFIG_JSON</code>	This environment variable is populated automatically by the Azure App Service platform and is used to configure the integrated authentication module. The value of this environment variable corresponds to the V2 (non-classic) authentication configuration for the current app in Azure Resource Manager. It's not intended to be configured explicitly.
<code>WEBSITE_AUTH_ENABLED</code>	Read-only. Injected into a Windows or Linux app to indicate whether App Service authentication is enabled.
<code>WEBSITE_AUTH_ENCRYPTION_KEY</code>	By default, the automatically generated key is used as the encryption key. To override, set to a desired key. This is recommended if you want to share tokens or sessions across multiple apps. If specified, it supersedes the <code>MACHINEKEY_DecryptionKey</code> setting.
<code>WEBSITE_AUTH_SIGNING_KEY</code>	By default, the automatically generated key is used as the signing key. To override, set to a desired key. This is recommended if you want to share tokens or sessions across multiple apps. If specified, it supersedes the <code>MACHINEKEY_ValidationKey</code> setting.

Managed identity

The following environment variables are related to [managed identities](#).

[Expand table](#)

Setting name	Description
<code>IDENTITY_ENDPOINT</code>	Read-only. The URL to retrieve the token for the app's managed identity .
<code>MSI_ENDPOINT</code>	Deprecated. Use <code>IDENTITY_ENDPOINT</code> .
<code>IDENTITY_HEADER</code>	Read-only. Value that must be added to the <code>x-IDENTITY-HEADER</code> header when making an HTTP GET request to <code>IDENTITY_ENDPOINT</code> . The value is rotated by the platform.
<code>MSI_SECRET</code>	Deprecated. Use <code>IDENTITY_HEADER</code> .

Health check

The following environment variables are related to [health checks](#).

[Expand table](#)

Setting name	Description
WEBSITE_HEALTHCHECK_MAXPINGFAILURES	The maximum number of failed pings before removing the instance. Set to a value between 2 and 100. When you're scaling up or out, App Service pings the Health check path to ensure new instances are ready. For more information, see Health check .
WEBSITE_HEALTHCHECK_MAXUNHEALTHYWORKERPERCENT	To avoid overwhelming healthy instances, no more than half of the instances will be excluded. For example, if an App Service Plan is scaled to four instances and three are unhealthy, at most two will be excluded. The other two instances (one healthy and one unhealthy) will continue to receive requests. In the worst-case scenario where all instances are unhealthy, none will be excluded. To override this behavior, set to a value between 1 and 100. A higher value means more unhealthy instances will be removed. The default is 50 (50%).

Push notifications

The following environment variables are related to the [push notifications](#) feature.

[Expand table](#)

Setting name	Description
WEBSITE_PUSH_ENABLED	Read-only. Added when push notifications are enabled.
WEBSITE_PUSH_TAG_WHITELIST	Read-only. Contains the tags in the notification registration.
WEBSITE_PUSH_TAGS_REQUIRING_AUTH	Read-only. Contains a list of tags in the notification registration that requires user authentication.
WEBSITE_PUSH_TAGS_DYNAMIC	Read-only. Contains a list of tags in the notification registration that were added automatically.

 Note

This article contains references to a term that Microsoft no longer uses. When the term is removed from the software, we'll remove it from this article.

Webjobs

The following environment variables are related to [WebJobs](#).

[Expand table](#)

Setting name	Description
WEBJOBS_RESTART_TIME	For continuous jobs, delay in seconds when a job's process goes down for any reason before relaunching it.
WEBJOBS_IDLE_TIMEOUT	For triggered jobs, timeout in seconds, after which the job is aborted if it's in idle, has no CPU time or output.
WEBJOBS_HISTORY_SIZE	For triggered jobs, maximum number of runs kept in the history directory per job. The default is 50.
WEBJOBS_STOPPED	Set to 1 to disable running any job, and stop all currently running jobs.
WEBJOBS_DISABLE_SCHEDULE	Set to 1 to turn off all scheduled triggering. Jobs can still be manually invoked.
WEBJOBS_ROOT_PATH	Absolute or relative path of webjob files. For a relative path, the value is combined with the default root path (<code>D:/home/site/wwwroot/</code> or <code>/home/site/wwwroot/</code>).

Setting name	Description
<code>WEBJOBS_LOG_TRIGGERED_JOBS_TO_APP_LOGS</code>	Set to true to send output from triggered WebJobs to the application logs pipeline (which supports file system, blobs, and tables).
<code>WEBJOBS_SHUTDOWN_FILE</code>	File that App Service creates when a shutdown request is detected. It's the web job process's responsibility to detect the presence of this file and initiate shutdown. When using the WebJobs SDK, this part is handled automatically.
<code>WEBJOBS_PATH</code>	Read-only. Root path of currently running job (will be under some temporary directory).
<code>WEBJOBS_NAME</code>	Read-only. Current job name.
<code>WEBJOBS_TYPE</code>	Read-only. Current job type (<code>triggered</code> or <code>continuous</code>).
<code>WEBJOBS_DATA_PATH</code>	Read-only. Current job metadata path to contain the job's logs, history, and any artifact of the job.
<code>WEBJOBS_RUN_ID</code>	Read-only. For triggered jobs, current run ID of the job.

Functions

 [Expand table](#)

Setting name	Description
<code>WEBSITE_FUNCTIONS_ARMCACHE_ENABLED</code>	Set to <code>0</code> to disable the functions cache.
<code>WEBSITE_MAX_DYNAMIC_APPLICATION_SCALE_OUT</code>	App settings reference for Azure Functions
<code>AzureWebJobsSecretStorageType</code>	App settings reference for Azure Functions
<code>FUNCTIONS_EXTENSION_VERSION</code>	App settings reference for Azure Functions
<code>FUNCTIONS_WORKER_RUNTIME</code>	App settings reference for Azure Functions
<code>AzureWebJobsStorage</code>	App settings reference for Azure Functions
<code>WEBSITE_CONTENTAZUREFILECONNECTIONSTRING</code>	App settings reference for Azure Functions
<code>WEBSITE_CONTENTSHARE</code>	App settings reference for Azure Functions
<code>WEBSITE_CONTENTOVERVNET</code>	App settings reference for Azure Functions
<code>WEBSITE_ENABLE_BROTLI_ENCODING</code>	App settings reference for Azure Functions
<code>WEBSITE_USE_PLACEHOLDER</code>	App settings reference for Azure Functions
<code>WEBSITE_PLACEHOLDER_MODE</code>	Read-only. Shows whether the function app is running on a placeholder host (<code>generalized</code>) or its own host (<code>specialized</code>).
<code>WEBSITE_DISABLE_ZIP_CACHE</code>	When your app runs from a ZIP package (<code>WEBSITE_RUN_FROM_PACKAGE=1</code>), the five most recently deployed ZIP packages are cached in the app's file system (D:\home\data\SitePackages). Set this variable to <code>1</code> to disable this cache. For Linux consumption apps, the ZIP package cache is disabled by default.

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

Customize the API Management developer portal on WordPress

Article • 01/10/2025

APPLIES TO: Developer | Basic | Standard | Premium

This article shows how to configure an open-source developer portal plugin (preview) to customize the API Management developer portal on WordPress. With the plugin, turn any WordPress site into a developer portal. Take advantage of site capabilities in WordPress to customize and add features to your developer portal including localization, collapsible and expandable menus, custom stylesheets, file downloads, and more.

In this article, you create a WordPress site on Azure App Service and configure the developer portal plugin on the WordPress site. Microsoft Entra ID is configured for authentication to the WordPress site and the developer portal.

Prerequisites

- An API Management instance. If needed, [create an instance](#).

ⓘ Note

Currently, the plugin isn't supported in the API Management v2 tiers.

- Enable and publish the developer portal. For steps, see [Tutorial: Access and customize the developer portal](#).
- Permissions to create an app registration in a Microsoft Entra tenant associated with your Azure subscription.
- Installation files for the developer portal WordPress plugin and customized WordPress theme from the [plugin repo ↗](#). Download the following zip files from the [dist ↗](#) folder in the repo:
 - `apim-devportal.zip` - Plugin file
 - `twentyninefour.zip` - Theme file

Step 1: Create WordPress on App Service

For this scenario, you create a managed WordPress site hosted on Azure App Service. The following are brief steps:

1. In the Azure portal, navigate to <https://portal.azure.com/#create/WordPress.WordPress>.
2. On the **Create WordPress on App Service** page, in the **Basics** tab, enter your project details, Web App details, and WordPress setup settings.

Store the WordPress admin username and password in a safe place. These credentials are required to sign into the WordPress admin site and install the plugin in a later step.

 **Caution**

Avoid using the default WordPress `admin` username, and create a strong password. [Learn more about WordPress password best practices](#)

3. On the **Add-ins** tab:
 - a. Select the recommended default values for **Email with Azure Communication Services**, **Azure CDN**, and **Azure Blob Storage**.
 - b. In **Virtual network**, select either the **New** value or an existing virtual network.
4. On the **Deployment** tab, leave **Add staging slot** unselected.
5. Select **Review + create** to run final validation.
6. Select **Create** to complete app service deployment.

It can take several minutes for the app service to deploy.

Step 2: Create a new Microsoft Entra app registration

In this step, create a new Microsoft Entra app. In later steps, you configure this app as an identity provider for authentication to your app service and to the developer portal in your API Management instance.

1. In the [Azure portal](#), navigate to **App registrations > + New registration**.
2. Provide the new app name, and in **Supported account types**, select **Accounts in this organizational directory only**. Select **Register**.
3. On the **Overview** page, copy and safely store the **Application (client) Id** and **Directory (tenant) Id**. You need these values in later steps to configure

authentication to your API Management instance and app service.

The screenshot shows the Azure portal's 'App registrations' section. A specific app registration named 'myapimportal' is selected. The left sidebar shows 'Manage' and various configuration tabs like 'Branding & properties', 'Authentication', 'Certificates & secrets', 'Token configuration', 'API permissions', and 'Expose an API'. The 'Overview' tab is currently active. On the right, under the 'Essentials' section, there are several fields: 'Display name' (myapimportal), 'Client credentials' (0 certificate, 1 secret), 'Redirect URIs' (1 web, 2 spa, 0 public client), 'Application ID URI' (api://xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx), and 'Managed application in local directory' (myapimportal). Two specific fields are highlighted with red boxes: 'Application (client) ID' (xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx) and 'Directory (tenant) ID' (xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx).

4. In the left menu, under **Manage**, select **Authentication** > **+ Add a platform**.
5. On the **Configure platforms** page, select **Web**.
6. On the **Configure Web** page, enter the following redirect URI, substituting the name of your app service, and select **Configure**:
`https://app-service-name.azurewebsites.net/.auth/login/aad/callback`
7. Select **+ Add a platform** again. Select **Single-page application**.
8. On the **Configure single-page application** page, enter the following redirect URI, substituting the name of your API Management instance, and select **Configure**:
`https://<apim-instance-name>.developer.azure-api.net/signin`
9. Select **+ Add a platform** again. Select **Single-page application** again.
10. On the **Configure single-page application** page, enter the following redirect URI, substituting the name of your API Management instance, and select **Configure**:
`https://<apim-instance-name>.developer.azure-api.net/`
11. In the left menu, under **Manage**, select **Token configuration** > **+ Add optional claim**.
12. On the **Add optional claim** page, select **ID** and then select the following claims: **email**, **family_name**, **given_name**, **onprem_sid**, **preferred_username**, **upn**. Select **Add**.
13. When prompted, select **Turn on the Microsoft Graph email, profile permission**. Select **Add**.

14. In the left menu, under **Manage** select **API permissions** and confirm that the following Microsoft Graph permissions are present: **email**, **profile**, **User.Read**.

Configured permissions					
Applications are authorized to call APIs when they are granted permissions by users/admins as part of the consent process. The list of configured permissions should include all the permissions the application needs. Learn more about permissions and consent					
+		Add a permission	✓ Grant admin consent for Microsoft		
API / Permissions name	Type	Description	Admin consent req...	Status	...
✓ Microsoft Graph (3)					...
email	Delegated	View users' email address	No		...
profile	Delegated	View users' basic profile	No		...
User.Read	Delegated	Sign in and read user profile	No		...

15. In the left menu, under **Manage**, select **Certificates & secrets** > **+ New client secret**.

16. Configure settings for the secret and select **Add**. Copy and safely store the secret's **Value** immediately after it's generated. You need this value in later steps to add the application to your API Management instance and app service for authentication.

17. In the left menu, under **Manage**, select **Expose an API**. Note the default **Application ID URI**. Optionally add custom scopes if needed.

Step 3: Enable authentication to the app service

In this step, add the Microsoft Entra app registration as an identity provider for authentication to the WordPress app service.

1. In the [portal](#), navigate to the WordPress app service.
2. In the left menu, under **Settings**, select **Authentication** > **Add identity provider**.
3. On the **Basics** tab, in **Identity provider**, select **Microsoft**.
4. Under **App registration**, select **Provide the details of an existing app registration**.
 - a. Enter the **Application (client) Id** and **Client secret** from the app registration that you created in the previous step.
 - b. In **Issuer URL**, enter either of the following values for the authentication endpoint: `https://login.microsoftonline.com/<tenant-id>` or `https://sts.windows.net/<tenantid>`. Replace `<tenant-id>` with the **Directory (tenant) Id** from the app registration.

ⓘ Important

Do not use the version 2.0 endpoint for the issuer URL (URL ending in `/v2.0`).

5. In **Allowed token audiences**, enter the **Application ID URI** from the app registration. Example: `api://<app-id>`.
6. Under **Additional checks**, select values appropriate for your environment, or use the default values.
7. Configure your desired the values for the remaining settings, or use the default values. Select **Add**.

 **Note**

If you want to allow guest users as well as signed-in users to access the developer portal on WordPress, you can enable unauthenticated access. In **Restrict access**, select **Allow unauthenticated access**. [Learn more](#)

The identity provider is added to the app service.

Step 4: Enable authentication to the API Management developer portal

Configure the same Microsoft Entra app registration as an identity provider for the API Management developer portal.

1. In the [portal](#), navigate to your API Management instance.
2. In the left menu, under **Developer portal**, select **Identities > + Add**.
3. On the **Add identity provider** page, select **Azure Active Directory (Microsoft Entra ID)**.
4. Enter the **Client Id**, **Client secret**, and **Signin tenant** values from the app registration that you created in a previous step. The **Signin tenant** is the **Directory (tenant) Id** from the app registration.
5. In **Client library**, select **MSAL**.
6. Accept default values for the remaining settings and select **Add**.
7. [Republish the developer portal](#) to apply the changes.
8. Test the authentication by signing into the developer portal at the following URL, substituting the name of your API Management instance: `https://<apim-instance->`

`name>.developer.azure-api.net/signin`. Select the **Azure Active Directory** (Microsoft Entra ID) button at the bottom to sign in.

When you open it for the first time, you may be prompted to consent to specific permissions.

Step 5: Configure developer portal settings in API Management

Update the settings of the developer portal in the API Management instance to enable CORS and to include the app service site as a portal origin.

1. In the [Azure portal](#), navigate to your API Management instance.
2. In the left menu, under **Developer portal**, select **Portal overview**.
3. On the **Portal overview** tab, select **Enable CORS**.
4. In the left menu, under **Developer portal**, select **Portal settings**.
5. On the **Self-hosted portal configuration** tab, enter the hostname of your WordPress on App Service site as a portal origin, substituting the name of your app service in the following URL: `https://<yourapp-service-name>.azurewebsites.net`
6. [Republish the developer portal](#) to apply the changes.

Also, update the `cors` policy configuration in the API Management instance to add the app service site as an allowed origin. This value is needed to allow calls from the developer portal's test console on the WordPress site.

Add the following `origin` value in your `cors` policy configuration:

XML

```
<cors ...>
  <allowed-origins>
    [...]
    <origin>https://<yourapp-service-name>.azurewebsites.net</origin>
  </allowed-origins>
</cors>
```

Learn more about how to [set or edit policies](#).

Step 6: Navigate to WordPress admin site and upload the customized theme

In this step, you upload the customized theme for the API Management developer portal to the WordPress admin site.

Important

We recommend that you upload the theme provided in the plugin repo. The theme is based on the Twenty Twenty Four theme and is customized to support the developer portal functionality in WordPress. If you choose to use a different theme, some functionality may not work as expected or require additional customization.

1. Open the WordPress admin website at the following URL, substituting the name of your app service: `http://<app-service-name>.azurewebsites.net/wp-admin`
When you open it for the first time, you may be prompted to consent to specific permissions.
2. Sign into the WordPress admin site using the username and password that you entered while creating WordPress on App Service.
3. In the left menu, select **Appearance > Themes** and then **Add New Theme**.
4. Select **Upload Theme**. Select **Choose File** to upload the API Management developer portal theme zip file that you downloaded previously. Select **Install Now**.
5. In the next screen, select **Replace active with uploaded**.
6. If prompted, select **Activate**.

Note

If your WordPress site includes a caching plug-in, clear the cache after activating the theme to ensure that the changes take effect.

Step 7: Install the developer portal plugin

1. In the WordPress admin site, in the left menu, select **Plugins > Add New Plugin**.
2. Select **Upload Plugin**. Select **Choose File** to upload the API Management developer portal plugin zip file (`apim-devportal.zip`) that you downloaded previously. Select **Install Now**.
3. After successful installation, select **Activate Plugin**.

4. In the left menu, select Azure API Management Developer Portal.
5. On the APIM Settings page, enter the following settings and select Save Changes:
 - APIM service name - Name of your API Management instance
 - Enable Create default pages and Create navigation menu

Step 8: Add a custom stylesheet

Add a custom stylesheet for the API Management developer portal.

1. In the WordPress admin site, in the left menu, select Appearance > Themes.
2. Select Customize and then navigate to Styles.
3. Select the pencil icon (Edit Styles).
4. In the Styles pane, select More (three dots) > Additional CSS.
5. In Additional CSS, enter the following CSS:

css

```
.apim-table {  
    width: 100%;  
    border: 1px solid #D1D1D1;  
    border-radius: 4px;  
    border-spacing: 0;  
}  
  
.apim-table th {  
    background: #E6E6E6;  
    font-weight: bold;  
    text-align: left;  
}  
  
.apim-table th,  
.apim-table td {  
    padding: .7em 1em;  
}  
  
.apim-table td {  
    border-top: #E6E6E6 solid 1px;  
}  
  
.apim-input,  
.apim-button,  
.apim-nav-link-btn {  
    border-radius: .33rem;  
    padding: 0.6rem 1rem;  
}
```

```
.apim-button,  
.apim-nav-link-btn {  
    background-color: var(--wp--preset--color--contrast);  
    border-color: var(--wp--preset--color--contrast);  
    border-width: 0;  
    color: var(--wp--preset--color--base);  
    font-size: var(--wp--preset--font-size--small);  
    font-weight: 500;  
    text-decoration: none;  
    cursor: pointer;  
    transition: all .25s ease;  
}  
  
.apim-nav-link-btn:hover {  
    background: var(--wp--preset--color--base);  
    color: var(--wp--preset--color--contrast);  
}  
  
.apim-button:hover {  
    background: var(--wp--preset--color--vivid-cyan-blue);  
}  
  
.apim-button:disabled {  
    background: var(--wp--preset--color--contrast-2);  
    cursor: not-allowed;  
}  
  
.apim-label {  
    display: block;  
    margin-bottom: 0.5rem;  
}  
  
.apim-input {  
    border: solid 1px var(--wp--preset--color--contrast);  
}  
  
.apim-grid {  
    display: grid;  
    grid-template-columns: 11em max-content;  
}  
  
.apim-link,  
.apim-nav-link {  
    text-align: inherit;  
    font-size: inherit;  
    padding: 0;  
    background: none;  
    border: none;  
    font-weight: inherit;  
    cursor: pointer;  
    text-decoration: none;  
    color: var(--wp--preset--color--vivid-cyan-blue);  
}
```

```
.apim-nav-link {  
    font-weight: 500;  
}  
  
.apim-link:hover,  
.apim-nav-link:hover {  
    text-decoration: underline;  
}  
  
#apim-signIn {  
    display: flex;  
    align-items: center;  
    gap: 24px;  
}
```

6. Save the changes. Select **Save** again to save the changes to the theme.

7. Log Out of the WordPress admin site.

Step 9: Sign into the API Management developer portal deployed on WordPress

Access the WordPress site to see your new API Management developer portal deployed on WordPress and hosted on App Service.

1. In a new browser window, navigate to your WordPress site, substituting the name of your app service in the following URL: `https://<yourapp-service-name>.azurewebsites.net`.
2. When prompted, sign in using Microsoft Entra ID credentials for a developer account. If unauthenticated access to the developer portal is enabled, select **Sign in** on the home page of the developer portal.

Note

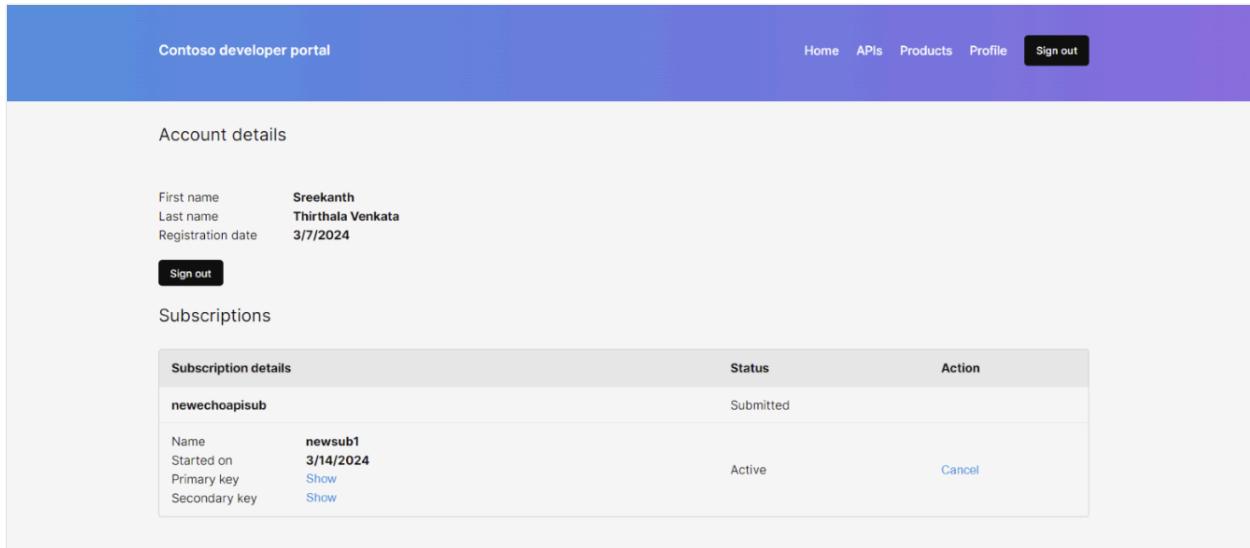
You can only sign in to the developer portal on WordPress using Microsoft Entra ID credentials. Basic authentication isn't supported.

You can now use the following features of the API Management developer portal:

- Sign into the portal
- See list of APIs
- Navigate to API details page and see list of operations
- Test the API using valid API keys
- See list of products

- Subscribe to a product and generate subscription keys
- Navigate to **Profile** tab with account and subscription details
- Sign out of the portal

The following screenshot shows a sample page of the API Management developer portal hosted on WordPress.



Troubleshooting

I don't see the latest developer portal pages on the WordPress site

If you don't see the latest developer portal pages when you visit the WordPress site, check that the developer portal plugin is installed, activated, and configured in the WordPress admin site. See [Install the developer portal plugin](#) for steps.

You might also need to clear the cache on your WordPress site or in the CDN, if one is configured. Alternatively, you might need to clear the cache on your browser.

I'm having problems signing in or out of the developer portal

If you're having problems signing in or out of the developer portal, clear the browser cache, or view the developer portal site in a separate browser session (using incognito or private browsing mode).

I don't see a sign-in button on the developer portal navigation bar

If you're using a custom theme different from the one provided in the plugin repo, you may need to add the sign-in functionality to the navigation bar manually. On the Home page, add the following shortcode block: [SignInButton]. [Learn more](#) in the WordPress documentation.

You can also sign in or sign out manually by entering the following URLs in your browser:

- Sign in: `https://<app-service-name>.azurewebsites.net/.auth/login/aad`
- Sign out: `https://<app-service-name>.azurewebsites.net/.auth/logout`

Related content

- [Create a WordPress site on Azure App Service](#)
- [Customize the developer portal](#)
- [Authorize developer accounts by using Microsoft Entra ID in Azure API Management.](#)

Feedback

Was this page helpful?



[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

WordPress on App Service: Frequently Asked Questions

Article • 12/13/2024

Are there limits on the number of sites, visits, storage, or bandwidth?

The allocated resources for an App Service plan and database tier determine the hosting capacity. For example:

- **App Service B1 Plan:** Includes 1 core, 1.75 GB RAM, and 10 GB storage.
- **Database B1ms Instance:** Offers 1 vCore, 2 GB RAM, and storage up to 16 GB.

There's no fixed limit on the number of sites you can host, but recommended app limits by SKU are:

[+] Expand table

SKU	Recommended Max Apps
B1, S1, P1v2, I1v1	8
B2, S2, P2v2, I2v1	16
B3, S3, P3v2, I3v1	32
P1v3, I1v2	16
P2v3, I2v2	32
P3v3, I3v2	64

Bandwidth is unlimited, but [charges apply for internet egress ↗](#).

How are security patches updated?

Azure manages security patches for core technologies, while WordPress-specific updates may require manual or semi-automated steps:

- **PHP Major Versions:** Update manually under **App Service > Settings > Configuration**.

- **WordPress Core:** Minor updates are automatic, while major updates need manual configuration.
- **Plugins and Themes:** Perform manual updates after backing up your site to avoid issues. WordPress also offers auto update options.

See [How to keep your WordPress website stack on Azure App Service up to date](#) for more information.

What security features are available to protect my website?

Azure App Service integrates robust security features to safeguard WordPress sites:

- [App Service Security](#): HTTPS, IP restrictions, certificates, authentication, and network isolation.
- [Easy Authentication](#): Built-in identity provider integration with minimal effort.
- [Azure Database for MySQL](#): Advanced protections for Azure MySQL servers, including encryption and backup capabilities.
- [Virtual Network \(VNET\)](#): Secure communication between Azure resources, the internet, and on-premises networks.
- [Managed Identities](#): Credential-free access to resources using Microsoft Entra tokens.
- [Microsoft Defender for Cloud](#): Proactive threat detection with DevSecOps integration.
- [Azure Key Vault](#): Secure storage for keys, secrets, and certificates.
- [Microsoft Entra ID](#): Single sign-On (SSO) for seamless authentication.

How can I set up WordPress Multisite?

WordPress Multisite allows managing multiple sites from a single installation. To enable: Set up a [subdirectory-based Multisite](#) or [subdomain-based Multisite](#).

Note

- Conversion to Multisite is permanent; reverting to a single site is unsupported.
- Switching between subdirectory and subdomain setups isn't allowed.

How do I access my WordPress website's database?

The database can be accessed using phpMyAdmin at: <https://<your-site-link>/phpmyadmin>. Use the `DATABASE_USERNAME` as the username and a generated token as the password (tokens can be retrieved via Kudu SSH).

How do I enable a custom domain for my WordPress website?

Custom domains can be set up with these resources:

- [Using custom domains with WordPress on Azure App Service ↗](#)
- [Configuring custom domains with Azure Front Door](#)

Does WordPress on App Service have email functionality?

Yes, email functionality is supported through [Azure Communication Services ↗](#). Custom email domains can be also be configured.

How can I update NGINX configurations for my WordPress website?

NGINX configurations can be updated using a [startup script](#). Detailed instructions are available in the [startup script guide ↗](#).

How can I access error logs for my WordPress website?

Access error logs for debugging via [App Service logs](#) or the [Kudu dashboard](#). Refer to the [documentation](#) for detailed steps.

How do I estimate pricing for hosting a WordPress site on Azure?

Use the [Azure Pricing Calculator](#) to estimate hosting costs, considering App Service, MySQL, CDN, Blob Storage, and other components. Use this [pricing estimate guide](#) for more information.

How can I debug and monitor my WordPress site?

Key tools for debugging and monitoring WordPress sites include:

- [App Service Logs](#)
- [Kudu](#)
- [SSH Access](#)

PhpMyAdmin

WordPress on App Service utilizes an Azure Database for MySQL Flexible Server, which is integrated into a VNET. This setup restricts database access to within the VNET.

WordPress on App Service includes phpMyAdmin by default. You can access it at:

<https://<your-site-link>/phpmyadmin>.

If you are using Managed Identities, you can log in to phpMyAdmin by using the value from DATABASE_USERNAME environment variable as the username and the token as the password. To find the token use your Kudu SSH to run the following command:

```
/usr/local/bin/fetch-mysql-access-token.sh
```

Or you can find the database username and password from App Service environment variables

What features can I use to boost my WordPress site's performance?

Enhance performance with these features / plugins:

- [Content Delivery Network \(CDN\)](#)
- [Azure Front Door \(AFD\)](#)
- [Blob Storage](#)
- Dynamic Caching
- [Image Compression \(Smush\)](#)
- [Scaling Up and Out](#)
- [Redis Cache](#)

What are the options for configuring and setting up my WordPress site?

Options for setting up WordPress include:

- [FTP File Transfers](#)
- [NGINX Configuration Updates ↗](#)
- [App Service settings](#)

How can I build a headless WordPress site?

Enable [WP REST APIs](#) and integrate with [Static Web Apps](#) to create a decoupled front-end experience. Learn more [here ↗](#).

What features are available for creating an enterprise-grade production website?

Key features include:

- [Staging slots ↗](#) for safe testing
- [Custom domains ↗](#) and SSL certificates
- [CI/CD ↗](#) pipelines for automated deployments
- [Startup scripts ↗](#) for configuration
- [Emails with custom domain](#)
- [Scaling](#) and [load testing](#) capabilities

What are common errors for WordPress on App Service, and how can I troubleshoot?

Typical issues and resolutions:

- [Debug Logs ↗](#): Enable for troubleshooting.
- [CORS Issues ↗](#): Adjust settings in CDN or Azure Front Door.
- [Existing WordPress Detected Warning ↗](#): Follow [troubleshooting steps](#).
- [Intl Extension issues ↗](#): Install via the configuration panel.

ⓘ Note: The author created this article with assistance from AI. [Learn more](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

Run background tasks with WebJobs in Azure App Service

Article • 04/04/2024

ⓘ Note

WebJobs for Windows container, Linux code, and Linux container is in preview.

WebJobs for Windows code is generally available and not in preview.

Deploy WebJobs by using the [Azure portal](#) to upload an executable or script. You can run background tasks in the Azure App Service.

If instead of the Azure App Service, you're using Visual Studio to develop and deploy WebJobs, see [Deploy WebJobs using Visual Studio](#).

Overview

WebJobs is a feature of [Azure App Service](#) that enables you to run a program or script in the same instance as a web app. There's no extra cost to use WebJobs.

You can use the Azure WebJobs SDK with WebJobs to simplify many programming tasks. For more information, see [What is the WebJobs SDK](#).

Azure Functions provides another way to run programs and scripts. For a comparison between WebJobs and Functions, see [Choose between Flow, Logic Apps, Functions, and WebJobs](#).

WebJob types

Supported file types for scripts or programs

Windows code

The following file types are supported:

- .cmd, .bat, .exe (using Windows cmd)
- .ps1 (using PowerShell)
- .sh (using Bash)
- .php (using PHP)

.py (using Python)

.js (using Node.js)

.jar (using Java)

The necessary runtimes to run these file types are already installed on the web app instance.

Continuous vs. triggered WebJobs

The following table describes the differences between *continuous* and *triggered* WebJobs:

[] Expand table

Continuous	Triggered
Starts immediately when the WebJob is created. To keep the job from ending, the program or script typically does its work inside an endless loop. If the job does end, you can restart it. Typically used with WebJobs SDK.	Starts only when triggered manually or on a schedule.
Runs on all instances that the web app runs on. You can optionally restrict the WebJob to a single instance.	Runs on a single instance that Azure selects for load balancing.
Supports remote debugging.	Doesn't support remote debugging.
Code is deployed under <code>\site\wwwroot\app_data\Jobs\Continuous</code> .	Code is deployed under <code>\site\wwwroot\app_data\Jobs\Triggered</code> .

! Note

A web app can time out after 20 minutes of inactivity, and only requests to the actual web app can reset the timer. Viewing the app's configuration in the Azure portal or making requests to the advanced tools site

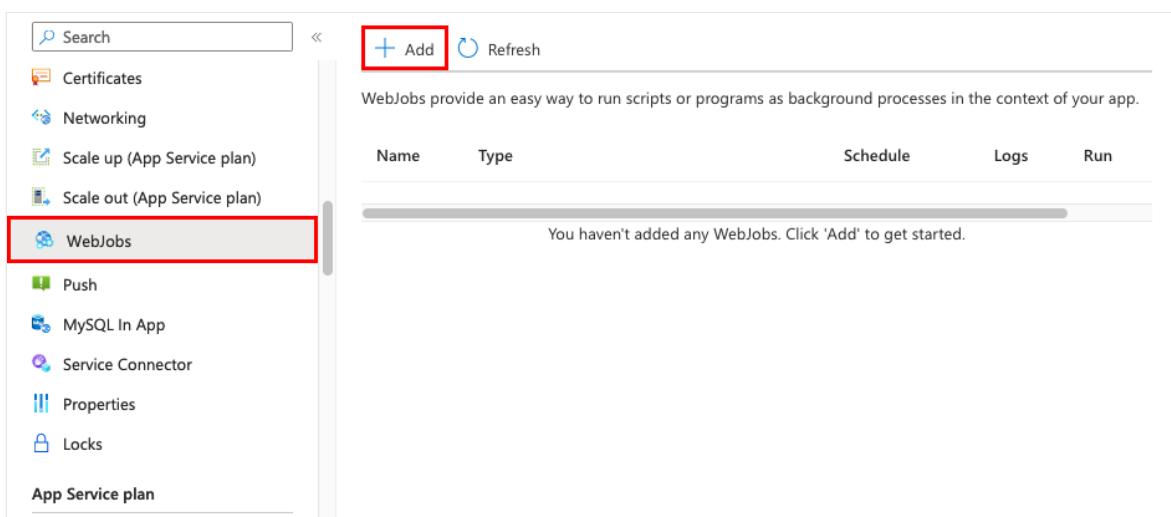
(https://<app_name>.scm.azurewebsites.net) doesn't reset the timer. If you set the web app that hosts your job to run continuously, run on a schedule, or use event-driven triggers, enable the **Always on** setting on your web app's Azure Configuration page. The Always on setting helps to make sure that these kinds of WebJobs run reliably. This feature is available only in the Basic, Standard, and Premium [pricing tiers](#).

Create a continuous WebJob

ⓘ Important

When you have source control configured for your application, Webjobs should be deployed as part of the source control integration. After source control is configured for your application, a WebJob can't be added from the Azure portal.

1. In the [Azure portal](#), go to the **App Service** page of your App Service web app, API app, or mobile app.
2. From the left pane, select **WebJobs**, then select **Add**.



3. Fill in the **Add WebJob** settings as specified in the table, then select **Create Webjob**.

Add WebJob X

my-demo-app

Name * ⓘ

File Upload *

 | Browse

Type * ⓘ

Scale * ⓘ

Create Webjob

[] Expand table

Setting	Sample value	Description
Name	myContinuousWebJob	A name that is unique within an App Service app. Must start with a letter or a number and must not contain special characters other than "-" and "_".
File Upload	ConsoleApp.zip	A <i>.zip</i> file that contains your executable or script file and any supporting files needed to run the program or script. The supported executable or script file types are listed in the Supported file types section.
Type	Continuous	The WebJob types are described earlier in this article.
Scale	Multi Instance	Available only for Continuous WebJobs. Determines whether the program or script runs on all instances or one instance. The option to run on multiple instances doesn't apply to the Free or Shared pricing tiers .

4. The new WebJob appears on the [WebJobs](#) page. If you see a message that says the WebJob was added, but you don't see it, select **Refresh**.

5. To stop or restart a continuous WebJob, right-click the WebJob in the list and select the **Stop** or **Run** button, then confirm your selection.

The screenshot shows the Azure portal's WebJobs management interface. At the top, there are 'Add' and 'Refresh' buttons. Below that, a message says 'WebJobs provide an easy way to run scripts or programs as background processes in the context of your app.' A table lists a single WebJob:

Name	Type	Status	Schedule	Logs	Run	Delete
myContinuousWebJob	Continuous	Running	Manual			

A context menu is open over the 'Run' button, showing a modal dialog titled 'Stop myContinuousWebJob'. The dialog contains the text 'Are you sure you want to stop the selected WebJob?' with two buttons: 'Stop' (highlighted with a red box) and 'Cancel'.

Create a manually triggered WebJob

1. In the [Azure portal](#), go to the App Service page of your App Service web app, API app, or mobile app.
2. From the left pane, select **WebJobs**, then select **Add**.

The screenshot shows the Azure portal's left sidebar with various service icons. The 'WebJobs' icon is highlighted with a red box. The main content area has a search bar and 'Add' and 'Refresh' buttons at the top. A message in the center says 'You haven't added any WebJobs. Click 'Add' to get started.' Below this, there is a table with columns 'Name', 'Type', 'Schedule', 'Logs', and 'Run'.

3. Fill in the **Add WebJob** settings as specified in the table, then select **Create Webjob**.

Add WebJob

my-demo-app

Name * ⓘ

File Upload *

 |

Type * ⓘ

Triggers * ⓘ

Create Webjob

Calling the webhook will trigger a

[+] Expand table

Setting	Sample value	Description
Name	myTriggeredWebJob	A name that is unique within an App Service app. Must start with a letter or a number and must not contain special characters other than "-" and "_".
File Upload	ConsoleApp1.zip	A .zip file that contains your executable or script file and any supporting files needed to run the program or script. The supported executable or script file types are listed in the Supported file types section.
Type	Triggered	The WebJob types are described previously in this article.
Triggers	Manual	

4. The new WebJob appears on the [WebJobs](#) page. If you see a message that says the WebJob was added, but you don't see it, select **Refresh**.
5. To run a manually triggered WebJob, right-click the WebJob in the list and select the **Run** button, then confirm your selection.

The screenshot shows the Azure portal's 'WebJobs' section. A table lists one job: 'myTriggeredWebJob' (Type: Triggered, Status: Ready, Schedule: Manual). A modal window titled 'Run myTriggeredWebJob' contains the message 'Are you sure you want to run the selected WebJob?' with 'Run' and 'Cancel' buttons. The 'Run' button is highlighted with a red box.

Create a scheduled WebJob

A scheduled Webjob is also triggered. You can schedule the trigger to occur automatically on the schedule you specify.

1. In the [Azure portal](#), go to the App Service page of your App Service web app, API app, or mobile app.
2. From the left pane, select **WebJobs**, then select **Add**.

The screenshot shows the Azure portal's 'WebJobs' section. The left sidebar has a 'WebJobs' link highlighted with a red box. The main area displays a message: 'You haven't added any WebJobs. Click 'Add' to get started.' At the top, there is a 'Search' bar, an 'Add' button (highlighted with a red box), and a 'Refresh' button.

3. Fill in the **Add WebJob** settings as specified in the table, then select **Create Webjob**.

Add WebJob

X

my-demo-app

Name * ⓘ

myScheduledWebJob

File Upload *

ConsoleApp1.zip

Browse

Type * ⓘ

Triggered

Triggers * ⓘ

Scheduled

CRON Expression * ⓘ

0 0/20 * * * *

Create Webjob

[+] Expand table

Setting	Sample value	Description
Name	myScheduledWebJob	A name that is unique within an App Service app. Must start with a letter or a number and must not contain special characters other than "-" and "_".
File Upload	ConsoleApp.zip	A .zip file that contains your executable or script file and any supporting files needed to run the program or script. The supported executable or script file types are listed in the Supported file types section.
Type	Triggered	The WebJob types are described earlier in this article.
Triggers	Scheduled	For the scheduling to work reliably, enable the Always On feature. Always On is available only in the Basic, Standard, and Premium pricing tiers.
CRON Expression	0 0/20 * * *	CRON expressions are described in the following section.

4. The new WebJob appears on the **WebJobs** page. If you see a message that says the WebJob was added, but you don't see it, select **Refresh**.
5. The scheduled WebJob is run at the schedule defined by the CRON expression. To run it manually at anytime, right-click the WebJob in the list and select the **Run** button, then confirm your selection.

The screenshot shows the Azure portal's 'WebJobs' section. At the top, there are 'Add' and 'Refresh' buttons. Below that, a descriptive text states: 'WebJobs provide an easy way to run scripts or programs as background processes in the context of your app.' A table lists a single job:

Name	Type	Schedule	Logs	Run
myScheduledWebJob	Triggered	Ready	0 0/20 * * * *	(highlighted with a red box)

A modal dialog box is displayed over the table, titled 'Run myScheduledWebJob'. It contains the question 'Are you sure you want to run the selected WebJob?' with two buttons: 'Run' (highlighted with a red box) and 'Cancel'.

NCRONTAB expressions

You can enter a **NCRONTAB expression** in the portal or include a `settings.job` file at the root of your WebJob `.zip` file, as in the following example:

```
JSON

{
  "schedule": "0 */15 * * *"
}
```

To learn more, see [Scheduling a triggered WebJob](#).

ⓘ Note

The default time zone used to run CRON expressions is Coordinated Universal Time (UTC). To have your CRON expression run based on another time zone, create an app setting for your function app named `WEBSITE_TIME_ZONE`. To learn more, see [NCRONTAB time zones](#).

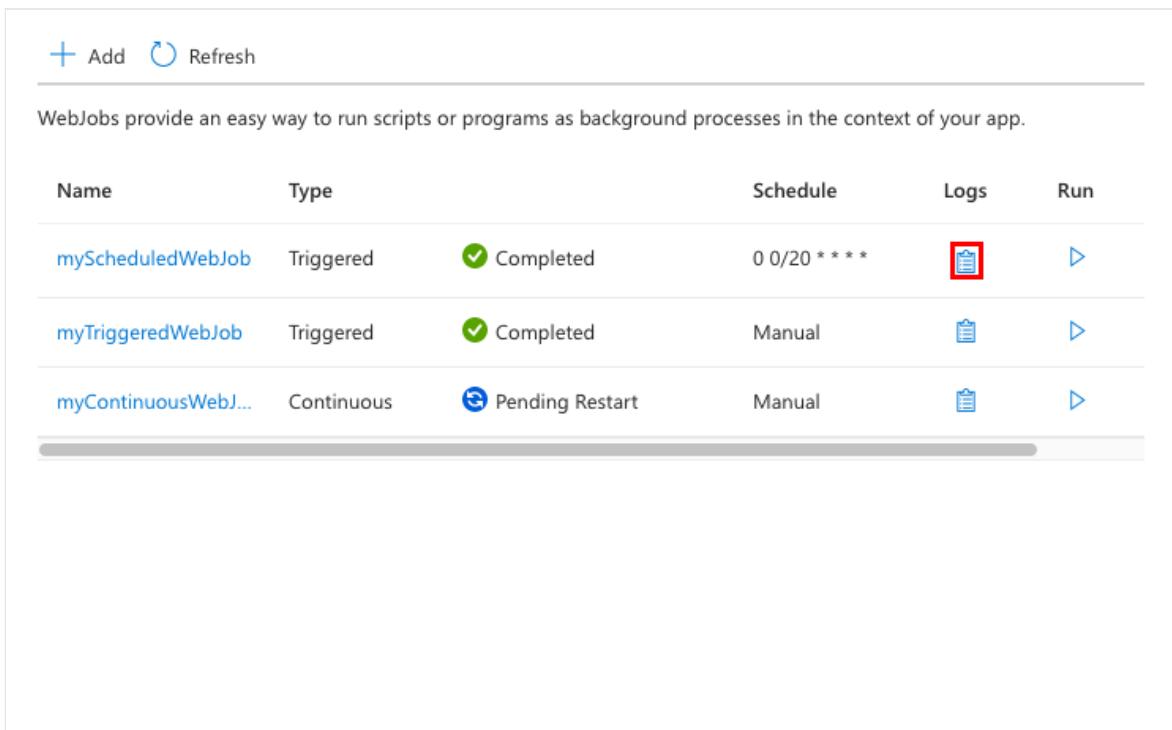
Manage WebJobs

You can manage the running state individual WebJobs running in your site in the [Azure portal](#). Go to **Settings > WebJobs**, choose the WebJob, and you can start and stop the WebJob. You can also view and modify the password of the webhook that runs the WebJob.

You can also [add an application setting](#) named `WEBJOBS_STOPPED` with a value of `1` to stop all WebJobs running on your site. You can use this method to prevent conflicting WebJobs from running both in staging and production slots. You can similarly use a value of `1` for the `WEBJOBS_DISABLE_SCHEDULE` setting to disable triggered WebJobs in the site or a staging slot. For slots, remember to enable the **Deployment slot setting** option so that the setting itself doesn't get swapped.

View the job history

1. For the WebJob you want to see, select **Logs**.



The screenshot shows the Azure portal's WebJobs management interface. At the top, there are 'Add' and 'Refresh' buttons. A descriptive text states: 'WebJobs provide an easy way to run scripts or programs as background processes in the context of your app.' Below this is a table listing three WebJobs:

Name	Type	Schedule	Logs	Run
myScheduledWebJob	Triggered	Completed	0 0/20 * * *	 
myTriggeredWebJob	Triggered	Completed	Manual	 
myContinuousWebJ...	Continuous	Pending Restart	Manual	 

2. In the **WebJob Details** page, select a time to see details for one run.



WebJobs

WebJob Details myScheduledWebJob

Run command: test.bat

Recent job runs

TIMING	STATUS
9 minutes ago (1 minute running time)	Success

Do more with [Microsoft Azure WebJobs SDK](#). The SDK integrates Microsoft Azure Storage, triggering a function in your program when items are added to Queues, Blobs, or Tables.

3. In the **WebJob Run Details** page, you can select **download** to get a text file of the logs, or select the **WebJobs** breadcrumb link at the top of the page to see logs for a different WebJob.

WebJob statuses

The following is a list of common WebJob statuses:

- **Initializing** The app has started and the WebJob is going through its initialization process.
- **Starting** The WebJob is starting up.
- **Running** The WebJob is running.
- **PendingRestart** A continuous WebJob exits in less than two minutes since it started for any reason, and App Service waits 60 seconds before restarting the WebJob. If the continuous WebJob exits after the two-minute mark, App Service doesn't wait the 60 seconds and restarts the WebJob immediately.
- **Stopped** The WebJob was stopped (usually from the Azure portal) and is currently not running and won't run until you start it again manually, even for a continuous or scheduled WebJob.

- **Aborted** This can occur for many of reasons, such as when a long-running WebJob reaches the timeout marker.

Next steps

The Azure WebJobs SDK can be used with WebJobs to simplify many programming tasks. For more information, see [What is the WebJobs SDK](#).

Develop and deploy WebJobs using Visual Studio

Article • 06/01/2022

This article explains how to use Visual Studio to deploy a console app project to a web app in [Azure App Service](#) as an [Azure WebJob](#). For information about how to deploy WebJobs by using the [Azure portal](#), see [Run background tasks with WebJobs in Azure App Service](#).

You can choose to develop a WebJob that runs as either a [.NET Core app](#) or a [.NET Framework app](#). Version 3.x of the [Azure WebJobs SDK](#) lets you develop WebJobs that run as either .NET Core apps or .NET Framework apps, while version 2.x supports only the .NET Framework. The way that you deploy a WebJobs project is different for .NET Core projects than for .NET Framework projects.

You can publish multiple WebJobs to a single web app, provided that each WebJob in a web app has a unique name.

WebJobs as .NET Core console apps

With version 3.x of the Azure WebJobs SDK, you can create and publish WebJobs as .NET Core console apps. For step-by-step instructions to create and publish a .NET Core console app to Azure as a WebJob, see [Get started with the Azure WebJobs SDK for event-driven background processing](#).

Note

.NET Core Web Apps and/or .NET Core WebJobs can't be linked with web projects. If you need to deploy your WebJob with a web app, [create your WebJobs as a .NET Framework console app](#).

Deploy to Azure App Service

Publishing a .NET Core WebJob to Azure App Service from Visual Studio uses the same tooling as publishing an ASP.NET Core app.

1. In **Solution Explorer**, right-click the project and select **Publish**.
2. In the **Publish** dialog box, select **Azure** for **Target**, and then select **Next**.

3. Select **Azure WebJobs** for **Specific target**, and then select **Next**.
4. Above **App Service instances** select the plus (+) button to **Create a new Azure WebJob**.
5. In the **App Service (Windows)** dialog box, use the hosting settings in the following table.

Setting	Suggested value	Description
Name	Globally unique name	Name that uniquely identifies your new function app.
Subscription	Choose your subscription	The Azure subscription to use.
Resource group	myResourceGroup	Name of the resource group in which to create your function app. Choose New to create a new resource group.
Hosting Plan	App Service plan	An App Service plan specifies the location, size, and features of the web server farm that hosts your app. You can save money when hosting multiple apps by configuring the web apps to share a single App Service plan. App Service plans define the region, instance size, scale count, and SKU (Free, Shared, Basic, Standard, or Premium). Choose New to create a new App Service plan. Free and Basic tiers don't support the Always On option to keep your site running continuously.

The screenshot shows the 'App Service (Windows)' creation dialog. At the top left is a cloud icon with a square overlay. To its right is the text 'App Service (Windows)' and a 'Create new' link. In the top right corner is a user profile icon for 'Contoso user@contoso.com'. Below these are four input fields: 'Name' (containing 'WebJobsApp'), 'Subscription' (set to 'Vendor Subscription'), 'Resource group' (set to 'myResourceGroup (Central US)'), and 'Hosting Plan' (set to 'WebJobsApp* (Central US, F1)'). At the bottom are three buttons: 'Export...', 'Create' (which is highlighted with a blue border), and 'Cancel'.

6. Select **Create** to create a WebJob and related resources in Azure with these settings and deploy your project code.
7. Select **Finish** to return to the **Publish** page.

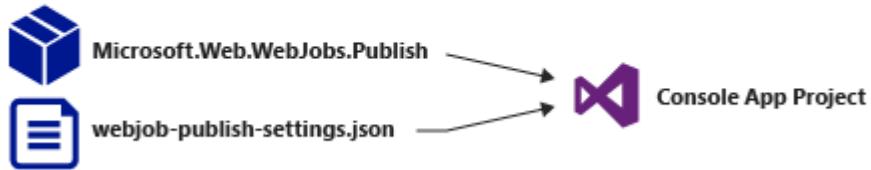
WebJobs as .NET Framework console apps

If you use Visual Studio to deploy a WebJobs-enabled .NET Framework console app project, it copies runtime files to the appropriate folder in the web app (*App_Data/jobs/continuous* for continuous WebJobs and *App_Data/jobs/triggered* for scheduled or on-demand WebJobs).

Visual Studio adds the following items to a WebJobs-enabled project:

- The [Microsoft.Web.WebJobs.Publish](#) NuGet package.
- A [webjob-publish-settings.json](#) file that contains deployment and scheduler settings.

Enable deployment as a WebJob



You can add these items to an existing console app project or use a template to create a new WebJobs-enabled console app project.

Deploy a project as a WebJob by itself, or link it to a web project so that it automatically deploys whenever you deploy the web project. To link projects, Visual Studio includes the name of the WebJobs-enabled project in a [webjobs-list.json](#) file in the web project.

Link WebJob project to web project



Prerequisites

Install Visual Studio 2022 with the [Azure development workload](#).

Enable WebJobs deployment for an existing console app project

You have two options:

- [Enable automatic deployment with a web project.](#)

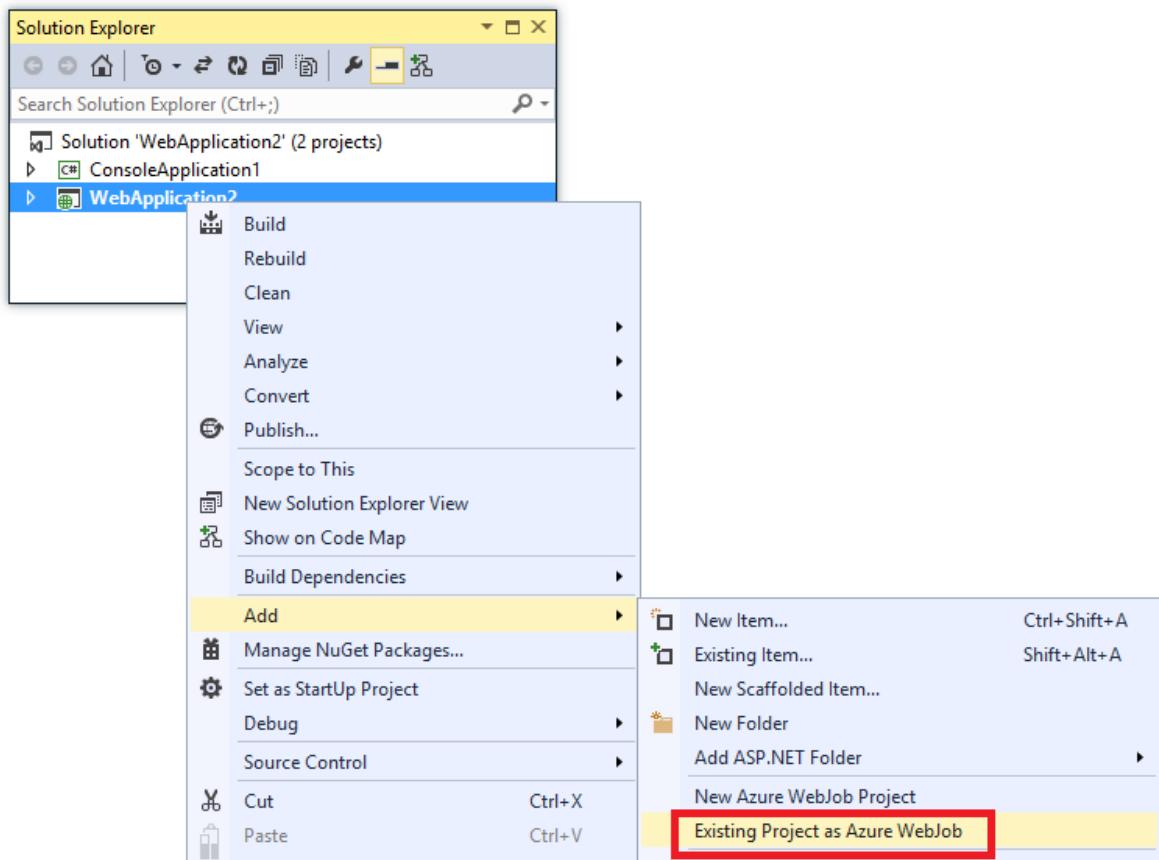
Configure an existing console app project so that it automatically deploys as a WebJob when you deploy a web project. Use this option when you want to run your WebJob in the same web app in which you run the related web application.

- [Enable deployment without a web project.](#)

Configure an existing console app project to deploy as a WebJob by itself, without a link to a web project. Use this option when you want to run a WebJob in a web app by itself, with no web application running in the web app. You might want to do so to scale your WebJob resources independently of your web application resources.

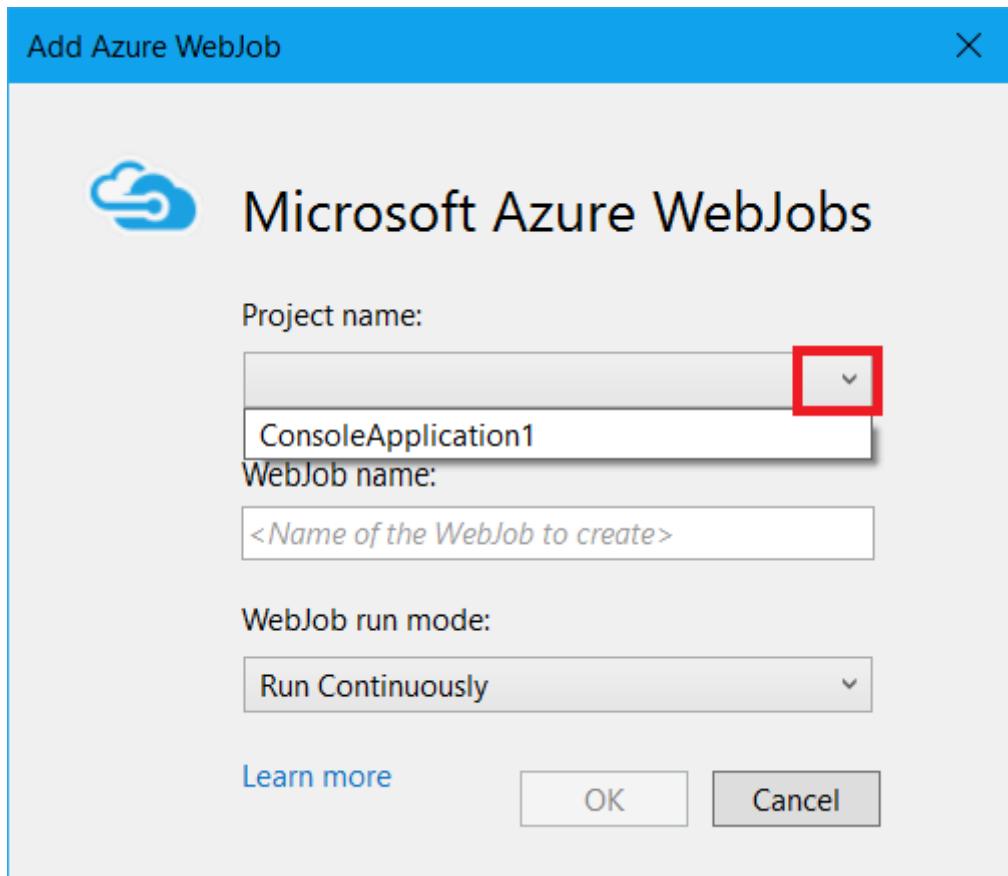
Enable automatic WebJobs deployment with a web project

1. Right-click the web project in **Solution Explorer**, and then select **Add > Existing Project as Azure WebJob**.



The [Add Azure WebJob](#) dialog box appears.

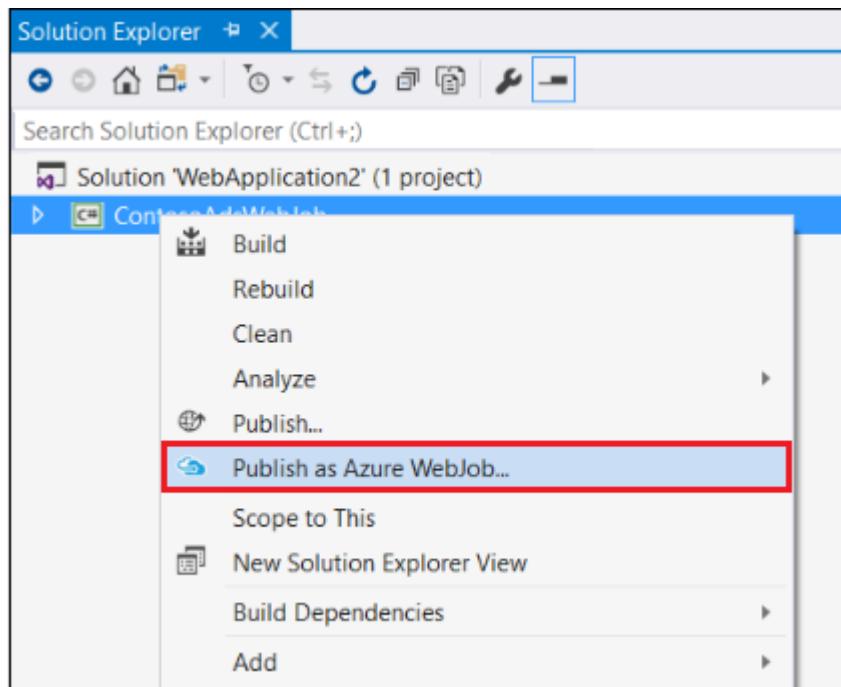
2. In the **Project name** drop-down list, select the console app project to add as a WebJob.



3. Complete the [Add Azure WebJob](#) dialog box, and then select OK.

Enable WebJobs deployment without a web project

1. Right-click the console app project in **Solution Explorer**, and then select **Publish as Azure WebJob**.



The [Add Azure WebJob](#) dialog box appears, with the project selected in the **Project name** box.

2. Complete the [Add Azure WebJob](#) dialog box, and then select **OK**.

The **Publish Web** wizard appears. If you don't want to publish immediately, close the wizard. The settings that you've entered are saved for when you do want to [deploy the project](#).

Create a new WebJobs-enabled project

To create a new WebJobs-enabled project, use the console app project template and enable WebJobs deployment as explained in [the previous section](#). As an alternative, you can use the WebJobs new-project template:

- [Use the WebJobs new-project template for an independent WebJob](#)

Create a project and configure it to deploy by itself as a WebJob, with no link to a web project. Use this option when you want to run a WebJob in a web app by itself, with no web application running in the web app. You might want to do so to scale your WebJob resources independently of your web application resources.

- [Use the WebJobs new-project template for a WebJob linked to a web project](#)

Create a project that is configured to deploy automatically as a WebJob when you deploy a web project in the same solution. Use this option when you want to run your WebJob in the same web app in which you run the related web application.

Note

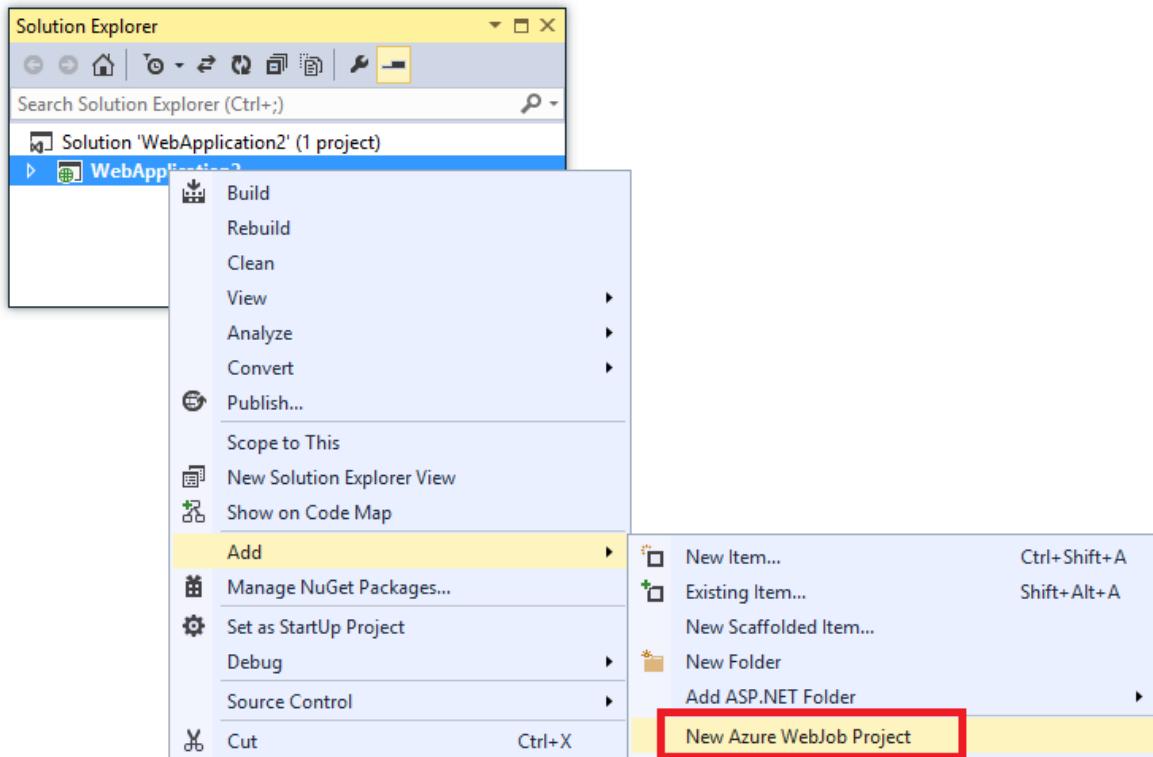
The WebJobs new-project template automatically installs NuGet packages and includes code in *Program.cs* for the **WebJobs SDK**. If you don't want to use the WebJobs SDK, remove or change the `host.RunAndBlock` statement in *Program.cs*.

Use the WebJobs new-project template for an independent WebJob

1. Select **File > New > Project**. In the **Create a new project** dialog box, search for and select **Azure WebJob (.NET Framework)** for C#.
2. Follow the previous directions to [make the console app project an independent WebJobs project](#).

Use the WebJobs new-project template for a WebJob linked to a web project

1. Right-click the web project in Solution Explorer, and then select Add > New Azure WebJob Project.



The [Add Azure WebJob](#) dialog box appears.

2. Complete the [Add Azure WebJob](#) dialog box, and then select OK.

webjob-publish-settings.json file

When you configure a console app for WebJobs deployment, Visual Studio installs the [Microsoft.Web.WebJobs.Publish](#) NuGet package and stores scheduling information in a `webjob-publish-settings.json` file in the project *Properties* folder of the WebJobs project. Here is an example of that file:

```
JSON

{
  "$schema": "http://schemastore.org/schemas/json/webjob-publish-
  settings.json",
  "webJobName": "WebJob1",
  "startTime": "null",
  "endTime": "null",
  "jobRecurrenceFrequency": "null",
  "interval": null,
  "runMode": "Continuous"
}
```

You can edit this file directly, and Visual Studio provides IntelliSense. The file schema is stored at <https://schemastore.org> and can be viewed there.

webjobs-list.json file

When you link a WebJobs-enabled project to a web project, Visual Studio stores the name of the WebJobs project in a *webjobs-list.json* file in the web project's *Properties* folder. The list might contain multiple WebJobs projects, as shown in the following example:

JSON

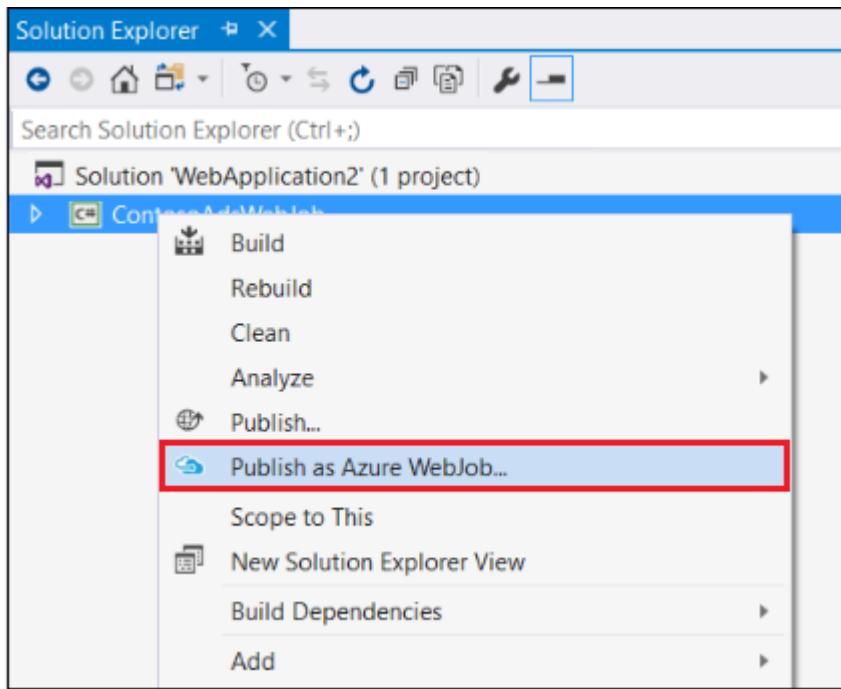
```
{  
  "$schema": "http://schemastore.org/schemas/json/webjobs-list.json",  
  "WebJobs": [  
    {  
      "filePath": "../ConsoleApplication1/ConsoleApplication1.csproj"  
    },  
    {  
      "filePath": "../WebJob1/WebJob1.csproj"  
    }  
  ]  
}
```

You can edit this file directly in Visual Studio, with IntelliSense. The file schema is stored at <https://schemastore.org>.

Deploy a WebJobs project

A WebJobs project that you've linked to a web project deploys automatically with the web project. For information about web project deployment, see [How-to guides > Deploy the app](#) in the left navigation.

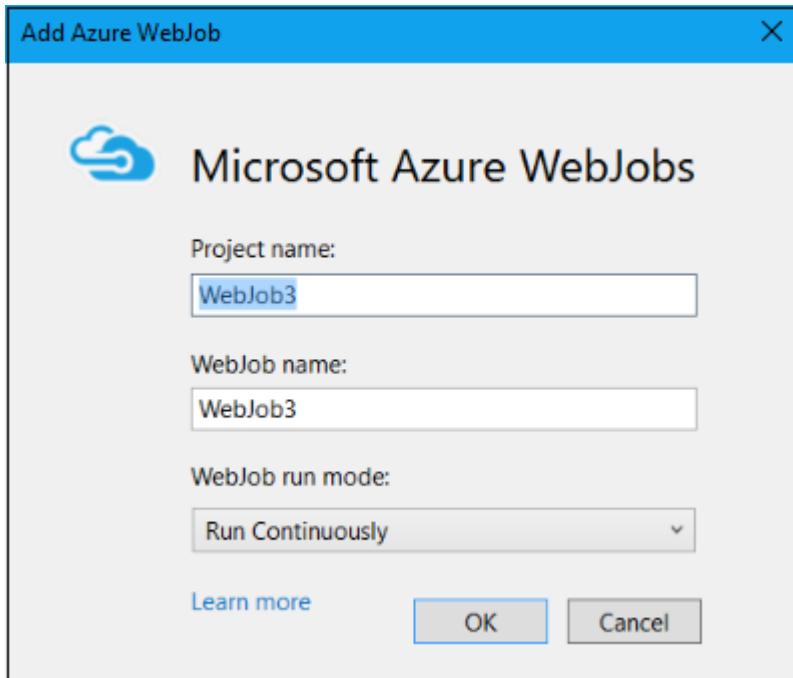
To deploy a WebJobs project by itself, right-click the project in **Solution Explorer** and select **Publish as Azure WebJob**.



For an independent WebJob, the same **Publish Web** wizard that is used for web projects appears, but with fewer settings available to change.

Add Azure WebJob dialog box

The **Add Azure WebJob** dialog box lets you enter the WebJob name and the run mode setting for your WebJob.



Some of the fields in this dialog box correspond to fields on the **Add WebJob** dialog box of the Azure portal. For more information, see [Run background tasks with WebJobs in Azure App Service](#).

WebJob deployment information:

- For information about command-line deployment, see [Enabling Command-line or Continuous Delivery of Azure WebJobs](#).
- If you deploy a WebJob, and then decide you want to change the type of WebJob and redeploy, delete the `webjobs-publish-settings.json` file. Doing so causes Visual Studio to redisplay the publishing options, so you can change the type of WebJob.
- If you deploy a WebJob and later change the run mode from continuous to non-continuous or vice versa, Visual Studio creates a new WebJob in Azure when you redeploy. If you change other scheduling settings, but leave run mode the same or switch between Scheduled and On Demand, Visual Studio updates the existing job instead of creating a new one.

WebJob types

The type of a WebJob can be either *triggered* or *continuous*:

- Triggered (default): A triggered WebJob starts based on a binding event, on a [schedule](#), or when you trigger it manually (on demand). It runs on a single instance that the web app runs on.
- Continuous: A [continuous](#) WebJob starts immediately when the WebJob is created. It runs on all web app scaled instances by default but can be configured to run as a single instance via `settings.job`.

Note

A web app can time out after 20 minutes of inactivity, and only requests to the actual web app can reset the timer. Viewing the app's configuration in the Azure portal or making requests to the advanced tools site (`https://<app_name>.scm.azurewebsites.net`) doesn't reset the timer. If you set the web app that hosts your job to run continuously, run on a schedule, or use event-driven triggers, enable the **Always on** setting on your web app's Azure Configuration page. The Always on setting helps to make sure that these kinds of WebJobs run reliably. This feature is available only in the Basic, Standard, and Premium pricing tiers.

Scheduling a triggered WebJob

When you publish a console app to Azure, Visual Studio sets the type of WebJob to **Triggered** by default, and adds a new `settings.job` file to the project. For triggered

WebJob types, you can use this file to set an execution schedule for your WebJob.

Use the *settings.job* file to set an execution schedule for your WebJob. The following example runs every hour from 9 AM to 5 PM:

JSON

```
{  
    "schedule": "0 0 9-17 * * *"  
}
```

This file is located at the root of the WebJobs folder with your WebJob's script, such as `wwwroot\app_data\jobs\triggered\{job_name}` or `wwwroot\app_data\jobs\continuous\{job_name}`. When you deploy a WebJob from Visual Studio, mark your *settings.job* file properties in Visual Studio as **Copy if newer**.

If you [create a WebJob from the Azure portal](#), the *settings.job* file is created for you.

CRON expressions

WebJobs uses the same CRON expressions for scheduling as the timer trigger in Azure Functions. To learn more about CRON support, see [Timer trigger for Azure Functions](#).

ⓘ Note

The default time zone used to run CRON expressions is Coordinated Universal Time (UTC). To have your CRON expression run based on another time zone, create an app setting for your function app named WEBSITE_TIME_ZONE. To learn more, see [NCrontab time zones](#).

settings.job reference

The following settings are supported by WebJobs:

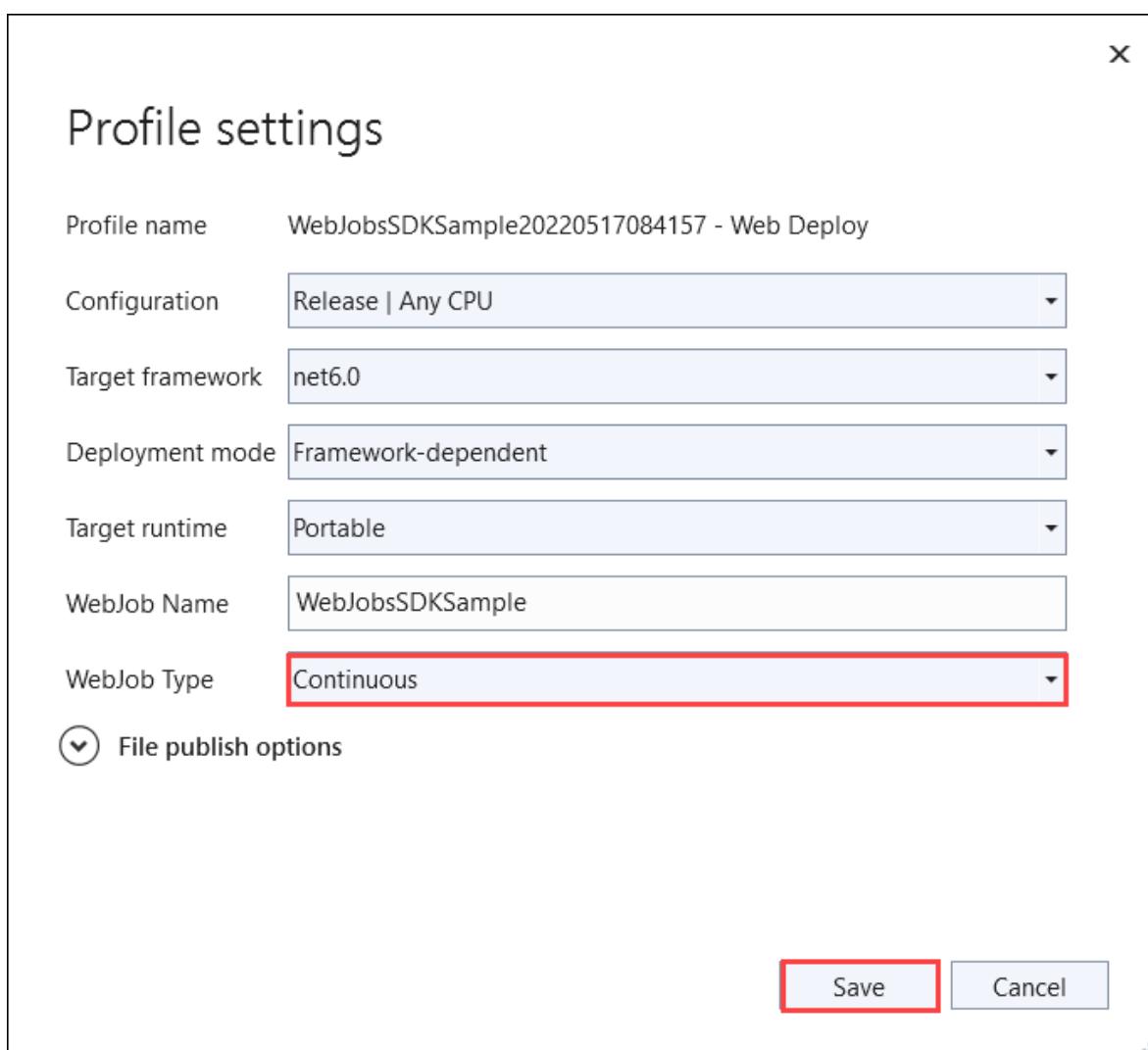
Setting	Type	Description
<code>is_in_place</code>	All	Allows the WebJob to run in place without first being copied to a temporary folder. For more information, see WebJob working directory .
<code>is_singleton</code>	Continuous	Only run the WebJob on a single instance when scaled out. For more information, see Set a continuous job as singleton .

Setting	Type	Description
<code>schedule</code>	Triggered	Run the WebJob on a CRON-based schedule. For more information, see NCRONTAB expressions .
<code>stopping_wait_time</code>	All	Allows control of the shutdown behavior. For more information, see Graceful shutdown .

Continuous execution

If you enable **Always on** in Azure, you can use Visual Studio to change the WebJob to run continuously:

1. If you haven't already done so, [publish the project to Azure](#).
2. In **Solution Explorer**, right-click the project and select **Publish**.
3. In the **Settings** section, choose **Show all settings**.
4. In the **Profile settings** dialog box, choose **Continuous** for **WebJob Type**, and then choose **Save**.



5. Select **Publish** in the **Publish** tab to republish the WebJob with the updated settings.

Next steps

[Learn more about the WebJobs SDK](#)

Tutorial: Get started with the Azure WebJobs SDK for event-driven background processing

Article • 01/17/2025

Get started with the Azure WebJobs SDK for Azure App Service to enable your web apps to run background tasks, scheduled tasks, and respond to events.

Use Visual Studio 2022 to create a .NET 8 console app that uses the WebJobs SDK to respond to Azure Storage Queue messages, run the project locally, and finally deploy it to Azure.

In this tutorial, you will learn how to:

- ✓ Create a console app
- ✓ Add a function
- ✓ Test locally
- ✓ Deploy to Azure
- ✓ Enable Application Insights logging
- ✓ Add input/output bindings

Prerequisites

- Visual Studio 2022 with the **Azure development** workload. [Install Visual Studio 2022](#).
- An Azure account with an active subscription. [Create an account for free ↗](#).

Create a console app

In this section, you start by creating a project in Visual Studio 2022. Next, you'll add tools for Azure development, code publishing, and functions that listen for triggers and call functions. Last, you'll set up console logging that disables a legacy monitoring tool and enables a console provider with default filtering.

Note

The procedures in this article are verified for creating a C# console app that runs on .NET 8.0.

Create a project

1. In Visual Studio, select **File > New > Project**.
2. Under **Create a new project**, select **Console Application (C#)**, and then select **Next**.
3. Under **Configure your new project**, name the project *WebJobsSDKSample*, and then select **Next**.
4. Choose your **Target framework** and select **Create**. This tutorial has been verified using .NET 6.0.

Install WebJobs NuGet packages

Install the latest WebJobs NuGet package. This package includes Microsoft.Azure.WebJobs (WebJobs SDK), which lets you publish your function code to WebJobs in Azure App Service.

1. Get the latest stable 4.x version of the [Microsoft.Azure.WebJobs.Extensions NuGet package](#).
2. In Visual Studio, go to **Tools > NuGet Package Manager**.
3. Select **Package Manager Console**. You'll see a list of NuGet cmdlets, a link to documentation, and a **PM>** entry point.
4. In the following command, replace `<4_X_VERSION>` with the current version number you found in step 1.

PowerShell

```
Install-Package Microsoft.Azure.WebJobs.Extensions -version  
<4_X_VERSION>
```

ⓘ Note

The sample code in this article works with package versions 4.x. Make sure you use a 4.x version because you get build errors when using package versions 5.x.

5. In the **Package Manager Console**, execute the command. The extension list appears and automatically installs.

Create the Host

The host is the runtime container for functions that listens for triggers and calls functions. The following steps create a host that implements [IHost](#), which is the Generic Host in ASP.NET Core.

1. Select the **Program.cs** tab, remove the existing contents, and add these `using` statements:

```
cs

using System.Threading.Tasks;
using Microsoft.Extensions.Hosting;
```

2. Also under **Program.cs**, add the following code:

```
cs

namespace WebJobsSDKSample
{
    class Program
    {
        static async Task Main()
        {
            var builder = new HostBuilder();
            builder.ConfigureWebJobs(b =>
            {
                b.AddAzureStorageCoreServices();
            });
            var host = builder.Build();
            using (host)
            {
                await host.RunAsync();
            }
        }
    }
}
```

In ASP.NET Core, host configurations are set by calling methods on the [HostBuilder](#) instance. For more information, see [.NET Generic Host](#). The `ConfigureWebJobs` extension method initializes the WebJobs host. In `ConfigureWebJobs`, initialize specific binding extensions, such as the Storage binding extension, and set properties of those extensions.

Enable console logging

Set up console logging that uses the [ASP.NET Core logging framework](#). This framework, `Microsoft.Extensions.Logging`, includes an API that works with a variety of built-in and third-party logging providers.

1. Get the latest stable version of the [Microsoft.Extensions.Logging.Console NuGet package](#), which includes `Microsoft.Extensions.Logging`.
2. In the following command, replace `<9_X_VERSION>` with the current version number you found in step 1. Each type of NuGet Package has a unique version number.

PowerShell

```
Install-Package Microsoft.Extensions.Logging.Console -version  
<9_X_VERSION>
```

3. In the **Package Manager Console**, fill in the current version number and execute the command. The extension list appears and automatically installs.
4. Under the tab **Program.cs**, add this `using` statement:

CS

```
using Microsoft.Extensions.Logging;
```

5. Continuing under **Program.cs**, add the `ConfigureLogging` method to `HostBuilder`, before the `Build` command. The `AddConsole` method adds console logging to the configuration.

CS

```
builder.ConfigureLogging((context, b) =>  
{  
    b.SetMinimumLevel(LogLevel.Error);  
    b.AddFilter("Function", LogLevel.Information);  
    b.AddFilter("Host", LogLevel.Debug);
```

```
b.AddConsole();  
});
```

This adds logging that captures log output for function executions at the `Information` level, the host at the `Debug` level, and the `Error` level for all other components. The `Main` method now looks like this:

```
cs  
  
static async Task Main()  
{  
    var builder = new HostBuilder();  
    builder.ConfigureWebJobs(b =>  
    {  
        b.AddAzureStorageCoreServices();  
    });  
    builder.ConfigureLogging((context, b) =>  
    {  
        b.SetMinimumLevel(LogLevel.Error);  
        b.AddFilter("Function", LogLevel.Information);  
        b.AddFilter("Host", LogLevel.Debug);  
        b.AddConsole();  
    });  
    var host = builder.Build();  
    using (host)  
    {  
        await host.RunAsync();  
    }  
}
```

This addition makes these changes:

- Disables [dashboard logging](#). The dashboard is a legacy monitoring tool, and dashboard logging is not recommended for high-throughput production scenarios.
- Adds the console provider with default [filtering](#).

Now, you can add a function that is triggered by messages arriving in an Azure Storage queue.

Add a function

A function is unit of code that runs on a schedule, is triggered based on events, or is run on demand. A trigger listens to a service event. In the context of the WebJobs SDK, triggered doesn't refer to the deployment mode. Event-driven or scheduled WebJobs

created using the SDK should always be deployed as continuous WebJobs with "Always on" enabled.

In this section, you create a function triggered by messages in an Azure Storage queue. First, you need to add a binding extension to connect to Azure Storage.

Install the Storage binding extension

Starting with version 3 of the WebJobs SDK, to connect to Azure Storage services you must install a separate Storage binding extension package.

ⓘ Note

Beginning with 5.x, `Microsoft.Azure.WebJobs.Extensions.Storage` has been [split by storage service](#) and has migrated the `AddAzureStorage()` extension method by service type. This version also requires you to update the version of the `Microsoft.Azure.WebJobs.Host.Storage` assembly used by the SDK.

1. Get the latest stable version of the [Microsoft.Azure.WebJobs.Extensions.Storage](#) NuGet package, version 5.x.
2. In the following command, replace `<5_X_VERSION>` with the current version number you found in step 1. Each type of NuGet Package has a unique version number.

PowerShell

```
Install-Package Microsoft.Azure.WebJobs.Extensions.Storage -Version <5_X_VERSION>
```

3. In the **Package Manager Console**, execute the command with the current version number at the `PM>` entry point.
4. Also run this command to update the `Microsoft.Azure.WebJobs.Host.Storage` package to version 4.1.0:

PowerShell

```
Install-Package Microsoft.Azure.WebJobs.Host.Storage -Version 4.1.0
```

5. Continuing in `Program.cs`, in the `ConfigureWebJobs` extension method, add the `AddAzureStorageQueues` method on the `HostBuilder` instance (before the `Build`

command) to initialize the Storage extension. At this point, the `ConfigureWebJobs` method looks like this:

```
cs

builder.ConfigureWebJobs(b =>
{
    b.AddAzureStorageCoreServices();
    b.AddAzureStorageQueues();
});
```

6. Add the following code in the `Main` method after the `builder` is instantiated:

```
C#

builder.UseEnvironment(EnvironmentName.Development);
```

Running in [development mode](#) reduces the [queue polling exponential backoff](#) that can significantly delay the amount of time it takes for the runtime to find the message and invoke the function. You should remove this line of code or switch to [Production](#) when you're done with development and testing.

The `Main` method should now look like the following example:

```
C#  
  
static async Task Main()
{
    var builder = new HostBuilder();
    builder.UseEnvironment(EnvironmentName.Development);
    builder.ConfigureWebJobs(b =>
    {
        b.AddAzureStorageCoreServices();
        b.AddAzureStorageQueues();
    });
    builder.ConfigureLogging((context, b) =>
    {
        b.SetMinimumLevel(LogLevel.Error);
        b.AddFilter("Function", LogLevel.Information);
        b.AddFilter("Host", LogLevel.Debug);
        b.AddConsole();
    });
    var host = builder.Build();
    using (host)
    {
        await host.RunAsync();
    }
}
```

Create a queue triggered function

The `QueueTrigger` attribute tells the runtime to call this function when a new message is written on an Azure Storage queue called `queue`. The contents of the queue message are provided to the method code in the `message` parameter. The body of the method is where you process the trigger data. In this example, the code just logs the message.

1. In Solution Explorer, right-click the project, select **Add > New Item**, and then select **Class**.
2. Name the new C# class file *Functions.cs* and select **Add**.
3. In *Functions.cs*, replace the generated template with the following code:

```
cs

using Microsoft.Azure.WebJobs;
using Microsoft.Extensions.Logging;

namespace WebJobsSDKSample
{
    public static class Functions
    {
        public static void ProcessQueueMessage([QueueTrigger("queue")]
string message, ILogger logger)
        {
            logger.LogInformation(message);
        }
    }
}
```

You should mark the *Functions* class as `public static` in order for the runtime to access and execute the method. In the above code sample, when a message is added to a queue named `queue`, the function executes and the `message` string is written to the logs. The queue being monitored is in the default Azure Storage account, which you create next.

The `message` parameter doesn't have to be a string. You can also bind to a JSON object, a byte array, or a `CloudQueueMessage` object. [See Queue trigger usage](#). Each binding type (such as queues, blobs, or tables) has a different set of parameter types that you can bind to.

Create an Azure storage account

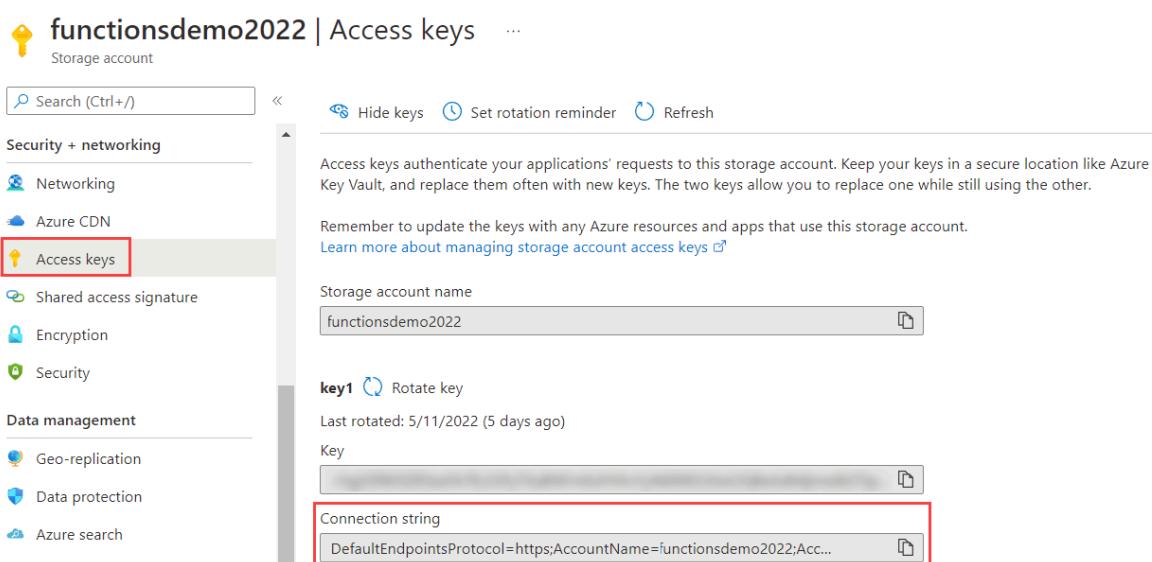
The Azure Storage Emulator that runs locally doesn't have all of the features that the WebJobs SDK needs. You'll create a storage account in Azure and configure the project to use it.

To learn how to create a general-purpose v2 storage account, see [Create an Azure Storage account](#).

Locate and copy your connection string

A connection string is required to configure storage. Keep this connection string for the next steps.

1. In the [Azure portal](#), navigate to your storage account and select **Settings**.
2. In **Settings**, select **Access keys**.
3. For the **Connection string** under **key1**, select the **Copy to clipboard** icon.



Configure storage to run locally

The WebJobs SDK looks for the storage connection string in the Application Settings in Azure. When you run locally, it looks for this value in the local configuration file or in environment variables.

1. Right-click the project, select **Add > New Item**, select **JavaScript JSON configuration file**, name the new file *appsettings.json* file, and select **Add**.
2. In the new file, add a `AzureWebJobsStorage` field, as in the following example:

```
JSON
```

```
{  
    "AzureWebJobsStorage": "{storage connection string}"  
}
```

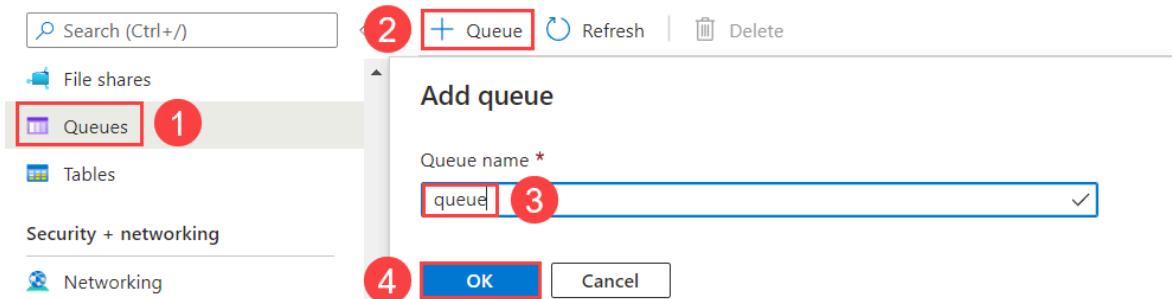
3. Replace *{storage connection string}* with the connection string that you copied previously.
4. Select the *appsettings.json* file in Solution Explorer and in the **Properties** window, set the **Copy to Output Directory** action to **Copy if newer**.

Because this file contains a connection string secret, you shouldn't store the file in a remote code repository. After publishing your project to Azure, you can add the same connection string app setting in your app in Azure App Service.

Test locally

Build and run the project locally and create a message queue to trigger the function.

1. In the Azure portal, navigate to your storage account and select the **Queues** tab (1). Select **+ Queue** (2) and enter **queue** as the Queue name (3). Then, select **OK** (4).



2. Click the new queue and select **Add message**.
3. In the **Add Message** dialog, enter *Hello World!* as the **Message text**, and then select **OK**. There is now a message in the queue.

Add message to queue

Message text *

✓

Expires in:

Days ▼

Message never expires

Encode the message body in Base64 ⓘ

OK

Cancel

4. Press **Ctrl+F5** to run the project.

The console shows that the runtime found your function. Because you used the `QueueTrigger` attribute in the `ProcessQueueMessage` function, the WebJobs runtime listens for messages in the queue named `queue`. When it finds a new message in this queue, the runtime calls the function, passing in the message string value.

5. Go back to the **Queue** window and refresh it. The message is gone, since it has been processed by your function running locally.
6. Close the console window or type **Ctrl+C**.

It's now time to publish your WebJobs SDK project to Azure.

Deploy to Azure

During deployment, you create an app service instance where you'll run your functions. When you publish a .NET console app to App Service in Azure, it automatically runs as a WebJob. To learn more about publishing, see [Develop and deploy WebJobs using Visual Studio](#).

Create Azure resources

1. In **Solution Explorer**, right-click the project and select **Publish**.
2. In the **Publish** dialog box, select **Azure** for **Target**, and then select **Next**.
3. Select **Azure WebJobs** for **Specific target**, and then select **Next**.
4. Above **App Service instances** select the plus (+) button to **Create a new Azure WebJob**.
5. In the **App Service (Windows)** dialog box, use the hosting settings in the following table.

[+] Expand table

Setting	Suggested value	Description
Name	Globally unique name	Name that uniquely identifies your new function app.
Subscription	Choose your subscription	The Azure subscription to use.
Resource group	myResourceGroup	Name of the resource group in which to create your function app. Choose New to create a new resource group.
Hosting Plan	App Service plan	An App Service plan specifies the location, size, and features of the web server farm that hosts your app. You can save money when hosting multiple apps by configuring the web apps to share a single App Service plan. App Service plans define the region, instance size, scale count, and SKU (Free, Shared, Basic, Standard, or Premium). Choose New to create a new App Service plan. Free and Basic tiers don't support the Always On option to keep your site running continuously.

The screenshot shows the 'App Service (Windows)' creation dialog. At the top left is a cloud icon with a square overlay. To its right is the text 'App Service (Windows)' and a 'Create new' link. On the right side, there's a user profile icon for 'Contoso' with the email 'user@contoso.com'. Below these are four input fields: 'Name' containing 'WebJobsApp', 'Subscription' set to 'Vendor Subscription', 'Resource group' set to 'myResourceGroup (Central US)', and 'Hosting Plan' set to 'WebJobsApp* (Central US, F1)'. At the bottom are three buttons: 'Export...', 'Create' (which is highlighted in blue), and 'Cancel'.

6. Select **Create** to create a WebJob and related resources in Azure with these settings and deploy your project code.
7. Select **Finish** to return to the **Publish** page.

Enable Always On

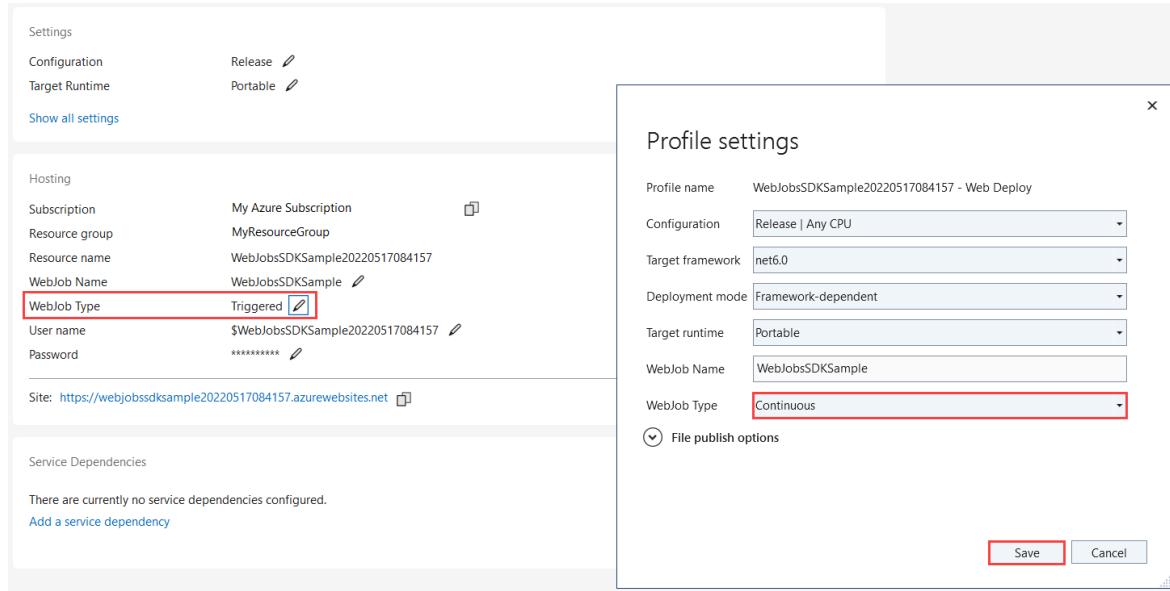
For a continuous WebJob, you should enable the Always on setting in the site so that your WebJobs run correctly. If you don't enable Always on, the runtime goes idle after a few minutes of inactivity.

1. In the **Publish** page, select the three dots above **Hosting** to show **Hosting profile** section actions and choose **Open in Azure portal**.
2. Under **Settings**, choose **Configuration > General settings**, set **Always on** to **On**, and then select **Save** and **Continue** to restart the site.

Publish the project

With the web app created in Azure, it's time to publish the WebJobs project.

1. In the **Publish** page under **Hosting**, select the edit button and change the **WebJob Type** to **Continuous** and select **Save**. This makes sure that the WebJob is running when messages are added to the queue. Triggered WebJobs are typically used only for manual webhooks.



2. Select the **Publish** button at the top right corner of the **Publish** page. When the operation completes, your WebJob is running on Azure.

Create a storage connection app setting

You need to create the same storage connection string setting in Azure that you used locally in your appsettings.json config file. This lets you more securely store the connection string and

1. In your **Publish** profile page, select the three dots above **Hosting** to show **Hosting profile section actions** and choose **Manage Azure App Service settings**.
2. In **Application settings**, choose **+ Add setting**.
3. In **New app setting name**, type **AzureWebJobsStorage** and select **OK**.
4. In **Remote**, paste in the connection string from your local setting and select **OK**.

The connection string is now set in your app in Azure.

Trigger the function in Azure

1. Make sure you're not running locally. Close the console window if it's still open. Otherwise, the local instance might be the first to process any queue messages you create.
2. In the **Queue** page in Visual Studio, add a message to the queue as before.
3. Refresh the **Queue** page, and the new message disappears because it has been processed by the function running in Azure.

Enable Application Insights logging

When the WebJob runs in Azure, you can't monitor function execution by viewing console output. To be able to monitor your WebJob, you should create an associated [Application Insights](#) instance when you publish your project.

Create an Application Insights instance

1. In your **Publish** profile page, select the three dots above **Hosting** to show **Hosting profile** section actions and choose **Open in Azure Portal**.
2. In the web app under **Monitoring**, choose **Application Insights**, and select **Turn on Application Insights**.
3. Verify the generated **Resource name** for the instance and the **Location**, and select **Apply** and then **Yes**.
4. Under **Settings**, choose **Environment variables** and verify that a new `APPINSIGHTS_INSTRUMENTATIONKEY` was created. This key is used to connect your WebJob instance to Application Insights.

To take advantage of [Application Insights](#) logging, you need to update your logging code as well.

Install the Application Insights extension

1. Get the latest stable version of the [Microsoft.Azure.WebJobs.Logging.ApplicationInsights](#) NuGet package, version 3.x.
2. In the following command, replace `<3_X_VERSION>` with the current version number you found in step 1. Each type of NuGet Package has a unique version number.

PowerShell

```
Install-Package Microsoft.Azure.WebJobs.Logging.ApplicationInsights -  
Version <3_X_VERSION>
```

3. In the **Package Manager Console**, execute the command with the current version number at the `PM>` entry point.

Initialize the Application Insights logging provider

Open `Program.cs` and add the following initializer in the `ConfigureLogging` after the call to `AddConsole`:

C#

```
// If the key exists in settings, use it to enable Application Insights.  
string instrumentationKey =  
    context.Configuration["APPINSIGHTS_INSTRUMENTATIONKEY"];  
if (!string.IsNullOrEmpty(instrumentationKey))  
{  
    b.AddApplicationInsightsWebJobs(o => o.InstrumentationKey =  
        instrumentationKey);  
}
```

The `Main` method code should now look like the following example:

C#

```
static async Task Main()  
{  
    var builder = new HostBuilder();  
    //builder.UseEnvironment(EnvironmentName.Development);  
    builder.ConfigureWebJobs(b =>  
    {  
        b.AddAzureStorageCoreServices();  
        b.AddAzureStorageQueues();  
    });  
    builder.ConfigureLogging((context, b) =>  
    {  
        b.SetMinimumLevel(LogLevel.Error);  
        b.AddFilter("Function", LogLevel.Information);  
        b.AddFilter("Host", LogLevel.Debug);  
        b.AddConsole();  
  
        // If the key exists in settings, use it to enable Application  
        // Insights.  
        string? instrumentationKey =  
            context.Configuration["APPINSIGHTS_INSTRUMENTATIONKEY"];  
        if (!string.IsNullOrEmpty(instrumentationKey))  
        {
```

```
        b.AddApplicationInsightsWebJobs(o => o.InstrumentationKey = instrumentationKey);
    }
});
var host = builder.Build();
using (host)
{
    await host.RunAsync();
}
}
```

This initializes the Application Insights logging provider with default [filtering](#). When running locally, all Information and higher-level logs are written to both the console and Application Insights. When running locally, Application Insights logging is only supported after you also add the `APPINSIGHTS_INSTRUMENTATIONKEY` to the `appsetting.json` file in the project.

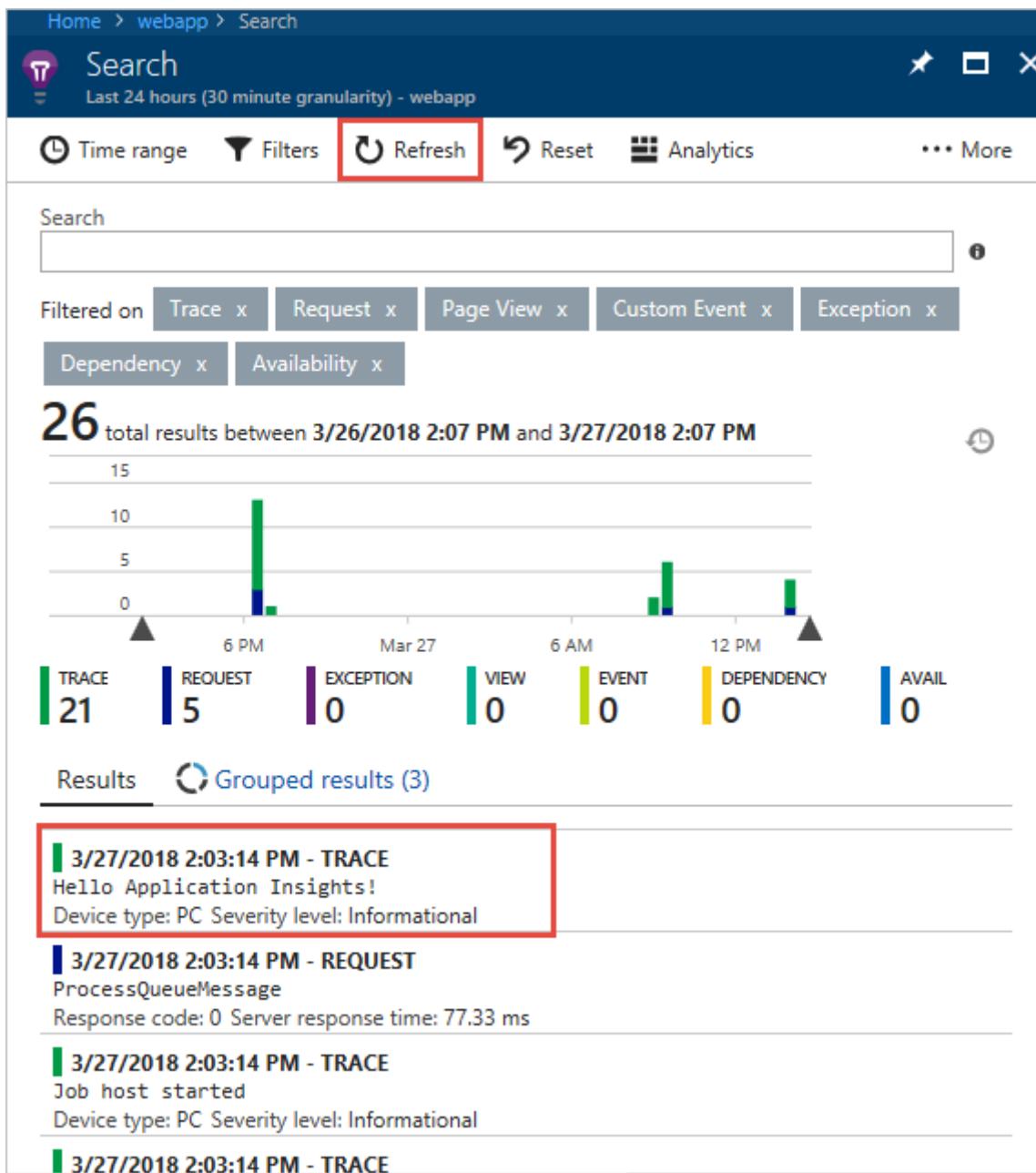
Republish the project and trigger the function again

1. In **Solution Explorer**, right-click the project and select **Publish**.
2. As before, use the Azure portal to create a queue message like you did [earlier](#), except enter *Hello App Insights!* as the message text.
3. In your **Publish** profile page, select the three dots above **Hosting** to show **Hosting profile section actions** and choose **Open in Azure Portal**.
4. In the web app under **Settings > Monitor**, choose **Application Insights**, and select **View Application Insights data**.
5. Select **Search** and then select **See all data in the last 24 hours**.

The screenshot shows the Azure Application Insights portal for a resource group named 'webapp'. The 'Essentials' section displays basic information about the application:

Setting	Value
Resource group (change)	webapp
Location	West US 2
Subscription name (change)	Windows Azure MSDN - Visual Studio Ultima...
Subscription ID	{subscription ID}
Type	ASP.NET
Instrumentation Key	aaaaaaaa-0b0b-1c1c-2d2d-333333333333

6. If you don't see the *Hello App Insights!* message, select **Refresh** periodically for several minutes. Logs don't appear immediately, because it takes a while for the Application Insights client to flush the logs it processes.



Add input/output bindings

Bindings simplify code that reads and writes data. Input bindings simplify code that reads data. Output bindings simplify code that writes data.

Add bindings

Input bindings simplify code that reads data. For this example, the queue message is the name of a blob, which you'll use to find and read a blob in Azure Storage. You will then use output bindings to write a copy of the file to the same container.

1. In `Functions.cs`, add a `using`:

```
CS
```

```
using System.IO;
```

2. Replace the `ProcessQueueMessage` method with the following code:

```
cs
```

```
public static void ProcessQueueMessage(
    [QueueTrigger("queue")] string message,
    [Blob("container/{queueTrigger}", FileAccess.Read)] Stream myBlob,
    [Blob("container/copy-{queueTrigger}", FileAccess.Write)] Stream
    outputBlob,
    ILogger logger)
{
    logger.LogInformation($"Blob name:{message} \n Size:
{myBlob.Length} bytes");
    myBlob.CopyTo(outputBlob);
}
```

In this code, `queueTrigger` is a [binding expression](#), which means it resolves to a different value at runtime. At runtime, it has the contents of the queue message.

This code uses output bindings to create a copy of the file identified by the queue message. The file copy is prefixed with `copy-`.

3. In `Program.cs`, in the `ConfigureWebJobs` extension method, add the `AddAzureStorageBlobs` method on the `HostBuilder` instance (before the `Build` command) to initialize the Storage extension. At this point, the `ConfigureWebJobs` method looks like this:

```
cs
```

```
builder.ConfigureWebJobs(b =>
{
    b.AddAzureStorageCoreServices();
    b.AddAzureStorageQueues();
    b.AddAzureStorageBlobs();
});
```

4. Create a blob container in your storage account.

- In the Azure portal, navigate to the **Containers** tab below **Data storage** and select **+ Container**
- In the **New container** dialog, enter `container` as the container name, and then select **Create**.

5. Upload the *Program.cs* file to the blob container. (This file is used here as an example; you could upload any text file and create a queue message with the file's name.)
 - a. Select the new container you created
 - b. Select the **Upload** button.

The screenshot shows the Azure Storage Container Overview page. At the top, there is a search bar labeled "Search (Ctrl+ /)" and an "Upload" button, which is highlighted with a red box. To the right of the upload button is a "Change access level" link. Below the search bar, there are three navigation links: "Overview" (which is selected and highlighted in grey), "Diagnose and solve problems", and "Access Control (IAM)". On the right side, there is a section titled "Authentication method: Access key (Sw)" and "Location: container". Below this, there is a search bar labeled "Search blobs by prefix (case-sensitive)". Under the "Settings" heading, there is a link to "Shared access tokens". On the far right, there is a table with one row, showing "Name" and "No results".

- c. Find and select *Program.cs*, and then select **OK**.

Republish the project

1. In **Solution Explorer**, right-click the project and select **Publish**.
2. In the **Publish** dialog, make sure that the current profile is selected and then select **Publish**. Results of the publish are detailed in the **Output** window.
3. Create a queue message in the queue you created earlier, with *Program.cs* as the text of the message.

Add message to queue

Message text *

Program.cs



Expires in: *

7

Days



Message never expires



Encode the message body in Base64 (i)

OK

Cancel

4. A copy of the file, *copy-Program.cs*, will appear in the blob container.

Next steps

This tutorial showed you how to create, run, and deploy a WebJobs SDK 3.x project.

[Learn more about the WebJobs SDK](#)

Feedback

Was this page helpful?

Yes

No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

How to use the Azure WebJobs SDK for event-driven background processing

Article • 06/06/2024

This article provides guidance on how to work with the Azure WebJobs SDK. To get started with WebJobs right away, see [Get started with the Azure WebJobs SDK](#).

WebJobs SDK versions

These are the key differences between version 3.x and version 2.x of the WebJobs SDK:

- Version 3.x adds support for .NET Core.
- In version 3.x, you'll install the Storage binding extension required by the WebJobs SDK. In version 2.x, the Storage bindings are included in the SDK.
- Visual Studio 2019 tooling for .NET Core (3.x) projects differs from tooling for .NET Framework (2.x) projects. To learn more, see [Develop and deploy WebJobs using Visual Studio - Azure App Service](#).

Several descriptions in this article provide examples for both WebJobs version 3.x and WebJobs version 2.x.

[Azure Functions](#) is built on the WebJobs SDK.

- Azure Functions version 2.x is built on WebJobs SDK version 3.x.
- Azure Functions version 1.x is built on WebJobs SDK version 2.x.

Source code repositories for both Azure Functions and WebJobs SDK use the WebJobs SDK numbering. Several sections of this how-to article link to Azure Functions documentation.

For more information, see [Compare the WebJobs SDK and Azure Functions](#)

WebJobs host

The host is a runtime container for functions. The Host listens for triggers and calls functions. In version 3.x, the host is an implementation of `IHost`. In version 2.x, you use the `JobHost` object. You create a host instance in your code and write code to customize its behavior.

This is a key difference between using the WebJobs SDK directly and using it indirectly through Azure Functions. In Azure Functions, the service controls the host, and you can't customize the host by writing code. Azure Functions lets you customize host behavior through settings in the `host.json` file. Those settings are strings, not code, and use of these strings limits the kinds of customizations you can do.

Host connections

The WebJobs SDK looks for Azure Storage and Azure Service Bus connections in the local.settings.json file when you run locally or in the environment of the WebJob when you run in Azure. By default, the WebJobs SDK requires a storage connection with the name `AzureWebJobsStorage`.

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 depend on the service to which you connect. However, a connection name can also refer to a collection of multiple configuration items, useful for configuring identity-based connections. 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 an Azure Blob trigger definition might be `Storage1`. As long as there's no single string value configured by an environment variable named `Storage1`, an environment variable named `Storage1__blobServiceUri` could be used to inform the `blobServiceUri` property of the connection. The connection properties are different for each service. Refer to the documentation for the component that uses the connection.

Identity-based connections

To use identity-based connections in the WebJobs SDK, make sure you are using the latest versions of WebJobs packages in your project. You should also ensure you have a reference to [Microsoft.Azure.WebJobs.Host.Storage](#). When setting up WebJobs within your HostBuilder, make sure to include a call to `AddAzureStorageCoreServices`, as this is what allows

`AzureWebJobsStorage` and other Storage triggers and bindings to use identity:

C#

```
builder.ConfigureWebJobs(b =>
{
    b.AddAzureStorageCoreServices();
    // other configurations...
});
```

Then, you can configure the `AzureWebJobsStorage` connection by setting environment variables (or Application Settings when hosted in App Service):

 Expand table

Environment variable	Description	Example value
<code>AzureWebJobsStorage__blobServiceUri</code>	The data plane URI	<a href="https://<storage_account_name>.blob.core.windows.net">https://<storage_account_name>.blob.core.windows.net

Environment variable	Description	Example value
	of the blob service of the storage account, using the HTTPS scheme.	
AzureWebJobsStorage__queueServiceUri	The data plane URI of the queue service of the storage account, using the HTTPS scheme.	https://<storage_account_name>.queue.core.windows.net

If you provide your configuration through any means other than environment variables, such as with an `appsettings.json`, you would instead need to provide structured configuration for the connection and its properties:

JSON

```
{
  "AzureWebJobsStorage": {
    "blobServiceUri": "https://<storage_account_name>.blob.core.windows.net",
    "queueServiceUri": "https://<storage_account_name>.queue.core.windows.net"
  }
}
```

You may omit the `queueServiceUri` property if you do not plan to use blob triggers.

When your code is run locally, this will default to using your developer identity per the behavior described for `DefaultAzureCredential`.

When your code is hosted in Azure App Service, the configuration shown above will default to the [system-assigned managed identity](#) for the resource. To instead use a [user-assigned identity](#) which has been assigned to the app, you need to add additional properties for your connection that specify which identity should be used. The `credential` property (`AzureWebJobsStorage__credential` as an environment variable) should be set to the string `"managedidentity"`. The `clientId` property (`AzureWebJobsStorage__clientId` as an environment variable) should be set to the client ID of the user-assigned managed identity to be used. As structured configuration, the complete object would be:

JSON

```
{
  "AzureWebJobsStorage": {
    "blobServiceUri": "https://<storage_account_name>.blob.core.windows.net",
    "queueServiceUri": "https://<storage_account_name>.queue.core.windows.net",
    "credential": "managedidentity",
    "clientId": "<user-assigned-identity-client-id>"
  }
}
```

The identity used for `AzureWebJobsStorage` should have role assignments granting it the [Storage Blob Data Owner](#), [Storage Queue Data Contributor](#), and [Storage Account Contributor](#) roles. You may omit both [Storage Queue Data Contributor](#) and [Storage Account Contributor](#) if you do not plan to use blob triggers.

The following table shows built-in roles that are recommended when using triggers in bindings in normal operation. Your application may require further permissions based on the code you write.

[] [Expand table](#)

Binding	Example built-in roles
Blob trigger	Storage Blob Data Owner and Storage Queue Data Contributor See above for requirements on <code>AzureWebJobsStorage</code> as well.
Blob (input)	Storage Blob Data Reader
Blob (output)	Storage Blob Data Owner
Queue trigger	Storage Queue Data Reader , Storage Queue Data Message Processor
Queue (output)	Storage Queue Data Contributor , Storage Queue Data Message Sender
Service Bus trigger ¹	Azure Service Bus Data Receiver , Azure Service Bus Data Owner
Service Bus (output)	Azure Service Bus Data Sender

¹ For triggering from Service Bus topics, the role assignment needs to have effective scope over the Service Bus subscription resource. If only the topic is included, an error will occur. Some clients, such as the Azure portal, don't expose the Service Bus subscription resource as a scope for role assignment. In such cases, the Azure CLI may be used instead. To learn more, see [Azure built-in roles for Azure Service Bus](#).

Connection strings in version 2.x

Version 2.x of the SDK doesn't require a specific name. Version 2.x lets you use your own names for these connection strings and allows you to store them elsewhere. You can set names in code using the [JobHostConfiguration](#), like this:

cs

```
static void Main(string[] args)
{
    var _storageConn = ConfigurationManager
        .ConnectionStrings["MyStorageConnection"].ConnectionString;

    //// Dashboard logging is deprecated; use Application Insights.
    //var _dashboardConn = ConfigurationManager
    //    .ConnectionStrings["MyDashboardConnection"].ConnectionString;

    JobHostConfiguration config = new JobHostConfiguration();
    config.StorageConnectionString = _storageConn;
    //config.DashboardConnectionString = _dashboardConn;
    JobHost host = new JobHost(config);
    host.RunAndBlock();
}
```

① Note

Because version 3.x uses the default .NET Core configuration APIs, there is no API to change connection string names. See [Develop and deploy WebJobs using Visual Studio](#)

Host development settings

You can run the host in development mode to make local development more efficient. Here are some of the settings that automatically change when you run in development mode:

[+] Expand table

Property	Development setting
Tracing.ConsoleLevel	TraceLevel.Verbose to maximize log output.
Queues.MaxPollingInterval	A low value to ensure queue methods are triggered immediately.
Singleton.ListenerLockPeriod	15 seconds to aid in rapid iterative development.

The process for enabling development mode depends on the SDK version.

Version 3.x

Version 3.x uses the standard ASP.NET Core APIs. Call the `UseEnvironment` method on the `HostBuilder` instance. Pass a string named `development`, as in this example:

cs

```
static async Task Main()
{
```

```
var builder = new HostBuilder();
builder.UseEnvironment("development");
builder.ConfigureWebJobs(b =>
{
    b.AddAzureStorageCoreServices();
});
var host = builder.Build();
using (host)
{
    await host.RunAsync();
}
}
```

Version 2.x

The `JobHostConfiguration` class has a `UseDevelopmentSettings` method that enables development mode. The following example shows how to use development settings. To make `config.IsDevelopment` return `true` when it runs locally, set a local environment variable named `AzureWebJobsEnv` with the value `Development`.

```
cs

static void Main()
{
    config = new JobHostConfiguration();

    if (config.IsDevelopment)
    {
        config.UseDevelopmentSettings();
    }

    var host = new JobHost(config);
    host.RunAndBlock();
}
```

Managing concurrent connections (version 2.x)

In version 3.x, the connection limit defaults to infinite connections. If for some reason you need to change this limit, you can use the `MaxConnectionsPerServer` property of the `WinHttpHandler` class.

In version 2.x, you control the number of concurrent connections to a host by using the `ServicePointManager.DefaultConnectionLimit` API. In 2.x, you should increase this value from the default of 2 before starting your WebJobs host.

All outgoing HTTP requests that you make from a function by using `HttpClient` flow through `ServicePointManager`. After you reach the value set in `DefaultConnectionLimit`, `ServicePointManager` starts queueing requests before sending them. Suppose your `DefaultConnectionLimit` is set to 2 and your code makes 1,000 HTTP requests. Initially, only two

requests are allowed through to the OS. The other 998 are queued until there's room for them. That means your `HttpClient` might time out because it appears to have made the request, but the request was never sent by the OS to the destination server. So you might see behavior that doesn't seem to make sense: your local `HttpClient` is taking 10 seconds to complete a request, but your service is returning every request in 200 ms.

The default value for ASP.NET applications is `Int32.MaxValue`, and that's likely to work well for WebJobs running in a Basic or higher App Service Plan. WebJobs typically need the **Always On** setting, and that's supported only by Basic and higher App Service Plans.

If your WebJob is running in a Free or Shared App Service Plan, your application is restricted by the App Service sandbox, which currently has a [connection limit of 300](#). With an unbound connection limit in `ServicePointManager`, it's more likely that the sandbox connection threshold will be reached and the site will shut down. In that case, setting `DefaultConnectionLimit` to something lower, like 50 or 100, can prevent this from happening and still allow for sufficient throughput.

The setting must be configured before any HTTP requests are made. For this reason, the WebJobs host shouldn't adjust the setting automatically. There could be HTTP requests that occur before the host starts, which could lead to unexpected behavior. The best approach is to set the value immediately in your `Main` method before initializing `JobHost`, as shown here:

C#

```
static void Main(string[] args)
{
    // Set this immediately so that it's used by all requests.
    ServicePointManager.DefaultConnectionLimit = Int32.MaxValue;

    var host = new JobHost();
    host.RunAndBlock();
}
```

Triggers

WebJobs SDK supports the same set of triggers and binding used by [Azure Functions](#). Please note that in the WebJobs SDK, triggers are function-specific and not related to the WebJob deployment type. WebJobs with event-triggered functions created using the SDK should always be published as a *continuous* WebJob, with *Always on* enabled.

Functions must be public methods and must have one trigger attribute or the [NoAutomaticTrigger](#) attribute.

Automatic triggers

Automatic triggers call a function in response to an event. Consider this example of a function that's triggered by a message added to Azure Queue storage. The function responds by reading a blob from Azure Blob storage:

```
cs

public static void Run(
    [QueueTrigger("myqueue-items")] string myQueueItem,
    [Blob("samples-workitems/{queueTrigger}", FileAccess.Read)] Stream myBlob,
    ILogger log)
{
    log.LogInformation($"BlobInput processed blob\n Name:{myQueueItem} \n Size:
{myBlob.Length} bytes");
}
```

The `QueueTrigger` attribute tells the runtime to call the function whenever a queue message appears in `myqueue-items`. The `Blob` attribute tells the runtime to use the queue message to read a blob in the *sample-workitems* container. The name of the blob item in the `samples-workitems` container is obtained directly from the queue trigger as a binding expression (`{queueTrigger}`).

① Note

A web app can time out after 20 minutes of inactivity, and only requests to the actual web app can reset the timer. Viewing the app's configuration in the Azure portal or making requests to the advanced tools site (https://<app_name>.scm.azurewebsites.net) doesn't reset the timer. If you set the web app that hosts your job to run continuously, run on a schedule, or use event-driven triggers, enable the **Always on** setting on your web app's **Azure Configuration** page. The Always on setting helps to make sure that these kinds of WebJobs run reliably. This feature is available only in the Basic, Standard, and Premium [pricing tiers](#).

Manual triggers

To trigger a function manually, use the `NoAutomaticTrigger` attribute, as shown here:

```
cs

[NoAutomaticTrigger]
public static void CreateQueueMessage(
    ILogger logger,
    string value,
    [Queue("outputqueue")] out string message)
{
    message = value;
```

```
        logger.LogInformation("Creating queue message: ", message);
    }
```

The process for manually triggering the function depends on the SDK version.

Version 3.x

CS

```
static async Task Main(string[] args)
{
    var builder = new HostBuilder();
    builder.ConfigureWebJobs(b =>
    {
        b.AddAzureStorageCoreServices();
        b.AddAzureStorage();
    });
    var host = builder.Build();
    using (host)
    {
        var jobHost = host.Services.GetService(typeof(IJobHost)) as JobHost;
        var inputs = new Dictionary<string, object>
        {
            { "value", "Hello world!" }
        };

        await host.StartAsync();
        await jobHost.CallAsync("CreateQueueMessage", inputs);
        await host.StopAsync();
    }
}
```

Version 2.x

CS

```
static void Main(string[] args)
{
    JobHost host = new JobHost();
    host.Call(typeof(Program).GetMethod("CreateQueueMessage"), new { value = "Hello
world!" });
}
```

Input and output bindings

Input bindings provide a declarative way to make data from Azure or third-party services available to your code. Output bindings provide a way to update data. The [Get started](#) article shows an example of each.

You can use a method return value for an output binding by applying the attribute to the method return value. See the example in [Using the Azure Function return value](#).

Binding types

The process for installing and managing binding types depends on whether you're using version 3.x or version 2.x of the SDK. You can find the package to install for a particular binding type in the "Packages" section of that binding type's Azure Functions [reference article](#). An exception is the Files trigger and binding (for the local file system), which isn't supported by Azure Functions.

Version 3.x

In version 3.x, the storage bindings are included in the `Microsoft.Azure.WebJobs.Extensions.Storage` package. Call the `AddAzureStorage` extension method in the `ConfigureWebJobs` method, as shown here:

```
cs

static async Task Main()
{
    var builder = new HostBuilder();
    builder.ConfigureWebJobs(b =>
    {
        b.AddAzureStorageCoreServices();
        b.AddAzureStorage();
    });
    var host = builder.Build();
    using (host)
    {
        await host.RunAsync();
    }
}
```

To use other trigger and binding types, install the NuGet package that contains them and call the `Add<binding>` extension method implemented in the extension. For example, if you want to use an Azure Cosmos DB binding, install `Microsoft.Azure.WebJobs.Extensions.CosmosDB` and call `AddCosmosDB`, like this:

```
cs

static async Task Main()
{
    var builder = new HostBuilder();
    builder.ConfigureWebJobs(b =>
    {
        b.AddAzureStorageCoreServices();
        b.AddCosmosDB();
    });
}
```

```
var host = builder.Build();
using (host)
{
    await host.RunAsync();
}
```

To use the Timer trigger or the Files binding, which are part of core services, call the `AddTimers` or `AddFiles` extension methods.

Version 2.x

These trigger and binding types are included in version 2.x of the `Microsoft.Azure.WebJobs` package:

- Blob storage
- Queue storage
- Table storage

To use other trigger and binding types, install the NuGet package that contains them and call a `Use<binding>` method on the `JobHostConfiguration` object. For example, if you want to use a Timer trigger, install `Microsoft.Azure.WebJobs.Extensions` and call `UseTimers` in the `Main` method, as shown here:

```
cs

static void Main()
{
    config = new JobHostConfiguration();
    config.UseTimers();
    var host = new JobHost(config);
    host.RunAndBlock();
}
```

To use the Files binding, install `Microsoft.Azure.WebJobs.Extensions` and call `UseFiles`.

ExecutionContext

WebJobs lets you bind to an [ExecutionContext](#). With this binding, you can access the [ExecutionContext](#) as a parameter in your function signature. For example, the following code uses the context object to access the invocation ID, which you can use to correlate all logs produced by a given function invocation.

```
cs

public class Functions
{
    public static void ProcessQueueMessage([QueueTrigger("queue")] string message,
```

```
        ExecutionContext executionContext,
        ILogger logger)
    {
        logger.LogInformation(${message}\n{executionContext.InvocationId});
    }
}
```

The process for binding to the [ExecutionContext](#) depends on your SDK version.

Version 3.x

Call the `AddExecutionContextBinding` extension method in the `ConfigureWebJobs` method, as shown here:

```
cs

static async Task Main()
{
    var builder = new HostBuilder();
    builder.ConfigureWebJobs(b =>
    {
        b.AddAzureStorageCoreServices();
        b.AddExecutionContextBinding();
    });
    var host = builder.Build();
    using (host)
    {
        await host.RunAsync();
    }
}
```

Version 2.x

The `Microsoft.Azure.WebJobs.Extensions` package mentioned earlier also provides a special binding type that you can register by calling the `UseCore` method. This binding lets you define an [ExecutionContext](#) parameter in your function signature, which is enabled like this:

```
cs

class Program
{
    static void Main()
    {
        config = new JobHostConfiguration();
        config.UseCore();
        var host = new JobHost(config);
        host.RunAndBlock();
    }
}
```

Binding configuration

You can configure the behavior of some triggers and bindings. The process for configuring them depends on the SDK version.

- **Version 3.x:** Set configuration when the `Add<Binding>` method is called in `ConfigureWebJobs`.
- **Version 2.x:** Set configuration by setting properties in a configuration object that you pass in to `JobHost`.

These binding-specific settings are equivalent to settings in the [host.json project file](#) in Azure Functions.

You can configure the following bindings:

- [Azure Cosmos DB trigger](#)
- [Event Hubs trigger](#)
- [Queue storage trigger](#)
- [SendGrid binding](#)
- [Service Bus trigger](#)

Azure Cosmos DB trigger configuration (version 3.x)

This example shows how to configure the Azure Cosmos DB trigger:

```
cs

static async Task Main()
{
    var builder = new HostBuilder();
    builder.ConfigureWebJobs(b =>
    {
        b.AddAzureStorageCoreServices();
        b.AddCosmosDB(a =>
        {
            a.ConnectionMode = ConnectionMode.Gateway;
            a.Protocol = Protocol.Https;
            a.LeaseOptions.LeasePrefix = "prefix1";

        });
    });
    var host = builder.Build();
    using (host)
    {
        await host.RunAsync();
    }
}
```

For more information, see the [Azure Cosmos DB binding](#) article.

Event Hubs trigger configuration (version 3.x)

This example shows how to configure the Event Hubs trigger:

```
cs

static async Task Main()
{
    var builder = new HostBuilder();
    builder.ConfigureWebJobs(b =>
    {
        b.AddAzureStorageCoreServices();
        b.AddEventHubs(a =>
        {
            a.BatchCheckpointFrequency = 5;
            a.EventProcessorOptions.MaxBatchSize = 256;
            a.EventProcessorOptions.PrefetchCount = 512;
        });
    });
    var host = builder.Build();
    using (host)
    {
        await host.RunAsync();
    }
}
```

For more information, see the [Event Hubs binding](#) article.

Queue storage trigger configuration

The following examples show how to configure the Queue storage trigger.

Version 3.x

```
cs

static async Task Main()
{
    var builder = new HostBuilder();
    builder.ConfigureWebJobs(b =>
    {
        b.AddAzureStorageCoreServices();
        b.AddAzureStorage(a => {
            a.BatchSize = 8;
            a.NewBatchThreshold = 4;
            a.MaxDequeueCount = 4;
            a.MaxPollingInterval = TimeSpan.FromSeconds(15);
        });
    });
    var host = builder.Build();
    using (host)
    {
        await host.RunAsync();
    }
}
```

```
    }  
}
```

For more information, see the [Queue storage binding](#) article.

Version 2.x

```
cs
```

```
static void Main(string[] args)  
{  
    JobHostConfiguration config = new JobHostConfiguration();  
    config.Queues.BatchSize = 8;  
    config.Queues.NewBatchThreshold = 4;  
    config.Queues.MaxDequeueCount = 4;  
    config.Queues.MaxPollingInterval = TimeSpan.FromSeconds(15);  
    JobHost host = new JobHost(config);  
    host.RunAndBlock();  
}
```

For more information, see the [host.json v1.x reference](#).

SendGrid binding configuration (version 3.x)

This example shows how to configure the SendGrid output binding:

```
cs
```

```
static async Task Main()  
{  
    var builder = new HostBuilder();  
    builder.ConfigureWebJobs(b =>  
    {  
        b.AddAzureStorageCoreServices();  
        b.AddSendGrid(a =>  
        {  
            a.FromAddress.Email = "samples@functions.com";  
            a.FromAddress.Name = "Azure Functions";  
        });  
    });  
    var host = builder.Build();  
    using (host)  
    {  
        await host.RunAsync();  
    }  
}
```

For more information, see the [SendGrid binding](#) article.

Service Bus trigger configuration (version 3.x)

This example shows how to configure the Service Bus trigger:

```
cs

static async Task Main()
{
    var builder = new HostBuilder();
    builder.ConfigureWebJobs(b =>
    {
        b.AddAzureStorageCoreServices();
        b.AddServiceBus(sbOptions =>
        {
            sbOptions.MessageHandlerOptions.AutoComplete = true;
            sbOptions.MessageHandlerOptions.MaxConcurrentCalls = 16;
        });
    });
    var host = builder.Build();
    using (host)
    {
        await host.RunAsync();
    }
}
```

For more details, see the [Service Bus binding](#) article.

Configuration for other bindings

Some trigger and binding types define their own custom configuration types. For example, the File trigger lets you specify the root path to monitor, as in the following examples.

Version 3.x

```
cs

static async Task Main()
{
    var builder = new HostBuilder();
    builder.ConfigureWebJobs(b =>
    {
        b.AddAzureStorageCoreServices();
        b.AddFiles(a => a.RootPath = @"c:\data\import");
    });
    var host = builder.Build();
    using (host)
    {
        await host.RunAsync();
    }
}
```

Version 2.x

```
cs

static void Main()
{
    config = new JobHostConfiguration();
    var filesConfig = new FilesConfiguration
    {
        RootPath = @"c:\data\import"
    };
    config.UseFiles(filesConfig);
    var host = new JobHost(config);
    host.RunAndBlock();
}
```

Binding expressions

In attribute constructor parameters, you can use expressions that resolve to values from various sources. For example, in the following code, the path for the `BlobTrigger` attribute creates an expression named `filename`. When used for the output binding, `filename` resolves to the name of the triggering blob.

```
cs

public static void CreateThumbnail(
    [BlobTrigger("sample-images/{filename}")] Stream image,
    [Blob("sample-images-sm/{filename}", FileAccess.Write)] Stream imageSmall,
    string filename,
    ILogger logger)
{
    logger.Info($"Blob trigger processing: {filename}");
    // ...
}
```

For more information about binding expressions, see [Binding expressions and patterns](#) in the Azure Functions documentation.

Custom binding expressions

Sometimes you want to specify a queue name, a blob name or container, or a table name in code rather than hard-coding it. For example, you might want to specify the queue name for the `QueueTrigger` attribute in a configuration file or environment variable.

You can do that by passing a custom name resolver during configuration. You include placeholders in trigger or binding attribute constructor parameters, and your resolver code provides the actual values to be used in place of those placeholders. You identify placeholders by surrounding them with percent (%) signs, as shown here:

```
cs
```

```
public static void WriteLog([QueueTrigger("%logqueue%")] string logMessage)
{
    Console.WriteLine(logMessage);
}
```

This code lets you use a queue named `logqueuetest` in the test environment and one named `logqueueprod` in production. Instead of a hard-coded queue name, you specify the name of an entry in the `appSettings` collection.

There's a default resolver that takes effect if you don't provide a custom one. The default gets values from app settings or environment variables.

Starting in .NET Core 3.1, the [ConfigurationManager](#) you use requires the [System.Configuration.ConfigurationManager NuGet package](#). The sample requires the following `using` statement:

```
cs

using System.Configuration;
```

Your `NameResolver` class gets the queue name from app settings, as shown here:

```
cs

public class CustomNameResolver : INameResolver
{
    public string Resolve(string name)
    {
        return ConfigurationManager.AppSettings[name].ToString();
    }
}
```

Version 3.x

You configure the resolver by using dependency injection. These samples require the following `using` statement:

```
cs

using Microsoft.Extensions.DependencyInjection;
```

You add the resolver by calling the [ConfigureServices](#) extension method on [HostBuilder](#), as in this example:

```
cs
```

```
static async Task Main(string[] args)
{
    var builder = new HostBuilder();
    var resolver = new CustomNameResolver();
    builder.ConfigureWebJobs(b =>
    {
        b.AddAzureStorageCoreServices();
    });
    builder.ConfigureServices(s => s.AddSingleton<INameResolver>(resolver));
    var host = builder.Build();
    using (host)
    {
        await host.RunAsync();
    }
}
```

Version 2.x

Pass your `NameResolver` class in to the `JobHost` object, as shown here:

```
cs

static void Main(string[] args)
{
    JobHostConfiguration config = new JobHostConfiguration();
    config.NameResolver = new CustomNameResolver();
    JobHost host = new JobHost(config);
    host.RunAndBlock();
}
```

Azure Functions implements `INameResolver` to get values from app settings, as shown in the example. When you use the WebJobs SDK directly, you can write a custom implementation that gets placeholder replacement values from whatever source you prefer.

Binding at runtime

If you need to do some work in your function before you use a binding attribute like `Queue`, `Blob`, or `Table`, you can use the `IBinder` interface.

The following example takes an input queue message and creates a new message with the same content in an output queue. The output queue name is set by code in the body of the function.

```
cs

public static void CreateQueueMessage(
    [QueueTrigger("inputqueue")] string queueMessage,
    IBinder binder)
{
```

```
        string outputQueueName = "outputqueue" + DateTime.Now.Month.ToString();
        QueueAttribute queueAttribute = new QueueAttribute(outputQueueName);
        CloudQueue outputQueue = binder.Bind<CloudQueue>(queueAttribute);
        outputQueue.AddMessageAsync(new CloudQueueMessage(queueMessage));
    }
```

For more information, see [Binding at runtime](#) in the Azure Functions documentation.

Binding reference information

The Azure Functions documentation provides reference information about each binding type. You'll find the following information in each binding reference article. (This example is based on Storage queue.)

- [Packages](#). The package you need to install to include support for the binding in a WebJobs SDK project.
- [Examples](#). Code samples. The C# class library example applies to the WebJobs SDK. Just omit the `FunctionName` attribute.
- [Attributes](#). The attributes to use for the binding type.
- [Configuration](#). Explanations of the attribute properties and constructor parameters.
- [Usage](#). The types you can bind to and information about how the binding works. For example: polling algorithm, poison queue processing.

Note

The HTTP, Webhooks, and Event Grid bindings are supported only by Azure Functions, not by the WebJobs SDK.

For a full list of bindings supported in Azure Functions runtime, see [Supported bindings](#).

Attributes for Disable, Timeout, and Singleton

With these attributes, you can control function triggering, cancel functions, and ensure that only one instance of a function runs.

Disable attribute

The [Disable](#) attribute lets you control whether a function can be triggered.

In the following example, if the app setting `Disable_TestJob` has a value of `1` or `True` (case insensitive), the function won't run. In that case, the runtime creates a log message *Function 'Functions.TestJob' is disabled.*

cs

```
[Disable("Disable_TestJob")]
public static void TestJob([QueueTrigger("testqueue2")] string message)
{
    Console.WriteLine("Function with Disable attribute executed!");
}
```

When you change app setting values in the Azure portal, the WebJob restarts to pick up the new setting.

The attribute can be declared at the parameter, method, or class level. The setting name can also contain binding expressions.

Timeout attribute

The [Timeout](#) attribute causes a function to be canceled if it doesn't finish within a specified amount of time. In the following example, the function would run for one day without the Timeout attribute. Timeout causes the function to be canceled after 15 seconds. When the Timeout attribute's "throwOnError" parameter is set to "true", the function invocation is terminated by having an exception thrown by the webjobs SDK when the timeout interval is exceeded. The default value of "throwOnError" is "false". When the Timeout attribute is used, the default behavior is to cancel the function invocation by setting the cancellation token while allowing the invocation to run indefinitely until the function code returns or throws an exception.

cs

```
[Timeout("00:00:15")]
public static async Task TimeoutJob(
    [QueueTrigger("testqueue2")] string message,
    CancellationToken token,
    TextWriter log)
{
    await log.WriteLineAsync("Job starting");
    await Task.Delay(TimeSpan.FromDays(1), token);
    await log.WriteLineAsync("Job completed");
}
```

You can apply the Timeout attribute at the class or method level, and you can specify a global timeout by using `JobHostConfiguration.FunctionTimeout`. Class-level or method-level timeouts override global timeouts.

Singleton attribute

The [Singleton](#) attribute ensures that only one instance of a function runs, even when there are multiple instances of the host web app. The Singleton attribute uses [distributed locking](#) to ensure that one instance runs.

In this example, only a single instance of the `ProcessImage` function runs at any given time:

```
cs

[Singleton]
public static async Task ProcessImage([BlobTrigger("images")] Stream image)
{
    // Process the image.
}
```

SingletonMode.Listener

Some triggers have built-in support for concurrency management:

- **QueueTrigger.** Set `JobHostConfiguration.Queues.BatchSize` to `1`.
- **ServiceBusTrigger.** Set `ServiceBusConfiguration.MessageOptions.MaxConcurrentCalls` to `1`.
- **FileTrigger.** Set `FileProcessor.MaxDegreeOfParallelism` to `1`.

You can use these settings to ensure that your function runs as a singleton on a single instance. To ensure that only a single instance of the function is running when the web app scales out to multiple instances, apply a listener-level singleton lock on the function (`[Singleton(Mode = SingletonMode.Listener)]`). Listener locks are acquired when the JobHost starts. If three scaled-out instances all start at the same time, only one of the instances acquires the lock and only one listener starts.

 **Note**

See this [GitHub Repo](#)  to learn more about how the `SingletonMode.Function` works.

Scope values

You can specify a *scope expression/value* on a singleton. The expression/value ensures that all executions of the function at a specific scope will be serialized. Implementing more granular locking in this way can allow for some level of parallelism for your function while serializing other invocations as dictated by your requirements. For example, in the following code, the scope expression binds to the `Region` value of the incoming message. When the queue contains three messages in regions East, East, and West, the messages that have region East are run serially. The message with region West is run in parallel with those in region East.

```
C#
```

```
[Singleton("{Region}")]
public static async Task ProcessWorkItem([QueueTrigger("workitems")] WorkItem
workItem)
{
    // Process the work item.
```

```
}

public class WorkItem
{
    public int ID { get; set; }
    public string Region { get; set; }
    public int Category { get; set; }
    public string Description { get; set; }
}
```

SingletonScope.Host

The default scope for a lock is `SingletonScope.Function`, meaning the lock scope (the blob lease path) is tied to the fully qualified function name. To lock across functions, specify `SingletonScope.Host` and use a scope ID name that's the same across all functions that you don't want to run simultaneously. In the following example, only one instance of `AddItem` or `RemoveItem` runs at a time:

C#

```
[Singleton("ItemsLock", SingletonScope.Host)]
public static void AddItem([QueueTrigger("add-item")] string message)
{
    // Perform the add operation.
}

[Singleton("ItemsLock", SingletonScope.Host)]
public static void RemoveItem([QueueTrigger("remove-item")] string message)
{
    // Perform the remove operation.
}
```

Viewing lease blobs

The WebJobs SDK uses [Azure blob leases](#) under the covers to implement distributed locking. The lease blobs used by Singleton can be found in the `azure-webjobs-host` container in the `AzureWebJobsStorage` storage account under the path "locks". For example, the lease blob path for the first `ProcessImage` example shown earlier might be `locks/061851c758f04938a4426aa9ab3869c0/WebJobs.Functions.ProcessImage`. All paths include the JobHost ID, in this case `061851c758f04938a4426aa9ab3869c0`.

Async functions

For information about how to code async functions, see the [Azure Functions documentation](#).

Cancellation tokens

For information about how to handle cancellation tokens, see the Azure Functions documentation on [cancellation tokens and graceful shutdown](#).

Multiple instances

If your web app runs on multiple instances, a continuous WebJob runs on each instance, listening for triggers and calling functions. The various trigger bindings are designed to efficiently share work collaboratively across instances, so that scaling out to more instances allows you to handle more load.

While some triggers may result in double-processing, queue and blob storage triggers automatically prevent a function from processing a queue message or blob more than once. For more information, see [Designing for identical input](#) in the Azure Functions documentation.

The timer trigger automatically ensures that only one instance of the timer runs, so you don't get more than one function instance running at a given scheduled time.

If you want to ensure that only one instance of a function runs even when there are multiple instances of the host web app, you can use the [Singleton](#) attribute.

Filters

Function Filters (preview) provide a way to customize the WebJobs execution pipeline with your own logic. Filters are similar to [ASP.NET Core filters](#). You can implement them as declarative attributes that are applied to your functions or classes. For more information, see [Function Filters](#).

Logging and monitoring

We recommend the logging framework that was developed for ASP.NET. The [Get started](#) article shows how to use it.

Log filtering

Every log created by an `ILogger` instance has an associated `Category` and `Level`. `LogLevel` is an enumeration, and the integer code indicates relative importance:

[] [Expand table](#)

LogLevel	Code
Trace	0

LogLevel	Code
Debug	1
Information	2
Warning	3
Error	4
Critical	5
None	6

You can independently filter each category to a particular `LogLevel`. For example, you might want to see all logs for blob trigger processing but only `Error` and higher for everything else.

Version 3.x

Version 3.x of the SDK relies on the filtering built into .NET Core. The `LogCategories` class lets you define categories for specific functions, triggers, or users. It also defines filters for specific host states, like `Startup` and `Results`. This enables you to fine-tune the logging output. If no match is found within the defined categories, the filter falls back to the `Default` value when deciding whether to filter the message.

`LogCategories` requires the following using statement:

```
cs

using Microsoft.Azure.WebJobs.Logging;
```

The following example constructs a filter that, by default, filters all logs at the `Warning` level. The `Function` and `results` categories (equivalent to `Host.Results` in version 2.x) are filtered at the `Error` level. The filter compares the current category to all registered levels in the `LogCategories` instance and chooses the longest match. This means that the `Debug` level registered for `Host.Triggers` matches `Host.Triggers.Queue` or `Host.Triggers.Blob`. This allows you to control broader categories without needing to add each one.

```
cs

static async Task Main(string[] args)
{
    var builder = new HostBuilder();
    builder.ConfigureWebJobs(b =>
    {
        b.AddAzureStorageCoreServices();
    });
    builder.ConfigureLogging(logging =>
    {
```

```

        logging.SetMinimumLevel(LogLevel.Warning);
        logging.AddFilter("Function", LogLevel.Error);

logging.AddFilter(LogCategories.CreateFunctionCategory("MySpecificFunctionName"),
    LogLevel.Debug);
logging.AddFilter(LogCategories.Results, LogLevel.Error);
logging.AddFilter("Host.Triggers", LogLevel.Debug);
});

var host = builder.Build();
using (host)
{
    await host.RunAsync();
}
}

```

Version 2.x

In version 2.x of the SDK, you use the `LogCategoryFilter` class to control filtering. The `LogCategoryFilter` has a `Default` property with an initial value of `Information`, meaning that any messages at the `Information`, `Warning`, `Error`, or `Critical` levels are logged, but any messages at the `Debug` or `Trace` levels are filtered away.

As with `LogCategories` in version 3.x, the `CategoryLevels` property allows you to specify log levels for specific categories so you can fine-tune the logging output. If no match is found within the `CategoryLevels` dictionary, the filter falls back to the `Default` value when deciding whether to filter the message.

The following example constructs a filter that by default filters all logs at the `Warning` level. The `Function` and `Host.Results` categories are filtered at the `Error` level. The `LogCategoryFilter` compares the current category to all registered `CategoryLevels` and chooses the longest match. So the `Debug` level registered for `Host.Triggers` will match `Host.Triggers.Queue` or `Host.Triggers.Blob`. This allows you to control broader categories without needing to add each one.

C#

```

var filter = new LogCategoryFilter();
filter.DefaultLevel = LogLevel.Warning;
filter.CategoryLevels[LogCategories.Function] = LogLevel.Error;
filter.CategoryLevels[LogCategories.Results] = LogLevel.Error;
filter.CategoryLevels["Host.Triggers"] = LogLevel.Debug;

config.LoggerFactory = new LoggerFactory()
    .AddApplicationInsights(instrumentationKey, filter.Filter)
    .AddConsole(filter.Filter);

```

Custom telemetry for Application Insights

The process for implementing custom telemetry for [Application Insights](#) depends on the SDK version. To learn how to configure Application Insights, see [Add Application Insights logging](#).

Version 3.x

Because version 3.x of the WebJobs SDK relies on the .NET Core generic host, a custom telemetry factory is no longer provided. But you can add custom telemetry to the pipeline by using dependency injection. The examples in this section require the following `using` statements:

```
cs

using Microsoft.ApplicationInsights.Extensibility;
using Microsoft.ApplicationInsights.Channel;
```

The following custom implementation of [ITelemetryInitializer](#) lets you add your own [ITelemetry](#) to the default [TelemetryConfiguration](#).

```
cs

internal class CustomTelemetryInitializer : ITelemetryInitializer
{
    public void Initialize(ITelemetry telemetry)
    {
        // Do something with telemetry.
    }
}
```

Call [ConfigureServices](#) in the builder to add your custom [ITelemetryInitializer](#) to the pipeline.

```
cs

static async Task Main()
{
    var builder = new HostBuilder();
    builder.ConfigureWebJobs(b =>
    {
        b.AddAzureStorageCoreServices();
    });
    builder.ConfigureLogging((context, b) =>
    {
        // Add logging providers.
        b.AddConsole();

        // If this key exists in any config, use it to enable Application Insights.
        string appInsightsKey =
            context.Configuration["APPINSIGHTS_INSTRUMENTATIONKEY"];
        if (!string.IsNullOrEmpty(appInsightsKey))
        {
            // This uses the options callback to explicitly set the instrumentation
            key.
```

```

        b.AddApplicationInsights(o => o.InstrumentationKey = appInsightsKey);
    }
});

builder.ConfigureServices(services =>
{
    services.AddSingleton<ITelemetryInitializer,
CustomTelemetryInitializer>();
});

var host = builder.Build();
using (host)
{
    await host.RunAsync();
}
}
}

```

When the [TelemetryConfiguration](#) is constructed, all registered types of [ITelemetryInitializer](#) are included. To learn more, see [Application Insights API for custom events and metrics](#).

In version 3.x, you no longer have to flush the [TelemetryClient](#) when the host stops. The .NET Core dependency injection system automatically disposes of the registered [ApplicationInsightsLoggerProvider](#), which flushes the [TelemetryClient](#).

Version 2.x

In version 2.x, the [TelemetryClient](#) created internally by the Application Insights provider for the WebJobs SDK uses [ServerTelemetryChannel](#). When the Application Insights endpoint is unavailable or throttling incoming requests, this channel [saves requests in the web app's file system and resubmits them later](#).

The [TelemetryClient](#) is created by a class that implements [ITelemetryClientFactory](#). By default, this is the [DefaultTelemetryClientFactory](#).

If you want to modify any part of the Application Insights pipeline, you can supply your own [ITelemetryClientFactory](#), and the host will use your class to construct a [TelemetryClient](#). For example, this code overrides [DefaultTelemetryClientFactory](#) to modify a property of [ServerTelemetryChannel](#):

C#

```

private class CustomTelemetryClientFactory : DefaultTelemetryClientFactory
{
    public CustomTelemetryClientFactory(string instrumentationKey, Func<string,
LogLevel, bool> filter)
        : base(instrumentationKey, new SamplingPercentageEstimatorSettings(),
filter)
    {
    }

    protected override ITelemetryChannel CreateTelemetryChannel()
    {
        ServerTelemetryChannel channel = new ServerTelemetryChannel();
    }
}

```

```
// Change the default from 30 seconds to 15 seconds.  
channel.MaxTelemetryBufferDelay = TimeSpan.FromSeconds(15);  
  
        return channel;  
    }  
}
```

The `SamplingPercentageEstimatorSettings` object configures [adaptive sampling](#). This means that in certain high-volume scenarios, Applications Insights sends a selected subset of telemetry data to the server.

After you create the telemetry factory, you pass it in to the Application Insights logging provider:

C#

```
var clientFactory = new CustomTelemetryClientFactory(instrumentationKey,  
filter.Filter);  
  
config.LoggerFactory = new LoggerFactory()  
.AddApplicationInsights(clientFactory);
```

Next steps

This article has provided code snippets that show how to handle common scenarios for working with the WebJobs SDK. For complete samples, see [azure-webjobs-sdk-samples ↗](#).

Reliability in Azure App Service

Article • 01/17/2025

This article describes reliability support in [Azure App Service](#). It covers both intra-regional resiliency with [availability zones](#) and information on [multi-region deployments](#).

Resiliency is a shared responsibility between you and Microsoft. This article also covers ways for you to build a resilient solution that meets your needs.

Azure App Service is an HTTP-based service for hosting web applications, REST APIs, and mobile back ends. Azure App Service adds the power of Microsoft Azure to your application, with capabilities for security, load balancing, autoscaling, and automated management. To explore how Azure App Service can bolster the reliability and resiliency of your application workload, see [Why use App Service?](#)

When you deploy Azure App Service, you can create multiple instances of an *App Service plan*, which represents the compute workers that run your application code. For more information, see [Azure App Service plan](#). Although the platform makes an effort to deploy the instances across different fault domains, it doesn't automatically spread the instances across availability zones.

Production deployment recommendations

For production deployments, you should:

- Use premium v3 App Service plans.
- [Enable zone redundancy](#), which requires your App Service plan to use a minimum of three instances.

Transient faults

Transient faults are short, intermittent failures in components. They occur frequently in a distributed environment like the cloud, and they're a normal part of operations. They correct themselves after a short period of time. It's important that your applications handle transient faults, usually by retrying affected requests.

All cloud-hosted applications should follow Azure's transient fault handling guidance when communicating with any cloud-hosted APIs, databases, and other components. To learn more about handling transient faults, see [Recommendations for handing transient faults](#).

Microsoft-provided SDKs usually handle transient faults. Because you host your own applications on Azure App Service, consider how to avoid causing transient faults by making sure that you:

- **Deploy multiple instances of your plan.** Azure App Service performs automated updates and other forms of maintenance on instances of your plan. If an instance becomes unhealthy, the service can automatically replace that instance with a new healthy instance. During the replacement process, there can be a short period when the previous instance is unavailable and a new instance isn't yet ready to serve traffic. You can mitigate the impact of this behavior by deploying multiple instances of your App Service plan.
- **Use deployment slots.** Azure App Service [deployment slots](#) allow for zero-downtime deployments of your applications. Use deployment slots to minimize the impact of deployments and configuration changes for your users. Using deployment slots also reduces the likelihood that your application restarts. Restarting causes a transient fault.
- **Avoid scaling up or down.** Instead, select a tier and instance size that meet your performance requirements under typical load. Only scale out instances to handle changes in traffic volume. Scaling up and down can trigger an application restart.

Availability zone support

Availability zones are physically separate groups of datacenters within each Azure region. When one zone fails, services can fail over to one of the remaining zones.

For more information on availability zones in Azure, see [What are availability zones?](#)

Azure App Service can be configured as *zone redundant*, which means that your resources are spread across multiple [availability zones](#). Spreading across multiple zones helps your production workloads achieve resiliency and reliability. Availability zone support is a property of the App Service plan.

Instance spreading with a zone-redundant deployment is determined using the following rules. These rules apply even as the app scales in and out:

- The minimum App Service plan instance count is three.
- The instances spread evenly if you specify a capacity larger than three, and the number of instances is divisible by three.
- Any instance counts beyond $3 \times N$ are spread across the remaining one or two zones.

When the App Service platform allocates instances for a zone-redundant App Service plan, it uses best effort zone balancing offered by the underlying Azure virtual machine scale sets. An App Service plan is *balanced* if each zone has either the same number of virtual machines, or plus or minus one, in all of the other zones. For more information, see [Zone balancing](#).

For App Service plans that aren't configured as zone redundant, virtual machine instances aren't resilient to availability zone failures. They can experience downtime during an outage in any zone in that region.

Regions supported

Zone-redundant App Service plans can be deployed in [any region that supports availability zones](#).

Requirements

- You must use either the [Premium v2](#) or [Premium v3](#) plan types.
- Availability zones are only supported on the newer App Service footprint. Even if you're using one of the supported regions, if availability zones aren't supported for your resource group, you receive an error. To ensure that your workloads land on a stamp that supports availability zones, you might need to create a new resource group, App Service plan, and App Service.

Multi-region support

Azure App Service is a single-region service. If the region becomes unavailable, your application is also unavailable.

Alternative multi-region solutions

To ensure that your application becomes less susceptible to a single-region failure, you need to deploy your application to multiple regions:

- Deploy your application to the instances in each region.
- Configure load balancing and failover policies.
- Replicate your data across the regions so that you can recover your last application state.

For example architectures that illustrates this approach, see:

- Reference architecture: Highly available multi-region web application.
- Multi-region App Service apps for disaster recovery

To follow along with a tutorial that creates a multi-region app, see [Tutorial: Create a highly available multi-region app in Azure App Service](#).

Backups

When you use Basic tier or higher, you can back up your App Service app to a file by using the App Service backup and restore capabilities. For more information, see [Backup and restore your app in Azure App Service](#).

This feature is useful if it's hard to redeploy your code, or if you store state on disk. For most solutions, you shouldn't rely on App Service backups. Use the other methods described in this article to support your resiliency requirements.

Service-level agreement (SLA)

The service-level agreement (SLA) for Azure App Service describes the expected availability of the service. It also describes the conditions that must be met to achieve that availability expectation. To understand those conditions, review the [Service Level Agreements \(SLA\) for Online Services](#).

When you deploy a zone-redundant App Service plan, the uptime percentage defined in the SLA increases.

Related content

- [Reliability in Azure](#)

Feedback

Was this page helpful?



[Provide product feedback](#) | Get help at Microsoft Q&A

Reliability in Azure App Service

Article • 01/17/2025

This article describes reliability support in [Azure App Service](#). It covers both intra-regional resiliency with [availability zones](#) and information on [multi-region deployments](#).

Resiliency is a shared responsibility between you and Microsoft. This article also covers ways for you to build a resilient solution that meets your needs.

Azure App Service is an HTTP-based service for hosting web applications, REST APIs, and mobile back ends. Azure App Service adds the power of Microsoft Azure to your application, with capabilities for security, load balancing, autoscaling, and automated management. To explore how Azure App Service can bolster the reliability and resiliency of your application workload, see [Why use App Service?](#)

When you deploy Azure App Service, you can create multiple instances of an *App Service plan*, which represents the compute workers that run your application code. For more information, see [Azure App Service plan](#). Although the platform makes an effort to deploy the instances across different fault domains, it doesn't automatically spread the instances across availability zones.

Production deployment recommendations

For production deployments, you should:

- Use premium v3 App Service plans.
- [Enable zone redundancy](#), which requires your App Service plan to use a minimum of three instances.

Transient faults

Transient faults are short, intermittent failures in components. They occur frequently in a distributed environment like the cloud, and they're a normal part of operations. They correct themselves after a short period of time. It's important that your applications handle transient faults, usually by retrying affected requests.

All cloud-hosted applications should follow Azure's transient fault handling guidance when communicating with any cloud-hosted APIs, databases, and other components. To learn more about handling transient faults, see [Recommendations for handing transient faults](#).

Microsoft-provided SDKs usually handle transient faults. Because you host your own applications on Azure App Service, consider how to avoid causing transient faults by making sure that you:

- **Deploy multiple instances of your plan.** Azure App Service performs automated updates and other forms of maintenance on instances of your plan. If an instance becomes unhealthy, the service can automatically replace that instance with a new healthy instance. During the replacement process, there can be a short period when the previous instance is unavailable and a new instance isn't yet ready to serve traffic. You can mitigate the impact of this behavior by deploying multiple instances of your App Service plan.
- **Use deployment slots.** Azure App Service [deployment slots](#) allow for zero-downtime deployments of your applications. Use deployment slots to minimize the impact of deployments and configuration changes for your users. Using deployment slots also reduces the likelihood that your application restarts. Restarting causes a transient fault.
- **Avoid scaling up or down.** Instead, select a tier and instance size that meet your performance requirements under typical load. Only scale out instances to handle changes in traffic volume. Scaling up and down can trigger an application restart.

Availability zone support

Availability zones are physically separate groups of datacenters within each Azure region. When one zone fails, services can fail over to one of the remaining zones.

For more information on availability zones in Azure, see [What are availability zones?](#)

Azure App Service can be configured as *zone redundant*, which means that your resources are spread across multiple [availability zones](#). Spreading across multiple zones helps your production workloads achieve resiliency and reliability. Availability zone support is a property of the App Service plan.

Instance spreading with a zone-redundant deployment is determined using the following rules. These rules apply even as the app scales in and out:

- The minimum App Service plan instance count is three.
- The instances spread evenly if you specify a capacity larger than three, and the number of instances is divisible by three.
- Any instance counts beyond $3 \times N$ are spread across the remaining one or two zones.

When the App Service platform allocates instances for a zone-redundant App Service plan, it uses best effort zone balancing offered by the underlying Azure virtual machine scale sets. An App Service plan is *balanced* if each zone has either the same number of virtual machines, or plus or minus one, in all of the other zones. For more information, see [Zone balancing](#).

For App Service plans that aren't configured as zone redundant, virtual machine instances aren't resilient to availability zone failures. They can experience downtime during an outage in any zone in that region.

Regions supported

Zone-redundant App Service plans can be deployed in [any region that supports availability zones](#).

Requirements

- You must use either the [Premium v2](#) or [Premium v3](#) plan types.
- Availability zones are only supported on the newer App Service footprint. Even if you're using one of the supported regions, if availability zones aren't supported for your resource group, you receive an error. To ensure that your workloads land on a stamp that supports availability zones, you might need to create a new resource group, App Service plan, and App Service.

Multi-region support

Azure App Service is a single-region service. If the region becomes unavailable, your application is also unavailable.

Alternative multi-region solutions

To ensure that your application becomes less susceptible to a single-region failure, you need to deploy your application to multiple regions:

- Deploy your application to the instances in each region.
- Configure load balancing and failover policies.
- Replicate your data across the regions so that you can recover your last application state.

For example architectures that illustrates this approach, see:

- Reference architecture: Highly available multi-region web application.
- Multi-region App Service apps for disaster recovery

To follow along with a tutorial that creates a multi-region app, see [Tutorial: Create a highly available multi-region app in Azure App Service](#).

Backups

When you use Basic tier or higher, you can back up your App Service app to a file by using the App Service backup and restore capabilities. For more information, see [Backup and restore your app in Azure App Service](#).

This feature is useful if it's hard to redeploy your code, or if you store state on disk. For most solutions, you shouldn't rely on App Service backups. Use the other methods described in this article to support your resiliency requirements.

Service-level agreement (SLA)

The service-level agreement (SLA) for Azure App Service describes the expected availability of the service. It also describes the conditions that must be met to achieve that availability expectation. To understand those conditions, review the [Service Level Agreements \(SLA\) for Online Services](#).

When you deploy a zone-redundant App Service plan, the uptime percentage defined in the SLA increases.

Related content

- [Reliability in Azure](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | Get help at Microsoft Q&A

Move an App Service app to another region

Article • 03/31/2023

ⓘ Important

Beginning 31 March 2025, we'll no longer place Azure App Service web applications in disaster recovery mode in the event of a disaster in an Azure region. We strongly encourage you to implement [commonly used disaster recovery techniques](#) to prevent loss of functionality or data for your web apps if there's a regional disaster.

This article describes how to bring App Service resources back online in a different Azure region during a disaster that impacts an entire Azure region. When a disaster brings an entire Azure region offline, all App Service apps hosted in that region are placed in disaster recovery mode. Features are available to help you restore the app to a different region or recover files from the impacted app.

App Service resources are region-specific and can't be moved across regions. You must restore the app to a new app in a different region, and then create mirroring configurations or resources for the new app.

Prerequisites

- None. [Restoring an automatic backup](#) usually requires Standard or Premium tier, but in disaster recovery mode, it's automatically enabled for your impacted app, regardless which tier the impacted app is in.

Prepare

Identify all the App Service resources that the impacted app currently uses. For example:

- App Service apps
- [App Service plans](#)
- [Deployment slots](#)
- Custom domains purchased in Azure
- [TLS/SSL certificates](#)
- [Azure Virtual Network integration](#)

- Hybrid connections.
- Managed identities
- Backup settings

Certain resources, such as imported certificates or hybrid connections, contain integration with other Azure services. For information on how to move those resources across regions, see the documentation for the respective services.

Restore app to a different region

1. Create a new App Service app in a *different* Azure region than the impacted app.
This is the target app in the disaster recovery scenario.
2. In the [Azure portal](#), navigate to the impacted app's management page. In a failed Azure region, the impacted app shows a warning text. Click the warning text.

The screenshot shows the Azure portal interface for an App Service named "drtestsite". At the top, there are navigation links: "Home >" followed by the app name "drtestsite" with a gear icon, and an "X" button. Below the name, it says "App Service". To the right of the name are several actions: "Browse", "Stop", "Swap", "Restart", "Delete", "Get publish profile", and three dots. A red box highlights a warning message: "⚠️ We are unable to retrieve comprehensive information about your app because it is operating in a limited state and may be down. Click here to restore your app contents from a snapshot." Below this, there is a list of details: "Resource group (change) DRTTesting", "Status", "---", "Location eastasiastage", "Subscription (change) DR OGF", and "Subscription ID 00000000-0000-0000-0000-000000000000".

3. In the **Restore Backup** page, configure the restore operation according to the following table. When finished, click **OK**.

Setting	Value	Description
Snapshot (Preview)	Select a snapshot.	The two most recent snapshots are available.
Restore destination	Existing app	Click the note below that says Click here to change the restore destination app and select the target app. In a disaster scenario, you can only restore the snapshot to an app in a different Azure region.

Setting	Value	Description
Restore site configuration	Yes	

Home > drtestsite >

Restore Backup

 Select Backup to Restore

Select a Backup on the app.

Snapshot (Preview) ⓘ

Snapshot taken at Tuesday, June 2, 2020, 11:41:15 AM PDT

 Select a target App Service App

Select to overwrite the current app or an existing app to restore content. Snapshots can only be restored to apps in the same App Service Plan.

 Overwriting the source app will result in data loss and will also cause extended period of downtime while the app is being restored. Make sure you have a backup of the current app content before overwriting the current app.

Restore destination ⓘ

Overwrite **Existing app**

 No restore destination app selected. Click here to select the restore destination app.

Select a slot to restore the backup to

 Advanced Settings

Advanced settings for restoring an app backup with options.

Restore site configuration

No Yes

OK

- Configure **everything else** in the target app to mirror the impacted app and verify your configuration.
- When you're ready for the custom domain to point to the target app, **remap the domain name**.

Recover app content only

If you only want to recover the files from the impacted app without restoring it, use the following steps:

1. In the [Azure portal](#), navigate to the impacted app's management page and click **Get publish profile**.

Home > drtestsite App Service

Browse Stop Swap Restart Delete Get publish profile ...

Get publish profile

We are unable to retrieve comprehensive information about your app because it is operating in a limited state and may be down. Click here to restore your app contents from a snapshot.

Resource group (change)
DRTesting

Status
--

Location
eastasiastage

Subscription (change)
DR OGF

Subscription ID
00000000-0000-0000-0000-000000000000

2. Open the downloaded file and find the publishing profile that contains `ReadOnly - FTP` in its name. This is the disaster recovery profile. For example:

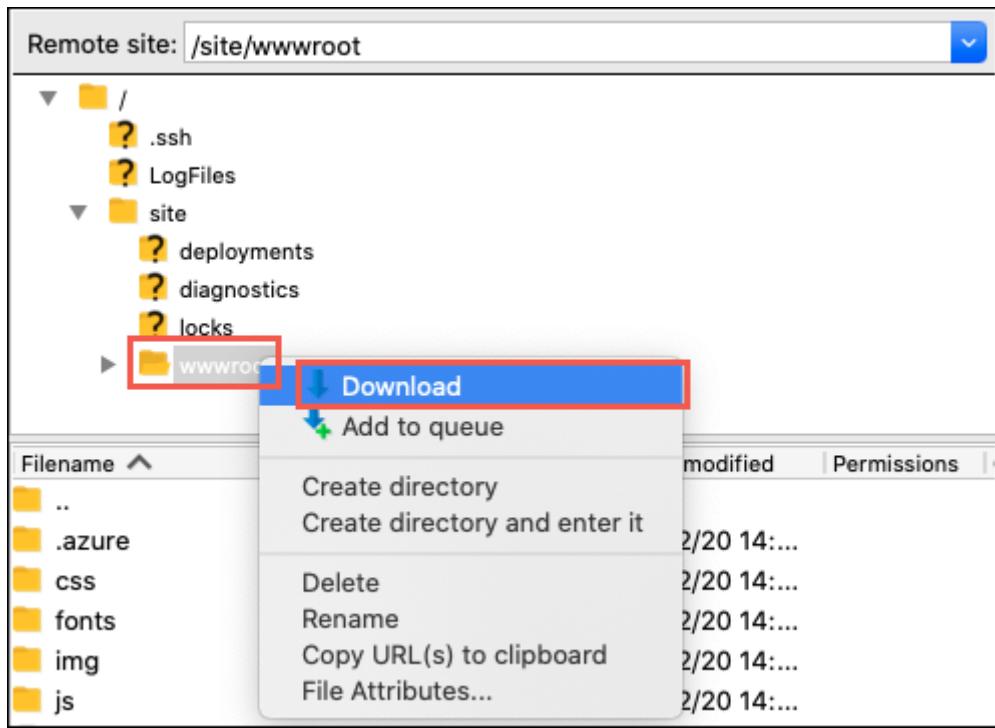
XML

```
<publishProfile profileName="%app-name% - ReadOnly - FTP"
publishMethod="FTP" publishUrl="ftp://%ftp-site%/site/wwwroot"
ftpPassiveMode="True" userName="%app-name%\%app-name%" userPWD=""
destinationAppUrl="http://%app-name%.azurewebsites.net"
SQLServerDBConnectionString="" mySQLDBConnectionString=""
hostingProviderForumLink="" controlPanelLink="http://windows.azure.com"
webSystem="WebSites">
  <databases />
</publishProfile>
```

Copy three attribute values:

- `publishUrl`: the FTP hostname
- `userName` and `userPWD`: the FTP credentials

3. Use the FTP client of your choice, connect to the impacted app's FTP host using the hostname and credentials.
4. Once connected, download the entire `/site/wwwroot` folder. The following screenshot shows how you download in [FileZilla](#).



Next steps

[Backup and restore](#)

CLI samples for Azure App Service

Article • 10/12/2022

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 into a local Git repository	Creates an App Service app and configures code push into 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 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	

Script	Description
Integrate with Azure Application Gateway	Creates an App Service app and integrates it with Application Gateway using service endpoint and access restrictions.
Connect app to resources	
Connect an app to a SQL Database	Creates an App Service app and a 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 Azure Cosmos DB	Creates an App Service app and an Azure Cosmos DB, then adds the Azure Cosmos DB connection details to the app settings.
Backup and restore app	
Backup and restore app	Creates an App Service app and creates a one-time backup for it, creates a backup schedule for it, and then 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

Article • 04/09/2023

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.

Script	Description
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	
Back up an app	Creates an App Service app and creates a one-time backup for it.
Create a scheduled backup for an app	Creates an App Service app and creates a scheduled backup for it.
Delete a backup for an app	Deletes an existing backup for an app.
Restore an app from backup	Restores an app from a previously completed backup.
Restore a backup across subscriptions	Restores a web app from a backup in another subscription.
Monitor app	
Monitor an app with web server logs	Creates an App Service app, enables logging for it, and downloads the logs to your local machine.

Azure Resource Manager templates for App Service

Article • 08/15/2024

The following table includes links to Azure Resource Manager templates for Azure App Service. For recommendations about avoiding common errors when you're creating app templates, see [Guidance on deploying apps with Azure Resource Manager templates](#).

To learn about the JSON syntax and properties for App Services resources, see [Microsoft.Web resource types](#).

[+] Expand table

Deploying an app	Description
App Service plan and basic Linux app	Deploys an App Service app that is configured for Linux.
App Service plan and basic Windows app	Deploys an App Service app that is configured for Windows.
App Service plan and basic Windows container app	Deploys an App Service app that is configured for a Windows container.
App linked to a GitHub repository	Deploys an App Service app that pulls code from GitHub.
App with custom deployment slots	Deploys an App Service app with custom deployment slots/environments.
App with Private Endpoint	Deploys an App Service app with a Private Endpoint.
Configuring an app	Description
App certificate from Key Vault	Deploys an App Service app certificate from an Azure Key Vault secret and uses it for TLS/SSL binding.
App with a custom domain and SSL	Deploys an App Service app with a custom host name, and gets an app certificate from Key Vault for TLS/SSL binding.
App with Java 8 and Tomcat 8	Deploys an App Service app with Java 8 and Tomcat 8 enabled. You can then run Java applications in Azure.
App with regional VNet integration	Deploys an App Service app with regional VNet integration enabled.
Protecting an app	Description

Deploying an app	Description
App integrated with Azure Application Gateway ↗	Deploys an App Service app and an Application Gateway, and isolates the traffic using service endpoint and access restrictions.
Linux app with connected resources	Description
App on Linux with MySQL ↗	Deploys an App Service app on Linux with Azure Database for MySQL.
App on Linux with PostgreSQL ↗	Deploys an App Service app on Linux with Azure Database for PostgreSQL.
App with connected resources	Description
App with MySQL ↗	Deploys an App Service app on Windows with Azure Database for MySQL.
App with PostgreSQL ↗	Deploys an App Service app on Windows with Azure Database for PostgreSQL.
App with a database in Azure SQL Database ↗	Deploys an App Service app and a database in Azure SQL Database at the Basic service level.
App with a Blob storage connection ↗	Deploys an App Service app with an Azure Blob storage connection string. You can then use Blob storage from the app.
App with an Azure Cache for Redis ↗	Deploys an App Service app with an Azure Cache for Redis.
App connected to a backend webapp ↗	Deploys two web apps (frontend and backend) securely connected together with VNet injection and Private Endpoint.
App connected to a backend webapp with staging slots ↗	Deploys two web apps (frontend and backend) with staging slots securely connected together with VNet injection and Private Endpoint.
Two apps in separate regions with Azure Front Door ↗	Deploys two identical web apps in separate regions with Azure Front Door to direct traffic.
App Service Environment	Description
Create an App Service environment v3 ↗	Creates an App Service environment v3, App Service plan, App Service, and all associated networking resources.

Feedback

Was this page helpful?

 Yes

 No

Provide product feedback  | Get help at Microsoft Q&A

Terraform samples for Azure App Service

Article • 07/03/2024

The following table includes links to Terraform scripts.

[+] Expand table

Script	Description
Create app	
Create two apps and connect securely with Private Endpoint and VNet integration	Creates two App Service apps and connect apps together with Private Endpoint and VNet integration.
Provision App Service and use slot swap to deploy	Provision App Service infrastructure with Azure deployment slots.
Create an Azure Windows web app with a backup	Create an Azure Windows web app with a backup schedule.

Feedback

Was this page helpful?



[Provide product feedback ↗](#)

Bicep files for App Service

Article • 08/15/2024

The following table includes links to Bicep files for Azure App Service. For quickstarts and further information about Bicep, see [Bicep documentation](#).

To learn about the Bicep syntax and properties for App Services resources, see [Microsoft.Web resource types](#).

[+] [Expand table](#)

Deploying an app	Description
App Service plan and basic Linux app	Deploys an App Service app that is configured for Linux.
App Service plan and basic Windows app	Deploys an App Service app that is configured for Windows.
Configuring an app	Description
App with log analytics module	Deploys an App Service app with log analytics.
App with regional VNet integration	Deploys an App Service app with regional VNet integration enabled.
App with connected resources	Description
App with CosmosDB	Deploys an App Service app on Linux with CosmosDB.
App with MySQL	Deploys an App Service app on Windows with Azure Database for MySQL.
App with a database in Azure SQL Database	Deploys an App Service app and a database in Azure SQL Database at the Basic service level.
App connected to a backend webapp	Deploys two web apps (frontend and backend) securely connected together with VNet injection and Private Endpoint.
App connected to a backend webapp with staging slots	Deploys two web apps (frontend and backend) with staging slots securely connected together with VNet injection and Private Endpoint.
App with a database, managed identity, and monitoring	Deploys an App Service App with a database, managed identity, and monitoring.
Two apps in separate regions with Azure Front Door	Deploys two identical web apps in separate regions with Azure Front Door to direct traffic.

Deploying an app	Description
App Service Environment	Description
Create an App Service environment v3 	Creates an App Service environment v3, App Service plan, App Service, and all associated networking resources.

Feedback

Was this page helpful?

 Yes No

[Provide product feedback !\[\]\(a6715b444c88530422c65146583686d0_img.jpg\)](#) | [Get help at Microsoft Q&A](#)

Environment variables and app settings in Azure App Service

Article • 10/16/2024

ⓘ Note

Starting June 1, 2024, all newly created App Service apps will have the option to generate a unique default hostname using the naming convention `<app-name>-<random-hash>. <region>.azurewebsites.net`. Existing app names will remain unchanged.

Example: `myapp-ds27dh7271aah175.westus-01.azurewebsites.net`

For further details, refer to [Unique Default Hostname for App Service Resource](#).

In [Azure App Service](#), certain settings are available to the deployment or runtime environment as environment variables. Some of these settings can be customized when you set them manually as [app settings](#). This reference shows the variables you can use or customize.

App environment

The following environment variables are related to the app environment in general.

[Expand table](#)

Setting name	Description	Example
<code>WEBSITE_SITE_NAME</code>	Read-only. App name.	
<code>WEBSITE_RESOURCE_GROUP</code>	Read-only. Azure resource group name that contains the app resource.	
<code>WEBSITE_OWNER_NAME</code>	Read-only. Contains the Azure subscription ID that owns the app, the resource group, and the webspace.	
<code>REGION_NAME</code>	Read-only. Region name of the app.	
<code>WEBSITE_PLATFORM_VERSION</code>	Read-only. App Service platform version.	
<code>HOME</code>	Read-only. Path to the home directory (for example, <code>D:\home</code> for Windows).	
<code>SERVER_PORT</code>	Read-only. The port the app should listen to.	
<code>WEBSITE_WARMUP_PATH</code>	A relative path to ping to warm up the app, beginning with a slash. The default is <code>/</code> , which pings the root path. The specific path can be pinged by an unauthenticated client, such as Azure Traffic Manager, even if App Service authentication is set to reject unauthenticated clients. (NOTE: This app setting doesn't change the path used by AlwaysOn.)	
<code>WEBSITE_COMPUTE_MODE</code>	Read-only. Specifies whether app runs on dedicated (<code>Dedicated</code>) or shared (<code>Shared</code>) VM/s.	

Setting name	Description	Example
WEBSITE_SKU	Read-only. SKU of the app. Possible values are <code>Free</code> , <code>Shared</code> , <code>Basic</code> , and <code>Standard</code> .	
SITE_BITNESS	Read-only. Shows whether the app is 32-bit (<code>x86</code>) or 64-bit (<code>AMD64</code>).	
WEBSITE_HOSTNAME	Read-only. Primary hostname for the app. Custom hostnames aren't accounted for here.	
WEBSITE_VOLUME_TYPE	Read-only. Shows the storage volume type currently in use.	
WEBSITE_NPM_DEFAULT_VERSION	Default npm version the app is using.	
WEBSOCKET_CONCURRENT_REQUEST_LIMIT	Read-only. Limit for websocket's concurrent requests. For <code>Standard</code> tier and above, the value is <code>-1</code> , but there's still a per VM limit based on your VM size (see Cross VM Numerical Limits).	
WEBSITE_PRIVATE_EXTENSIONS	Set to <code>0</code> to disable the use of private site extensions.	
WEBSITE_TIME_ZONE	By default, the time zone for the app is always UTC. You can change it to any of the valid values that are listed in Default Time Zones . If the specified value isn't recognized, UTC is used.	Atlantic Standard Time
WEBSITE_ADD_SITENAME_BINDINGS_IN_APPHOST_CONFIG	After slot swaps, the app may experience unexpected restarts. This is because after a swap, the hostname binding configuration goes out of sync, which by itself doesn't cause restarts. However, certain underlying storage events (such as storage volume failovers) may detect these discrepancies and force all worker processes to restart. To minimize these types of restarts, set the app setting value to <code>1</code> on all slots (default is <code>0</code>). However, don't set this value if you're running a Windows Communication Foundation (WCF) application. For more information, see Troubleshoot swaps	
WEBSITE_PROACTIVE_AUTOHEAL_ENABLED	By default, a VM instance is proactively "autohealed" when it's using more than 90% of allocated memory for more than 30 seconds, or when 80% of the total requests in the last two minutes take longer than 200 seconds. If a VM instance has triggered one of these rules, the recovery process is an overlapping restart of the instance. Set to <code>false</code> to disable this recovery behavior. The default is <code>true</code> . For more information, see Proactive Auto Heal .	
WEBSITE_PROACTIVE_CRASHMONITORING_ENABLED	Whenever the w3wp.exe process on a VM instance of your app crashes due to an unhandled exception for more than three times in 24 hours, a debugger process is attached to the main worker process on that instance, and collects a memory dump when the worker process crashes again. This memory dump is then analyzed and the call stack of the thread that caused the	

Setting name	Description	Example
	crash is logged in your App Service's logs. Set to <code>false</code> to disable this automatic monitoring behavior. The default is <code>true</code> . For more information, see Proactive Crash Monitoring .	
<code>WEBSITE_DAAS_STORAGE_SASURI</code>	During crash monitoring (proactive or manual), the memory dumps are deleted by default. To save the memory dumps to a storage blob container, specify the SAS URI.	
<code>WEBSITE_CRASHMONITORING_ENABLED</code>	Set to <code>true</code> to enable crash monitoring manually. You must also set <code>WEBSITE_DAAS_STORAGE_SASURI</code> and <code>WEBSITE_CRASHMONITORING_SETTINGS</code> . The default is <code>false</code> . This setting has no effect if remote debugging is enabled. Also, if this setting is set to <code>true</code> , proactive crash monitoring is disabled.	
<code>WEBSITE_CRASHMONITORING_SETTINGS</code>	A JSON with the following format: <pre>{"StartTimeUtc": "2020-02-10T08:21", "MaxHours": "<elapsed-hours-from-StartTimeUtc>", "MaxDumpCount": "<max-number-of-crash-dumps>"}.</pre> Required to configure crash monitoring if <code>WEBSITE_CRASHMONITORING_ENABLED</code> is specified. To only log the call stack without saving the crash dump in the storage account, add <code>,"UseStorageAccount": "false"</code> in the JSON.	
<code>REMOTEDEBUGGINGVERSION</code>	Remote debugging version.	
<code>WEBSITE_CONTENTAZUREFILECONNECTIONSTRING</code>	By default, App Service creates a shared storage for you at app creation. To use a custom storage account instead, set to the connection string of your storage account. For functions, see App settings reference for Functions .	<code>DefaultEndpointsProtocol=https;AccountName=<name>;AccountKey=<key></code>
<code>WEBSITE_CONTENTSHARE</code>	When you use specify a custom storage account with <code>WEBSITE_CONTENTAZUREFILECONNECTIONSTRING</code> , App Service creates a file share in that storage account for your app. To use a custom name, set this variable to the name you want. If a file share with the specified name doesn't exist, App Service creates it for you.	<code>myapp123</code>
<code>WEBSITE_SCM_ALWAYS_ON_ENABLED</code>	Read-only. Shows whether Always On is enabled (1) or not (0).	
<code>WEBSITE_SCM_SEPARATE_STATUS</code>	Read-only. Shows whether the Kudu app is running in a separate process (1) or not (0).	
<code>WEBSITE_DNS_ATTEMPTS</code>	Number of times to try name resolve.	
<code>WEBSITE_DNS_TIMEOUT</code>	Number of seconds to wait for name resolve	

Variable prefixes

The following table shows environment variable prefixes that App Service uses for various purposes.

[Expand table](#)

Setting name	Description
APPSETTING_	Signifies that a variable is set by the customer as an app setting in the app configuration. It's injected into a .NET app as an app setting.
MAINSITE_	Signifies a variable is specific to the app itself.
SCMSITE_	Signifies a variable is specific to the Kudu app.
SQLCONNSTR_	Signifies a SQL Server connection string in the app configuration. It's injected into a .NET app as a connection string.
SQLAZURECONNSTR_	Signifies an Azure SQL Database connection string in the app configuration. It's injected into a .NET app as a connection string.
POSTGRESQLCONNSTR_	Signifies a PostgreSQL connection string in the app configuration. It's injected into a .NET app as a connection string.
CUSTOMCONNSTR_	Signifies a custom connection string in the app configuration. It's injected into a .NET app as a connection string.
MYSQLCONNSTR_	Signifies a MySQL Database connection string in the app configuration. It's injected into a .NET app as a connection string.
AZUREFILESTORAGE_	A connection string to a custom share for a custom container in Azure Files.
AZUREBLOBSTORAGE_	A connection string to a custom storage account for a custom container in Azure Blob Storage.
NOTIFICATIONHUBCONNSTR_	Signifies a connection string to a notification hub in Azure Notification Hubs.
SERVICEBUSCONNSTR_	Signifies a connection string to an instance of Azure Service Bus.
EVENTHUBCONNSTR_	Signifies a connection string to an event hub in Azure Event Hubs.
DOCDBCONNSTR_	Signifies a connection string to a database in Azure Cosmos DB.
REDISCACHECONNSTR_	Signifies a connection string to a cache in Azure Cache for Redis.
FILESHARESTORAGE_	Signifies a connection string to a custom file share.

Deployment

The following environment variables are related to app deployment. For variables related to App Service build automation, see [Build automation](#).

[Expand table](#)

Setting name	Description
DEPLOYMENT_BRANCH	For local Git or cloud Git deployment (such as GitHub), set to the branch in Azure you want to deploy to. By default, it's <code>master</code> .
WEBSITE_RUN_FROM_PACKAGE	Set to <code>1</code> to run the app from a local ZIP package, or set to the URL of an external URL to run the app from a remote ZIP package. For more information, see Run your app in Azure App Service directly from a ZIP package .
WEBSITE_USE_ZIP	Deprecated. Use <code>WEBSITE_RUN_FROM_PACKAGE</code> .
WEBSITE_RUN_FROM_ZIP	Deprecated. Use <code>WEBSITE_RUN_FROM_PACKAGE</code> .

Setting name	Description
SCM_MAX_ZIP_PACKAGE_COUNT	Your app keeps 5 of the most recent zip files deployed using zip deploy . You can keep more or less by setting the app setting to a different number.
WEBSITE_WEBDEPLOY_USE_SCM	Set to <code>false</code> for WebDeploy to stop using the Kudu deployment engine. The default is <code>true</code> . To deploy to Linux apps using Visual Studio (WebDeploy/MSDeploy), set it to <code>false</code> .
MSDEPLOY_RENAME_LOCKED_FILES	Set to <code>1</code> to attempt to rename DLLs if they can't be copied during a WebDeploy deployment. This setting isn't applicable if <code>WEBSITE_WEBDEPLOY_USE_SCM</code> is set to <code>false</code> .
WEBSITE_DISABLE_SCM_SEPARATION	By default, the main app and the Kudu app run in different sandboxes. When you stop the app, the Kudu app is still running, and you can continue to use Git deploy and MSDeploy. Each app has its own local files. Turning off this separation (setting to <code>true</code>) is a legacy mode that's no longer fully supported.
WEBSITE_ENABLE_SYNC_UPDATE_SITE	Set to <code>1</code> ensure that REST API calls to update <code>site</code> and <code>siteconfig</code> are completely applied to all instances before returning. The default is <code>1</code> if deploying with an ARM template, to avoid race conditions with subsequent ARM calls.
WEBSITE_START_SCM_ON_SITE_CREATION	In an ARM template deployment, set to <code>1</code> in the ARM template to pre-start the Kudu app as part of app creation.
WEBSITE_START_SCM_WITH_PRELOAD	For Linux apps, set to <code>true</code> to force preloading the Kudu app when Always On is enabled by pinging its URL. The default is <code>false</code> . For Windows apps, the Kudu app is always preloaded.

Build automation

Kudu (Windows)

Kudu build configuration applies to native Windows apps and is used to control the behavior of Git-based (or ZIP-based) deployments.

[Expand table](#)

Setting name	Description	Example
SCM_BUILD_ARGS	Add things at the end of the msbuild command line, such that it overrides any previous parts of the default command line.	To do a clean build: <code>-t:Clean;Compile</code>
SCM_SCRIPT_GENERATOR_ARGS	Kudu uses the <code>azure site deploymentscript</code> command described here to generate a deployment script. It automatically detects the language framework type and determines the parameters to pass to the command. This setting overrides the automatically generated parameters.	To treat your repository as plain content files: <code>--basic -p <folder-to-deploy></code>
SCM_TRACE_LEVEL	Build trace level. The default is <code>1</code> . Set to higher values, up to <code>4</code> , for more tracing.	<code>4</code>
SCM_COMMAND_IDLE_TIMEOUT	Time out in seconds for each command that the build process launches to wait before without producing any output. After that, the command is considered idle and killed. The default is <code>60</code> (one minute). In Azure, there's also a general idle request timeout that disconnects clients after 230 seconds. However, the command will still continue running server-side after that.	
SCM_LOGSTREAM_TIMEOUT	Time-out of inactivity in seconds before stopping log streaming. The default is <code>1800</code> (30 minutes).	
SCM_SITEEXTENSIONS_FEED_URL	URL of the site extensions gallery. The default is https://www.nuget.org/api/v2/ . The URL of the old feed is http://www.siteextensions.net/api/v2/ .	
SCM_USE_LIBGIT2SHARP_REPOSITORY	Set to <code>0</code> to use git.exe instead of libgit2sharp for git operations.	

Setting name	Description	Example
WEBSITE_LOAD_USER_PROFILE	In case of the error <code>The specified user does not have a valid profile.</code> during ASP.NET build automation (such as during Git deployment), set this variable to <code>1</code> to load a full user profile in the build environment. This setting is only applicable when <code>WEBSITE_COMPUTE_MODE</code> is <code>Dedicated</code> .	
WEBSITE_SCM_IDLE_TIMEOUT_IN_MINUTES	Time out in minutes for the SCM (Kudu) site. The default is <code>20</code> .	
SCM_DO_BUILD_DURING_DEPLOYMENT	With ZIP deploy , the deployment engine assumes that a ZIP file is ready to run as-is and doesn't run any build automation. To enable the same build automation as in Git deploy , set to <code>true</code> .	

Language-specific settings

This section shows the configurable runtime settings for each supported language framework. Additional settings are available during [build automation](#) at deployment time.

.NET

[\[+\] Expand table](#)

Setting name	Description
PORT	Read-only. For Linux apps, port that the .NET runtime listens to in the container.
WEBSITE_ROLE_INSTANCE_ID	Read-only. ID of the current instance.
HOME	Read-only. Directory that points to shared storage (<code>/home</code>).
DUMP_DIR	Read-only. Directory for the crash dumps (<code>/home/logs/dumps</code>).
APP_SVC_RUN_FROM_COPY	Linux apps only. By default, the app is run from <code>/home/site/wwwroot</code> , a shared directory for all scaled-out instances. Set this variable to <code>true</code> to copy the app to a local directory in your container and run it from there. When using this option, be sure not to hard-code any reference to <code>/home/site/wwwroot</code> . Instead, use a path relative to <code>/home/site/wwwroot</code> .
MACHINEKEY_Decryption	For Windows native apps or Windows containerized apps, this variable is injected into app environment or container to enable ASP.NET cryptographic routines (see machineKey Element). To override the default <code>decryption</code> value, configure it as an App Service app setting, or set it directly in the <code>machineKey</code> element of the <code>Web.config</code> file.
MACHINEKEY_DecryptionKey	For Windows native apps or Windows containerized apps, this variable is injected into the app environment or container to enable ASP.NET cryptographic routines (see machineKey Element). To override the automatically generated <code>decryptionKey</code> value, configure it as an App Service app setting, or set it directly in the <code>machineKey</code> element of the <code>Web.config</code> file.
MACHINEKEY_Validation	For Windows native apps or Windows containerized apps, this variable is injected into the app environment or container to enable ASP.NET cryptographic routines (see machineKey Element). To override the default <code>validation</code> value, configure it as an App Service app setting, or set it directly in the <code>machineKey</code> element of the <code>Web.config</code> file.
MACHINEKEY_ValidationKey	For Windows native apps or Windows containerized apps, this variable is injected into the app environment or container to enable ASP.NET cryptographic routines (see machineKey Element). To override the automatically generated <code>validationKey</code> value, configure it as an App Service app setting, or set it directly in the <code>machineKey</code> element of the <code>Web.config</code> file.

WordPress

[Expand table](#)

Application Setting	Scope	Value	Max	Description
<code>WEBSITES_ENABLE_APP_SERVICE_STORAGE</code>	Web App	true	-	When set to TRUE, file contents are preserved during restarts.
<code>WP_MEMORY_LIMIT</code>	WordPress	128M	512M	Frontend or general wordpress PHP memory limit (per script). Can't be more than <code>PHP_MEMORY_LIMIT</code>
<code>WP_MAX_MEMORY_LIMIT</code>	WordPress	256M	512M	Admin dashboard PHP memory limit (per script). Generally Admin dashboard/ backend scripts takes lot of memory compared to frontend scripts. Can't be more than <code>PHP_MEMORY_LIMIT</code> .
<code>PHP_MEMORY_LIMIT</code>	PHP	512M	512M	Memory limits for general PHP script. It can only be decreased.
<code>FILE_UPLOADS</code>	PHP	On	-	Can be either On or Off. Note that values are case sensitive. Enables or disables file uploads.
<code>UPLOAD_MAX_FILESIZE</code>	PHP	50M	256M	Max file upload size limit. Can be increased up to 256M.
<code>POST_MAX_SIZE</code>	PHP	128M	256M	Can be increased up to 256M. Generally should be more than <code>UPLOAD_MAX_FILESIZE</code> .
<code>MAX_EXECUTION_TIME</code>	PHP	120	120	Can only be decreased. Please break down the scripts if it is taking more than 120 seconds. Added to avoid bad scripts from slowing the system.
<code>MAX_INPUT_TIME</code>	PHP	120	120	Max time limit for parsing the input requests. Can only be decreased.
<code>MAX_INPUT_VARS</code>	PHP	10000	10000	-
<code>DATABASE_HOST</code>	Database	-	-	Database host used to connect to WordPress.
<code>DATABASE_NAME</code>	Database	-	-	Database name used to connect to WordPress.
<code>DATABASE_USERNAME</code>	Database	-	-	Database username used to connect to WordPress.
<code>DATABASE_PASSWORD</code>	Database	-	-	Database password used to connect to the MySQL database. To change the MySQL database password, see update admin password . Whenever the MySQL database password is changed, the Application Settings also need to be updated.
<code>WORDPRESS_ADMIN_EMAIL</code>	Deployment only	-	-	WordPress admin email.
<code>WORDPRESS_ADMIN_PASSWORD</code>	Deployment only	-	-	WordPress admin password. This is only for deployment purposes. Modifying this value has no effect on the WordPress installation. To change the WordPress admin password, see resetting your password .
<code>WORDPRESS_ADMIN_USER</code>	Deployment only	-	-	WordPress admin username
<code>WORDPRESS_ADMIN_LOCALE_CODE</code>	Deployment only	-	-	Database username used to connect to WordPress.

Domain and DNS

[Expand table](#)

Setting name	Description	Example
<code>WEBSITE_DNS_SERVER</code>	IP address of primary DNS server for outgoing connections (such as to a back-end service). The default DNS server for App Service is Azure DNS, whose IP address is <code>168.63.129.16</code> . If your app uses VNet integration or is in an App Service environment , it inherits the DNS server configuration from the VNet by default.	<code>10.0.0.1</code>
<code>WEBSITE_DNS_ALT_SERVER</code>	IP address of fallback DNS server for outgoing connections. See <code>WEBSITE_DNS_SERVER</code> .	
<code>WEBSITE_ENABLE_DNS_CACHE</code>	Allows successful DNS resolutions to be cached. By Default expired DNS cache entries will be flushed & in addition to the existing cache to be flushed every 4.5 mins.	

TLS/SSL

For more information, see [Use a TLS/SSL certificate in your code in Azure App Service](#).

[Expand table](#)

Setting name	Description
<code>WEBSITE_LOAD_CERTIFICATES</code>	Comma-separate thumbprint values to the certificate you want to load in your code, or <code>*</code> to allow all certificates to be loaded in code. Only certificates added to your app can be loaded.
<code>WEBSITE_PRIVATE_CERTS_PATH</code>	Read-only. Path in a Windows container to the loaded private certificates.
<code>WEBSITE_PUBLIC_CERTS_PATH</code>	Read-only. Path in a Windows container to the loaded public certificates.
<code>WEBSITE_INTERMEDIATE_CERTS_PATH</code>	Read-only. Path in a Windows container to the loaded intermediate certificates.
<code>WEBSITE_ROOT_CERTS_PATH</code>	Read-only. Path in a Windows container to the loaded root certificates.

Deployment slots

For more information on deployment slots, see [Set up staging environments in Azure App Service](#).

[Expand table](#)

Setting name	Description	Example
<code>WEBSITE_OVERRIDE_STICKY_EXTENSION_VERSIONS</code>	By default, the versions for site extensions are specific to each slot. This prevents unanticipated application behavior due to changing extension versions after a swap. If you want the extension versions to swap as well, set to <code>0</code> on <i>all slots</i> .	
<code>WEBSITE_OVERRIDE_PRESERVE_DEFAULT_STICKY_SLOT_SETTINGS</code>	Designates certain settings as sticky or not swappable by default . Default is <code>true</code> . Set this setting to <code>false</code> or <code>0</code> for <i>all deployment slots</i> to make them swappable instead. There's no fine-grain control for specific setting types.	
<code>WEBSITE_SWAP_WARMUP_PING_PATH</code>	Path to ping to warm up the target slot in a swap, beginning with a slash. The default is <code>/</code> , which pings the root path over HTTP.	<code>/statuscheck</code>
<code>WEBSITE_SWAP_WARMUP_PING_STATUSES</code>	Valid HTTP response codes for the warm-up operation during a swap. If the returned status code isn't in the list, the warmup and swap operations are stopped. By default, all response codes are valid.	<code>200,202</code>
<code>WEBSITE_SLOT_NUMBER_OF_TIMEOUTS_BEFORE_RESTART</code>	During a slot swap, maximum number of timeouts after which we force restart the site on a specific VM instance. The default is <code>3</code> .	

Setting name	Description	Example
<code>WEBSITE_SLOT_MAX_NUMBER_OF_TIMEOUTS</code>	During a slot swap, maximum number of timeout requests for a single URL to make before giving up. The default is <code>5</code> .	
<code>WEBSITE_SKIP_ALL_BINDINGS_IN_APPHOST_CONFIG</code>	Set to <code>true</code> or <code>1</code> to skip all bindings in <code>applicationHost.config</code> . The default is <code>false</code> . If your app triggers a restart because <code>applicationHost.config</code> is updated with the swapped hostnames of the slots, set this variable to <code>true</code> to avoid a restart of this kind. If you're running a Windows Communication Foundation (WCF) app, don't set this variable.	

Custom containers

For more information on custom containers, see [Run a custom container in Azure](#).

[Expand table](#)

Setting name	Description	Example
<code>WEBSITES_ENABLE_APP_SERVICE_STORAGE</code>	For Linux custom containers: set to <code>true</code> to enable the <code>/home</code> directory to be shared across scaled instances. The default is <code>false</code> for Linux custom containers. For Windows containers: set to <code>true</code> to enable the <code>c:\home</code> directory to be shared across scaled instances. The default is <code>true</code> for Windows containers.	
<code>WEBSITES_CONTAINER_START_TIME_LIMIT</code>	Amount of time in seconds to wait for the container to complete start-up before restarting the container. Default is <code>230</code> . You can increase it up to the maximum of <code>1800</code> .	
<code>WEBSITES_CONTAINER_STOP_TIME_LIMIT</code>	Amount of time in seconds to wait for the container to terminate gracefully. Default is <code>5</code> . You can increase to a maximum of <code>120</code> .	
<code>DOCKER_REGISTRY_SERVER_URL</code>	URL of the registry server, when running a custom container in App Service. For security, this variable isn't passed on to the container.	<code>https://<server-name>.azurecr.io</code>
<code>DOCKER_REGISTRY_SERVER_USERNAME</code>	Username to authenticate with the registry server at <code>DOCKER_REGISTRY_SERVER_URL</code> . For security, this variable isn't passed on to the container.	
<code>DOCKER_REGISTRY_SERVER_PASSWORD</code>	Password to authenticate with the registry server at <code>DOCKER_REGISTRY_SERVER_URL</code> . For security, this variable isn't passed on to the container.	
<code>DOCKER_ENABLE_CI</code>	Set to <code>true</code> to enable the continuous deployment for custom containers. The default is <code>false</code> for custom containers.	
<code>WEBSITE_PULL_IMAGE_OVER_VNET</code>	Connect and pull from a registry inside a Virtual Network or on-premises. Your app will need to be connected to a Virtual Network using VNet integration feature. This setting is also needed for Azure Container Registry with Private Endpoint.	
<code>WEBSITES_WEB_CONTAINER_NAME</code>	In a Docker Compose app, only one of the containers can be internet accessible. Set to the name of the container defined in the configuration file to override the default container selection. By default, the internet accessible container is the first container to define port 80 or 8080, or, when no such container is found, the first container defined in the configuration file.	
<code>WEBSITES_PORT</code>	For a custom container, the custom port number on the container for App Service to route requests to. By default, App Service attempts automatic	

Setting name	Description	Example
	port detection of ports 80 and 8080. This setting isn't injected into the container as an environment variable.	
WEBSITE_CPU_CORES_LIMIT	By default, a Windows container runs with all available cores for your chosen pricing tier. To reduce the number of cores, set to the number of desired cores limit. For more information, see Customize the number of compute cores .	
WEBSITE_MEMORY_LIMIT_MB	By default all Windows Containers deployed in Azure App Service have a memory limit configured depending on the App Service Plan SKU. Set to the desired memory limit in MB. The cumulative total of this setting across apps in the same plan must not exceed the amount allowed by the chosen pricing tier. For more information, see Customize container memory .	

Scaling

[Expand table](#)

Setting name	Description
WEBSITE_INSTANCE_ID	Read-only. Unique ID of the current VM instance, when the app is scaled out to multiple instances.
WEBSITE_IIS_SITE_NAME	Deprecated. Use <code>WEBSITE_INSTANCE_ID</code> .
WEBSITE_DISABLE_OVERLAPPED_RECYCLING	Overlapped recycling makes it so that before the current VM instance of an app is shut down, a new VM instance starts. In some cases, it can cause file locking issues. You can try turning it off by setting to <code>1</code> .
WEBSITE_DISABLE_CROSS_STAMP_SCALE	By default, apps are allowed to scale across stamps if they use Azure Files or a Docker container. Set to <code>1</code> or <code>true</code> to disable cross-stamp scaling within the app's region. The default is <code>0</code> . Custom Docker containers that set <code>WEBSITES_ENABLE_APP_SERVICE_STORAGE</code> to <code>true</code> or <code>1</code> can't scale cross-stamps because their content isn't completely encapsulated in the Docker container.

Logging

[Expand table](#)

Setting name	Description	Example
WEBSITE_HTTPLOGGING_ENABLED	Read-only. Shows whether the web server logging for Windows native apps is enabled (<code>1</code>) or not (<code>0</code>).	
WEBSITE_HTTPLOGGING_RETENTION_DAYS	Retention period in days of web server logs, if web server logs are enabled for a Windows native or Linux app.	10
WEBSITE_HTTPLOGGING_CONTAINER_URL	SAS URL of the blob storage container to store web server logs for Windows native apps, if web server logs are enabled. If not set, web server logs are stored in the app's file system (default shared storage).	
DIAGNOSTICS_AZUREBLOBRETENTIONINDAYS	Retention period in days of application logs for Windows native apps, if application logs are enabled.	10
DIAGNOSTICS_AZUREBLOBCONTAINERSASURL	SAS URL of the blob storage container to store application logs for Windows native apps, if application logs are enabled.	
APPSERVICEAPPLOGS_TRACE_LEVEL	Minimum log level to ship to Log Analytics for the AppServiceAppLogs log type.	

Setting name	Description	Example
DIAGNOSTICS_LASTRESORTFILE	The filename to create, or a relative path to the log directory, for logging internal errors for troubleshooting the listener. The default is <code>logging-errors.txt</code> .	
DIAGNOSTICS_LOGGINGSETTINGSFILE	Path to the log settings file, relative to <code>D:\home</code> or <code>/home</code> . The default is <code>site\diagnostics\settings.json</code> .	
DIAGNOSTICS_TEXTTRACELOGDIRECTORY	The log folder, relative to the app root (<code>D:\home\site\wwwroot</code> or <code>/home/site/wwwroot</code>).	<code>..\..\LogFiles\Application</code>
DIAGNOSTICS_TEXTTRACEMAXLOGFILESIZEBYTES	Maximum size of the log file in bytes. The default is <code>131072</code> (128 KB).	
DIAGNOSTICS_TEXTTRACEMAXLOGFOLDERSIZEBYTES	Maximum size of the log folder in bytes. The default is <code>1048576</code> (1 MB).	
DIAGNOSTICS_TEXTTRACEMAXNUMLOGFILES	Maximum number of log files to keep. The default is <code>20</code> .	
DIAGNOSTICS_TEXTTRACETURNOFFPERIOD	Time out in milliseconds to keep application logging enabled. The default is <code>43200000</code> (12 hours).	
WEBSITE_LOG_BUFFERING	By default, log buffering is enabled. Set to <code>0</code> to disable it.	
WEBSITE_ENABLE_PERF_MODE	For native Windows apps, set to <code>TRUE</code> to turn off IIS log entries for successful requests returned within 10 seconds. This is a quick way to do performance benchmarking by removing extended logging.	

Performance counters

The following are 'fake' environment variables that don't exist if you enumerate them, but return their value if you look them up individually. The value is dynamic and can change on every lookup.

[Expand table](#)

Setting name	Description
WEBSITE_COUNTERS_ASPNET	A JSON object containing the ASP.NET perf counters.
WEBSITE_COUNTERS_APP	A JSON object containing sandbox counters.
WEBSITE_COUNTERS_CLR	A JSON object containing CLR counters.
WEBSITE_COUNTERS_ALL	A JSON object containing the combination of the other three variables.

Caching

[Expand table](#)

Setting name	Description
WEBSITE_LOCAL_CACHE_OPTION	Whether local cache is enabled. Available options are: <ul style="list-style-type: none"> - <code>Default</code>: Inherit the stamp-level global setting. - <code>Always</code>: Enable for the app. - <code>OnStorageUnavailability</code> - <code>Disabled</code>: Disabled for the app.
WEBSITE_LOCAL_CACHE_READWRITE_OPTION	Read-write options of the local cache. Available options are: <ul style="list-style-type: none"> - <code>ReadOnly</code>: Cache is read-only. - <code>WriteButDiscardChanges</code>: Allow writes to local cache but discard changes made locally.

Setting name	Description
WEBSITE_LOCAL_CACHE_SIZEINMB	Size of the local cache in MB. Default is 1000 (1 GB).
WEBSITE_LOCALCACHE_READY	Read-only flag indicating if the app using local cache.
WEBSITE_DYNAMIC_CACHE	Due to network file shared nature to allow access for multiple instances, the dynamic cache improves performance by caching the recently accessed files locally on an instance. Cache is invalidated when file is modified. The cache location is %SYSTEMDRIVE%\local\DynamicCache (same %SYSTEMDRIVE%\local quota is applied). To enable full content caching, set to 1, which includes both file content and directory/file metadata (timestamps, size, directory content). To conserve local disk use, set to 2 to cache only directory/file metadata (timestamps, size, directory content). To turn off caching, set to 0. For Windows apps and for Linux apps created with the WordPress template , the default is 1. For all other Linux apps, the default is 0.
WEBSITE_READONLY_APP	When using dynamic cache, you can disable write access to the app root (D:\home\site\wwwroot or /home/site/wwwroot) by setting this variable to 1. Except for the App_Data directory, no exclusive locks are allowed, so that deployments don't get blocked by locked files.

Networking

The following environment variables are related to [hybrid connections](#) and [VNET integration](#).

[Expand table](#)

Setting name	Description
WEBSITE_RELAYS	Read-only. Data needed to configure the Hybrid Connection, including endpoints and service bus data.
WEBSITE_REWRITE_TABLE	Read-only. Used at runtime to do the lookups and rewrite connections appropriately.
WEBSITE_VNET_ROUTE_ALL	By default, if you use regional VNet Integration , your app only routes RFC1918 traffic into your VNet. Set to 1 to route all outbound traffic into your VNet and be subject to the same NSGs and UDRs. The setting lets you access non-RFC1918 endpoints through your VNet, secure all outbound traffic leaving your app, and force tunnel all outbound traffic to a network appliance of your own choosing.
WEBSITE_PRIVATE_IP	Read-only. IP address associated with the app when integrated with a VNet . For Regional VNet Integration, the value is an IP from the address range of the delegated subnet, and for Gateway-required VNet Integration, the value is an IP from the address range of the point-to-site address pool configured on the Virtual Network Gateway. This IP is used by the app to connect to the resources through the VNet. Also, it can change within the described address range.
WEBSITE_PRIVATE_PORTS	Read-only. In VNet integration, shows which ports are useable by the app to communicate with other nodes.
WEBSITE_CONTENTOVERVNET	If you are mounting an Azure File Share on the App Service and the Storage account is restricted to a VNET, ensure to enable this setting with a value of 1.

Key vault references

The following environment variables are related to [key vault references](#).

[Expand table](#)

Setting name	Description
WEBSITE_KEYVAULT_REFERENCES	Read-only. Contains information (including statuses) for all Key Vault references that are currently configured in the app.
WEBSITE_SKIP_CONTENTSHARE_VALIDATION	If you set the shared storage connection of your app (using WEBSITE_CONTENTAZUREFILECONNECTIONSTRING) to a Key Vault reference, the app can't resolve the key vault reference at app creation or update if one of the following conditions is true: - The app accesses the key vault with a system-assigned identity. - The app accesses the key vault with a user-assigned identity, and the key vault is locked with a

Setting name	Description
<code>VNet</code>	To avoid errors at create or update time, set this variable to <code>1</code> .
<code>WEBSITE_DELAY_CERT_DELETION</code>	This env var can be set to <code>1</code> by users in order to ensure that a certificate that a worker process is dependent upon isn't deleted until it exits.

CORS

The following environment variables are related to Cross-Origin Resource Sharing (CORS) configuration.

[Expand table](#)

Setting name	Description
<code>WEBSITE_CORS_ALLOWED_ORIGINS</code>	Read-only. Shows the allowed origins for CORS.
<code>WEBSITE_CORS_SUPPORT_CREDENTIALS</code>	Read-only. Shows whether setting the <code>Access-Control-Allow-Credentials</code> header to <code>true</code> is enabled (<code>True</code>) or not (<code>False</code>).

Authentication & Authorization

The following environment variables are related to [App Service authentication](#).

[Expand table](#)

Setting name	Description
<code>WEBSITE_AUTH_DISABLE_IDENTITY_FLOW</code>	When set to <code>true</code> , disables assigning the thread principal identity in ASP.NET-based web applications (including v1 Function Apps). This is designed to allow developers to protect access to their site with auth, but still have it use a separate sign-in mechanism within their app logic. The default is <code>false</code> .
<code>WEBSITE_AUTH_HIDE_DEPRECATED_SID</code>	<code>true</code> or <code>false</code> . The default value is <code>false</code> . This is a setting for the legacy Azure Mobile Apps integration for Azure App Service. Setting this to <code>true</code> resolves an issue where the SID (security ID) generated for authenticated users might change if the user changes their profile information. Changing this value may result in existing Azure Mobile Apps user IDs changing. Most apps don't need to use this setting.
<code>WEBSITE_AUTH_NONCE_DURATION</code>	A <code>timespan</code> value in the form <code>_hours_:_minutes_:_seconds_</code> . The default value is <code>00:05:00</code> , or 5 minutes. This setting controls the lifetime of the cryptographic nonce generated for all browser-driven logins. If a sign-in fails to complete in the specified time, the sign-in flow will be retried automatically. This application setting is intended for use with the V1 (classic) configuration experience. If using the V2 authentication configuration schema, you should instead use the <code>login.nonce.nonceExpirationInterval</code> configuration value.
<code>WEBSITE_AUTH_PRESERVE_URL_FRAGMENT</code>	When set to <code>true</code> and users select on app links that contain URL fragments, the sign-in process will ensure that the URL fragment part of your URL doesn't get lost in the sign-in redirect process. For more information, see Customize sign-in and sign-out in Azure App Service authentication .
<code>WEBSITE_AUTH_USE_LEGACY CLAIMS</code>	To maintain backward compatibility across upgrades, the authentication module uses the legacy claims mapping of short to long names in the <code>/auth/me</code> API, so certain mappings are excluded (e.g. "roles"). To get the more modern version of the claims mappings, set this variable to <code>False</code> . In the "roles" example, it would be mapped to the long claim name " <code>http://schemas.microsoft.com/ws/2008/06/identity/claims/role</code> ".
<code>WEBSITE_AUTH_DISABLE_WWWAUTHENTICATE</code>	<code>true</code> or <code>false</code> . The default value is <code>false</code> . When set to <code>true</code> , removes the WWW-Authenticate HTTP response header from module-generated HTTP 401 responses. This application setting is intended for use with the V1 (classic) configuration experience. If using the V2 authentication configuration schema, you should instead use the <code>identityProviders.azureActiveDirectory.login.disableWwwAuthenticate</code> configuration value.

Setting name	Description
<code>WEBSITE_AUTH_STATE_DIRECTORY</code>	A local file system directory path where tokens are stored when the file-based token store is enabled. The default value is <code>%HOME%\Data\auth</code> . This application setting is intended for use with the V1 (classic) configuration experience. If using the V2 authentication configuration schema, you should instead use the <code>login.tokenStore.FileSystem.directory</code> configuration value.
<code>WEBSITE_AUTH_TOKEN_CONTAINER_SASURL</code>	A fully qualified blob container URL. Instructs the auth module to store and load all encrypted tokens to the specified blob storage container instead of using the default local file system.
<code>WEBSITE_AUTH_TOKEN_REFRESH_HOURS</code>	Any positive decimal number. The default value is <code>72</code> (hours). This setting controls the amount of time after a session token expires that the <code>/auth/refresh</code> API can be used to refresh it. Refresh attempts after this period will fail and end users will be required to sign-in again. This application setting is intended for use with the V1 (classic) configuration experience. If using the V2 authentication configuration schema, you should instead use the <code>login.tokenStore.tokenRefreshExtensionHours</code> configuration value.
<code>WEBSITE_AUTH_TRACE_LEVEL</code>	Controls the verbosity of authentication traces written to Application Logging . Valid values are <code>Off</code> , <code>Error</code> , <code>Warning</code> , <code>Information</code> , and <code>Verbose</code> . The default value is <code>Verbose</code> .
<code>WEBSITE_AUTH_VALIDATE_NONCE</code>	<code>true</code> or <code>false</code> . The default value is <code>true</code> . This value should never be set to <code>false</code> except when temporarily debugging cryptographic nonce validation failures that occur during interactive logins. This application setting is intended for use with the V1 (classic) configuration experience. If using the V2 authentication configuration schema, you should instead use the <code>login.nonce.validateNonce</code> configuration value.
<code>WEBSITE_AUTH_V2_CONFIG_JSON</code>	This environment variable is populated automatically by the Azure App Service platform and is used to configure the integrated authentication module. The value of this environment variable corresponds to the V2 (non-classic) authentication configuration for the current app in Azure Resource Manager. It's not intended to be configured explicitly.
<code>WEBSITE_AUTH_ENABLED</code>	Read-only. Injected into a Windows or Linux app to indicate whether App Service authentication is enabled.
<code>WEBSITE_AUTH_ENCRYPTION_KEY</code>	By default, the automatically generated key is used as the encryption key. To override, set to a desired key. This is recommended if you want to share tokens or sessions across multiple apps. If specified, it supersedes the <code>MACHINEKEY_DecryptionKey</code> setting.
<code>WEBSITE_AUTH_SIGNING_KEY</code>	By default, the automatically generated key is used as the signing key. To override, set to a desired key. This is recommended if you want to share tokens or sessions across multiple apps. If specified, it supersedes the <code>MACHINEKEY_ValidationKey</code> setting.

Managed identity

The following environment variables are related to [managed identities](#).

[Expand table](#)

Setting name	Description
<code>IDENTITY_ENDPOINT</code>	Read-only. The URL to retrieve the token for the app's managed identity .
<code>MSI_ENDPOINT</code>	Deprecated. Use <code>IDENTITY_ENDPOINT</code> .
<code>IDENTITY_HEADER</code>	Read-only. Value that must be added to the <code>x-IDENTITY-HEADER</code> header when making an HTTP GET request to <code>IDENTITY_ENDPOINT</code> . The value is rotated by the platform.
<code>MSI_SECRET</code>	Deprecated. Use <code>IDENTITY_HEADER</code> .

Health check

The following environment variables are related to [health checks](#).

[Expand table](#)

Setting name	Description
<code>WEBSITE_HEALTHCHECK_MAXPINGFAILURES</code>	The maximum number of failed pings before removing the instance. Set to a value between <code>2</code> and <code>100</code> . When you're scaling up or out, App Service pings the Health check path to ensure new instances are ready. For more information, see Health check .
<code>WEBSITE_HEALTHCHECK_MAXUNHEALTHYWORKERPERCENT</code>	To avoid overwhelming healthy instances, no more than half of the instances will be excluded. For example, if an App Service Plan is scaled to four instances and three are unhealthy, at most two will be excluded. The other two instances (one healthy and one unhealthy) will continue to receive requests. In the worst-case scenario where all instances are unhealthy, none will be excluded. To override this behavior, set to a value between <code>1</code> and <code>100</code> . A higher value means more unhealthy instances will be removed. The default is <code>50</code> (50%).

Push notifications

The following environment variables are related to the [push notifications](#) feature.

[Expand table](#)

Setting name	Description
<code>WEBSITE_PUSH_ENABLED</code>	Read-only. Added when push notifications are enabled.
<code>WEBSITE_PUSH_TAG_WHITELIST</code>	Read-only. Contains the tags in the notification registration.
<code>WEBSITE_PUSH_TAGS_REQUIRING_AUTH</code>	Read-only. Contains a list of tags in the notification registration that requires user authentication.
<code>WEBSITE_PUSH_TAGS_DYNAMIC</code>	Read-only. Contains a list of tags in the notification registration that were added automatically.

 **Note**

This article contains references to a term that Microsoft no longer uses. When the term is removed from the software, we'll remove it from this article.

Webjobs

The following environment variables are related to [WebJobs](#).

[Expand table](#)

Setting name	Description
<code>WEBJOBS_RESTART_TIME</code>	For continuous jobs, delay in seconds when a job's process goes down for any reason before relaunching it.
<code>WEBJOBS_IDLE_TIMEOUT</code>	For triggered jobs, timeout in seconds, after which the job is aborted if it's in idle, has no CPU time or output.
<code>WEBJOBS_HISTORY_SIZE</code>	For triggered jobs, maximum number of runs kept in the history directory per job. The default is <code>50</code> .
<code>WEBJOBS_STOPPED</code>	Set to <code>1</code> to disable running any job, and stop all currently running jobs.
<code>WEBJOBS_DISABLE_SCHEDULE</code>	Set to <code>1</code> to turn off all scheduled triggering. Jobs can still be manually invoked.
<code>WEBJOBS_ROOT_PATH</code>	Absolute or relative path of webjob files. For a relative path, the value is combined with the default root path (<code>D:/home/site/wwwroot/</code> or <code>/home/site/wwwroot/</code>).

Setting name	Description
<code>WEBJOBS_LOG_TRIGGERED_JOBS_TO_APP_LOGS</code>	Set to true to send output from triggered WebJobs to the application logs pipeline (which supports file system, blobs, and tables).
<code>WEBJOBS_SHUTDOWN_FILE</code>	File that App Service creates when a shutdown request is detected. It's the web job process's responsibility to detect the presence of this file and initiate shutdown. When using the WebJobs SDK, this part is handled automatically.
<code>WEBJOBS_PATH</code>	Read-only. Root path of currently running job (will be under some temporary directory).
<code>WEBJOBS_NAME</code>	Read-only. Current job name.
<code>WEBJOBS_TYPE</code>	Read-only. Current job type (<code>triggered</code> or <code>continuous</code>).
<code>WEBJOBS_DATA_PATH</code>	Read-only. Current job metadata path to contain the job's logs, history, and any artifact of the job.
<code>WEBJOBS_RUN_ID</code>	Read-only. For triggered jobs, current run ID of the job.

Functions

 [Expand table](#)

Setting name	Description
<code>WEBSITE_FUNCTIONS_ARMCACHE_ENABLED</code>	Set to <code>0</code> to disable the functions cache.
<code>WEBSITE_MAX_DYNAMIC_APPLICATION_SCALE_OUT</code>	App settings reference for Azure Functions
<code>AzureWebJobsSecretStorageType</code>	App settings reference for Azure Functions
<code>FUNCTIONS_EXTENSION_VERSION</code>	App settings reference for Azure Functions
<code>FUNCTIONS_WORKER_RUNTIME</code>	App settings reference for Azure Functions
<code>AzureWebJobsStorage</code>	App settings reference for Azure Functions
<code>WEBSITE_CONTENTAZUREFILECONNECTIONSTRING</code>	App settings reference for Azure Functions
<code>WEBSITE_CONTENTSHARE</code>	App settings reference for Azure Functions
<code>WEBSITE_CONTENTOVERVNET</code>	App settings reference for Azure Functions
<code>WEBSITE_ENABLE_BROTLI_ENCODING</code>	App settings reference for Azure Functions
<code>WEBSITE_USE_PLACEHOLDER</code>	App settings reference for Azure Functions
<code>WEBSITE_PLACEHOLDER_MODE</code>	Read-only. Shows whether the function app is running on a placeholder host (<code>generalized</code>) or its own host (<code>specialized</code>).
<code>WEBSITE_DISABLE_ZIP_CACHE</code>	When your app runs from a ZIP package (<code>WEBSITE_RUN_FROM_PACKAGE=1</code>), the five most recently deployed ZIP packages are cached in the app's file system (D:\home\data\SitePackages). Set this variable to <code>1</code> to disable this cache. For Linux consumption apps, the ZIP package cache is disabled by default.

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

az appservice

Reference

ⓘ Note

This command group has commands that are defined in both Azure CLI and at least one extension. Install each extension to benefit from its extended capabilities. [Learn more](#) about extensions.

Manage App Service plans.

Commands

[Expand table](#)

Name	Description	Type	Status
az appservice ase	Manage App Service Environments.	Core	GA
az appservice ase create	Create app service environment.	Core	GA
az appservice ase create-inbound-services	Private DNS Zone for Internal (ILB) App Service Environments.	Core	Preview
az appservice ase delete	Delete app service environment.	Core	GA
az appservice ase list	List app service environments.	Core	GA
az appservice ase list-addresses	List VIPs associated with an app service environment v2.	Core	GA
az appservice ase list-plans	List app service plans associated with an app service environment.	Core	GA
az appservice ase send-test-notification	Send a test upgrade notification in app service environment v3.	Core	Preview
az appservice ase show	Show details of an app service environment.	Core	GA

Name	Description	Type	Status
az appservice ase update	Update app service environment.	Core	GA
az appservice ase upgrade	Upgrade app service environment v3.	Core	Preview
az appservice domain	Manage custom domains.	Core	Preview
az appservice domain create	Create and purchase a custom domain.	Core	Preview
az appservice domain show-terms	Show the legal terms for purchasing and creating a custom domain.	Core	Preview
az appservice hybrid-connection	A method that sets the key a hybrid-connection uses.	Core	GA
az appservice hybrid-connection set-key	Set the key that all apps in an appservice plan use to connect to the hybrid-connections in that appservice plan.	Core	GA
az appservice kube	Manage Kubernetes Environments.	Extension	Preview
az appservice kube create	Create a Kubernetes Environment.	Extension	Preview
az appservice kube delete	Delete kubernetes environment.	Extension	Preview
az appservice kube list	List kubernetes environments by subscription or resource group.	Extension	Preview
az appservice kube show	Show the details of a kubernetes environment.	Extension	Preview
az appservice kube update	Update a Kubernetes Environment. Currently not supported.	Extension	Preview
az appservice kube wait	Wait for a Kubernetes Environment to reach a desired state.	Extension	Preview
az appservice list-locations	List regions where a plan sku is available.	Core	GA

Name	Description	Type	Status
az appservice plan	Manage app service plans.	Core and Extension	GA
az appservice plan create	Create an app service plan.	Core	GA
az appservice plan create (appservice-kube extension)	Create an app service plan.	Extension	GA
az appservice plan delete	Delete an app service plan.	Core	GA
az appservice plan list	List app service plans.	Core	GA
az appservice plan list (appservice-kube extension)	List app service plans.	Extension	GA
az appservice plan show	Get the app service plans for a resource group or a set of resource groups.	Core	GA
az appservice plan show (appservice-kube extension)	Get the app service plans for a resource group or a set of resource groups.	Extension	GA
az appservice plan update	Update an app service plan.	Core	GA
az appservice plan update (appservice-kube extension)	Update an app service plan. See https://docs.microsoft.com/azure/app-service/app-service-plan-manage#move-an-app-to-another-app-service-plan to learn more.	Extension	GA
az appservice vnet-integration	A method that lists the virtual network integrations used in an appservice plan.	Core	GA
az appservice vnet-integration list	List the virtual network integrations used in an appservice plan.	Core	GA

az appservice list-locations

 Edit

List regions where a plan sku is available.

Azure CLI

```
az appservice list-locations --sku {B1, B2, B3, D1, F1, FREE, I1, I1MV2,  
I1V2, I2, I2MV2, I2V2, I3, I3MV2, I3V2, I4MV2, I4V2, I5MV2, I5V2, I6V2,  
P0V3, P1MV3, P1V2, P1V3, P2MV3, P2V2, P2V3, P3MV3, P3V2, P3V3, P4MV3, P5MV3,  
S1, S2, S3, SHARED, WS1, WS2, WS3}  
    [--hyperv-workers-enabled]  
    [--linux-workers-enabled]
```

Examples

List regions where a plan sku is available. (autogenerated)

Azure CLI

```
az appservice list-locations --sku F1
```

Required Parameters

--sku

The pricing tiers, e.g., F1(Free), D1(Shared), B1(Basic Small), B2(Basic Medium), B3(Basic Large), S1(Standard Small), P1V2(Premium V2 Small), P2V2(Premium V2 Medium), P3V2(Premium V2 Large), P0V3(Premium V3 Extra Small), P1V3(Premium V3 Small), P2V3(Premium V3 Medium), P3V3(Premium V3 Large), P1MV3(Premium Memory Optimized V3 Small), P2MV3(Premium Memory Optimized V3 Medium), P3MV3(Premium Memory Optimized V3 Large), P4MV3(Premium Memory Optimized V3 Extra Large), P5MV3(Premium Memory Optimized V3 Extra Extra Large), I1(Isolated Small), I2 (Isolated Medium), I3 (Isolated Large), I1V2 (Isolated V2 I1V2), I2V2 (Isolated V2 I2V2), I3V2 (Isolated V2 I3V2), I4V2 (Isolated V2 I4V2), I5V2 (Isolated V2 I5V2), I6V2 (Isolated V2 I6V2), I1MV2 (Isolated Memory Optimized V2 I1MV2), I2MV2 (Isolated Memory Optimized V2 I2MV2), I3MV2 (Isolated Memory Optimized V2 I3MV2), I4MV2 (Isolated Memory Optimized V2 I4MV2), I5MV2 (Isolated Memory Optimized V2 I5MV2), WS1 (Logic Apps Workflow Standard 1), WS2 (Logic Apps Workflow Standard 2), WS3 (Logic Apps Workflow Standard 3).

Accepted values: B1, B2, B3, D1, F1, FREE, I1, I1MV2, I1V2, I2, I2MV2, I2V2, I3, I3MV2, I3V2, I4MV2, I4V2, I5MV2, I5V2, I6V2, P0V3, P1MV3, P1V2, P1V3, P2MV3, P2V2, P2V3, P3MV3, P3V2, P3V3, P4MV3, P5MV3, S1, S2, S3, SHARED, WS1, WS2, WS3

Optional Parameters

--hyperv-workers-enabled

Get regions which support hosting web apps on Windows Container workers.

--linux-workers-enabled

Get regions which support hosting web apps on Linux workers.

▼ Global Parameters

--debug

Increase logging verbosity to show all debug logs.

--help -h

Show this help message and exit.

--only-show-errors

Only show errors, suppressing warnings.

--output -o

Output format.

Accepted values: json, jsonc, none, table, tsv, yaml, yamlc

Default value: json

--query

JMESPath query string. See <http://jmespath.org/> for more information and examples.

--subscription

Name or ID of subscription. You can configure the default subscription using `az account set -s NAME_OR_ID`.

--verbose

Increase logging verbosity. Use --debug for full debug logs.

Az.Websites

Reference

ARM (Azure Resource Manager) Web App and App Service Plan commands.

App Service

 Expand table

Add-AzWebAppAccessRestrictionRule	Adds an Access Restriction rule to an Azure Web App.
Add-AzWebAppTrafficRouting	Add a routing Rule to the Slot.
Edit-AzWebAppBackupConfiguration	Edits the current configuration backup for an Azure Web App
Get-AzAppServiceEnvironment	Gets App Service Environment. If only Resource Group is specified, it will return a list of ASE in the Resource Group.
Get-AzAppServicePlan	Gets an Azure App Service plan in the specified resource group.
Get-AzDeletedWebApp	Gets deleted web apps in the subscription.
Get-AzStaticWebApp	Description for Gets the details of a static site.
Get-AzStaticWebAppBuild	Description for Gets the details of a static site build.
Get-AzStaticWebAppBuildAppSetting	Description for Gets the application settings of a static site build.
Get-AzStaticWebAppBuildFunction	Description for Gets the functions of a particular static site build.
Get-AzStaticWebAppBuildFunctionAppSetting	Description for Gets the application settings of a static site build.
Get-AzStaticWebAppConfiguredRole	Description for Lists the roles configured for the static site.
Get-AzStaticWebAppCustomDomain	Description for Gets an existing custom domain for a particular static site.
Get-AzStaticWebAppFunction	Description for Gets the functions of a static

	site.
Get-AzStaticWebAppFunctionAppSetting	Description for Gets the application settings of a static site.
Get-AzStaticWebAppSecret	Description for Lists the secrets for an existing static site.
Get-AzStaticWebAppSetting	Description for Gets the application settings of a static site.
Get-AzStaticWebAppUser	Description for Gets the list of users of a static site.
Get-AzStaticWebAppUserProvidedFunctionApp	Description for Gets the details of the user provided function app registered with a static site build
Get-AzWebApp	Gets Azure Web Apps in the specified resource group.
Get-AzWebAppAccessRestrictionConfig	Gets Access Restriction configuration for an Azure Web App.
Get-AzWebAppBackup	
Get-AzWebAppBackupConfiguration	
Get-AzWebAppBackupList	
Get-AzWebAppCertificate	Gets an Azure Web App certificate.
Get-AzWebAppContainerContinuousDeploymentUrl	Get-AzWebAppContainerContinuousDeploymentUrl will return container continuous deployment url
Get-AzWebAppContinuousWebJob	Get or list continuous web for an app.
Get-AzWebAppPublishingProfile	Gets an Azure Web App publishing profile.
Get-AzWebAppSlot	Gets an Azure Web App slot.
Get-AzWebAppSlotConfigName	Get the list of Web App Slot Config names
Get-AzWebAppSlotContinuousWebJob	Get or list continuous web for a deployment slot.
Get-AzWebAppSlotPublishingProfile	Gets an Azure Web App slot publishing profile.
Get-AzWebAppSlotTriggeredWebJob	Get or list triggered web for a deployment slot.
Get-AzWebAppSlotTriggeredWebJobHistory	Get or list triggered web job's history for a deployment slot.

Get-AzWebAppSlotWebJob	List webjobs for a deployment slot.
Get-AzWebAppSnapshot	Gets the snapshots available for a web app.
Get-AzWebAppSSLBinding	Gets an Azure Web App certificate SSL binding.
Get-AzWebAppTrafficRouting	Get a routing Rule for the given Slot name.
Get-AzWebAppTriggeredWebJob	Get or list triggered web for an app.
Get-AzWebAppTriggeredWebJobHistory	Get or list triggered web job's history for an app.
Get-AzWebAppWebJob	List webjobs for an app.
Import-AzWebAppKeyVaultCertificate	Import an SSL certificate to a web app from Key Vault.
New-AzAppServiceEnvironment	Creates an App Service Environment including the recommended Route Table and Network Security Group
New-AzAppServiceEnvironmentInboundServices	Creates inbound services for App Service Environment. For ASEv2 ILB, this will create an Azure Private DNS Zone and records to map to the internal IP. For ASEv3 it will in addition ensure subnet has Network Policy disabled and will create a private endpoint.
New-AzAppServicePlan	Creates an Azure App Service plan in a given Geo location.
New-AzStaticWebApp	Description for Creates a new static site in an existing resource group, or updates an existing static site.
New-AzStaticWebAppBuildAppSetting	Description for Creates or updates the app settings of a static site build.
New-AzStaticWebAppBuildFunctionAppSetting	Description for Creates or updates the function app settings of a static site build.
New-AzStaticWebAppCustomDomain	Description for Creates a new static site custom domain in an existing resource group and static site.
New-AzStaticWebAppFunctionAppSetting	Description for Creates or updates the function app settings of a static site.
New-AzStaticWebAppSetting	Description for Creates or updates the app settings of a static site.

New-AzStaticWebAppUserRoleInvitationLink	Description for Creates an invitation link for a user with the role
New-AzWebApp	Creates an Azure Web App.
New-AzWebAppAzureStoragePath	Creates an object that represents an Azure Storage path to be mounted in a Web App. It is meant to be used as a parameter (-AzureStoragePath) to Set-AzWebApp and Set-AzWebAppSlot
New-AzWebAppBackup	Creates an Azure Web App Backup.
New-AzWebAppCertificate	Creates an App service managed certificate for an Azure Web App.
New-AzWebAppDatabaseBackupSetting	Creates a new Azure Web App Backup setting.
New-AzWebAppSlot	Creates an Azure Web App slot.
New-AzWebAppSSLBinding	Creates an SSL certificate binding for an Azure Web App.
Publish-AzWebApp	Deploys an Azure Web App from a ZIP, JAR, or WAR file using zipdeploy.
Register-AzStaticWebAppUserProvidedFunctionApp	Description for Register a user provided function app with a static site build
Remove-AzAppServiceEnvironment	Remove App Service Environment.
Remove-AzAppServicePlan	Removes an Azure App Service plan.
Remove-AzStaticWebApp	Description for Deletes a static site.
Remove-AzStaticWebAppAttachedRepository	Description for Detaches a static site.
Remove-AzStaticWebAppBuild	Description for Deletes a static site build.
Remove-AzStaticWebAppCustomDomain	Description for Deletes a custom domain.
Remove-AzStaticWebAppUser	Description for Deletes the user entry from the static site.
Remove-AzWebApp	Removes an Azure Web App.
Remove-AzWebAppAccessRestrictionRule	Removes an Access Restriction rule from an Azure Web App.
Remove-AzWebAppBackup	
Remove-AzWebAppCertificate	Removes an App service managed certificate for an Azure Web App.

Remove-AzWebAppContinuousWebJob	Delete a continuous web job for an app.
Remove-AzWebAppSlot	
Remove-AzWebAppSlotContinuousWebJob	Delete a continuous web job for a deployment slot.
Remove-AzWebAppSlotTriggeredWebJob	Delete a triggered web job for a deployment slot.
Remove-AzWebAppSSLBinding	Removes an SSL binding from an uploaded certificate.
Remove-AzWebAppTrafficRouting	Remove a routing Rule from the Slot.
Remove-AzWebAppTriggeredWebJob	Delete a triggered web job for an app.
Reset-AzStaticWebAppApiKey	Description for Resets the api key for an existing static site.
Reset-AzWebAppPublishingProfile	Resets the publishing profile for the specified Web App.
Reset-AzWebAppSlotPublishingProfile	
Restart-AzWebApp	Restarts an Azure Web App.
Restart-AzWebAppSlot	Restarts an Azure Web App Slot.
Restore-AzDeletedWebApp	Restores a deleted web app to a new or existing web app.
Restore-AzWebAppBackup	Restores an Azure Web App Backup.
Restore-AzWebAppSnapshot	Restores a web app snapshot.
Set-AzAppServicePlan	Sets an Azure App Service plan.
Set-AzWebApp	Modifies an Azure Web App.
Set-AzWebAppSlot	Modifies an Azure Web App slot.
Set-AzWebAppSlotConfigName	Set Web App Slot Config names
Start-AzWebApp	Starts an Azure Web App.
Start-AzWebAppContinuousWebJob	Start a continuous web job for an app.
Start-AzWebAppSlot	Starts an Azure Web App slot.
Start-AzWebAppSlotContinuousWebJob	Start a continuous web job for a deployment slot.

Start-AzWebAppSlotTriggeredWebJob	Run a triggered web job for a deployment slot.
Start-AzWebAppTriggeredWebJob	Run a triggered web job for an app.
Stop-AzWebApp	Stops an Azure Web App.
Stop-AzWebAppContinuousWebJob	Stop a continuous web job for an app.
Stop-AzWebAppSlot	Stops an Azure Web App slot.
Stop-AzWebAppSlotContinuousWebJob	Stop a continuous web job for a deployment slot.
Switch-AzWebAppSlot	Swap two slots within a Web App
Test-AzStaticWebAppCustomDomain	Description for Validates a particular custom domain can be added to a static site.
Unregister-AzStaticWebAppBuildUserProvidedFunctionApp	Description for Detach the user provided function app from the static site build
Unregister-AzStaticWebAppUserProvidedFunctionApp	Description for Detach the user provided function app from the static site
Update-AzStaticWebApp	Description for Creates a new static site in an existing resource group, or updates an existing static site.
Update-AzStaticWebAppUser	Description for Updates a user entry with the listed roles
Update-AzWebAppAccessRestrictionConfig	Updates the inheritance of Main site Access Restiction config to SCM Site for an Azure Web App.
Update-AzWebAppTrafficRouting	Update a routing Rule to the Slot.

Azure App Service

Article • 05/07/2024

Azure App Service lets you run web apps, mobile app back ends, and API apps in Azure. For a more detailed overview, see the [Azure App Service product page](#).

Azure subscription and service limits, quotas, and constraints

Article • 01/23/2025

This document lists some of the most common Microsoft Azure limits, which are also sometimes called quotas.

- To learn more about Azure pricing, see the [Azure pricing](#) overview and details page.
- The Azure pricing page provides details for specific services; for example, [Windows Virtual Machines](#).
- You can also use the Azure [pricing calculator](#) to estimate your costs.
- See [What is Microsoft Billing?](#) for tips to help manage your costs.

How to manage limits

Note

Some services have adjustable limits.

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. Some services with adjustable limits use different headers with information about adjusting the limit.

When a service doesn't have adjustable limits, the following tables use the header **Limit** without any additional information about adjusting the limit. In those cases, the default and the maximum limits are the same.

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* are often 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 but only a hard limit.

[Free Azure trial subscriptions](#) aren't eligible for limit or quota increases. If you have this type of subscription, you can upgrade to a [Pay-as-you-go](#) one. For more information, see [Upgrade your Azure account](#) and the overviews for [Try Azure for free or pay as you go](#).

Some limits are managed at a regional level. You decide what your quotas must be for your workload in any one region, and then request that amount for each region into which you want to deploy.

For example, with virtual central processing unit (vCPU) quotas:

- To request a quota increase with support for vCPUs, you decide how many vCPUs 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 doesn't increase in any other region; only West Europe has the 30-vCPU quota.

See [Resolve errors for resource quotas](#) for more information about how to determine quotas for specific regions.

General limits

- See [Naming rules and restrictions for Azure resources](#) for limits on resource names.
- See [Understand how Azure Resource Manager throttles requests](#) to learn about Resource Manager API read and write limits.

Azure management group limits

The following limits apply to [Azure management groups](#).

Resource	Limit
Management groups per Microsoft Entra 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](#).

Azure subscription limits

The following limits apply when you use Azure Resource Manager and Azure resource groups.

[+] [Expand table](#)

Resource	Limit
Azure subscriptions associated with a Microsoft Entra 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 ²	80,000
Subscription-level deployments per location	800 ³
Locations of Subscription-level deployments	10

¹You can apply up to 50 tags directly to a subscription. Within the subscription, each resource or resource group is also limited to 50 tags. However, the subscription can contain an unlimited number of tags that are dispersed across resources and resource groups.

²Resource Manager returns a [list of tag name and values](#) in the subscription only when the number of unique tags is 80,000 or less. A unique tag is defined by the combination of resource ID, tag name, and tag value. For example, two resources with the same tag name and value would be calculated as two unique tags. You still can find a resource by tag when the number exceeds 80,000.

³Deployments are automatically deleted from the history as you near the limit. For more information, see [Automatic deletions from deployment history](#).

Azure resource group limits

[+] [Expand table](#)

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

Resource	Limit
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

[+] [Expand table](#)

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
Resource definition size	1 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](#).

Microsoft Entra ID limits

Here are the usage constraints and other service limits for the Microsoft Entra service.

[+] [Expand table](#)

Category	Limit
Tenants	<ul style="list-style-type: none"> A single user can belong to a maximum of 500 Microsoft Entra tenants as a member or a guest. Create a maximum of 200 tenants. Limit of 300 license-based subscriptions (such as Microsoft 365 subscriptions) per tenant
Domains	<ul style="list-style-type: none"> You can add no more than 5,000 managed domain names. If you set up all of your domains for federation with on-premises Active Directory, you can add no more than 2,500 domain names in each tenant.
Resources	<ul style="list-style-type: none"> By default, a maximum of 50,000 Microsoft Entra resources can be created in a single tenant by users of the Microsoft Entra ID Free edition. If you have at least one verified domain, the default Microsoft Entra service quota for your organization is extended to 300,000 Microsoft Entra resources. The Microsoft Entra service quota for organizations created by self-service sign-up remains 50,000 Microsoft Entra resources, even after you perform 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

Category	Limit
	<p>Microsoft Entra pricing page.</p> <p>To go beyond the default quota, you must contact Microsoft Support.</p> <ul style="list-style-type: none"> A non-admin user can create no more than 250 Microsoft Entra resources. Both active resources and deleted resources that are available to restore count toward this quota. Only deleted Microsoft Entra resources that were deleted fewer than 30 days ago are available to restore. Deleted Microsoft Entra 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. Resource limitations apply to all directory objects in a given Microsoft Entra tenant, including users, groups, applications, and service principals.
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 Microsoft Entra resource. Only User, Group, TenantDetail, Device, Application, and ServicePrincipal entities can be extended with string-type or binary-type single-valued attributes.
Applications	<ul style="list-style-type: none"> A maximum of 100 users and service principals can be owners of a single application. A user, group, or service principal can have a maximum of 1,500 app role assignments. The limitation is on the service principal, user, or group across all app roles and not on the number of assignments on a single app role. A user can have credentials configured for a maximum of 48 apps using password-based single sign-on. This limit only applies for credentials configured when the user is directly assigned the app, not when the user is a member of a group that is assigned. A group can have credentials configured for a maximum of 48 apps using password-based single sign-on. See more limits in Validation differences by supported account types.
Application manifest	<p>A maximum of 1,200 entries can be added to the application manifest.</p> <p>See more limits in Validation differences by supported account types.</p>
Groups	<ul style="list-style-type: none"> A non-admin user can create a maximum of 250 groups in a Microsoft Entra organization. Any Microsoft Entra admin who can manage groups in the organization can also create an unlimited number of groups (up to the Microsoft Entra object limit). If you assign a role to a user to remove the limit for that user, assign a less privileged, built-in role such as User Administrator or Groups Administrator. A Microsoft Entra organization can have a maximum of 15,000 dynamic groups and dynamic administrative units combined. A maximum of 500 role-assignable groups can be created in a single Microsoft Entra organization (tenant). A maximum of 100 users can be owners of a single group. Any number of Microsoft Entra resources can be members of a single group. A user can be a member of any number of groups. When security groups are being used in combination with SharePoint Online, a user can be a part of 2,049 security groups in total. This includes both direct and indirect group memberships. When this limit is exceeded, authentication and search results become unpredictable. By default, the number of members in a group that you can synchronize from your on-premises Active Directory to Microsoft Entra ID by using Microsoft Entra Connect is limited to 50,000 members. If you need to sync a group membership that's over this limit, you must onboard the Microsoft Entra Connect Sync V2 endpoint API. When you select a list of groups, you can assign a group expiration policy to a maximum of 500 Microsoft 365 groups. There is no limit when the policy is applied to all Microsoft 365 groups.
	<p>At this time, the following scenarios are supported 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 Microsoft Entra join and device registration.
	<p>The following scenarios are <i>not</i> supported with nested groups:</p> <ul style="list-style-type: none"> App role assignment, for both access and provisioning. Assigning groups to an app is supported, but any groups nested within the directly assigned group won't have access. Group-based licensing (assigning a license automatically to all members of a group). Microsoft 365 Groups.
Application Proxy	<ul style="list-style-type: none"> A maximum of 500 transactions* per second per Application Proxy application. A maximum of 750 transactions per second for the Microsoft Entra organization.

Category	Limit
	*A transaction is defined as a single HTTP request and response for a unique resource. When clients are throttled, they'll receive a 429 response (too many requests). Transaction metrics are collected on each connector and can be monitored using performance counters under the object name <code>Microsoft AAD App Proxy Connector</code> .
Access Panel	There's no limit to the number of applications per user that can be displayed in the Access Panel, regardless of the number of assigned licenses.
Reports	A maximum of 1,000 rows can be viewed or downloaded in any report. Any other data is truncated.
Administrative units	<ul style="list-style-type: none"> A Microsoft Entra resource can be a member of no more than 30 administrative units. A maximum of 100 restricted management administrative units in a tenant. A Microsoft Entra organization can have a maximum of 15,000 dynamic groups and dynamic administrative units combined.
Microsoft Entra roles and permissions	<ul style="list-style-type: none"> A maximum of 100 Microsoft Entra custom roles can be created in a Microsoft Entra organization. A maximum of 150 Microsoft Entra custom role assignments for a single principal at any scope. A maximum of 100 Microsoft Entra built-in role assignments for a single principal at non-tenant scope (such as an administrative unit or Microsoft Entra object). There is no limit to Microsoft Entra built-in role assignments at tenant scope. For more information, see Assign Microsoft Entra roles at different scopes. 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 Microsoft Entra 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 might take up to 15 minutes or you might have to sign out and sign back in before admin role membership additions and revocations take effect.
Conditional Access Policies	A maximum of 195 policies can be created in a single Microsoft Entra organization (tenant).
Terms of use	You can add no more than 40 terms to a single Microsoft Entra organization (tenant).
Multitenant organizations	<ul style="list-style-type: none"> A maximum of 5 active tenants, including the owner tenant. The owner tenant can add more than 5 pending tenants, but they won't be able to join the multitenant organization if the limit is exceeded. This limit is applied at the time a pending tenant joins a multitenant organization. A maximum of 100,000 internal users per active tenant. This limit is applied at the time a pending tenant joins a multitenant organization.

Azure API Center limits

[Expand table](#)

Resource	Free plan ¹	Standard plan ²
Maximum number of APIs	200 ³	10,000
Maximum number of versions per API	5	100
Maximum number of definitions per version	5	5
Maximum number of deployments per API	10	10
Maximum number of environments	20	20
Maximum number of workspaces	1 (Default)	1 (Default)
Maximum number of custom metadata properties per entity ³	10	20
Maximum number of child properties in custom metadata property of type "object"	10	10
Maximum requests per minute (data plane)	3,000	6,000
Maximum number of APIs accessed through data plane API	5	10,000
Maximum number of API definitions analyzed	10	2,000 ⁴
Maximum number of linked API sources ⁵	1	3

Resource	Free plan ¹	Standard plan ²
Maximum number of APIs synchronized from a linked API source	200	2,000 ⁴

¹ Free plan provided for 90 days, then service is soft-deleted. Use of full service features including API analysis and access through the data plane API is limited.

² To increase a limit in the Standard plan, contact [support](#).

³ Custom metadata properties assigned to APIs, deployments, and environments.

⁴ Process can take a few minutes to up to 24 hours to complete.

⁵ Sources such as linked API Management instances.

Azure API Management limits

This section provides information about limits that apply to Azure API Management instances in different [service tiers](#), including the following:

- [API Management classic tiers](#)
- [API Management v2 tiers](#)
- [API Management workspaces](#)
- [Developer portal in API Management v2 tiers](#)

Limits - API Management classic tiers

For certain API Management resources, limits are set only in the Consumption tier; in other API Management classic tiers, where indicated, these resources are unlimited. However, your practical upper limit depends on service configuration including pricing tier, service capacity, number of scale units, policy configuration, API definitions and types, number of concurrent requests, and other factors.

To request a limit increase, create a support request from the Azure portal. For more information, see [Azure support plans](#).

[Expand table](#)

Resource	Consumption	Developer	Basic	Standard	Premium
Maximum number of scale units	N/A (automatic scaling)	1	2	4	31 per region
Cache size (per unit)	External only	10 MiB	50 MiB	1 GiB	5 GiB
Concurrent back-end connections ¹ per HTTP authority	Unlimited	1,024	2,048 per unit	2,048 per unit	2,048 per unit
Maximum cached response size	2 MiB	2 MiB	2 MiB	2 MiB	2 MiB
Maximum policy document size	16 KiB	256 KiB	256 KiB	256 KiB	256 KiB
Maximum custom gateway domains per service instance	N/A	20	N/A	N/A	20
Maximum number of CA certificates per service instance	N/A	10	10	10	10
Maximum number of service instances per Azure subscription	20	Unlimited	Unlimited	Unlimited	Unlimited
Maximum number of subscriptions per service instance	500	Unlimited	Unlimited	Unlimited	Unlimited
Maximum number of client certificates per service instance	50	Unlimited	Unlimited	Unlimited	Unlimited
Maximum number of APIs per service instance	50	Unlimited	Unlimited	Unlimited	Unlimited
Maximum number of API operations per service instance	1,000	Unlimited	Unlimited	Unlimited	Unlimited
Maximum total request duration	30 seconds	Unlimited	Unlimited	Unlimited	Unlimited
Maximum request payload size	1 GiB	Unlimited	Unlimited	Unlimited	Unlimited
Maximum buffered payload size	2 MiB	500 MiB	500 MiB	500 MiB	500 MiB

Resource	Consumption	Developer	Basic	Standard	Premium
Maximum request/response payload size in diagnostic logs	8,192 bytes	8,192 bytes	8,192 bytes	8,192 bytes	8,192 bytes
Maximum request URL size ²	16,384 bytes	Unlimited	Unlimited	Unlimited	Unlimited
Maximum character length of URL path segment	1,024	1,024	1,024	1,024	1,024
Maximum character length of named value	4,096	4,096	4,096	4,096	4,096
Maximum size of API schema used by validation policy	4 MB	4 MB	4 MB	4 MB	4 MB
Maximum number of schemas	100	100	100	100	100
Maximum size of request or response body in validate-content policy	100 KiB	100 KiB	100 KiB	100 KiB	100 KiB
Maximum number of self-hosted gateways ³	N/A	25	N/A	N/A	25
Maximum number of active WebSocket connections per unit ⁴	N/A	2,500	5,000	5,000	5,000
Maximum number of tags supported by an API Management resource	15	15	15	15	15
Maximum number of credential providers per service instance	1,000	1,000	1,000	1,000	1,000
Maximum number of connections per credential provider	10,000	10,000	10,000	10,000	10,000
Maximum number of access policies per connection	100	100	100	100	100
Maximum number of authorization requests per minute per connection	250	250	250	250	250
Maximum number of workspaces per service instance	N/A	N/A	N/A	N/A	100

¹ Connections are pooled and reused unless explicitly closed by the backend.

² Includes an up to 2048-bytes long query string.

³ 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.

⁴ Up to a maximum of 60,000 connections per service instance.

Limits - API Management v2 tiers

To request a limit increase, create a support request from the Azure portal. For more information, see [Azure support plans](#).

[\[\]](#) Expand table

Resource	Basic v2	Standard v2	Premium v2
Maximum number of scale units	10	10	30
Maximum cache size per service instance	250 MB	1 GB	5 GB
Maximum number of APIs per service instance	150	500	2,500
Maximum number of API operations per service instance	3,000	10,000	20,000
Maximum number of subscriptions per service instance	500	2,000	4,000
Maximum number of products per service instance	50	200	400
Maximum number of users per service instance	300	2,000	4,000
Maximum number of groups per service instance	20	100	200
Maximum number of authorization servers per service instance	10	500	500
Maximum number of policy fragments per service instance	50	50	100
Maximum number of OpenID Connect providers per service instance	10	10	20

Resource	Basic v2	Standard v2	Premium v2
Maximum number of certificates per service instance	100	100	100
Maximum number of backends per service instance	100	100	100
Maximum number of caches per service instance	100	100	100
Maximum number of named values per service instance	100	100	100
Maximum number of loggers per service instance	100	100	100
Maximum number of schemas per service instance	100	100	100
Maximum number of schemas per API	100	100	100
Maximum number of tags per service instance	100	100	100
Maximum number of tags per API	100	100	100
Maximum number of version sets per service instance	100	100	100
Maximum number of releases per API	100	100	100
Maximum number of operations per API	100	100	100
Maximum number of GraphQL resolvers per service instance	100	100	100
Maximum number of GraphQL resolvers per API	100	100	100
Maximum number of APIs per product	100	100	100
Maximum number of APIs per subscription	100	100	100
Maximum number of products per subscription	100	100	100
Maximum number of groups per product	100	100	100
Maximum number of tags per product	100	100	100
Concurrent back-end connections ¹ per HTTP authority	2,048	2,048	2,048
Maximum cached response size	2 MiB	2 MiB	2 MiB
Maximum policy document size	256 KiB	256 KiB	256 KiB
Maximum request payload size	1 GiB	1 GiB	1 GiB
Maximum buffered payload size	2 MiB	2 MiB	2 MiB
Maximum request/response payload size in diagnostic logs	8,192 bytes	8,192 bytes	8,192 bytes
Maximum request URL size ²	16,384 bytes	16,384 bytes	16,384 bytes
Maximum length of URL path segment	1,024 characters	1,024 characters	1,024 characters
Maximum character length of named value	4,096 characters	4,096 characters	4,096 characters
Maximum size of request or response body in validate-content policy	100 KiB	100 KiB	100 KiB
Maximum size of API schema used by validation policy	4 MB	4 MB	4 MB
Maximum number of active WebSocket connections per unit ³	5,000	5,000	5,000

¹ Connections are pooled and reused unless explicitly closed by the backend.

² Includes an up to 2048-bytes long query string.

³ Up to a maximum of 60,000 connections per service instance.

Limits - API Management workspaces

The following are resource limits per [workspace](#) in Azure API Management:

[+] [Expand table](#)

Resource	Workspace - Premium tier
Maximum number of workspaces per instance	100
Maximum number of scale units per premium workspace gateway	12
Maximum number of APIs (including versions and revisions)	200
Maximum number of API operations	5,000
Maximum number of operations per API	100
Maximum number of releases per API	100
Maximum number of schemas per API	100
Maximum number of subscriptions per API	200
Maximum number of tags per API	100
Maximum number of backends	200
Maximum number of certificates	200
Maximum number of groups	50
Maximum number of loggers	50
Maximum number of named values	200
Maximum number of policy fragments	50
Maximum number of products	100
Maximum number of APIs per product	200
Maximum number of groups per product	200
Maximum number of subscriptions per product	1,000
Maximum number of tags per product	50
Maximum number of schemas	500
Maximum number of subscriptions	5,000
Maximum number of tags	200
Maximum number of groups per user	200
Maximum number of version sets	50

Limits - Developer portal in API Management v2 tiers

[Expand table](#)

Item	Basic v2	Standard v2	Premium v2
Maximum number of media files to upload	15	15	15
Maximum size of a media file	500 KB	500 KB	500 KB
Maximum number of pages	30	50	50
Maximum number of widgets ¹	30	50	50
Maximum size of metadata per page	350 KB	350 KB	350 KB
Maximum size of metadata per widget ¹	350 KB	350 KB	350 KB
Maximum number of client requests per minute	200	200	200

¹ Limit for built-in widgets such as text, images, or APIs list. Currently, custom widgets and custom HTML code widgets aren't supported in the v2 tiers.

Azure App Service limits

[Expand table](#)

Resource	Free	Shared	Basic	Standard	Premium (v1-v3)	Isolated
Web, mobile, or API apps ¹ per Azure App Service plan	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 ¹² 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
WebSockets per instance (Windows) ⁷	5	35	350	Unlimited	Unlimited	Unlimited
WebSockets per instance (Linux) ⁷	5	N/A	~50K	~50K	~50K	~50K
Outbound IP connections per instance	600	600	Depends on instance size ⁸	Depends on instance size ⁸	Depends on instance size ⁸	16,000
Concurrent debugger connections per application	1	1	1	5	5	5
App Service Certificates per subscription	Not supported	Not supported	10	10	10	10
Custom domains per app	0 (azurewebsites.net subdomain only)	500	500	500	500	500
Custom domain SSL support	Not supported, wildcard certificate for *.azurewebsites.net available by default	Not supported, wildcard certificate for *.azurewebsites.net available by default	Unlimited SNI SSL connections	Unlimited SNI SSL and 1 IP SSL connections included	Unlimited SNI SSL and 1 IP SSL connections included	Unlimited SNI SSL and 1 IP SSL connections included

Resource	Free	Shared	Basic	Standard	Premium (v1-v3)	Isolated
Hybrid connections			5 per plan	25 per plan	220 per app	220 per app
Virtual Network Integration			X	X	X	X
Private Endpoints			100 per app	100 per app	100 per app	
Integrated load balancer	X	X	X	X	X	X ⁹
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 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)	Scheduled backups every hour, a maximum of 50 backups per day (manual + scheduled)
Autoscale			X	X	X	X
WebJobs ¹⁰	X	X	X	X	X	X
Endpoint monitoring		X	X	X	X	X
Staging slots per app			5	20	20	
Testing in Production			X	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 ¹¹			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).

⁷If you scale a Windows app in the Basic tier to two instances, you have 350 concurrent connections for each of the two instances. For Windows apps on Standard tier and above, there are no theoretical limits to WebSockets, but other factors can limit the number of WebSockets. 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). Linux apps are limited 5 concurrent WebSocket connections on Free SKU and ~50k concurrent WebSocket connections per instance on all other SKUs.

⁸ The maximum IP connections are per instance and depend on the instance size: 1,920 per B1/S1/P0V3/P1V3 instance, 3,968 per B2/S2/P2V3 instance, 8,064 per B3/S3/P3V3 instance.

⁹ 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.

¹⁰ 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.

¹¹ Only issuing standard certificates (wildcard certificates aren't available). Limited to only one free certificate per custom domain.

¹² Total storage usage across all apps deployed in a single App Service Environment (regardless of how they're allocated across different resource groups).

Azure Automation limits

Process automation

[Expand table](#)

Resource	Limit	Notes
Maximum number of active Automation accounts in a subscription in a region	10	Enterprise and CSP subscriptions can create Automation accounts in any of the public regions supported by the service . Create a Support request to request for Quota increase. Learn more .
	2	Pay-as-you-go, Sponsored, MSDN, MPN, Azure Pass subscriptions can create Automation accounts in any of the public regions supported by the service. Create a Support request to request for Quota increase. Learn more .
	1	Free trial and Azure for Student subscriptions can create only one Automation account per region per subscription. Allowed list of regions: EastUS, EastUS2, WestUS, NorthEurope, SoutheastAsia, and JapanWest2 ²
Maximum number of concurrent running jobs at the same instance of time per Automation account	50	When this limit is reached, the subsequent requests to create a job fail. The client receives an error response. Enterprise and CSP subscription in public regions. Create a Support request to request for Quota increase. Learn more .
	10	Pay-as-you-go, Sponsored, MSDN, MPN, Azure Pass subscriptions in public regions. Create a support request to request for a Quota increase.
	5	Free trial and Azure for Student Azure in open subscriptions in public regions ² .
Maximum number of new jobs that can be submitted every 30 seconds per Azure Automation account	100	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 job stream limit on Azure Automation portal	200KB	Portal limit to show the job logs.
Maximum number of modules that can be imported every 30 seconds per Automation account	5	
Maximum size of a module	100 MB	

Resource	Limit	Notes
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.
Maximum runtime allowed per runbook ¹	3 hours	Applies to Azure sandboxes only.
Maximum number of system hybrid runbook workers per Automation Account	4,000	
Maximum number of user hybrid runbook workers per Automation Account	4,000	
Maximum number of concurrent jobs that can be run on a single Hybrid Runbook Worker	50	
Maximum runbook job parameter size	512 kilobytes	
Maximum runbook parameters	50	If you reach the 50-parameter limit, you can pass a JSON or XML string to a parameter and parse it with the runbook.
Maximum webhook payload size	512 kilobytes	
Maximum days that job data is retained	30 days	
Maximum PowerShell workflow state size	5 MB	Applies to PowerShell workflow runbooks when checkpointing workflow.
Maximum number of tags supported by an Automation account	15	
Maximum number of characters in the value field of a variable	1048576	

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.

²Free subscriptions including [Azure Free Account](#) and [Azure for Students](#) aren't eligible for limit or quota changes. If you have a free subscription, you can [upgrade](#) to pay-as-you-go subscription. ³Limits for Government clouds: 200 concurrent running jobs at the same instance of time per Automation account, no limit on number of Automation accounts per subscription.

Change Tracking and Inventory

The following table shows the tracked item limits per machine for change tracking.

[Expand table](#)

Resource	Limit	Notes
File	500	
File size	5 MB	
Registry	250	

Resource	Limit	Notes
Windows software	250	Doesn't include software updates.
Linux packages	1,250	
Services	250	
Daemon	250	

Azure Update Manager

The following are the Dynamic scope recommended limits for **each dynamic scope**:

[Expand table](#)

Resource	Limit
Resource associations	1000
Number of tag filters	50
Number of Resource Group filters	50

The following are the limits for schedule patching:

[Expand table](#)

Indicator	Public Cloud Limit	Mooncake/Fairfax Limit
Number of schedules per subscription per region	250	250
Total number of resource associations to a schedule	3,000	3,000
Resource associations on each dynamic scope	1,000	1,000
Number of dynamic scopes per resource group or subscription per region	250	250
Number of dynamic scopes per schedule	200	100
Total number of subscriptions attached to all dynamic scopes per schedule	200	100

Azure App Configuration

[Expand table](#)

Resource	Limit	Comment
Configuration stores for Free tier	One store per region per subscription.	
Configuration stores for Standard tier	Unlimited stores per subscription.	
Configuration stores for Premium tier	Unlimited stores per subscription.	
Configuration store requests for Free tier	1,000 requests per day	Once the quota is exhausted, HTTP status code 429 is returned for all requests until the end of the day.
Configuration store requests for Standard tier	30,000 per hour	Once the quota is exhausted, requests may return HTTP status code 429 indicating Too Many Requests - until the end of the hour.
Configuration store requests for Premium tier	No quota limit on requests.	
Throughput for Free tier	No guaranteed throughput.	

Resource	Limit	Comment
Throughput for Standard tier	Allow up to 300 requests per second (RPS) for read requests and up to 60 RPS for write requests.	
Throughput for Premium tier	Allow up to 450 requests per second (RPS) for read requests and up to 100 RPS for write requests.	
Storage for Free tier	10 MB	There is no limit on the number of keys and labels as long as their total size is below the storage limit.
Storage for Standard tier	1 GB	There is no limit on the number of keys and labels as long as their total size is below the storage limit.
Storage for Premium tier	4 GB	There is no limit on the number of keys and labels as long as their total size is below the storage limit.
Keys and values	10 KB	For a single key-value item, including all metadata.
Snapshots storage for Free tier	10 MB	Snapshots storage is extra and in addition to "Storage for Free Tier". Storage for both archived and active snapshots is counted towards this limit.
Snapshots storage for Standard tier	1 GB	Snapshots storage is extra and in addition to "Storage for Standard Tier". Storage for both archived and active snapshots is counted towards this limit.
Snapshots storage for Premium tier	4 GB	Snapshots storage is extra and in addition to "Storage for Premium Tier". Storage for both archived and active snapshots is counted towards this limit.
Snapshot size	1 MB	

Azure Cache for Redis limits

[Expand table](#)

Resource	Limit
Cache size	1.2 TB
Databases	64
Maximum connected clients	40,000
Azure Cache for Redis replicas, for high availability	3
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

[Expand table](#)

Resource	Limit
Web or worker roles per deployment ¹	25
Instance input endpoints per deployment	25

Resource	Limit
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 AI Search limits

Pricing tiers determine the capacity and limits of your search service. These tiers include:

- **Free:** Multitenant service that's shared with other Azure subscribers and helps with evaluations and small development projects
- **Basic:** Provides dedicated computing resources for production workloads at a smaller scale and with up to three replicas for highly available query workloads
- **Standard:** Includes S1, S2, S3, and S3 High Density; is for larger production workloads; multiple levels exist within the Standard tier for you to choose a resource configuration that best matches your workload profile

Limits per subscription

You can create multiple *billable* search services (Basic and higher), up to the maximum number of services allowed at each tier, per region. For example, you could create up to 16 services at the Basic tier and another 16 services at the S1 tier within the same subscription and region. You could then create an additional 16 Basic services in another region for a combined total of 32 Basic services under the same subscription. For more information about tiers, see [Choose a tier \(or SKU\) for Azure AI Search](#).

Maximum service limits can be raised upon request. If you need more services within the same subscription, [file a support request](#).

[Expand table](#)

Resource	Free ¹	Basic	S1	S2	S3	S3 HD	L1	L2
Maximum services per region	1	16	16	8	6	6	6	6
Maximum search units (SU) ²	N/A	3 SU	36 SU	36 SU	36 SU	36 SU	36 SU	36 SU

¹ You can have one free search service per Azure subscription. The free tier is based on infrastructure shared with other customers. Because the hardware isn't dedicated, scale-up isn't supported, and storage is limited to 50 MB. A free search service might be deleted after extended periods of inactivity to make room for more services.

² Search units (SU) are billing units, allocated as either a *replica* or a *partition*. You need both. To learn more about SU combinations, see [Estimate and manage capacity of a search service](#).

Limits per search service

The following table covers SLA, partition counts, and replica counts at the service level.

[Expand table](#)

Resource	Free	Basic	S1	S2	S3	S3 HD	L1	L2
Service level agreement (SLA)	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Partitions	N/A	3 ¹	12	12	12	3	12	12
Replicas	N/A	3	12	12	12	12	12	12

¹ Basic tier supports three partitions and three replicas, for a total of nine search units (SU) on [new search services](#) created after April 3, 2024. Older basic services are limited to one partition and three replicas.

A search service is subject to a maximum storage limit (partition size multiplied by the number of partitions) or by a hard limit on the [maximum number of indexes](#) or [indexers](#), whichever comes first.

Service level agreements (SLAs) apply to billable services having two or more replicas for query workloads, or three or more replicas for query and indexing workloads. The number of partitions isn't an SLA consideration. For more information, see [Reliability in Azure AI Search](#).

Free services don't have fixed partitions or replicas and they share resources with other subscribers.

Partition storage (GB)

Per-service storage limits vary by two things: [service creation date](#), and [region](#). There are higher limits for newer services in most supported regions.

This table shows the progression of storage quota increases in GB over time. Higher capacity partitions were brought online starting in April 2024, in the regions listed in the footnotes. Higher capacity is limited to [new search services](#). There's no in-place upgrade at this time.

Service creation date	Basic	S1	S2	S3/HD	L1	L2
Before April 3, 2024	2	25	100	200	1,024	2,048
April 3, 2024 through May 17, 2024 ¹	15	160	512	1,024	1,024	2,048
After May 17, 2024 ²	15	160	512	1,024	2,048	4,096

¹ Higher capacity storage for Basic, S1, S2, S3 in these regions. **Americas:** Brazil South, Canada Central, Canada East, East US, East US 2, Central US, North Central US, South Central US, West US, West US 2, West US 3, West Central US. **Europe:** France Central, Italy North, North Europe, Norway East, Poland Central, Switzerland North, Sweden Central, UK South, UK West. **Middle East:** UAE North. **Africa:** South Africa North. **Asia Pacific:** Australia East, Australia Southeast, Central India, Jio India West, East Asia, Southeast Asia, Japan East, Japan West, Korea Central, Korea South.

² Higher capacity storage for L1 and L2. More regions provide higher capacity at every billable tier. **Europe:** Germany North, Germany West Central, Switzerland West. **Azure Government:** Texas, Arizona, Virginia. **Africa:** South Africa North. **Asia Pacific:** China North 3, China East 3.

A few regions still run on older infrastructure, subject to the April 3 limits. Before creating a new service, check [supported regions](#) to make sure your region of choice provides the extra capacity.

See [Service limits in Azure AI Search](#) for more details about limits, including document size, queries per second, keys, requests, and responses.

Azure AI Services limits

The following limits are for the number of Azure AI services resources per Azure subscription. There is a limit of only one allowed 'Free' account, per resource type, per subscription. Each of the Azure AI services may have other limitations, for more information, see [Azure AI services](#).

Type	Limit	Example
A mixture of Azure AI services resources	Maximum of 200 total Azure AI services resources per region.	100 Azure AI Vision resources in West US, 50 Azure AI Speech resources in West US, and 50 Azure AI Language resources in West US.
A single type of Azure AI services resources.	Maximum of 100 resources per region	100 Azure AI Vision resources in West US 2, and 100 Azure AI Vision resources in East US.

Azure Chaos Studio limits

See [Azure Chaos Studio service limits](#) for Azure Chaos Studio limits.

Azure Communications Gateway limits

Some of the following default limits and quotas can be increased. To request a change, create an [Azure portal support request](#), and describe the limit that you need to change.

The following restrictions apply to all Azure Communications Gateways:

- All traffic must use IPv4.
- All traffic must use TLS 1.2 or greater. Earlier versions aren't supported.
- The number of active calls is limited to 15% of the number of users assigned to Azure Communications Gateway. For the definition of users, see [Plan and manage costs for Azure Communications Gateway](#).
- The number of calls being actively transcoded is limited to 5% of the total number of active calls.

Azure Communications Gateway also has limits on SIP signaling.

[Expand table](#)

Resource	Limit
Maximum SIP message size	10 Kilobytes
Maximum length of an SDP message body	128 Kilobytes
Maximum length of request URI	256 Bytes
Maximum length of Contact header URI	256 Bytes
Maximum length of the userinfo part of a URI	256 Bytes
Maximum length of domain name in From header	255 Bytes
Maximum length of a SIP header's name	32 Bytes
Maximum length of a SIP body name	64 Bytes
Maximum length of a Supported, Require or Proxy-Require header	256 Bytes
Maximum length of a SIP option-tag	32 Bytes

Some endpoints might add parameters in the following headers to an in-dialog message when those parameters weren't present in the dialog-creating message. In that case, Azure Communications Gateway strips the parameters, because RFC 3261 doesn't permit this behavior.

- Request URI
- To header
- From header

The Provisioning API has a rate limit of 100 requests per minute, applied across all the resources. A batch request to update multiple resources counts as one request.

Azure Container Apps limits

See [Quotas in Azure Container Apps](#) for Azure Container Apps limits.

The amount of disk space available to your application varies based on the associated workload profile. Available disk space determines the image size limit you can deploy to your container apps.

For dedicated workload profiles, the image size limit is per instance.

[Expand table](#)

Display name	Name	Image Size Limit (GB)
Consumption	consumption	8*
Dedicated-D4	D4	90
Dedicated-D8	D8	210
Dedicated-D16	D16	460
Dedicated-D32	D32	940
Dedicated-E4	E4	90
Dedicated-E8	E8	210
Dedicated-E16	E16	460
Dedicated-E32	E32	940
Dedicated-NC24-A100 (preview)	NC24-A100	210
Dedicated-NC48-A100 (preview)	NC48-A100	460
Dedicated-NC96-A100 (preview)	NC96-A100	940

* The image size limit for a consumption workload profile is a shared among both image and app. For example, logs used by your app are subject to this size limit.

Azure Cosmos DB limits

See [Limits in Azure Cosmos DB](#) for Azure Cosmos DB limits.

Azure Data Explorer limits

The following table describes the maximum limits for Azure Data Explorer clusters.

[Expand table](#)

Resource	Limit
Clusters per region per subscription	20
Instances per cluster	1,000
Number of databases in a cluster	10,000
Number of follower clusters (data share consumers) per leader cluster (data share producer)	100

Note

You can request higher limits for *Number of databases in a cluster* and *Clusters per region per subscription*. To request an increase, contact [Azure Support](#).

The following table describes the limits on management operations performed on Azure Data Explorer clusters.

[Expand table](#)

Scope	Operation	Limit
Cluster	read (for example, get a cluster)	500 per 5 minutes
Cluster	write (for example, create a database)	1,000 per hour

Azure Database for MySQL

See [Limitations in Azure Database for MySQL](#) for Azure Database for MySQL limits.

Azure Database for PostgreSQL

See [Limitations in Azure Database for PostgreSQL](#) for Azure Database for PostgreSQL limits.

Azure Deployment Environments limits

 Expand table

Subscription	Runtime limit per deployment	Runtime limit per month per region per subscription	Storage limit per Environment
Enterprise	30 min	5000 min	1 GB
Pay as you go	10 min	200 min	1 GB
Azure Pass	10 min	200 min	1 GB
MSDN	10 min	200 min	1 GB
CSP	10 min	200 min	1 GB
Free trial	10 min	200 min	1 GB
Azure for students	10 min	200 min	1 GB

Azure Files and Azure File Sync

See [Scalability and performance targets for Azure Files and Azure File Sync](#) to learn more about the limits for Azure Files and Azure File Sync.

Azure Functions limits

 Expand table

Resource	Flex Consumption plan	Premium plan	Dedicated plan/ASE	Container Apps	Consumption plan
Default timeout duration (min)	30	30	30 ¹	30 ¹⁶	5
Max timeout duration (min)	unbounded ⁹	unbounded ⁹	unbounded ²	unbounded ¹⁷	10
Max outbound connections (per instance)	unbounded	unbounded	unbounded	unbounded	600 active (1200 total)
Max request size (MB) ³	210	210	210	210	210
Max query string length ³	4096	4096	4096	4096	4096
Max request URL length ³	8192	8192	8192	8192	8192
ACU per instance	210-840	100-840/210-250 ¹⁰	varies	100	varies
Max memory (GB per instance)	4⁴	3.5-14	1.75-256/8-256	varies	1.5
Max instance count (Windows/Linux)	100/20	varies by SKU/100 ¹¹	10-300 ¹⁸	200/100	1000 ¹⁵
Function apps per	100	100	unbounded ⁴	unbounded ⁴	100

Resource	Flex Consumption plan	Premium plan	Dedicated plan/ASE	Container Apps	Consumption plan
plan ¹³					
App Service plans	n/a	100 per resource group	100 per resource group	n/a	100 per region ²
Deployment slots per app ¹²	n/a	3	1-20 ¹¹	not supported	2
Storage (temporary) ⁵	0.8 GB	21-140 GB	11-140 GB	n/a	0.5 GB
Storage (persisted)	0 GB ⁷	250 GB	10-1000 GB ¹¹	n/a	1 GB ^{6,7}
Custom domains per app	500	500	500	not supported	500 ⁷
Custom domain SSL support	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	not supported	unbounded SNI SSL connection included

Notes on service limits:

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](#). A grace period of 10 minutes is given during platform updates.
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. For Consumption plans on Linux, the storage is currently 1.5 GB.
6. Consumption plan uses an Azure Files share for persisted storage. When you provide your own Azure Files share, the specific share size limits depend on the storage account you set for [WEBSITE_CONTENTAZUREFILECONNECTIONSTRING](#).
7. On Linux, you must [explicitly mount your own Azure Files share](#).
8. 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.
9. There is no maximum execution timeout duration enforced. However, the grace period given to a function execution is 60 minutes [during scale in](#) and 10 minutes during platform updates.
10. 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.
11. See [App Service limits](#) for details.
12. Including the production slot.
13. There's currently a limit of 5000 function apps in a given subscription.
14. Flex Consumption plan instance sizes are currently defined as either 2,048 MB or 4,096 MB. For more information, see [Instance memory](#).
15. Flex Consumption plan has a regional subscription quota that limits the total memory usage of all instances across a given region. For more information, see [Instance memory](#).
16. When the [minimum number of replicas](#) is set to zero, the default timeout depends on the specific triggers used in the app.
17. When the [minimum number of replicas](#) is set to one or more.
18. On Container Apps, you can set the [maximum number of replicas](#), which is honored as long as there's enough cores quota available.

See [Azure Functions hosting options](#) for more information.

Azure Health Data Services

Azure Health Data Services limits

Health Data Services is a set of managed API services based on open standards and frameworks. Health Data Services enables workflows to improve healthcare and offers scalable and secure healthcare solutions. Health Data Services includes Fast Healthcare Interoperability Resources (FHIR) service, the Digital Imaging and Communications in Medicine (DICOM) service, and MedTech service.

FHIR service is an implementation of the FHIR specification within Health Data Services. It enables you to combine in a single workspace one or more FHIR service instances with optional DICOM and MedTech service instances. Azure API for FHIR is generally available as a stand-alone service offering.

Each FHIR service instance in Azure Health Data Services has a storage limit of 4 TB by default. If you have more data, you can ask Microsoft to increase storage up to 100 TB for your FHIR service. To request storage greater than 4 TB, [create a support request](#) on the Azure portal and use the issue type Service and Subscription limit (quotas).

[\[+\] Expand table](#)

Quota Name	Default Limit	Maximum Limit	Notes
Workspace	10	Contact support	Limit per subscription
FHIR	10	Contact support	Limit per workspace
DICOM	10	Contact support	Limit per workspace
MedTech	10	N/A	Limit per workspace, can't be increased

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.

[\[+\] Expand table](#)

Quota Name	Default Limit	Maximum Limit	Notes
Request Units (RUs)	100,000 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 Kubernetes Service limits

[\[+\] Expand table](#)

Resource	Limit
Maximum clusters per subscription globally	5,000
Maximum clusters per subscription per region ¹	100
Maximum nodes per cluster with Virtual Machine Scale Sets and Standard Load Balancer SKU	5,000 across all node pools Note: If you're unable to scale up to 5,000 nodes per cluster, see Best Practices for Large Clusters .
Maximum nodes per node pool (Virtual Machine Scale Sets node pools)	1000
Maximum node pools per cluster	100
Maximum pods per node: with Kubenet networking plug-in ¹ (Azure CNI) ²	Maximum: 250 Azure CLI default: 110 Azure Resource Manager template default: 110 Azure portal deployment default: 30
Maximum pods per node: with Azure Container Networking Interface (Azure CNI) ²	Maximum: 250 Maximum recommended for Windows Server containers: 110 Default: 30

Resource	Limit
Open Service Mesh (OSM) AKS addon	Kubernetes Cluster Version: AKS Supported Versions OSM controllers per cluster: 1 Pods per OSM controller: 1600 Kubernetes service accounts managed by OSM: 160
Maximum load-balanced kubernetes services per cluster with Standard Load Balancer SKU	300
Maximum nodes per cluster with Virtual Machine Availability Sets and Basic Load Balancer SKU	100

¹ More are allowed upon request.

² Windows Server containers must use Azure CNI networking plug-in. Kubenet isn't supported for Windows Server containers.

[\[+\] Expand table](#)

Kubernetes Control Plane tier	Limit
Standard tier	Automatically scales Kubernetes API server based on load. Larger control plane component limits and API server/etcdb instances.
Free tier	Limited resources with inflight requests limit of 50 mutating and 100 read-only calls. Recommended node limit of 10 nodes per cluster. Best for experimenting, learning, and simple testing. Not advised for production/critical workloads.

Azure Lab Services

The following limits are for the number of Azure Lab Services resources.

Per resource type

[\[+\] Expand table](#)

Grouping	Resource type	Limit
Per subscription	Labs	980
Per resource group	Labs	800
	Lab plans	800
Per lab	Schedules	250
	Virtual machines (VMs)	400

Per region - Lab plans and labs

[\[+\] Expand table](#)

Subscription type	Lab plan limits	Lab limits
Default	2	2
Pay As You Go	500	500
MPN	500	500
Azure In Open	500	500
Enterprise Agreement	500	500
MSDN	500	500
Sponsored	100	15

Subscription type	Lab plan limits	Lab limits
CSP	500	500
Azure Pass	100	25
Free Trial	100	15
Azure for Students	100	15

For more information about Azure Lab Services capacity limits, see [Capacity limits in Azure Lab Services](#).

Contact support to request an increase your limit.

Azure Load Testing limits

See [Service limits in Azure Load Testing](#) for Azure Load Testing limits.

Azure Machine Learning limits

See [Manage and increase quotas and limits for resources with Azure Machine Learning](#) for the latest values for Azure Machine Learning Compute quotas.

Azure Maps limits

! Note

Azure Maps Gen1 Price Tier Retirement

Gen1 pricing tier is now deprecated and will be retired on 9/15/26. Gen2 pricing tier replaces Gen1 (both S0 and S1) pricing tier. If your Azure Maps account has Gen1 pricing tier selected, you can switch to Gen2 pricing before it's retired, otherwise it will automatically be updated. For more information, see [Manage the pricing tier of your Azure Maps account](#).

For Azure Maps queries per second limits, see [Azure Maps QPS rate limits](#)

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 Gen1 (S1) and Gen2 pricing tier.

! Expand table

Resource	Limit
Maximum storage per Azure subscription	1 GB
Maximum size per file upload	100 MB

! Note

Azure Maps Data service Retirement

The Azure Maps Data service (both [v1] and [v2]) is now deprecated and will be retired on 9/16/24. The Azure Maps [Data Registry](#) service is replacing the Data service. For more information, see [How to create data registry](#).

Azure Managed Grafana limits

! Expand table

Limit	Description	Essential	Standard
Alert rules	Maximum number of alert rules that can be created.	Not supported	500 per instance
Dashboards	Maximum number of dashboards that can be created.	20 per instance	Unlimited
Data sources	Maximum number of datasources that can be created.	5 per instance	Unlimited
API keys	Maximum number of API keys that can be created.	2 per instance	100 per instance
Data query timeout	Maximum wait duration for the reception of data query response headers, before Grafana times out.	200 seconds	200 seconds
Data source query size	Maximum number of bytes that are read/accepted from responses of outgoing HTTP requests.	80 MB	80 MB
Render image or PDF report wait time	Maximum duration for an image or report PDF rendering request to complete before Grafana times out.	Not supported	220 seconds
Instance count	Maximum number of instances in a single subscription per Azure region.	1	50
Requests per IP	Maximum number of requests per IP per second.	90 requests per second	90 requests per second
Requests per HTTP host	Maximum number of requests per HTTP host per second. The HTTP host stands for the Host header in incoming HTTP requests, which can describe each unique host client.	45 requests per second	45 requests per second

Azure Monitor limits

For Azure Monitor limits, see [Azure Monitor service limits](#).

Azure Data Factory limits

Azure Data Factory is a multitenant service that has the following default limits in place to make sure customer subscriptions are protected from each other's workloads. To raise the limits up to the maximum for your subscription, contact support.

[Expand table](#)

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	5,000
Total CPU cores for Azure-SSIS Integration Runtimes under one subscription	64	Find out how to request a quota increase from 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 doesn't apply to Self-hosted IR.	3,000	3,000
Concurrent Pipeline activity runs per subscription per Azure Integration Runtime region Pipeline activities execute on integration runtime, including Lookup, GetMetadata, and Delete. This limit doesn't apply to Self-hosted IR.	1,000	1,000
Concurrent authoring operations per subscription per Azure Integration Runtime region Including test connection, browse folder list and table list, preview data. This limit doesn't apply to Self-hosted IR.	200	200
Concurrent Data Integration Units ¹ consumption per subscription per Azure Integration Runtime region	Region group 1 ² : 6,000 Region group 2 ² : 3,000 Region group 3 ² : 1,500 Region group 2 ² : 3,000	Region group 1 ² : 6,000 Region group 2 ² : 3,000 Region group 3 ² : 1,500

Resource	Default limit	Maximum limit
	Region group 3 ² : 1,500	
Concurrent Data Integration Units ¹ consumption per subscription per Azure Integration Runtime region in managed virtual network	2,400	2,400
Maximum activities per pipeline, which includes inner activities for containers	80	120
Maximum number of linked integration runtimes that can be created against a single self-hosted integration runtime	100	100
Maximum number of nodes that can be created against a single self-hosted integration runtime	4	4
Maximum parameters per pipeline	50	50
ForEach items	100,000	100,000
ForEach parallelism	20	50
Maximum queued runs per pipeline	100	100
Characters per expression	8,192	8,192
Minimum tumbling window trigger interval	5 min	15 min
Minimum timeout for pipeline activity runs	10 min	10 min
Maximum timeout for pipeline activity runs	7 days	7 days
Bytes per object for pipeline objects ³	200 KB	200 KB
Bytes per object for dataset and linked service objects ³	100 KB	2,000 KB
Bytes per payload for each activity run ⁴	896 KB	896 KB
Data Integration Units ¹ per copy activity run	256	256
Write API calls	1,200/h	1,200/h
	This limit is imposed by Azure Resource Manager, not Azure Data Factory.	
Read API calls	12,500/h	12,500/h
	This limit is imposed by Azure Resource Manager, not Azure Data Factory.	
Monitoring queries per minute	1,000	1,000
Maximum time of data flow debug session	8 hrs	8 hrs
Concurrent number of data flows per integration runtime	50	50
Concurrent number of data flows per integration runtime in managed vNet	50	50
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	2 GB

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.

[+] [Expand table](#)

Region group	Regions
Region group 1	Central US, East US, East US 2, North Europe, West Europe, West US, West US 2
Region group 2	Australia East, Australia Southeast, Brazil South, Central India, Japan East, North Central US, South Central US, Southeast Asia, West Central US
Region group 3	Other regions

If managed virtual network is enabled, the data integration unit (DIU) in all region groups are 2,400.

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.

Web service call limits

Azure Resource Manager has limits for API calls. You can make API calls at a rate within the [Azure Resource Manager API limits](#).

Azure NetApp Files

Azure NetApp Files has a regional limit for capacity. The standard capacity limit for each subscription is 25 TiB, per region, across all service levels. To increase the capacity, use the [Service and subscription limits \(quotas\)](#) support request.

To learn more about the limits for Azure NetApp Files, see [Resource limits for Azure NetApp Files](#).

Azure Policy limits

There's a maximum count for each object type for Azure Policy. For definitions, an entry of *Scope* means the [management group](#) or subscription. For assignments and exemptions, an entry of *Scope* means the management group, subscription, resource group, or individual resource.

[Expand table](#)

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	400
Policy or initiative assignments	Exclusions (notScopes)	400
Policy rule	Nested conditionals	512
Remediation task	Resources	50,000
Policy definition, initiative, or assignment request body	Bytes	1,048,576

Policy rules have more limits to the number of conditions and their complexity. For more information, go to [Policy rule limits](#) for more details.

Azure Quantum limits

Provider Limits & Quota

The Azure Quantum Service supports both first and third-party service providers. Third-party providers own their limits and quotas. Users can view offers and limits in the Azure portal when configuring third-party providers.

You can find the published quota limits for Microsoft's first party Optimization Solutions provider below.

Learn & Develop SKU

[Expand table](#)

Resource	Limit
CPU-based concurrent jobs	up to 5 ¹ concurrent jobs
FPGA-based concurrent jobs	up to 2 ¹ concurrent jobs
CPU-based solver hours	20 hours per month
FPGA-based solver hours	1 hour per month

While on the Learn & Develop SKU, you **cannot** request an increase on your quota limits. Instead you should switch to the Performance at Scale SKU.

Performance at Scale SKU

[Expand table](#)

Resource	Default Limit	Maximum Limit
CPU-based concurrent jobs	up to 100 ¹ concurrent jobs	same as default limit
FPGA-based concurrent jobs	up to 10 ¹ concurrent jobs	same as default limit
Solver hours	1,000 hours per month	up to 50,000 hours per month

Reach out to Azure Support to request a limit increase.

For more information, please review the [Azure Quantum pricing page](#). Review the relevant provider pricing pages in the Azure portal for details on third-party offerings.

¹ Describes the number of jobs that can be queued at the same time.

Azure RBAC limits

The following limits apply to [Azure role-based access control \(Azure RBAC\)](#).

[Expand table](#)

Area	Resource	Limit
Azure role assignments	Azure role assignments per Azure subscription	4,000
	Azure role assignments per management group	500

Area	Resource	Limit
	Size of description for Azure role assignments	2 KB
	Size of condition for Azure role assignments	8 KB
Azure custom roles		
	Azure custom roles per tenant	5,000
	Azure custom roles per tenant (for Microsoft Azure operated by 21Vianet)	2,000
	Size of role name for Azure custom roles	512 chars
	Size of description for Azure custom roles	2 KB
	Number of assignable scopes for Azure custom roles	2,000

Azure SignalR Service limits

[\[+\] Expand table](#)

Resource	Default limit	Maximum limit
Azure SignalR Service units per instance for Free tier	1	1
Azure SignalR Service units per instance for Standard/Premium_P1 tier	100	100
Azure SignalR Service units per instance for Premium_P2 tier	100 - 1,000	100 - 1,000
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
Concurrent connections per unit for Free tier	20	20
Concurrent connections per unit for Standard/Premium 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/Premium tier	1,000,000	1,000,000
Additional messages per unit per day for Standard/Premium tier	Unlimited	Unlimited

To request an update to your subscription's default limits, open a support ticket.

For more information about how connections and messages are counted, see [Messages and connections in Azure SignalR Service](#).

If your requirements exceed the limits, switch from Free tier to Standard tier and add units. For more information, see [How to scale an Azure SignalR Service instance?](#).

If your requirements exceed the limits of a single instance, add instances. For more information, see [How to enable Geo-Replication in Azure SignalR Service](#).

Azure Spring Apps limits

See [Quotas and service plans for Azure Spring Apps](#) to learn more about the limits for Azure Spring Apps.

Azure Storage limits

This section lists the following limits for Azure Storage:

- [Standard storage account limits](#)
- [Azure Storage resource provider limits](#)

- [Azure Blob Storage limits](#)
- [Azure Queue storage limits](#)
- [Azure Table storage limits](#)

Standard storage account limits

The following table describes default limits for Azure general-purpose v2 (GPv2), general-purpose v1 (GPv1), and Blob storage accounts. The *ingress* limit refers to all data that is sent to a storage account. The *egress* limit refers to all data that is received from a storage account.

Microsoft recommends that you use a GPv2 storage account for most scenarios. You can easily upgrade a GPv1 or a Blob storage account to a GPv2 account with no downtime and without the need to copy data. For more information, see [Upgrade to a GPv2 storage account](#).

 **Note**

You can request higher capacity and ingress limits. To request an increase, contact [Azure Support](#).

 Expand table

Resource	Limit
Maximum number of storage accounts with standard endpoints per region per subscription, including standard and premium storage accounts.	250 by default, 500 by request ¹
Maximum number of storage accounts with Azure DNS zone endpoints (preview) per region per subscription, including standard and premium storage accounts.	5000 (preview)
Default maximum storage account capacity	5 PiB ²
Maximum number of blob containers, blobs, directories and subdirectories (if Hierarchical Namespace is enabled), file shares, tables, queues, entities, or messages per storage account.	No limit
Default maximum request rate per general-purpose v2 and Blob storage account in the following regions:	40,000 requests per second ²
<ul style="list-style-type: none"> • East Asia • Southeast Asia • Australia East • Brazil South • Canada Central • China East 2 • China North 3 • North Europe • West Europe • France Central • Germany West Central • Central India • Japan East • Jio India West • Korea Central • Norway East • South Africa North • Sweden Central • UAE North • UK South • Central US • East US • East US 2 • USGov Virginia • USGov Arizona • North Central US • South Central US • West US • West US 2 • West US 3 	

Resource	Limit
Default maximum request rate per general-purpose v2 and Blob storage account in regions that aren't listed in the previous row.	20,000 requests per second ²
Default maximum ingress per general-purpose v2 and Blob storage account in the following regions:	60 Gbps ²
<ul style="list-style-type: none"> • East Asia • Southeast Asia • Australia East • Brazil South • Canada Central • China East 2 • China North 3 • North Europe • West Europe • France Central • Germany West Central • Central India • Japan East • Jio India West • Korea Central • Norway East • South Africa North • Sweden Central • UAE North • UK South • Central US • East US • East US 2 • USGov Virginia • USGov Arizona • North Central US • South Central US • West US • West US 2 • West US 3 	
Default maximum ingress per general-purpose v2 and Blob storage account in regions that aren't listed in the previous row.	25 Gbps ²
Default maximum ingress for general-purpose v1 storage accounts (all regions)	10 Gbps ²
Default maximum egress for general-purpose v2 and Blob storage accounts in the following regions:	200 Gbps ²
<ul style="list-style-type: none"> • East Asia • Southeast Asia • Australia East • Brazil South • Canada Central • China East 2 • China North 3 • North Europe • West Europe • France Central • Germany West Central • Central India • Japan East • Jio India West • Korea Central • Norway East • South Africa North • Sweden Central • UAE North • UK South • Central US • East US • East US 2 • USGov Virginia • USGov Arizona • North Central US 	

Resource	Limit
<ul style="list-style-type: none"> • South Central US • West US • West US 2 • West US 3 	
Default maximum egress for general-purpose v2 and Blob storage accounts in regions that aren't listed in the previous row.	50 Gbps ²
Maximum egress for general-purpose v1 storage accounts (US regions)	20 Gbps if RA-GRS/GRS is enabled, 30 Gbps for LRS/ZRS
Maximum egress for general-purpose v1 storage accounts (non-US regions)	10 Gbps if RA-GRS/GRS is enabled, 15 Gbps for LRS/ZRS
Maximum number of IP address rules per storage account	400
Maximum number of virtual network rules per storage account	400
Maximum number of resource instance rules per storage account	200
Maximum number of private endpoints per storage account	200

¹ With a quota increase, you can create up to 500 storage accounts with standard endpoints per region. For more information, see [Increase Azure Storage account quotas](#).

² Azure Storage standard accounts support higher capacity limits and higher limits for ingress and egress by request. To request an increase in account limits, contact [Azure Support](#).

Azure Storage resource provider limits

The following limits apply only when you perform management operations by using Azure Resource Manager with Azure Storage and the Storage Resource Provider. The limits apply per subscription per region of the resource in the request.

[] [Expand table](#)

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

[] [Expand table](#)

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. These factors include but aren't 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](#)², upload larger blobs or blocks. Specifically, call the [Put Blob](#) or [Put Block](#) operation with a blob or block size that is greater than 256 KiB.

² Page blobs aren't yet supported in accounts that have a hierarchical namespace enabled.

The following table describes the maximum block and blob sizes permitted by service version.

[Expand table](#)

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
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

[Expand table](#)

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.

[Expand table](#)

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
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, <code>PartitionKey</code> , <code>RowKey</code> , and <code>Timestamp</code>)
Maximum total size of an individual	Varies by property type. For more information, see Property Types in Understanding the Table Service

Resource	Target
property in an entity	Data Model .
Size of the PartitionKey	A string up to 1024 characters in size
Size of the RowKey	A string up to 1024 characters 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

Azure subscription creation limits

See [Billing accounts and scopes in the Azure portal](#) to learn more about creating limits for Azure subscriptions.

Azure Virtual Desktop Service limits

The following table describes the maximum limits for Azure Virtual Desktop.

[Expand table](#)

Azure Virtual Desktop Object	Per Parent Container Object	Service Limit
Workspace	Microsoft Entra tenant	1300
HostPool	Workspace	400
Application group	Microsoft Entra tenant	500 ¹
RemoteApp	Application group	500
Role Assignment	Any Azure Virtual Desktop Object	200
Session Host	HostPool	10,000

¹If you require over 500 Application groups then please raise a support ticket via the Azure portal.

All other Azure resources used in Azure Virtual Desktop such as Virtual Machines, Storage, Networking etc. are all subject to their own resource limitations documented in the relevant sections of this article. To visualise the relationship between all the Azure Virtual Desktop objects, review this article [Relationships between Azure Virtual Desktop logical components](#).

To get started with Azure Virtual Desktop, use the [getting started guide](#). For deeper architectural content for Azure Virtual Desktop, use the [Azure Virtual Desktop section of the Cloud Adoption Framework](#). For pricing information for Azure Virtual Desktop, add "Azure Virtual Desktop" within the Compute section of the [Azure Pricing Calculator](#).

Azure VMware Solution limits

The following table describes the maximum limits for Azure VMware Solution.

[Expand table](#)

Resource	Limit
vSphere clusters per private cloud	12
Minimum number of ESXi hosts per cluster	3 (hard-limit)

Resource	Limit
Maximum number of ESXi hosts per cluster	16 (hard-limit)
Maximum number of ESXi hosts per private cloud	96
Maximum number of vCenter Servers per private cloud	1 (hard-limit)
Maximum number of HCX site pairings	25 (any edition)
Maximum number of HCX service meshes	10 (any edition)
Maximum number of Azure VMware Solution ExpressRoute linked private clouds from a single location to a single Virtual Network Gateway	4 The virtual network gateway used determines the actual max linked private clouds. For more information, see About ExpressRoute virtual network gateways If you exceed this threshold use Azure VMware Solution Interconnect to aggregate private cloud connectivity within the Azure region.
Maximum Azure VMware Solution ExpressRoute throughput	10 Gbps (use Ultra Performance Gateway SKU with FastPath enabled)** The virtual network gateway used determines the actual bandwidth. For more information, see About ExpressRoute virtual network gateways Azure VMware Solution ExpressRoutes do not have any port speed limitations and will perform above 10 Gbps; however, rates over 10 Gbps are not guaranteed due to QoS.
Maximum number of Azure Public IPv4 addresses assigned to NSX	2,000
Maximum number of Azure VMware Solution Interconnects per private cloud	10
Maximum number of Azure ExpressRoute Global Reach connections per Azure VMware Solution private cloud	8
vSAN capacity limits	75% of total usable (keep 25% available for SLA)
VMware Site Recovery Manager - Maximum number of protected Virtual Machines	3,000
VMware Site Recovery Manager - Maximum number of Virtual Machines per recovery plan	2,000
VMware Site Recovery Manager - Maximum number of protection groups per recovery plan	250
VMware Site Recovery Manager - RPO Values	5 min or higher * (hard-limit)
VMware Site Recovery Manager - Maximum number of virtual machines per protection group	500
VMware Site Recovery Manager - Maximum number of recovery plans	250

* For information about Recovery Point Objective (RPO) lower than 15 minutes, see [How the 5 Minute Recovery Point Objective Works](#) in the *vSphere Replication Administration guide*.

** This is the soft and recommended limit but can support higher throughput based on the scenario.

For other VMware-specific limits, use the [VMware by Broadcom configuration maximum tool](#).

Azure Web PubSub limits

[] Expand table

Resource	Default limit	Maximum limit
Azure Web PubSub Service units per instance for Free tier	1	1
Azure Web PubSub Service units per instance for Standard/Premium_P1 tier	100	100

Resource	Default limit	Maximum limit
Azure Web PubSub Service units per instance for Premium_P2 tier	100 - 1,000	100 - 1,000
Azure Web PubSub Service units per subscription per region for Free tier	5	5
Total Azure Web PubSub Service unit counts per subscription per region	150	Unlimited
Concurrent connections per unit for Free tier	20	20
Concurrent connections per unit for Standard/Premium 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/Premium tier	1,000,000	1,000,000
Additional messages per unit per day for Standard/Premium tier	Unlimited	Unlimited

To request an update to your subscription's default limits, open a support ticket.

For more information about how connections and messages are counted in billing, see [Billing model in Azure Web PubSub Service](#).

If your requirements exceed the limits, scale up from Free tier to Standard/Premium tier or scale out units. For more information, see [How to scale an Azure Web PubSub Service instance](#).

If your requirements exceed the limits of a single instance, add instances. For more information, see [How to use Geo-Replication in Azure Web PubSub](#).

Backup limits

For a summary of Azure Backup support settings and limitations, see [Azure Backup Support Matrices](#).

Batch limits

[\[+\] Expand table](#)

Resource	Default limit	Maximum limit
Azure Batch accounts per region per subscription	1-3	50
Dedicated cores per Batch account	0-900 ¹	Contact support
Low-priority cores per Batch account	0-100 ¹	Contact support
Active jobs and job schedules per Batch account (completed jobs have no limit)	100-300	1,000 ²
Pools per Batch account	0-100 ¹	500 ²
Private endpoint connections per Batch account	100	100

¹ For capacity management purposes, the default quotas for new Batch accounts in some regions and for some subscription types have been reduced from the above range of values. In some cases, these limits have been reduced to zero. When you create a new Batch account, [check your quotas](#) and [request an appropriate core or service quota increase](#), if necessary. Alternatively, consider reusing Batch accounts that already have sufficient quota or user subscription pool allocation Batch accounts to maintain core and VM family quota across all Batch accounts on the subscription. Service quotas like active jobs or pools apply to each distinct Batch account even for user subscription pool allocation Batch accounts.

² To request an increase beyond this limit, contact Azure Support.

Note

Default limits vary depending on the type of subscription you use to create a Batch account. Cores quotas shown are for Batch accounts in Batch service mode. [View the quotas in your Batch account](#).

Classic deployment model limits

The following limits apply if you use a classic deployment model instead of the Azure Resource Manager deployment model.

[Expand table](#)

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
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

[Expand table](#)

Resource	Actual Limit
Standard sku container groups per region per subscription	100
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	100
Standard sku cores (CPUs) for K80 GPU per region per subscription	0
Standard sku cores (CPUs) for V100 GPU per region per subscription	0
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.

Azure Container Registry limits

The following table details the features and limits of the Basic, Standard, and Premium Azure Container Registry service tiers.

[Expand table](#)

Resource	Basic	Standard	Premium
Included storage ¹ (GiB)	10	100	500
Storage limit (TiB)	40	40	40
Maximum image layer size (GiB)	200	200	200
Maximum manifest size (MiB)	4	4	4
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	Supported
Content trust	N/A	N/A	Supported
Private link with private endpoints	N/A	N/A	Supported
• Private endpoints	N/A	N/A	200
Public IP network rules	N/A	N/A	100
Service endpoint VNet access	N/A	N/A	Preview
• Virtual network rules	N/A	N/A	100
Customer-managed keys	N/A	N/A	Supported
Repository-scoped permissions	Supported	Supported	Supported
• Tokens	100	500	50,000
• Scope maps	100	500	50,000
• Actions	500	500	500
• Repositories per scope map ⁵	500	500	500
Anonymous pull access	N/A	Preview	Preview

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. Both resources, ACR, and the device must be in the same region to achieve a fast download speed.

³ A [docker pull](#) translates to multiple read operations based on the number of layers in the image, plus the manifest retrieval.

⁴ A [docker push](#) translates to multiple write operations, based on the number of layers that must be pushed. A [docker push](#) includes *ReadOps* to retrieve a manifest for an existing image.

⁵ Individual *actions* of `content/delete`, `content/read`, `content/write`, `metadata/read`, `metadata/write` corresponds to the limit of Repositories per scope map.

Azure Content Delivery Network limits

[Expand table](#)

Resource	Limit
Azure Content Delivery Network profiles	25
Content Delivery Network endpoints per profile	25
Custom domains per endpoint	25
Maximum origin group per profile	10
Maximum origin per origin group	10
Maximum number of rules per CDN endpoint	25
Maximum number of match conditions per rule	10
Maximum number of actions per rule	5
Maximum bandwidth per profile*	75 Gbps
Maximum requests per second per profile	100,000
HTTP header size limit (per header)	32 KB

*These two limits are only applicable to Azure CDN Standard from Microsoft (classic). If the traffic is not globally distributed and concentrated in one or two regions, or if a higher quota limit is needed, create an [Azure Support request](#).

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.

Azure 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.

[Expand table](#)

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.

Azure 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.

[Expand table](#)

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.

Azure Data Share limits

Azure Data Share enables organizations to simply and securely share data with their customers and partners.

[Expand table](#)

Resource	Limit
Maximum number of Data Share resources per Azure subscription	100
Maximum number of sent shares per Data Share resource	200
Maximum number of received shares per Data Share resource	100
Maximum number of invitations per sent share	200
Maximum number of share subscriptions per sent share	200
Maximum number of datasets per share	200
Maximum number of snapshot schedules per share	1

Azure 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.

[Expand table](#)

Resource	Limit	Comments
Maximum number of services per subscription, per region	10	To request an increase for this limit, contact support.

Azure Device Update for IoT Hub limits

Limits can be adjusted only for the Standard SKU. Limit adjustment requests are evaluated on a case-by-case basis, and approvals aren't guaranteed.

Limit adjustment requests aren't accepted for the Free SKU. Also, Free SKU instances can't be upgraded to Standard SKU instances.

The following table shows the limits for the Device Update for IoT Hub resource in Azure Resource Manager.

[Expand table](#)

Resource	Standard SKU limit	Free SKU limit	Adjustable for Standard SKU?
Accounts per subscription	50	1	No

Resource	Standard SKU limit	Free SKU limit	Adjustable for Standard SKU?
Instances per account	50	1	No
Length of account name	3-24 characters	3-24 characters	No
Length of instance name	3-36 characters	3-36 characters	No

The following table shows the limits associated with various Device Update operations.

[Expand table](#)

Operation	Standard SKU limit	Free SKU limit	Adjustable for Standard SKU?
Number of devices per instance	1 million	10	Yes
Number of device groups per instance	100	10	Yes
Number of device classes per instance	80	10	Yes
Number of active deployments per instance	50, including one reserved for cancellations	5, including one reserved for cancellations	Yes
Number of total deployments per instance, including all active, inactive, and canceled deployments that aren't deleted	100	20	No
Number of update providers per instance	25	2	No
Number of update names per provider per instance	25	2	No
Number of update versions per update provider and name per instance	100	5	No
Total number of updates per instance	100	10	No
Maximum single update file size	2 GB	2 GB	Yes
Maximum combined size of all files in a single import action	2 GB	2 GB	Yes
Maximum number of files in a single update	10	10	No
Total data storage included per instance	100 GB	5 GB	No

⚠ Note

Canceled or inactive deployments count toward your total deployment limit. Make sure to clean up these deployments periodically so you aren't prevented from creating new deployments.

Azure Digital Twins limits

⚠ Note

Some areas of this service have adjustable limits, and others do not. The following tables use the *Adjustable?* column to represent this condition. When the limit can be adjusted, the *Adjustable?* value is Yes.

Functional limits

The following table lists the functional limits of Azure Digital Twins.

[Expand table](#)

Area	Capability	Default limit	Adjustable?
Azure resource	Number of Azure Digital Twins instances in a region, per subscription	10	Yes

Area	Capability	Default limit	Adjustable?
Digital twins	Number of twins in an Azure Digital Twins instance	2,000,000	Yes
Digital twins	Number of digital twins that can be imported in a single Import Jobs API job	2,000,000	No
Digital twins	Number of incoming relationships to a single twin	50,000	No
Digital twins	Number of outgoing relationships from a single twin	50,000	No
Digital twins	Total number of relationships in an Azure Digital Twins instance	20,000,000	Yes
Digital twins	Number of relationships that can be imported in a single Import Jobs API job	10,000,000	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
Digital twins	Maximum size of a string property value (UTF-8)	4 KB	No
Digital twins	Maximum size of a property name	1 KB	No
Routing	Number of endpoints for a single Azure Digital Twins instance	6	No
Routing	Number of routes for a single Azure Digital Twins instance	6	Yes
Models	Number of models within a single Azure Digital Twins instance	10,000	Yes
Models	Number of models that can be imported in a single API call (not using the Import Jobs API)	250	No
Models	Number of models that can be imported in a single Import Jobs API job	10,000	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	1000	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 lists the rate limits of different APIs.

[Expand table](#)

API	Capability	Default limit	Adjustable?
Jobs API	Number of requests per second	1	Yes
Jobs API	Number of bulk jobs running concurrently (including import and delete)	1	Yes
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	500	Yes
Digital Twins API	Number of create/update/delete operations per second on a single twin or its incoming/outgoing relationships	10	No

API	Capability	Default limit	Adjustable?
Digital Twins API	Number of outstanding operations on a single twin or its incoming/outgoing relationships	500	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 in its spec documentation in GitHub: [Digital Twins Definition Language \(DTDL\) - version 3](#).

Query latency details are described in [Query language](#). Limitations of particular query language features can be found in the [query reference documentation](#).

Azure Event Grid limits

! Note

The following limits listed in this article are per region.

Event Grid throttle limits

Event Grid offers a standard tier and basic tier. Event Grid standard tier enables pub-sub using Message Queuing Telemetry Transport (MQTT) broker functionality and pull-delivery of messages through the Event Grid namespace. Event Grid basic tier enables push delivery using Event Grid custom topics, Event Grid system topics, Event domains, and Event Grid partner topics. See [Choose the right Event Grid tier](#). This article describes the quota and limits for both tiers.

Event Grid Namespace resource limits

[Azure Event Grid namespaces](#) enables MQTT messaging, and HTTP pull delivery. The following limits apply to namespace resources in Azure Event Grid.

! [Expand table](#)

Limit description	Limit
Event Grid namespaces per Azure subscription	50
Maximum throughput units per Event Grid namespace	40
IP Firewall rules per Event Grid namespace	16

MQTT limits in Event Grid namespace

The following limits apply to MQTT in Azure Event Grid namespace resource.

! Note

Throughput units (TUs) define the ingress and egress event rate capacity in namespaces. They allow you to control the capacity of your namespace resource for message ingress and egress.

[Expand table](#)

Limit description	Limit
MQTT sessions per Event Grid namespace	10,000 per throughput unit (TU)
Sessions per Event Grid namespace	10,000 per TU
Session Expiry Interval	8 hours, configurable on the Event Grid namespace
Inbound MQTT publishing requests per Event Grid namespace	1,000 messages per second per TU
Inbound MQTT bandwidth per Event Grid namespace	1 MB per second per TU
Inbound MQTT publishing requests per session	100 messages per second
Inbound MQTT bandwidth per session	1 MB per second
Inbound in-flight MQTT messages*	100 messages
Inbound in-flight MQTT bandwidth*	64 KB
Outbound MQTT publishing requests per Event Grid namespace	1,000 messages per second per TU
Outbound MQTT bandwidth per Event Grid namespace	1 MB per second per TU
Outbound MQTT publishing requests per session	100 messages per second
Outbound MQTT bandwidth per session	1 MB per second
Outbound in-flight MQTT messages*	100 messages
Outbound in-flight MQTT bandwidth*	64 KB
Max message size	512 KB
Segments per topic/ topic filter	8
Topic size	256 B
MQTTv5 response topic	256 B
MQTTv5 topic aliases	10 per session
MQTTv5 total size of all user properties	32 KB
MQTTv5 content type size	256 B
MQTTv5 correlation data size	256 B
Connect requests	200 requests per second per TU
MQTTv5 authentication data size	8 KB
Maximum keep-alive interval	1160
Topic filters per MQTT SUBSCRIBE packet	10
Subscribe and unsubscribe requests per Event Grid namespace	200 requests per second
Subscribe and unsubscribe requests per session	5 requests per second
Subscriptions per MQTT session	50
Subscriptions per Event Grid namespace	1 million
Subscriptions per MQTT topic	Unlimited, as long as they don't exceed the limit for subscriptions per Event Grid namespace or session
Registered client resources	10,000 clients per TU
CA certificates	10
Client groups	10

Limit description	Limit
Topic spaces	10
Topic templates	10 per topic space
Permission bindings	100

* For MQTTv5, learn more about [flow control support](#).

Events limits in Event Grid namespace

The following limits apply to events in Azure Event Grid namespace resource.

[\[+\] Expand table](#)

Limit description	Limit
Event Grid namespace topics	100 per TU
Event ingress	1,000 events per second or 1 MB per second per TU (whichever comes first)
Event egress (push and pull APIs)	Up to 2,000 events per second or 2 MB per second per TU
Event egress (acknowledge, release, reject, and renew lock APIs)	Up to 2,000 events per second or 2 MB per second per TU
Maximum event retention on Event Grid namespace topics	7 days
Subscriptions per topic	500
Maximum event size	1 MB
Batch size	1 MB
Events per request	1,000

Custom topic, system topic, and partner topic resource limits

The following limits apply to Azure Event Grid custom topic, system topic, and partner topic resources.

[\[+\] Expand table](#)

Limit description	Limit
Custom topics per Azure subscription	100 When the limit is reached, you can consider a different region or consider using domains, which can support 100,000 topics.
Event subscriptions per topic	500 This limit can't be increased.
Publish rate for a custom or a partner topic (ingress)	5,000 events or 5 MB per second (whichever comes first). An event is counted for limits and pricing purposes as a 64KB data chunk. So, if the event is 128 KB, it counts as two events.
Event size	1 MB This limit can't be increased.
Maximum event retention on topics	1 day. This limit can't be increased.
Number of incoming events per batch	5,000 This limit can't be increased
Private endpoint connections per topic	64 This limit can't be increased
IP Firewall rules per topic	128

Domain resource limits

The following limits apply to Azure Event Grid domain resource.

[Expand table](#)

Limit description	Limit
Domains per Azure subscription	100
Topics per domain	100,000
Event subscriptions per topic within a domain	500 This limit can't be increased
Domain scope event subscriptions	50 This limit can't be increased
Publish rate for a domain (ingress)	5,000 events or 5 MB per second (whichever comes first). An event is counted for limits and pricing purposes as a 64KB data chunk. So, if the event is 128 KB, it counts as two events.
Maximum event retention on domain topics	1 day. This limit can't be increased.
Private endpoint connections per domain	64
IP Firewall rules per topic	128

Azure 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.

[Expand table](#)

Limit	Notes	Value
Size of an event hub name	-	256 characters
Size of a consumer group name	Kafka protocol doesn't require the creation of a consumer group. AMQP: 50 characters	Kafka: 256 characters AMQP: 50 characters
Number of non-epoch receivers per consumer group	-	5
Number of authorization rules per namespace	Subsequent requests for authorization rule creation are rejected.	12
Number of calls to the GetRuntimeInformation method	-	50 per second
Number of virtual networks (VNet)	-	128
Number of IP Config rules	-	128
Maximum length of a schema group name		50
Maximum length of a schema name		100
Size in bytes per schema		1 MB
Number of properties per schema group		1024
Size in bytes per schema group property key		256
Size in bytes per schema group property value		1024

Basic vs. standard vs. premium vs. dedicated tiers

The following table shows limits that are different for Basic, Standard, Premium, and Dedicated tiers.

ⓘ Note

- In the table, CU is [capacity unit](#), PU is [processing unit](#), and TU is [throughput unit](#).
- You can configure [TUs](#) for a Basic or Standard tier namespace or [PUs](#) for a Premium tier namespace.
- When you [create a dedicated cluster](#), one CU is assigned to the cluster. If you enable the [Support scaling](#) option while you create the cluster, you can scale out by increasing CUs or scale in by decreasing CUs for the cluster yourself. For step-by-step instructions, see [Scale dedicated cluster](#). For clusters that don't support the [Support scaling](#) feature, [submit a ticket](#) to adjust CUs for the cluster.

[Expand table](#)

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	1,000 No limit per CU
Number of Kafka consumer groups per namespace	NA	1,000	1,000	1,000
Number of brokered connections per namespace	100	5,000	10,000 per PU For example, if the namespace is assigned 3 PUs, the limit is 30,000.	100,000 per CU
Maximum retention period of event data	1 day	7 days	90 days	90 days
Event storage for retention	84 GB per TU	84 GB per TU	1 TB per PU	10 TB per CU
Maximum TUs or PUs or CUs	40 TUs	40 TUs	16 PUs	20 CUs
Number of partitions per event hub	32	32	100 per event hub, but there's a limit of 200 per PU at the namespace level. For example, if a namespace is assigned 2 PUs, the limit for total number of partitions in all event hubs in the namespace is $2 * 200 = 400$.	1,024 per event hub 2,000 per CU
Number of namespaces per subscription	1,000	1,000	1,000	1,000 (50 per CU)
Number of event hubs per namespace	10	10	100 per PU	1,000
Capture	N/A	Pay per hour	Included	Included
Size of compacted event hub	N/A	1 GB per partition	250 GB per partition	250 GB per partition
Size of the schema registry (namespace) in megabytes	N/A	25	100	1,024
Number of schema groups in a schema registry or namespace	N/A	1: excluding the default group	100 1 MB per schema	1,000 1 MB per schema
Number of schema versions across all schema groups	N/A	25	1,000	10,000

Limit	Basic	Standard	Premium	Dedicated
Throughput per unit	Ingress: 1 MB/sec or 1000 events per second Egress: 2 MB/sec or 4,096 events per second	Ingress: 1 MB/sec or 1,000 events per second Egress: 2 MB/sec or 4,096 events per second	No limits per PU *	No limits per CU *

* Depends on factors such as resource allocation, number of partitions, and storage.

ⓘ Note

You can publish events individually or batched. The publication limit (according to SKU) applies regardless of whether it's a single event or a batch. Publishing events larger than the maximum threshold will be rejected.

Azure IoT Central limits

IoT Central limits the number of applications you can deploy in a subscription to 100. To learn more, see [Azure IoT Central quota and limits](#).

Azure 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](#).

[Expand table](#)

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

The following table lists the limits that apply to IoT Hub resources.

[Expand table](#)

Resource	Limit
Maximum paid IoT hubs per Azure subscription	50
Maximum free IoT hubs per Azure subscription	1
Maximum number of characters in a device ID	128
Maximum number of device identities returned in a single call	1,000
IoT Hub message maximum retention for device-to-cloud messages	7 days
Maximum size of device-to-cloud message	256 KB
Maximum size of device-to-cloud batch	AMQP and HTTP: 256 KB for the entire batch MQTT: 256 KB for each message
Maximum messages in device-to-cloud batch	500
Maximum size of cloud-to-device message	64 KB
Maximum TTL for cloud-to-device messages	2 days
Maximum delivery count for cloud-to-device messages	100

Resource	Limit
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 (beyond built-in 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**

The total number of devices plus modules that can be registered to a single IoT hub is capped at 1,000,000.

IoT Hub throttles requests when the following quotas are exceeded.

[Expand table](#)

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 concurrent file uploads per device.
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).

Throttle	Per-hub value
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).

Azure IoT Hub Device Provisioning Service limits

The following table lists the limits that apply to Azure IoT Hub Device Provisioning Service resources.

[Expand table](#)

Resource	Limit	Adjustable?
Maximum device provisioning services per Azure subscription	10	No
Maximum number of registrations	1,000,000	No
Maximum number of individual enrollments	1,000,000	No
Maximum number of enrollment groups (<i>X.509 certificate</i>)	100	No
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.

[Expand table](#)

Rate	Per-unit value	Adjustable?
Operations	1,000/min/service	No
Device registrations	1,000/min/service	No
Device polling operation	5/10 sec/device	No

Azure 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):

[Expand table](#)

Key type	HSM key CREATE key	HSM key All other transactions	Software key CREATE key	Software key All other transactions
RSA 2,048-bit	10	2,000	20	4,000
RSA 3,072-bit	10	500	20	1,000
RSA 4,096-bit	10	250	20	500
ECC P-256	10	2,000	20	4,000
ECC P-384	10	2,000	20	4,000
ECC P-521	10	2,000	20	4,000
ECC SECP256K1	10	2,000	20	4,000

ⓘ Note

In the previous table, we see that for RSA 2,048-bit software keys, 4,000 GET transactions per 10 seconds are allowed. For RSA 2,048-bit HSM-keys, 2,000 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 $2,000/250 = 8$.

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:

- 4,000 RSA 2,048-bit software-key GET transactions
- 2,000 RSA 2,048-bit HSM-key GET transactions
- 250 RSA 4,096-bit HSM-key GET transactions
- 248 RSA 4,096-bit HSM-key GET transactions and 16 RSA 2,048-bit HSM-key GET transactions

Secrets, managed storage account keys, and vault transactions:

[+] Expand table

Transactions type	Maximum transactions allowed in 10 seconds, per vault per region ¹
Secret CREATE secret	300
All other transactions	4,000

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.

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 cannot 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

[+] Expand table

Transactions type	Maximum key vault object versions allowed
Back up 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](#).

Resource type: Managed HSM

This section describes service limits for resource type `managed HSM`.

Object limits

Expand table

Item	Limits
Number of HSM instances per subscription per region	5
Number of keys per HSM instance	5000
Number of versions per key	100
Number of custom role definitions per HSM instance	50
Number of role assignments at HSM scope	50
Number of role assignments at each individual key scope	10

Transaction limits for administrative operations (number of operations per second per HSM instance)

Expand table

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 three 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 three 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 one 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)

[Expand table](#)

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.

[Expand table](#)

Operation	P-256	P-256K	P-384	P-521
Create Key	1	1	1	1
Delete Key (soft-delete)	10	10	10	10
Purge Key	10	10	10	10
Backup Key	10	10	10	10
Restore Key	10	10	10	10
Get Key Information	1100	1100	1100	1100
Sign	260	260	165	56
Verify	130	130	82	28

AES key operations (number of operations per second per HSM instance)

- Encrypt and Decrypt operations assume a 4KB packet size.
- Throughput limits for Encrypt/Decrypt apply to AES-CBC and AES-GCM algorithms.
- Throughput limits for Wrap/Unwrap apply to AES-KW algorithm.

[Expand table](#)

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

Operation	128-bit	192-bit	256-bit
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

Azure Managed Identity limits

- Each managed identity counts towards the object quota limit in a Microsoft Entra tenant as described in [Microsoft Entra service limits and restrictions](#).
- The rate at which managed identities can be created have the following limits:
 - Per Microsoft Entra tenant per Azure region: 400 create operations per 20 seconds.
 - Per Azure Subscription per Azure region : 80 create operations per 20 seconds.
- The rate at which a user-assigned managed identity can be assigned with an Azure resource :
 - Per Microsoft Entra tenant per Azure region: 400 assignment operations per 20 seconds.
 - Per Azure Subscription per Azure region : 300 assignment operations per 20 seconds.

Azure 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

ⓘ Expand table

Resource	Default Limit
Media Services accounts in a single subscription	100 (fixed)

Asset limits

ⓘ Expand table

Resource	Default Limit
Assets per Media Services account	1,000,000

Storage (media) limits

ⓘ Expand table

Resource	Default Limit
File size	In some scenarios, there is a limit on the maximum file size supported for processing in Media Services. ⁽¹⁾

Resource	Default Limit
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.

² The storage accounts must be from the same Azure subscription.

Jobs (encoding & analyzing) limits

 Expand table

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

 Expand table

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

 Expand table

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 reuse 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

[Expand table](#)

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.

Azure Media Services v2 (legacy)

For limits specific to Media Services v2 (legacy), see [Media Services v2 (legacy)]

Azure Mobile Services limits

[Expand table](#)

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/ WebSockets	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](#).

Microsoft Entra service limits

Here are the usage constraints and other service limits for the Microsoft Entra service.

[Expand table](#)

Category	Limit
Tenants	<ul style="list-style-type: none"> A single user can belong to a maximum of 500 Microsoft Entra tenants as a member or a guest. Create a maximum of 200 tenants. Limit of 300 license-based subscriptions (such as Microsoft 365 subscriptions) per tenant
Domains	<ul style="list-style-type: none"> You can add no more than 5,000 managed domain names. If you set up all of your domains for federation with on-premises Active Directory, you can add no more than 2,500 domain names in each tenant.
Resources	<ul style="list-style-type: none"> By default, a maximum of 50,000 Microsoft Entra resources can be created in a single tenant by users of the Microsoft Entra ID Free edition. If you have at least one verified domain, the default Microsoft Entra service quota for your organization is extended to 300,000 Microsoft Entra resources. The Microsoft Entra service quota for organizations created by self-service sign-up remains 50,000 Microsoft Entra resources, even after you perform 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 Microsoft Entra pricing page. To go beyond the default quota, you must contact Microsoft Support. A non-admin user can create no more than 250 Microsoft Entra resources. Both active resources and deleted resources that are available to restore count toward this quota. Only deleted Microsoft Entra resources that were deleted fewer than 30 days ago are available to restore. Deleted Microsoft Entra 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. Resource limitations apply to all directory objects in a given Microsoft Entra tenant, including users, groups, applications, and service principals.
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 Microsoft Entra resource. Only User, Group, TenantDetail, Device, Application, and ServicePrincipal entities can be extended with string-type or binary-type single-valued attributes.
Applications	<ul style="list-style-type: none"> A maximum of 100 users and service principals can be owners of a single application. A user, group, or service principal can have a maximum of 1,500 app role assignments. The limitation is on the service principal, user, or group across all app roles and not on the number of assignments on a single app role. A user can have credentials configured for a maximum of 48 apps using password-based single sign-on. This limit only applies for credentials configured when the user is directly assigned the app, not when the user is a member of a group that is assigned. A group can have credentials configured for a maximum of 48 apps using password-based single sign-on. See more limits in Validation differences by supported account types.
Application manifest	<p>A maximum of 1,200 entries can be added to the application manifest.</p> <p>See more limits in Validation differences by supported account types.</p>
Groups	<ul style="list-style-type: none"> A non-admin user can create a maximum of 250 groups in a Microsoft Entra organization. Any Microsoft Entra admin who can manage groups in the organization can also create an unlimited number of groups (up to the Microsoft Entra object limit). If you assign a role to a user to remove the limit for that user, assign a less privileged, built-in role such as User Administrator or Groups Administrator. A Microsoft Entra organization can have a maximum of 15,000 dynamic groups and dynamic administrative units combined. A maximum of 500 role-assignable groups can be created in a single Microsoft Entra organization (tenant). A maximum of 100 users can be owners of a single group. Any number of Microsoft Entra resources can be members of a single group. A user can be a member of any number of groups. When security groups are being used in combination with SharePoint Online, a user can be a part of 2,049 security groups in total. This includes both direct and indirect group memberships. When this limit is exceeded, authentication and search results become unpredictable. By default, the number of members in a group that you can synchronize from your on-premises Active Directory to Microsoft Entra ID by using Microsoft Entra Connect is limited to 50,000 members. If you need to sync a group membership that's over this limit, you must onboard the Microsoft Entra Connect Sync V2 endpoint API. When you select a list of groups, you can assign a group expiration policy to a maximum of 500 Microsoft 365 groups. There is no limit when the policy is applied to all Microsoft 365 groups.
At this time, the following scenarios are supported with nested groups:	
<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). 	

Category	Limit
	<ul style="list-style-type: none"> • Restricting access to self-serve password reset. • Restricting which users can do Microsoft Entra join and device registration. <p>The following scenarios are <i>not</i> supported with nested groups:</p> <ul style="list-style-type: none"> • App role assignment, for both access and provisioning. Assigning groups to an app is supported, but any groups nested within the directly assigned group won't have access. • Group-based licensing (assigning a license automatically to all members of a group). • Microsoft 365 Groups.
Application Proxy	<ul style="list-style-type: none"> • A maximum of 500 transactions* per second per Application Proxy application. • A maximum of 750 transactions per second for the Microsoft Entra organization. <p>*A transaction is defined as a single HTTP request and response for a unique resource. When clients are throttled, they'll receive a 429 response (too many requests). Transaction metrics are collected on each connector and can be monitored using performance counters under the object name <code>Microsoft AAD App Proxy Connector</code>.</p>
Access Panel	There's no limit to the number of applications per user that can be displayed in the Access Panel, regardless of the number of assigned licenses.
Reports	A maximum of 1,000 rows can be viewed or downloaded in any report. Any other data is truncated.
Administrative units	<ul style="list-style-type: none"> • A Microsoft Entra resource can be a member of no more than 30 administrative units. • A maximum of 100 restricted management administrative units in a tenant. • A Microsoft Entra organization can have a maximum of 15,000 dynamic groups and dynamic administrative units combined.
Microsoft Entra roles and permissions	<ul style="list-style-type: none"> • A maximum of 100 Microsoft Entra custom roles can be created in a Microsoft Entra organization. • A maximum of 150 Microsoft Entra custom role assignments for a single principal at any scope. • A maximum of 100 Microsoft Entra built-in role assignments for a single principal at non-tenant scope (such as an administrative unit or Microsoft Entra object). There is no limit to Microsoft Entra built-in role assignments at tenant scope. For more information, see Assign Microsoft Entra roles at different scopes. • 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 Microsoft Entra 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 might take up to 15 minutes or you might have to sign out and sign back in before admin role membership additions and revocations take effect.
Conditional Access Policies	A maximum of 195 policies can be created in a single Microsoft Entra organization (tenant).
Terms of use	You can add no more than 40 terms to a single Microsoft Entra organization (tenant).
Multitenant organizations	<ul style="list-style-type: none"> • A maximum of 5 active tenants, including the owner tenant. The owner tenant can add more than 5 pending tenants, but they won't be able to join the multitenant organization if the limit is exceeded. This limit is applied at the time a pending tenant joins a multitenant organization. • A maximum of 100,000 internal users per active tenant. This limit is applied at the time a pending tenant joins a multitenant organization.

Azure 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 have 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 manually increased by support in the past and are currently seeing limits lower than

what is listed in the following tables, [open an online customer support request at no charge](#)

[Expand table](#)

Resource	Limit
Virtual networks	1,000
Subnets per virtual network	3,000
Virtual network peerings per virtual network	500
Virtual network gateways (VPN gateways) per virtual network	1
Virtual network gateways (ExpressRoute gateways) per virtual network	1
DNS servers per virtual network	20
Private IP addresses per virtual network	65,536
Total Private Addresses for a group of Peered Virtual networks	128,000
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 (The limit applies separately to source and destination)	4,000
Application security groups	3,000
Application security groups per IP configuration, per NIC	20
Application security groups referenced as source/destination per NSG rule	10
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

[Expand table](#)

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
Standard Public IP addresses ¹	10	Contact support

Resource	Default limit	Maximum limit
Public IP prefixes	limited by number of Standard Public IPs in a subscription	Contact support
Public IP prefix length	/28	Contact support
Custom IP prefixes	5	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.

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](#).

[+] [Expand table](#)

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

Azure Load Balancer limits

Standard Load Balancer

[+] [Expand table](#)

Resource	Limit
Load balancers	1,000
Frontend IP configurations	600
Rules (Load Balancer + Inbound NAT) per resource	1,500
Rules per NIC (across all IPs on a NIC) ¹	300
High-availability ports rule	1 per internal frontend
Outbound rules per Load Balancer	600
Backend pool size	5,000

Resource	Limit
Azure global Load Balancer Backend pool size	300
Backend IP configurations per frontend ²	10,000
Backend IP configurations across all frontends	500,000

¹ Each NIC can have a total of 300 rules (load balancing, inbound NAT, and outbound rules combined) configured across all IP configurations on the NIC. ² Backend IP configurations are aggregated across all load balancer rules including load balancing, inbound NAT, and outbound rules. Each rule a backend pool instance is configured to counts as one configuration.

Load Balancer doesn't apply any throughput limits. However, throughput limits for virtual machines and virtual networks still apply. For more information, see [Virtual machine network bandwidth](#).

Gateway Load Balancer

[Expand table](#)

Resource	Limit
Resources chained per Load Balancer (LB frontend configurations or VM NIC IP configurations combined)	100

All limits for Standard Load Balancer also apply to Gateway Load Balancer.

Basic Load Balancer

[Expand table](#)

Resource	Limit
Load balancers	1,000
Rules per resource	250
Rules per NIC (across all IPs on a NIC)	300
Frontend IP configurations ³	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)

³ 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.

Azure Application Gateway limits

The following table applies to v1, v2, Standard, and WAF SKUs unless otherwise stated.

[Expand table](#)

Resource	Limit	Note
Azure Application Gateway	1,000 per region per subscription	
Frontend IP configurations	4	IPv4 - 1 public and 1 private. IPv6 - 1 public and 1 private.
Frontend ports	100 ¹	
Backend address pools	100	

Resource	Limit	Note
Backend targets 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. For more information, see Frequently asked questions about Application Gateway .
HTTP load-balancing rules	400 ¹	
Backend HTTP settings	100 ¹	
Instances per gateway	V1 SKU - 32 V2 SKU - 125	
SSL certificates	100 ¹	1 per HTTP listener
Maximum SSL certificate size	V1 SKU - 10 KB V2 SKU - 16 KB	
Maximum trusted client CA certificate size	25 KB	25 KB is the maximum aggregated size of root and intermediate certificates contained in an uploaded pem or cer file.
Maximum trusted client CA certificates	200	100 per SSL Profile
Authentication certificates	100	
Trusted root certificates	100	
Request timeout minimum	1 second	
Request timeout maximum to private backend	24 hours	
Request timeout maximum to external backend	4 minutes	
Number of sites	100 ¹	1 per HTTP listener
URL maps per listener	1	
Host names per listener	5	
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	32 KB	
Maximum header size	32 KB	
Maximum header field size for HTTP/2	8 KB	
Maximum header size for HTTP/2	16 KB	
Maximum requests per	1000	The total number of requests that can share the same frontend HTTP/2 connection

Resource	Limit	Note
HTTP/2 connection		
Maximum file upload size (Standard SKU)	V1 - 2 GB V2 - 4 GB	This maximum size limit is shared with the request body
Maximum file upload size (WAF SKU)	V1 Medium - 100 MB V1 Large - 500 MB V2 - 750 MB V2 (with CRS 3.2 or DRS) - 4 GB ³	1 MB - Minimum Value 100 MB - Default value V2 with CRS 3.2 or DRS - can be turned On/Off
Maximum request size limit Standard SKU (without files)	V1 - 2 GB V2 - 4 GB	
Maximum request size limit WAF SKU (without files)	V1 or V2 (with CRS 3.1 and older) - 128 KB V2 (with CRS 3.2 or DRS) - 2 MB ³	8 KB - Minimum Value 128 KB - Default value V2 with CRS 3.2 or DRS - can be turned On/Off
Maximum request inspection limit WAF SKU	V1 or V2 (with CRS 3.1 and older) - 128 KB V2 (with CRS 3.2 or DRS) - 2 MB ³	8 KB - Minimum Value 128 KB - Default value V2 with CRS 3.2 or DRS - can be turned On/Off
Maximum Private Link Configurations	2	1 for public IP, 1 for private IP
Maximum Private Link IP Configurations	8	
Maximum WAF custom rules per WAF policy	100	
Maximum WAF match conditions per custom rule	10	This limit is not enforced by the WAF. Adding more than 10 match conditions can lead to performance degradation
WAF IP address ranges per match condition	540 600 - with CRS 3.2 or DRS	
Maximum WAF exclusions per Application Gateway	40 200 - with CRS 3.2 or DRS	
WAF string match values per match condition	10	

¹ The number of resources listed in the table applies to standard Application Gateway SKUs and WAF-enabled SKUs running CRS 3.2 or DRS. For WAF-enabled SKUs running CRS 3.1 or lower, the supported number is 40. For more information, see [WAF engine](#).

² Limit is per Application Gateway instance not per Application Gateway resource.

³ Must define the value via WAF Policy for Application Gateway.

Azure Application Gateway for Containers limits

[Expand table](#)

Resource	Limit
Application Gateway for Containers	1000 per subscription
Associations	1 per gateway
Frontends	5 per gateway

[Expand table](#)

Resource	Limit
Resource naming	128 characters
Namespace naming	128 characters
Listeners per gateway	64 listeners per gateway resource (enforced by Gateway API)
Total AGC references	5 per ALB controller
Total certificate references	100 per AGC
Total listeners	200 per AGC
Total routes	200 per AGC
Total rules	200 per AGC
Total services	100 per AGC
Total endpoints	5000 per AGC

Azure Bastion limits

An instance is an optimized Azure VM that is created when you configure Azure Bastion. When you configure Azure Bastion using the Basic SKU, 2 instances are created. If you use the Standard SKU, you can specify the number of instances between 2-50.

[Expand table](#)

Workload Type*	Session Limit per Instance**
Light	25
Medium	20
Heavy	2

*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

Public DNS zones

[Expand table](#)

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 ¹
Number of Alias records for a single Azure resource	20

¹If you need to increase these quota limits, contact Azure Support.

Public DNS zone operations

[Expand table](#)

Operation	Limit (per zone)
Create	40/min
Delete	40/min
Get	1000/min
List	60/min
List By Resource Group	60/min (per resource group)
Update	40/min

Public DNS resource record operations

[Expand table](#)

Operation	Limit (per zone)
Create	200/min
Delete	200/min
Get	2000/min
List By DNS Zone	60/min
List By Type	60/min
Update	200/min

Private DNS

Private DNS zones

[Expand table](#)

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 autoregistration enabled	100
Number of private DNS zones a virtual network can get linked to with autoregistration enabled	1
Number of private DNS zones a virtual network can get linked	1000

Private DNS zone operations

[Expand table](#)

Operation	Limit (per subscription)
Create	40/min
Delete	40/min
Get	200/min (per zone)

Operation	Limit (per subscription)
List by subscription	60/min
List by resource group	100/min (per resource group)
Update	40/min

Private DNS resource record operations

[Expand table](#)

Operation	Limit (per zone)
Create	60/min
Delete	60/min
Get	200/min
List	100/min
Update	60/min

Virtual network links operations

[Expand table](#)

Operation	Limit (per zone)
Create	60/min
Delete	60/min
Get	100/min
List by virtual network	20/min
Update	60/min

Azure-provided DNS resolver VM limits

[Expand table](#)

Resource	Limit
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. These limits apply to the default Azure resolver, not the DNS private resolver.

DNS Private Resolver¹

[Expand table](#)

Resource	Limit
DNS private resolvers per subscription	15
DNS private resolvers per virtual network	1
Inbound endpoints per DNS private resolver	5

Resource	Limit
Outbound endpoints per DNS private resolver	5
Forwarding rules per DNS forwarding ruleset	1000
Virtual network links per DNS forwarding ruleset	500
DNS forwarding ruleset linked to a virtual network	1
Outbound endpoints per DNS forwarding ruleset	2
DNS forwarding rulesets per outbound endpoint	2
Target DNS servers per forwarding rule	6
QPS per endpoint	10,000

¹Different limits might be enforced by the Azure portal until the portal is updated. Use PowerShell to provision elements up to the most current limits.

Azure Firewall limits

[Expand table](#)

Resource	Limit
Azure Firewalls per virtual network	1
Max Data throughput	100 Gbps for Premium, 30 Gbps for Standard, 250 Mbps for Basic (preview) SKU For more information, see Azure Firewall performance .
Rule limits	20,000 unique source/destinations in network rules Unique source/destinations in network = (Source addresses + Source IP Groups) * (Destination addresses + Destination Fqdn count + Destination IP Groups) * (IP protocols count) * (Destination ports) You can track the Firewall Policy network rule count in the policy analytics under the Insights tab. As a proxy, you can also monitor your Firewall Latency Probe metrics to ensure it stays within 20 ms even during peak hours.
Total size of rules within a single Rule Collection Group	1 MB for Firewall policies created before July 2022 2 MB for Firewall policies created after July 2022
Number of Rule Collection Groups in a firewall policy	50 for Firewall policies created before July 2022 90 for Firewall policies created after July 2022
Maximum DNAT rules (Maximum external destinations)	250 maximum [number of firewall public IP addresses + unique destinations (destination address, port, and protocol)] The DNAT limitation is due to the underlying platform. For example, you can configure 500 UDP rules to the same destination IP address and port (one unique destination), while 500 rules to the same IP address but to 500 different ports exceeds the limit (500 unique destinations). If you need more than 250, you'll need to add another firewall in a separate virtual network
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	It is recommended to have a maximum of 50 unique IP Groups per classic firewall. Maximum of 200 unique IP Groups per firewall policy. Maximum 5000 individual IP addresses or IP prefixes per each IP Group.

Resource	Limit
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.
TLS inspection timeout	120 seconds

Azure Front Door (classic) limits

- In addition to the following limits, there are [composite limit on the number of routing rules, front-end domains, protocols, and paths](#).

[Expand table](#)

Resource	Classic tier limit
Azure Front Door resources per subscription	100
Front-end hosts, which include custom domains per resource	500
Routing rules per resource	500
Rules per Rule set	25
Back-end pools per resource ²	50
Back ends per back-end pool	100
Path patterns to match for a routing rule	25
URLs in a single cache purge call	100
Maximum bandwidth ¹	75 Gbps
Maximum requests per second per profile ¹	100,000
HTTP header size limit (per header)	32 KB
Custom web application firewall rules per policy	100
Web application firewall policy per subscription	100
Web application firewall match conditions per custom rule	10
Web application firewall IP address ranges per custom rule	600
Web application firewall string match values per match condition	10
Web application firewall string match value length	256
Web application firewall POST body parameter name length	256
Web application firewall HTTP header name length	256
Web application firewall cookie name length	256
Web application firewall exclusion limit	100
Web application firewall HTTP request body inspection limit	128 KB
Web application firewall custom response body length	32 KB

¹If the traffic isn't globally distributed and concentrated in one or more regions, or if a higher quota limited is needed, create an [Azure support request](#).

²To 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.

Azure Front Door Standard and Premium service limits

- Maximum of 500 total Standard and Premium profiles per subscription.
- In addition to the following limits, there are [composite limit on the number of routes, domains, protocols, and paths](#).

[+] [Expand table](#)

Resource	Standard tier limit	Premium tier limit
Maximum profiles per subscription	500	500
Maximum endpoint per profile	10	25
Maximum custom domain per profile	100	500
Maximum origin groups per profile	100	200
Maximum origins per origin group	50	50
Maximum origins per profile	100	200
Maximum origin timeout	16 - 240 secs	16 - 240 secs
Maximum routes per profile	100	200
Maximum rule set per profile	100	200
Maximum rules per route	100	100
Maximum rules per rule set	100	100
Maximum bandwidth ¹	75 Gbps	75 Gbps
Maximum requests per second per profile ¹	100,000	100,000
Path patterns to match for a routing rule	25	50
URLs in a single cache purge call	100	100
Maximum security policy per profile	100	200
Maximum associations per security policy	110	225
Maximum secrets per profile	100	500
HTTP header size limit (per header)	32 KB	32 KB
Web Application Firewall (WAF) policy per subscription	100	100
WAF custom rules per policy	100	100
WAF match conditions per custom rule	10	10
WAF custom regex rules per policy	5	5
WAF IP address ranges per match conditions	600	600
WAF string match values per match condition	10	10
WAF string match value length	256	256
WAF POST body parameter name length	256	256
WAF HTTP header name length	256	256
WAF cookie name length	256	256

Resource	Standard tier limit	Premium tier limit
WAF exclusion per policy	100	100
WAF HTTP request body and file upload inspection limit	128 KB	128 KB
WAF custom response body length	32 KB	32 KB

¹If the traffic isn't globally distributed and concentrated in one or more regions, or if a higher quota limited is need, create an [Azure support request](#).

Timeout values

From Client to Front Door

- Front Door has an idle TCP connection timeout of 61 seconds.

Front Door to application back-end

- After the HTTP request gets forwarded to the back end, Azure Front Door waits for 60 seconds (Standard and Premium) or 30 seconds (classic) for the first packet from the back end. Then it returns a 503 error to the client, or 504 for a cached request. You can configure this value using the *originResponseTimeoutSeconds* field in Azure Front Door Standard and Premium API, or the *sendRecvTimeoutSeconds* field in the Azure Front Door (classic) API.
- After the back end receives the first packet, if the origin pauses for any reason in the middle of the response body beyond the *originResponseTimeoutSeconds* or *sendRecvTimeoutSeconds*, the response will be canceled.
- Front Door takes advantage of HTTP keep-alive to keep connections open for reuse from previous requests. These connections have an idle timeout of 90 seconds. Azure Front Door would disconnect idle connections after reaching the 90-second idle timeout. This timeout value can't be configured.

Upload and download data limit

[Expand table](#)

With chunked transfer encoding (CTE)		Without HTTP chunking
Download	There's no limit on the download size.	There's no limit on the download size.
Upload	There's no limit as long as each CTE upload is less than 2 GB.	The size can't be larger than 2 GB.

Other limits

- Maximum URL size - 8,192 bytes - Specifies maximum length of the raw URL (scheme + hostname + port + path + query string of the URL)
- Maximum Query String size - 4,096 bytes - Specifies the maximum length of the query string, in bytes.
- Maximum HTTP response header size from health probe URL - 4,096 bytes - Specified the maximum length of all the response headers of health probes.
- Maximum rules engine action header value character: 640 characters.
- Maximum rules engine condition header value character: 256 characters.
- Maximum ETag header size: 128 bytes
- Maximum endpoint name for Standard and Premium: 46 characters.

For more information about limits that apply to Rules Engine configurations, see [rules engine terminology](#)

Azure Network Watcher limits

[Expand table](#)

Resource	Limit
Network Watcher instances per region per subscription	1 (One instance in a region to enable access to the service in the region)
Connection monitors per region per subscription	100
Maximum test groups per a connection monitor	20
Maximum sources and destinations per a connection monitor	100
Maximum test configurations per a connection monitor	20
Packet capture sessions per region per subscription	10,000 (Number of sessions only, not saved captures)
VPN troubleshoot operations per subscription	1 (Number of operations at one time)

Azure Route Server limits

[Expand table](#)

Resource	Limit
Number of BGP peers	8
Number of routes each BGP peer can advertise to Azure Route Server ¹	1,000
Number of VMs in the virtual network (including peered virtual networks) that Azure Route Server can support	4,000
Number of virtual networks that Azure Route Server can support	500
Number of total on-premises and Azure Virtual Network prefixes that Azure Route Server can support	10,000

¹ If your NVA advertises more routes than the limit, the BGP session gets dropped.

 Note

The total number of routes advertised from VNet address space and Route Server towards ExpressRoute circuit, when [Branch-to-branch](#) enabled, must not exceed 1,000. For more information, see [Route advertisement limits](#) of ExpressRoute.

Azure ExpressRoute limits

[Expand table](#)

Resource	Limit
ExpressRoute circuits per subscription	50 (Submit a support request to increase limit)
ExpressRoute circuits per region per subscription, with Azure Resource Manager	10
Maximum number of circuits in the same peering location linked to the same virtual network	4
Maximum number of circuits in different peering locations linked to the same virtual network	Standard / ERGw1Az - 4 High Perf / ERGw2Az - 8 Ultra Performance / ErGw3Az - 16
Maximum number of IPs for ExpressRoute provider circuit with Fastpath	25,000
Maximum number of IPs for ExpressRoute Direct 10 Gbps with Fastpath	100,000
Maximum number of IPs for ExpressRoute Direct 100 Gbps with Fastpath	200,000
Maximum number of flows for ExpressRoute Traffic Collector	300,000

Route advertisement limits

[Expand table](#)

Resource	Local / Standard SKU	Premium SKU
Maximum number of IPv4 on-prem routes advertised over Azure private peering to the ExpressRoute circuit	4,000	10,000
Maximum number of IPv6 on-prem routes advertised over Azure private peering to the ExpressRoute circuit	100	100
Maximum number of IPv4 Virtual Network routes advertised by the Gateway to the ExpressRoute circuit over Azure private peering	1,000	1,000
Maximum number of IPv6 Virtual Network routes advertised by the Gateway to the ExpressRoute circuit over Azure private peering	100	100
Maximum number of IPv4 routes advertised to Microsoft peering from on-premises	200	200
Maximum number of IPv6 routes advertised to Microsoft peering from on-premises	200	200

Virtual networks links allowed for each ExpressRoute circuit limit

[Expand table](#)

Circuit size	Local / Standard SKU	Premium SKU
50 Mbps	10	20
100 Mbps	10	25
200 Mbps	10	25
500 Mbps	10	40
1 Gbps	10	50
2 Gbps	10	60
5 Gbps	10	75
10 Gbps	10	100
40 Gbps*	10	100
100 Gbps*	10	100

*100-Gbps ExpressRoute Direct Only

Note

Global Reach connections count against the limit of virtual network connections per ExpressRoute Circuit. For example, a 10 Gbps Premium Circuit would allow for 5 Global Reach connections and 95 connections to the ExpressRoute Gateways or 95 Global Reach connections and 5 connections to the ExpressRoute Gateways or any other combination up to the limit of 100 connections for the circuit.

ExpressRoute gateway performance limits

The following tables provide an overview of the different types of gateways, their respective limitations, and their expected performance metrics.

Maximum supported limits

This table applies to both the Azure Resource Manager and classic deployment models.

[Expand table](#)

Gateway SKU	Megabits per second	Packets per second	Supported number of VMs in the virtual network ¹	Flow count limit	Number of routes learned by gateway
Standard/ERGw1Az	1,000	100,000	2,000	200,000	4,000
High Performance/ERGw2Az	2,000	200,000	4,500	400,000	9,500
Ultra Performance/ErGw3Az	10,000	1,000,000	11,000	1,000,000	9,500
ErGwScale (per scale unit 1-40)	1,000 per scale unit	100,000 per scale unit	2,000 per scale unit	100,000 per scale unit	60,000 total per gateway

¹ The values in the table are estimates and vary depending on the CPU utilization of the gateway. If the CPU utilization is high and the number of supported VMs is exceeded, the gateway will start to drop packets.

ⓘ Note

ExpressRoute can facilitate up to 11,000 routes that span virtual network address spaces, on-premises networks, and any relevant virtual network peering connections. To ensure stability of your ExpressRoute connection, refrain from advertising more than 11,000 routes to ExpressRoute. The maximum number of routes advertised by gateway is 1,000 routes.

ⓘ Important

- Application performance depends on multiple factors, such as end-to-end latency and the number of traffic flows that the application opens. The numbers in the table represent the upper limit that the application can theoretically achieve in an ideal environment. Additionally, we perform routine host and OS maintenance on the ExpressRoute virtual network gateway, to maintain reliability of the service. During a maintenance period, the control plane and data path capacity of the gateway is reduced.
- During a maintenance period, you might experience intermittent connectivity problems to private endpoint resources.
- ExpressRoute supports a maximum TCP and UDP packet size of 1,400 bytes. Packet sizes larger than 1,400 bytes will get fragmented.
- Azure Route Server can support up to 4,000 VMs. This limit includes VMs in virtual networks that are peered. For more information, see [Azure Route Server limitations](#).

Azure NAT Gateway limits

The following limits apply to NAT gateway resources managed through Azure Resource Manager per region per subscription. Learn how to [view your current resource usage against your subscription limits](#).

[\[+\] Expand table](#)

Resource	Limit
Public IP addresses	16 per NAT gateway
Subnets	800 per NAT gateway
Data throughput ¹	50 Gbps
NAT gateways for Enterprise and CSP agreements ²	1,000 per subscription per region
NAT gateways for Sponsored and pay-as-you-go ²	100 per subscription per region
NAT gateways for Free Trial and all other offer types ²	15 per subscription per region
Packets processed	1M - 5M packets per second
Connections to same destination endpoint	50,000 connections to the same destination per public IP
Connections total	2M connections per NAT gateway

¹ The total data throughput of 50 Gbps is split between outbound and inbound (return) data through a NAT gateway resource. Data throughput is rate limited at 25 Gbps for outbound data and 25 Gbps for inbound (response) data through NAT gateway.

² Default limits for NAT gateways vary by offer category type, such as Free Trial, pay-as-you-go, and CSP. For example, the default for Enterprise Agreement subscriptions is 1000.

Azure Private Link limits

The following limits apply to Azure private link:

[Expand table](#)

Resource	Limit
Number of private endpoints per virtual network	1000
Number of private endpoints across peered virtual networks	4000
Number of private endpoints per subscription	64000
Number of private link services per subscription	800
Number of private link services per Standard Load Balancer	8
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 subscriptions allowed in visibility setting on private link service	100
Number of subscriptions allowed in auto-approval setting on private link service	100
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
Number of private IP addresses on private endpoint network interface	500

Azure Traffic Manager limits

[Expand table](#)

Resource	Limit
Profiles per subscription	200 ¹
Endpoints per profile	200

¹If you need to increase these limits, contact Azure Support.

Azure VPN Gateway limits

Unless stated otherwise, the following limits apply to Azure VPN Gateway resources and virtual network gateways.

[Expand table](#)

Resource	Limit
VNet Address Prefixes	600 per VPN gateway
Aggregate BGP routes	4,000 per VPN gateway
Local Network Gateway address prefixes	1000 per local network gateway

Resource	Limit
S2S connections	Limit depends on the gateway SKU. See the Limits by gateway SKU table.
P2S connections	Limit depends on the gateway SKU. See the Limits by gateway SKU table.
P2S route limit - IKEv2	256 for non-Windows / 25 for Windows
P2S route limit - OpenVPN	1000
Max. flows	500K inbound and 500K outbound for VpnGw1-5/AZ
Traffic Selector Policies	100
Custom APIPA BGP addresses	32
Supported number of VMs in the virtual network	Limit depends on the gateway SKU. See the Limits by gateway SKU table.

Limits by gateway SKU

[+] [Expand table](#)

VPN Gateway Generation	SKU	S2S/VNet-to-VNet Tunnels	P2S SSTP Connections	P2S IKEv2/OpenVPN Connections	Aggregate Throughput Benchmark	BGP	Zone-redundant	Supported Number of VMs in the Virtual Network
Generation1	Basic	Max. 10	Max. 128	Not Supported	100 Mbps	Not Supported	No	200
Generation1	VpnGw1	Max. 30	Max. 128	Max. 250	650 Mbps	Supported	No	450
Generation1	VpnGw2	Max. 30	Max. 128	Max. 500	1 Gbps	Supported	No	1300
Generation1	VpnGw3	Max. 30	Max. 128	Max. 1000	1.25 Gbps	Supported	No	4000
Generation1	VpnGw1AZ	Max. 30	Max. 128	Max. 250	650 Mbps	Supported	Yes	1000
Generation1	VpnGw2AZ	Max. 30	Max. 128	Max. 500	1 Gbps	Supported	Yes	2000
Generation1	VpnGw3AZ	Max. 30	Max. 128	Max. 1000	1.25 Gbps	Supported	Yes	5000
Generation2	VpnGw2	Max. 30	Max. 128	Max. 500	1.25 Gbps	Supported	No	685
Generation2	VpnGw3	Max. 30	Max. 128	Max. 1000	2.5 Gbps	Supported	No	2240
Generation2	VpnGw4	Max. 100*	Max. 128	Max. 5000	5 Gbps	Supported	No	5300
Generation2	VpnGw5	Max. 100*	Max. 128	Max. 10000	10 Gbps	Supported	No	6700
Generation2	VpnGw2AZ	Max. 30	Max. 128	Max. 500	1.25 Gbps	Supported	Yes	2000
Generation2	VpnGw3AZ	Max. 30	Max. 128	Max. 1000	2.5 Gbps	Supported	Yes	3300
Generation2	VpnGw4AZ	Max. 100*	Max. 128	Max. 5000	5 Gbps	Supported	Yes	4400
Generation2	VpnGw5AZ	Max. 100*	Max. 128	Max. 10000	10 Gbps	Supported	Yes	9000

For more information about gateway SKUs and limits, see [About gateway SKUs](#).

Gateway performance limits

The table in this section lists the results of performance tests for VpnGw SKUs. A VPN tunnel connects to a VPN gateway instance. Each instance throughput is mentioned in the throughput table in the previous section and is available aggregated across all tunnels connecting to that instance. The table shows the observed bandwidth and packets per second throughput per tunnel for the different gateway SKUs. All testing was performed between gateways (endpoints) within Azure across different regions with 100 connections and under standard load conditions. We used publicly available iPerf and CTSTraffic tools to measure performances for site-to-site connections.

- The best performance was obtained when we used the GCMAES256 algorithm for both IPsec Encryption and Integrity.
- Average performance was obtained when using AES256 for IPsec Encryption and SHA256 for Integrity.
- The lowest performance was obtained when we used DES3 for IPsec Encryption and SHA256 for Integrity.

[Expand table](#)

Generation	SKU	Algorithms used	Throughput observed per tunnel	Packets per second per tunnel observed
Generation1	VpnGw1	GCMAES256	650 Mbps	62,000
		AES256 & SHA256	500 Mbps	47,000
		DES3 & SHA256	130 Mbps	12,000
Generation1	VpnGw2	GCMAES256	1.2 Gbps	100,000
		AES256 & SHA256	650 Mbps	61,000
		DES3 & SHA256	140 Mbps	13,000
Generation1	VpnGw3	GCMAES256	1.25 Gbps	120,000
		AES256 & SHA256	700 Mbps	66,000
		DES3 & SHA256	140 Mbps	13,000
Generation1	VpnGw1AZ	GCMAES256	650 Mbps	62,000
		AES256 & SHA256	500 Mbps	47,000
		DES3 & SHA256	130 Mbps	12,000
Generation1	VpnGw2AZ	GCMAES256	1.2 Gbps	110,000
		AES256 & SHA256	650 Mbps	61,000
		DES3 & SHA256	140 Mbps	13,000
Generation1	VpnGw3AZ	GCMAES256	1.25 Gbps	120,000
		AES256 & SHA256	700 Mbps	66,000
		DES3 & SHA256	140 Mbps	13,000
Generation2	VpnGw2	GCMAES256	1.25 Gbps	120,000
		AES256 & SHA256	550 Mbps	52,000
		DES3 & SHA256	130 Mbps	12,000
Generation2	VpnGw3	GCMAES256	1.5 Gbps	140,000
		AES256 & SHA256	700 Mbps	66,000
		DES3 & SHA256	140 Mbps	13,000
Generation2	VpnGw4	GCMAES256	2.3 Gbps	220,000
		AES256 & SHA256	700 Mbps	66,000
		DES3 & SHA256	140 Mbps	13,000
Generation2	VpnGw5	GCMAES256	2.3 Gbps	220,000
		AES256 & SHA256	700 Mbps	66,000
		DES3 & SHA256	140 Mbps	13,000
Generation2	VpnGw2AZ	GCMAES256	1.25 Gbps	120,000
		AES256 & SHA256	550 Mbps	52,000
		DES3 & SHA256	130 Mbps	12,000
Generation2	VpnGw3AZ	GCMAES256	1.5 Gbps	140,000
		AES256 & SHA256	700 Mbps	66,000
		DES3 & SHA256	140 Mbps	13,000
Generation2	VpnGw4AZ	GCMAES256	2.3 Gbps	220,000
		AES256 & SHA256	700 Mbps	66,000
		DES3 & SHA256	140 Mbps	13,000
Generation2	VpnGw5AZ	GCMAES256	2.3 Gbps	220,000
		AES256 & SHA256	700 Mbps	66,000
		DES3 & SHA256	140 Mbps	13,000

Azure Virtual WAN limits

[Expand table](#)

Resource	Limit
VPN (branch) connections per hub	1,000
Aggregate throughput per Virtual WAN Site-to-site VPN gateway	20 Gbps
Throughput per Virtual WAN VPN connection (2 tunnels)	2 Gbps with 1 Gbps/IPsec tunnel
Point-to-site users per hub	100,000
Aggregate throughput per Virtual WAN User VPN (Point-to-site) gateway	200 Gbps
Aggregate throughput per Virtual WAN ExpressRoute gateway	20 Gbps
ExpressRoute circuit connections per hub	8
VNet connections per hub	500 minus total number of hubs in Virtual WAN
Aggregate throughput per Virtual WAN hub router	50 Gbps for VNet to VNet transit
VM workload across all VNets connected to a single Virtual WAN hub	2000 (If you want to raise the limit or quota above the default limit, see hub settings).
Total number of routes the hub can accept from its connected resources (virtual networks, branches, other virtual hubs, etc.)	10,000

Azure Notification Hubs limits

[Expand table](#)

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](#).

Microsoft Dev Box limits

[Expand table](#)

Subscription type	VM Cores	Network Connections	Dev centers	Dev box definitions	Dev box projects
Pay as you go	20	5	2	200	500
Azure Pass	20	5	2	200	500
CSP	20	5	2	200	500
Free trial	0	0	0	0	0
Azure for Students	0	0	0	0	0
Enterprise	80	10	5	200	500
MSDN	n/a	5	2	200	500

Microsoft Purview limits

See [Classic Microsoft Purview data governance limits](#) for the most current Microsoft Purview quotas.

Microsoft Sentinel limits

See [Service limits for Microsoft Sentinel](#) for Microsoft Sentinel limits.

Azure 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](#).

[+] Expand table

Quota name	Scope	Value	Notes
Maximum number of namespaces per Azure subscription	Namespace	1000 (default and maximum)	This limit is based on the <code>Microsoft.ServiceBus</code> provider, not based on the tier. Therefore, it's the total number of namespaces across all tiers. 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. Total size limit for a premium namespace per messaging unit is 1 TB. Total size of all entities in a namespace can't exceed this limit.	Defined upon creation/updation of the queue or topic. Subsequent incoming messages are rejected, and an exception is received by the calling code. Currently, a large message (size > 1 MB) sent to a queue is counted twice. And, a large message (size > 1 MB) sent to a topic is counted X + 1 times, where X is the number of subscriptions to the topic.
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. Each partitioned queue or topic counts toward the quota of 1,000 entities per namespace.	Subsequent requests for creation of a new partitioned topic or queue in the namespace are rejected. As a result, if configured through the Azure portal , an error message is generated. If called from the management API, the exception <code>QuotaExceededException</code> is received by the calling code. If you want to have more partitioned entities in a basic or a standard tier namespace, create additional namespaces.
Maximum size of any messaging entity path: queue or topic	Entity	260 characters.	
Maximum size of any messaging entity name: namespace, subscription, or subscription rule	Entity	50 characters.	
Maximum size of a message ID	Entity	128	
Maximum size of a message	Entity	128	

Quota name	Scope	Value	Notes
session ID			
Message size for a queue, topic, or subscription entity	Entity	256 KB for Standard tier 100 MB for Premium tier on AMQP, and 1 MB for Premium on HTTP and SBMP.	Incoming messages that exceed these quotas are rejected, and an exception is received by the calling code.
		The maximum size for batches is 256 KB for the Standard tier, and 1 MB for the Premium tier.	
		The message size includes the size of properties (system and user) and the size of payload. The size of system properties varies depending on your scenario.	
Message property size for a queue, topic, or subscription entity	Entity	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.	The exception <code>SerializationException</code> is generated.
		Maximum number of header properties in property bag: <code>byte/int.MaxValue</code> .	
Number of subscriptions per topic	Entity	2,000 per-topic for the Standard tier and Premium tier.	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.
Number of SQL filters per topic	Entity	2,000	Subsequent requests for creation of additional filters on the topic are rejected, and an exception is received by the calling code.
Number of correlation filters per topic	Entity	100,000	Subsequent requests for creation of additional filters on the topic are rejected, and an exception is received by the calling code.
Size of SQL filters or actions	Namespace	Maximum length of filter condition string: 1,024 (1 K). Maximum length of rule action string: 1,024 (1 K).	Subsequent requests for creation of additional filters are rejected, and an exception is received by the calling code.
		Maximum number of expressions per rule action: 32.	
Number of shared access authorization rules per namespace, queue, or topic	Entity, namespace	Maximum number of rules per entity type: 12. Rules that are configured on a Service Bus namespace apply to all types: queues, topics.	Subsequent requests for creation of additional rules are rejected, and an exception is received by the calling code.
Number of messages per transaction	Transaction	100 For both <code>Send()</code> and <code>SendAsync()</code> operations.	Additional incoming messages are rejected, and an exception stating "Can't send more than 100 messages in a single transaction" is received by the calling code.
Maximum number of messages deleted in <code>DeleteMessagesAsync</code> call	Entity	4000	
Maximum number of messages returned in <code>PeekMessagesAsync</code> call	Entity	250	

Quota name	Scope	Value	Notes
Number of virtual network and IP filter rules	Namespace	128	

Azure Site Recovery limits

The following limits apply to Azure Site Recovery.

[Expand table](#)

Limit identifier	Limit
Number of vaults per subscription	500
Number of protected disks per subscription (Both Data and OS)	3000
Number of appliances 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

Azure SQL Database limits

For Azure SQL Database limits see:

- [Overview of Azure SQL Managed Instance resource limits](#)
- [Resource limits for single databases using the vCore purchasing model](#)
- [Resource limits for elastic pools using the vCore purchasing model](#)

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.

Azure Synapse limits for workspaces

For Pay-As-You-Go, Free Trial, Azure Pass, and Azure for Students subscription offer types:

[Expand table](#)

Resource	Default limit	Maximum limit
Synapse workspaces in an Azure subscription	2	2

For other subscription offer types:

[Expand table](#)

Resource	Default limit	Maximum limit
Synapse workspaces in an Azure subscription per region	20	100

Azure Synapse limits for Apache Spark

For Pay-As-You-Go, Free Trial, Azure Pass, and Azure for Students subscription offer types:

[Expand table](#)

Resource	Memory Optimized cores	GPU cores
Spark cores in a Synapse workspace	12	48

For other subscription offer types:

[Expand table](#)

Resource	Memory Optimized cores	GPU cores
Spark cores in a Synapse workspace	50	50

For additional limits for Spark pools, see [Concurrency and API rate limits for Apache Spark pools in Azure Synapse Analytics](#).

Azure Synapse limits for pipelines

[Expand table](#)

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	Find out how to request a quota increase from support .
Total CPU cores for Azure-SSIS Integration Runtimes under one workspace	256	Find out how to request a quota increase from 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
Concurrent authoring operations per workspace per Azure Integration Runtime region Including test connection, browse folder list and table list, preview data. This limit does not apply to Self-hosted IR.	200	200
Concurrent Data Integration Units ¹ consumption per workspace per Azure Integration Runtime region	Region group 1 ² : 6,000 Region group 2 ² : 3,000 Region group 3 ² : 1,500 Managed virtual network: Find out how to request a quota increase from support .	Region group 1 ² : 6,000 Region group 2 ² : 3,000 Region group 3 ² : 1,500 Managed virtual network ² : 2,400
Maximum activities per pipeline, which includes inner activities for containers	40	40
Maximum number of linked integration runtimes that can be created against a single self-hosted integration runtime	100	Find out how to request a quota increase from support .
Maximum parameters per pipeline	50	50
ForEach items	100,000	100,000
ForEach parallelism	20	50

Resource	Default limit	Maximum limit
Maximum queued runs per pipeline	100	100
Characters per expression	8,192	8,192
Minimum tumbling window trigger interval	5 min	15 min
Maximum timeout for pipeline activity runs	7 days	7 days
Bytes per object for pipeline objects ³	200 KB	200 KB
Bytes per object for dataset and linked service objects ³	100 KB	2,000 KB
Bytes per payload for each activity run ⁴	896 KB	896 KB
Data Integration Units ¹ per copy activity run	256	256
Write API calls	1,200/h	1,200/h
		This limit is imposed by Azure Resource Manager, not Azure Synapse Analytics.
Read API calls	12,500/h	12,500/h
		This limit is imposed by Azure Resource Manager, not Azure Synapse Analytics.
Monitoring queries per minute	1,000	1,000
Maximum time of data flow debug session	8 hrs	8 hrs
Concurrent number of data flows per integration runtime	50	Find out how to request a quota increase from support ↗ .
Concurrent number of data flows per integration runtime in managed vNet	20	Find out how to request a quota increase from 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	Find out how to request a quota increase from 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.

[+] [Expand table](#)

Region group	Regions
Region group 1	Central US, East US, East US 2, North Europe, West Europe, West US, West US 2
Region group 2	Australia East, Australia Southeast, Brazil South, Central India, Japan East, North Central US, South Central US, Southeast Asia, West Central US
Region group 3	Other regions

If managed virtual network is enabled, the data integration unit (DIU) in all region groups are 2,400.

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.

Azure Synapse limits for dedicated SQL pools

For details of capacity limits for dedicated SQL pools in Azure Synapse Analytics, see [dedicated SQL pool resource limits](#).

Azure Resource Manager limits for web service calls

Azure Resource Manager has limits for API calls. You can make API calls at a rate within the [Azure Resource Manager API limits](#).

Azure virtual machine disk limits

You can attach a number of data disks to an Azure virtual machine (VM). Based on the scalability and performance targets for a VM's data disks, you can determine the number and type of disk that you need to meet your performance and capacity requirements.

Important

For optimal performance, limit the number of highly utilized disks attached to the virtual machine to avoid possible throttling. If all attached disks aren't highly utilized at the same time, the virtual machine can support a larger number of disks. Additionally, when creating a managed disk from an existing managed disk, only 49 disks can be created concurrently. More disks can be created after some of the initial 49 have been created.

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.

 [Expand table](#)

Resource	Limit
Standard managed disks	50,000
Standard SSD managed disks	50,000
Premium SSD managed disks	50,000
Premium SSD v2 managed disks	1,000
Premium SSD v2 managed disks capacity ²	32,768
Ultra disks	1,000
Ultra disk capacity ²	32,768
Standard_LRS snapshots ¹	75,000
Standard_ZRS snapshots ¹	75,000
Managed image	50,000

¹An individual disk can have 500 incremental snapshots.

²This is the default max but higher capacities are supported by request. To request an increase in capacity, request a quota increase or contact Azure Support.

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.

For unmanaged disks, 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.

See [sizes for virtual machines in Azure](#) for more information.

For VM Applications

When working with VM applications in Azure, you may encounter an error message that says "Operation could not be completed as it results in exceeding approved UnmanagedStorageAccountCount quota." This error occurs when you have reached the limit for the number of unmanaged storage accounts that you can use.

When you publish a VM application, Azure needs to replicate it across multiple regions. To do this, Azure creates an unmanaged storage account for each region. The number of unmanaged storage accounts that an application uses is determined by the number of replicas across all applications.

As a general rule, each storage account can accommodate up to 200 simultaneous connections. Below are options for resolving the "UnmanagedStorageAccountCount" error:

- Use page blobs for your source application blobs. Unmanaged accounts are only used for block blob replication. Page blobs have no such limits.
- Reduce the number of replicas for your VM Application versions or delete applications you no longer need.
- File a support request to obtain a quota increase.

See [VM Applications overview](#) for more information.

Azure disk encryption sets

A limit of 5,000 disk encryption sets are allowed per region and per subscription. [Contact Azure support](#) to increase the quota.

See the following documentation to learn more about encryption restrictions:

- [Linux](#)
- [Windows](#) virtual machines

Azure-managed virtual machine disks

Standard HDD managed disks

[] Expand table

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
Base IOPS per disk	Up to 500	Up to 1,300	Up to 2,000	Up to 2,000							
*Expanded IOPS per disk	N/A	N/A	N/A	N/A	N/A	Up to 1,500	Up to 3,000	Up to 3,000	Up to 3,000	Up to 3,000	Up to 3,000
Base throughput per disk	Up to 60 MB/s	Up to 300 MB/s	Up to 500 MB/s	Up to 500 MB/s							

Standard Disk Type	S4	S6	S10	S15	S20	S30	S40	S50	S60	S70	S80
*Expanded throughput per disk	N/A	N/A	N/A	N/A	N/A	Up to 150 MB/s	Up to 300 MB/s	Up to 500 MB/s			

* Only applies to disks with performance plus (preview) enabled.

Standard SSD managed disks

[Expand table](#)

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
Base IOPS per disk	Up to 500	Up to 2,000	Up to 4,000	Up to 6,000										
*Expanded IOPS per disk	N/A	Up to 1,500	Up to 3,000	Up to 6,000	Up to 6,000	Up to 6,000	Up to 6,000							
Base throughput per disk	Up to 100 MB/s	Up to 400 MB/s	Up to 600 MB/s	Up to 750 MB/s										
*Expanded throughput per disk	N/A	Up to 150 MB/s	Up to 300 MB/s	Up to 600 MB/s	Up to 750 MB/s	Up to 750 MB/s	Up to 750 MB/s							
Max burst IOPS per disk	600	600	600	600	600	600	600	600	600	1000				
Max burst throughput per disk	150 MB/s	250 MB/s												
Max burst duration	30 min													

* Only applies to disks with performance plus (preview) enabled.

Premium SSD managed disks: Per-disk limits

[Expand table](#)

Premium SSD sizes	P1	P2	P3	P4	P6	P10	P15	P20	P30	P40	P50	P60	P70	P80
Disk size in GiB	4	8	16	32	64	128	256	512	1,024	2,048	4,096	8,192	16,384	32,767
Base provisioned IOPS per disk	120	120	120	120	240	500	1,100	2,300	5,000	7,500	7,500	16,000	18,000	20,000
**Expanded provisioned IOPS per disk	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	8,000	16,000	20,000	20,000	20,000	20,000
Base provisioned	25 MB/s	25 MB/s	25 MB/s	25 MB/s	50 MB/s	100 MB/s	125 MB/s	150 MB/s	200 MB/s	250 MB/s	250 MB/s	500 MB/s	750 MB/s	900 MB/s

Premium SSD sizes	P1	P2	P3	P4	P6	P10	P15	P20	P30	P40	P50	P60	P70	P80
Throughput per disk														
**Expanded provisioned throughput per disk	N/A	300 MB/s	600 MB/s	900 MB/s	900 MB/s	900 MB/s	900 MB/s							
Max burst IOPS per disk	3,500	3,500	3,500	3,500	3,500	3,500	3,500	3,500	30,000*	30,000*	30,000*	30,000*	30,000*	30,000*
Max burst throughput per disk	170 MB/s	1,000 MB/s*												
Max burst duration	30 min	Unlimited*	Unlimited*	Unlimited*	Unlimited*	Unlimited*	Unlimited*							
Eligible for reservation	No	Yes, up to one year												

*Applies only to disks with on-demand bursting enabled.

** Only applies to disks with performance plus (preview) enabled.

Premium SSD managed disks: Per-VM limits

[Expand table](#)

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

[Expand table](#)

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

[Expand table](#)

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

¹Ingress refers to all data from requests that are sent to a storage account. Egress refers to all data from responses that are received from a storage account.

Premium unmanaged virtual machine disks: Per-disk limits

[Expand table](#)

Premium storage disk type	P10	P20	P30	P40	P50
Disk size	128 GiB	512 GiB	1,024 GiB (1 TB)	2,048 GiB (2 TB)	4,095 GiB (4 TB)
Maximum IOPS per disk	500	2,300	5,000	7,500	7,500
Maximum throughput per disk	100 MB/sec	150 MB/sec	200 MB/sec	250 MB/sec	250 MB/sec
Maximum number of disks per storage account	280	70	35	17	8

Premium unmanaged virtual machine disks: Per-VM limits

[Expand table](#)

Resource	Limit
Maximum IOPS per VM	80,000 IOPS with GS5 VM
Maximum throughput per VM	2,000 MB/sec with GS5 VM

Azure StorSimple System limits

[Expand table](#)

Limit identifier	Limit	Comments
Maximum number of storage account credentials	64	
Maximum number of volume containers	64	
Maximum number of volumes	255	
Maximum number of schedules per bandwidth template	168	A schedule for every hour, every day of the week.
Maximum size of a tiered volume on physical devices	64 TB for StorSimple 8100 and StorSimple 8600	StorSimple 8100 and StorSimple 8600 are physical devices.
Maximum size of a tiered volume on virtual devices in Azure	30 TB for StorSimple 8010 64 TB for StorSimple 8020	StorSimple 8010 and StorSimple 8020 are virtual devices in Azure that use Standard storage and Premium storage, respectively.
Maximum size of a locally pinned volume on physical devices	9 TB for StorSimple 8100 24 TB for StorSimple 8600	StorSimple 8100 and StorSimple 8600 are physical devices.
Maximum number of iSCSI connections	512	
Maximum number of iSCSI connections from initiators	512	
Maximum number of access control records per device	64	
Maximum number of volumes per backup policy	24	
Maximum number of backups retained per backup policy	64	

Limit identifier	Limit	Comments
Maximum number of schedules per backup policy	10	
Maximum number of snapshots of any type that can be retained per volume	256	This 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.
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.
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.

Azure Stream Analytics limits

[+] [Expand table](#)

Limit identifier	Limit	Comments
Maximum number of streaming units per subscription per region	83	To request an increase in streaming units for your subscription beyond 83, 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	66	There's a hard limit of 66 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 1 or more SUs.
Maximum number of characters in a query	512000	There's a hard limit of 512k characters in an Azure Stream Analytics job query.

Azure Virtual Machines limits

Azure Virtual Machines limits

[+] [Expand table](#)

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.

Azure Virtual Machines limits - Azure Resource Manager

The following limits apply when you use Azure Resource Manager and Azure resource groups.

[+] [Expand table](#)

Resource	Limit
VMs per subscription ♂	25,000 ¹ per region.
VM total cores per subscription ♂	20 ¹ per region. Contact support to increase limit.
Azure Spot VM total cores per subscription ♂	20 ¹ per region. Contact support to increase limit.
VM per series, such as Dv2 and F, cores per subscription ♂	20 ¹ per region. Contact support to increase limit.
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

Resource	Limit
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.

Azure Compute Gallery limits

There are limits per subscription for deploying resources when you use Compute Galleries:

- 100 compute galleries per subscription and per region
- 1,000 image definitions per subscription and per region
- 10,000 image versions per subscription and per region

Managed Run Command limit

The maximum allowed Managed Run Commands is currently limited to 25.

Azure Virtual Machine Scale Sets limits

[+] [Expand table](#)

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 per subscription per region	2,500
Maximum number of nodes supported in VMSS for IB cluster	100

Azure Virtual Network Manager limits

[+] [Expand table](#)

Category	Limitation
General Limitations	
Cross-tenant Support	Only with static membership network groups
Azure Subscriptions	Policy application limited to < 15,000 subscriptions
Policy Enforcement Mode	No addition to network group if set to Disabled

Category	Limitation
Policy Evaluation Cycle	Standard evaluation cycle not supported
Subscription Movement	Moving subscription to another tenant not supported
Limits for Connectivity Configurations	
Virtual Networks in a Connected Group	A connected group can include up to 250 VNets by default, expandable to 1000 upon request using this form ↗ .
Private Endpoints	1000 private endpoints per connected group
Hub-and-Spoke Configuration	Max 1000 virtual networks peered to the hub
Direct Connectivity	Up to 250 VNets by default, expandable to 1000 upon request using this form ↗ .
Group Membership	A virtual network can be part of up to two connected groups, expandable to 1000 upon request using this form ↗ .
Overlapping IP Spaces	Communication to overlapped IP address is dropped
Limits for Security Admin Rules	
IP Prefixes	Max 1,000 IP prefixes combined
Admin Rules	Max 100 admin rules at one level
Limits for User Defined Routes	
User Defined Routes per Route Table	Max 1,000

Dev tunnels limits

The following limits apply to [dev tunnels ↗](#). The limits reset monthly.

[Expand table](#)

Resource	Limit
Bandwidth	5 GB per user
Tunnels	10 per user
Active connections	1000 per port
Ports	10 per tunnel
HTTP request rate	1500/min per port
Data transfer rate	Up to 20 MB/s per tunnel
Max web-forwarding HTTP request body size	16 MB

For questions on these limits, open an issue in our [GitHub repo ↗](#).

Network security perimeter limits

Scale limitations

Network security perimeter functionality can be used to support deployments of PaaS resources with common public network controls with following scale limitations:

[Expand table](#)

Limitation	Description
Number of network security perimeters	Supported up to 100 as recommended limit per subscription.
Profiles per network security perimeters	Supported up to 200 as recommended limit.
Number of rule elements per profile	Supported up to 200 as hard limit.
Number of PaaS resources across subscriptions associated with the same network security perimeter	Supported up to 1000 as recommended limit.

Other limitations

Network security perimeter has other limitations as follows:

[Expand table](#)

Limitation/Issue	Description
Resource group move operation cannot be performed if multiple network security perimeters are present	If there are multiple network security perimeters present in the same resource group, then the network security perimeter cannot be moved across resource groups/subscriptions.
Associations must be removed before deleting network security perimeter	Forced delete option is currently unavailable. Thus all associations must be removed before deleting a network security perimeter. Only remove associations after taking precautions for allowing access previously controlled by network security perimeter.
Resource names cannot be longer than 44 characters to support network security perimeter	The network security perimeter resource association created from the Azure portal has the format <code>{resourceName}-{perimeter-guid}</code> . To align with the requirement name field can't have more than 80 characters, resources names would have to be limited to 44 characters.
Service endpoint traffic is not supported.	It's recommended to use private endpoints for IaaS to PaaS communication. Currently, service endpoint traffic can be denied even when an inbound rule allows 0.0.0.0/0.

 Note

Refer to individual PaaS documentation for respective limitations for each service.

Next steps

Continue to the following resources to learn more:

- [Understand Azure Limits and Increases](#)
- [Sizes for virtual machines in Azure](#)
- [Sizes for Cloud Services \(classic\)](#)
- [Naming rules and restrictions for Azure resources](#)

Feedback

Was this page helpful? [!\[\]\(a63d68a1743546a44dc83151245ddcbb_img.jpg\) Yes](#) [!\[\]\(f347fe73540d6e1683b18e6d716574ed_img.jpg\) No](#)

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

Routine (planned) maintenance for Azure App Service

Article • 10/29/2024

Routine maintenance covers behind-the-scenes updates to Azure App Service. Types of maintenance can be performance improvements, bug fixes, new features, or security updates. App Service maintenance can be on the service itself or the underlying operating system.

Important

A breaking change or deprecation of functionality is not a part of routine maintenance. For more information, see [Modern Lifecycle Policy](#).

Microsoft service quality and uptime guarantees continue to apply during maintenance periods. Notifications mention maintenance periods to help customers get visibility into platform changes.

What to expect

Like personal computers, mobile phones, and other devices, machines in the cloud need the latest updates. Unlike physical devices, cloud solutions like Azure App Service provide ways to handle routine maintenance with more ease. There's no need to stop working and wait until patches are installed. Any workload can be shifted to different hardware in a matter of seconds and while updates are installed. The updates happen monthly but can vary, depending on your organization's needs and other factors.

Because a typical cloud solution consists of multiple applications, databases, storage accounts, functions, and other resources, parts of your solutions can undergo maintenance at different times. Some of this coordination is related to geography, region, datacenters, and availability zones. It can also be due to the cloud, where not everything is touched simultaneously. For more information, see [Safe deployment practices](#).

The following screenshot shows an example of a maintenance event.

The screenshot shows the Azure Service Health interface for 'Planned maintenance'. It lists 13 planned maintenance events. One specific event is highlighted with a red box:

Issue name	Tracking ID	Services	Regions	Start Time	End Time
End of Routine Planned Maintenance fo... (highlighted)	ABCD-EF2	App Service	UK South	2022-11-16T00:00:39Z	2022-11-26T00:00:39Z
Routine Advanced Maintenance advisor... (highlighted)	ABCD-EF1	App Service	North Europe, South Cent...	2022-11-18T00:00:00Z	2022-11-21T00:00:00Z

Below the table, there's a summary section with a red box around the 'Impact' category:

Impact Category: no impact expected

Maintenance summary: This notification is for upcoming scheduled maintenance to your App Service Plan in the South Central US and North Europe regions. Azure App Service will begin upgrading your resources as part of routine scheduled maintenance in around **7 days**. Once started in a given region, upgrades generally take between **24-78 hours to complete**. You may receive more than one notification per region if a subscription has resources on multiple scale units in a given region. Maintenance is optimized for non-business hours of a given region. During this maintenance, your Azure App Service application(s) may see a brief drop in connectivity for a few seconds.

Maintenance time/date:

- Start: 18 November 2022 00:00 UTC
- End: 21 November 2022 00:00 UTC

Expected duration of impact: no impact expected

In order from top to bottom, the example shows:

- A descriptive title of the maintenance event.
- Affected regions and subscriptions.
- The expected maintenance window.

Frequently asked questions

Why is the maintenance taking so long?

Fundamentally, routine maintenance delivers the latest updates to the platform and service. It's hard to predict how the maintenance will affect individual apps down to a specific time, so notifications tend to be more general. The time ranges in notifications don't reflect the experiences at the app level, but rather the overall operation across all resources. Apps that undergo maintenance instantly restart on freshly updated machines and continue working. There's no downtime when requests and traffic aren't served.

Why am I getting so many notifications?

A typical scenario is that customers have multiple applications that are upgraded at different times. To avoid sending notifications for each of them, we send one notification that captures multiple resources. We send the notification at the beginning and

throughout the maintenance window. You might receive multiple reminders for the same rollout if the time window is long, so you can more easily correlate any restarts, interruptions, or other issues.

How is routine maintenance related to SLA?

Platform maintenance shouldn't affect application uptime or availability. Applications continue to stay online while platform maintenance occurs.

Platform maintenance might cause applications to be cold started on new virtual machines, which can lead to delays. An application is still considered to be online while it's cold starting. To minimize or avoid cold starts, consider using [local cache for Windows apps](#) and [health check](#).

We don't expect sites to incur any service-level agreement (SLA) violations during the maintenance windows.

How does the upgrade ensure the smooth operation of my apps?

Azure App Service represents a fleet of scale units that provide hosting of web applications and solutions to customers. Each scale unit is divided into upgrade domains and availability zones. This division optimizes placements of bigger App Service plans and smooth deployments, because not all machines in each scale unit are updated at once.

Maintenance operations upgrade machines iteratively while App Service monitors the health of the fleet. If there's a problem, the system can stop the rollout. For more information about this process, see the blog post [Demystifying the magic behind App Service OS updates ↗](#).

Are business hours reflected?

Yes, business hours are reflected for the time zone of the region. Maintenance operations are optimized to start outside the standard business hours of 9 AM to 5 PM. Statistically, that's the best time for any interruptions and restarts of workloads because there's less stress on the system (in customer applications and transitively on the platform itself). If resources are still upgrading by 9 AM in a given region, the upgrade will safely pause before the next critical step and until the end of business hours.

What are my options to control routine maintenance?

If you run your workloads in an isolated product via App Service Environment v3, you can schedule the upgrades if necessary. For more information about this capability, see the blog post [Control and automate planned maintenance for App Service Environment v3](#).

Can I prepare my apps better for restarts?

If your applications need extra time during restarts to come online, consider using [health check](#). A typical pattern for needing extra time is heavy dependency on external resources during application warmup or startup.

You can use health check to inform the platform that your application isn't ready to receive requests yet. The system can use that information to route requests to other instances in your App Service plan. For such cases, we recommend that you have at least two instances in the plan.

My applications have been online, but things are worse since these notifications started showing up. What changed?

Updates and maintenance events have been happening to the platform since its inception. The frequency of updates decreased over time, so the number of interruptions also decreased and uptime increased. However, you now have more visibility into all changes. Increased visibility might cause the perception that more changes are happening.

Next steps

Get more information about maintenance notifications by reading the blog post [Routine Planned Maintenance Notifications for Azure App Service](#).

Feedback

Was this page helpful?



[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

Routine maintenance for Azure App Service, restarts, and downtime

Article • 09/19/2024

Azure App Service is a Platform as a Service (PaaS) for hosting web applications, REST APIs, and mobile back ends. One of the benefits of the offering is that planned maintenance is performed behind the scenes. Our customers can focus on deploying, running, and maintaining their application code instead of worrying about maintenance activities for the underlying infrastructure. Azure App Service maintenance is a robust process designed to avoid or minimize downtime to hosted applications. This process remains largely invisible to the users of hosted applications. However, our customers are often curious if downtime they experience is a result of our planned maintenance, especially if they seem to coincide in time.

Background

Our planned maintenance mechanism revolves around the architecture of the scale units that host the servers on which deployed applications run. Any given scale unit contains several different types of roles that all work together. The two roles that are most relevant to our planned maintenance update mechanism are the Worker and File Server roles. For a more detailed description of all the different roles and other details about the App Service architecture, review [Inside the Azure App Service Architecture](#)

There are different ways that an update strategy could be designed and those different designs would each have their own benefits and downsides. One of the strategies that we use for major updates is that these updates don't run on servers / roles that are currently used by our customers. Instead, our update process updates instances in waves and the instances undergoing updates aren't used by applications. Instances being used by applications are gradually swapped out and replaced by updated instances. The resulting effect on an application is that the application experiences a start, or restart. From a statistical perspective and from empirical observations, applications restarts are much less disruptive than performing maintenance on servers that are actively being used by applications.

Instance update details

There are two slightly different scenarios that play out during every Planned Maintenance cycle. These two scenarios are related to the updates performed on the Worker and File Server roles. At a high level, both these scenarios appear similar from an

end-user perspective but there are some important differences that can sometimes cause some unexpected behavior.

When a File Server role needs to be updated, the storage volume used by the application needs to be migrated from one File Server instance to another. During this change, an updated File Server role is added to the application. This causes a worker process restart simultaneously on all worker instances in that App Service Plan. The worker process restart is overlapped - the update mechanism starts the new worker process first, lets it complete its start-up, sends new requests to the new worker process. Once the new worker process is responding, the existing requests have 30 seconds by default to complete in the old worker process, then the old worker process is stopped.

When a Worker role is updated, the update mechanism similarly swaps in a new updated Worker role. The worker is swapped as follows - An updated Worker is added to the ASP, the application is started on the new Worker, our infrastructure waits for the application to start-up, new requests are sent to the new worker instance, requests are allowed to complete on the old instance, then the old worker instance is removed from the ASP. This sequence usually occurs once for each worker instance in the ASP and is spread out over minutes or hours depending on the size of the plan and scale unit.

The main differences between these two scenarios are:

- A File Server role change results in a simultaneous overlapped worker process restart on all instances, whereas a Worker change results in an application start on a single instance.
- A File Server role change means that the application restarts on the same instance as it was running before, whereas a Worker change results in the application running on a different instance after start-up.

The overlapped restart mechanism results in zero downtime for most applications and planned maintenance isn't even noticed. If the application takes some time to start, the application can experience some minimal downtime associated with application slowness or failures during or shortly after the process starts. Our platform keeps attempting to start the application until successful but if the application fails to start altogether, a longer downtime can occur. The downtime persists until some corrective action is taken, such as manually restarting the application on that instance.

Unexpected failure handling

While this article focuses largely on planned maintenance activities, it's worth mentioning that similar behavior can occur as a result of the platform recovering from unexpected failures. If an unexpected hardware failure occurs that affects a Worker role,

the platform similarly replaces it by a new worker. The application starts on this new Worker role. When a failure or latency affects a File Server role that is associated with the application, a new File Server role replaces it. A worker process restart occurs on all the Worker roles. This fact is important to consider when evaluating strategies for improving uptime for your applications.

Strategies for increased uptime

Most of our hosted applications experience limited or no downtime during planned maintenance. However, this fact isn't helpful if your specific applications have a more complicated start-up behavior and are therefore susceptible to downtime when restarted. If applications are experiencing downtime every time they're restarted, addressing the downtime is even more pressing. There are several features available in our App Service product offering that are designed to further minimize downtime in these scenarios. Broadly speaking there are two categories of strategies that can be employed:

- Improving application start-up consistency
- Minimizing application restarts

Improving application start-up speed and ensuring it's consistently successful has a higher success rate statistically. We recommend reviewing options that are available in this area first. Some of them are fairly easy to implement and can yield large improvements. Start-up consistency strategies utilize both App Service features and techniques related to application code or configuration. Minimizing restarts is a group of options that can be used if we can't improve application start-up to be consistent enough. These options are typically more expensive and less reliable as they usually protect against a subset of restarts. Avoiding all restarts isn't possible. Using both types of strategies is something that is highly effective.

Strategies for start-up consistency

Application Initialization (AppInit)

When an application starts on a Windows Worker, the Azure App Service infrastructure tries to determine when the application is ready to serve requests before external requests are routed to this worker. By default, a successful request to the root (/) of the application is a signal that the application is ready to serve requests. For some applications, this default behavior isn't sufficient to ensure that the application is fully warmed up. Typically that happens if the root of the application has limited

dependencies but other paths rely on more libraries or external dependencies to work. The [IIS Application Initialization Module](#) works well to fine tune warm-up behavior. At a high level, it allows the application owner to define which path or paths serve as indicators that the application is in fact ready to serve requests. For a detailed discussion of how to implement this mechanism, review the following article: [App Service Warm-Up Demystified ↗](#). When correctly implemented, this feature can result in zero downtime even if the application start-up is more complex.

Linux applications can utilize a similar mechanism by using the WEBSITE_WARMUP_PATH application setting.

Health Check

[Health Check](#) is a feature that is designed to handle unexpected code and platform failures during normal application execution but can also be helpful to augment start-up resiliency. Health Check performs two different healing functions - removing a failing instance from the load balancer, and replacing an entire instance. We can utilize the removal of an instance from the load balancer to handle intermittent start-up failures. If an instance returns failures after start-up despite employing all other strategies, health check can remove that instance from the load balancer until that instance starts returning 200 status code to health check requests again. This feature therefore acts as a fail-safe to minimize any post start-up downtime that occurs. This feature can be useful if the post start-up failures are transient and don't require process restart.

Auto-Heal

Auto-Heal for [Windows ↗](#) and [Linux ↗](#) is another feature that is designed for normal application execution but can be used for improving start-up behavior as well. If we know that the application sometimes enters an unrecoverable state after start-up, Health Check won't be suitable. However, auto-heal can automatically restart the worker process which can be useful in that scenario. We can configure an auto-heal rule that monitors failed requests and trigger a process restart on a single instance.

Application start-up testing

Testing the start-up of an application exhaustively can be overlooked. Start-up testing in combination with other factors such as dependency failures, library load failures, network issues etc. poses a bigger challenge. A relatively small failure rate for start-up can go unnoticed but can result in a high failure rate when there are multiple instances being restarted every update cycle. A plan with 20 instances and an application with a five-percent failure rate in start-up, results in three instances failing to start on average

every update cycle. There are usually three application restarts per instance (20 instance moves and 2 File Server related restarts per instance).

We recommend testing several scenarios

- General start-up testing (one instance at a time) to establish individual instance start-up success rate. This simplest scenario should approach 100 percent before moving on to other more complicated scenarios.
- Simulate start-up dependency failure. If the app has any dependency on other Azure or non-Azure services, simulate downtime in those dependencies to reveal application behavior under those conditions.
- Simultaneous start-up of many instances - preferably more instances than in production. Testing with many instances often reveals dependency failures that are often used during start-up only, such as KeyVault references, App Configuration, databases etc. These dependencies should be tested for burst volume of requests that a simultaneous instance restart generates.
- Adding an instance under full load - making sure ApInit is configured correctly and application can be initialized fully before requests are sent to the new instance. Manual scaling out is an easy way to replicate an instance move during maintenance.
- Overlapped worker process restart - again testing whether ApInit is configured correctly and if requests can complete successfully as the old worker process completes and new worker process starts up. Changing an environment variable under load can simulate what a File Server change does.
- Multiple apps in a plan - if there are multiple apps in the same plan, perform all these tests simultaneously across all apps.

Start-up logging

Having the ability to retroactively troubleshoot start-up failures in production is a consideration that is separate from using testing to improve start-up consistency. However, it's equally or more important since despite all our efforts, we might not be able to simulate all types of real-world failures in a test or QA environment. It's also commonly the weakest area for logging as initializing the logging infrastructure is another start-up activity that must be performed. The order of operations for initializing the application is an important consideration for this reason and can become a chicken and egg type of problem. For example, if we need to configure logging based on a KeyVault reference, and we fail to obtain the KeyVault value, how do we log this failure? We might want to consider duplicating start-up logging using a separate logging mechanism that doesn't depend on any other external factors. For example, logging these types of start-up failures to the local disk. Simply turning on a general logging

feature, such as [.NET Core stdout logging](#), can be counter-productive as this logging keeps generating log data even after start-up, and that can fill up the disk over time. This feature can be used strategically for troubleshooting reproducible start-up failures.

Strategies for minimizing restarts

The following strategies can significantly reduce the number of restarts that an application experiences during planned maintenance. Some of the strategies in this section can also give more control over when these restarts occur. In general, these strategies, while effective, can't avoid restarts altogether. The main reason is that some restarts occur due to unexpected failures rather than planned maintenance.

Important

Completely avoiding restarts is not possible. The following strategies can help reduce the number of restarts.

Local Cache

[Local Cache](#) is a feature that is designed to improve resiliency due to external storage failures. At a high level, it creates a copy of the application content on the local disk of the instance on which it runs. This isolates the application from unexpected storage failures but also prevents restarts due to File Server changes. Utilizing this feature can vastly reduce the number of restarts during public maintenance - typically it can remove about two thirds of those restarts. Since it primarily avoids simultaneous worker process restarts, the observed improvement on application start-up consistency can be even bigger. Local Cache does have some design implications and changes to application behavior so it's important to fully test the application to ensure that the application is compatible with this feature.

Planned maintenance notifications and paired regions

If we want to reduce the risk of update-related restarts in production, we can utilize [Planned Maintenance Notifications](#) to find out when any given application will be updated. We can then set up a copy of the application in a [Paired Region](#) and route traffic to our secondary application copy during maintenance in the primary copy. This option can be costly as the window for this maintenance is fairly wide so the secondary application copy needs to run on sufficient instances for at least several days. This option can be less costly if we already have a secondary application set up for general

resiliency. This option can reduce the number of restarts but like other options in this category can't eliminate all restarts.

Controlling planned maintenance window in ASE v3

Controlling the window for maintenance is only available in our isolated ASE v3 environments. If we're using an ASE already, or it's feasible to use an ASE, doing so allows our customers to [Control Planned Maintenance](#) behavior to a high degree. It isn't possible to control the time of the planned maintenance in a multitenant environment.

Feedback

Was this page helpful?



Yes



No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

Language runtime support policy for App Service

Article • 05/13/2024

This article describes the language runtime support policy for updating existing stacks and retiring end-of-support stacks in Azure App Service. This policy clarifies existing practices and doesn't represent a change to customer commitments.

Updates to existing stacks

App Service updates existing stacks after they become available from each community. App Service updates major versions of stacks but can't guarantee any specific minor or patch versions. The platform controls minor and patch versions. For example, App Service updates Node 18 but doesn't guarantee a specific Node 18.x.x version. If you need a specific minor or patch version, you can use a [custom container](#).

Retirements

App Service follows community support timelines for the lifecycle of the runtime. After community support for a language reaches the end of support, your applications continue to run unchanged. However, App Service can't provide security patches or related customer support for that runtime version past its end-of-support date. If your application has any problems past the end-of-support date for that version, you should move up to a supported version to receive the latest security patches and features.

Important

If you're running apps that use an unsupported language version, you need to upgrade to a supported language version before you can get support for those apps.

Notifications

End-of-support dates for runtime versions are determined independently by their respective stacks and are outside the control of App Service. App Service sends reminder notifications to subscription owners for upcoming end-of-support runtime versions when they become available for each language.

Roles that receive notifications include account administrators, service administrators, and coadministrators. Contributors, readers, or other roles don't directly receive notifications unless they opt in to receive notification emails, using [Service Health Alerts](#).

Timelines for language runtime version support

To learn more about specific timelines for the language support policy, see the following resources:

- [.NET and ASP.NET Core ↗](#)
- [.NET Framework and ASP.NET ↗](#)
- [Node ↗](#)
- [Java ↗](#)
- [Python ↗](#)
- [PHP ↗](#)
- [Go ↗](#)

Configure language versions

To learn more about how to update language versions for your App Service applications, see the following resources:

- [.NET ↗](#)
- [Node ↗](#)
- [Java ↗](#)
- [Python ↗](#)
- [PHP ↗](#)

Java-specific runtime statement of support

- [JDK versions and maintenance](#)
- [Security updates](#)
- [Deprecation and retirement](#)
- [Local development](#)

JDK versions and maintenance

Microsoft and Adoptium builds of OpenJDK are provided and supported on App Service for Java 8, 11, 17, and 21. These binaries are provided as a no-cost, multi-platform, production-ready distribution of the OpenJDK for Azure. They contain all the

components for building and running Java SE applications. For local development or testing, you can install the Microsoft build of OpenJDK from the [downloads page](#).

Windows

 [Expand table](#)

Java stack name	Windows version	Java distribution
Java SE, Java 8	Windows Server 2022	Adoptium Temurin 8
Java SE, Java 11	Windows Server 2022	MSFT OpenJDK 11
Java SE, Java 17	Windows Server 2022	MSFT OpenJDK 17
Java SE, Java 21	Windows Server 2022	MSFT OpenJDK 21
Tomcat 8.5, Java 8	Windows Server 2022	Adoptium Temurin 8
Tomcat 8.5, Java 11	Windows Server 2022	MSFT OpenJDK 11
Tomcat 9.0, Java 8	Windows Server 2022	Adoptium Temurin 8
Tomcat 9.0, Java 11	Windows Server 2022	MSFT OpenJDK 11
Tomcat 9.0, Java 17	Windows Server 2022	MSFT OpenJDK 17
Tomcat 9.0, Java 21	Windows Server 2022	MSFT OpenJDK 21
Tomcat 10.0, Java 8	Windows Server 2022	Adoptium Temurin 8
Tomcat 10.0, Java 11	Windows Server 2022	MSFT OpenJDK 11
Tomcat 10.0, Java 17	Windows Server 2022	MSFT OpenJDK 17
Tomcat 10.0, Java 21	Windows Server 2022	MSFT OpenJDK 21
Tomcat 10.1, Java 11	Windows Server 2022	MSFT OpenJDK 11
Tomcat 10.1, Java 17	Windows Server 2022	MSFT OpenJDK 17
Tomcat 10.1, Java 21	Windows Server 2022	MSFT OpenJDK 21

If you're [pinned](#) to an older minor version of Java, your app might be using the deprecated [Azul Zulu for Azure](#) binaries provided through [Azul Systems](#). You can keep using these binaries for your app, but any security patches or improvements are available only in new versions of the OpenJDK, so we recommend that you periodically update your Web Apps to a later version of Java.

Major version updates are provided through new runtime options in Azure App Service. Customers update to these newer versions of Java by configuring their App Service deployment and are responsible for testing and ensuring the major update meets their needs.

Supported JDKs are automatically patched on a quarterly basis in January, April, July, and October of each year. For more information on Java on Azure, see [this support document](#).

Security updates

Patches and fixes for major security vulnerabilities are released as soon as they become available in Microsoft builds of the OpenJDK. A "major" vulnerability has a base score of 9.0 or higher on the [NIST Common Vulnerability Scoring System, version 2](#).

Tomcat 8.5 reached [End of Life as of March 31, 2024](#) and Tomcat 10.0 reached [End of Life as of October 31, 2022](#).

While the runtimes are still available on Azure App Service, Tomcat 8.5 or 10.0 won't receive security updates.

When possible, migrate your applications to Tomcat 9.0 or Tomcat 10.1. Tomcat 9.0 and Tomcat 10.1 are available on Azure App Service. For more information, see the [official Tomcat site](#).

Community support for Java 7 ended on July 29, 2022 and [Java 7 was retired from App Service](#). If you have a web app running on Java 7, upgrade to Java 8 or 11 immediately.

Deprecation and retirement

If a supported Java runtime is retired, Azure developers using the affected runtime receive a deprecation notice at least six months before the runtime is retired.

- [Reasons to move to Java 11](#)
- [Java 7 migration guide](#)

Local development

Developers can download the Microsoft Build of OpenJDK for local development from [our download site](#).

Product support for the [Microsoft Build of OpenJDK](#) is available through Microsoft when developing for Azure or [Azure Stack](#) with a [qualified Azure support plan](#).

Operating system functionality in Azure App Service

Article • 01/08/2024

This article describes the baseline operating system functionality that's available to all Windows apps running in [Azure App Service](#). This functionality includes file, network, and registry access, along with diagnostics logs and events.

ⓘ Note

Linux apps in App Service run in their own containers. You have root access to the container but no access to the host operating system. Likewise, for [apps running in Windows containers](#), you have administrative access to the container but no access to the host operating system.

App Service plan tiers

App Service runs customer apps in a multitenant hosting environment. Apps deployed in the Free and Shared tiers run in worker processes on shared virtual machines (VMs). Apps deployed in the Standard and Premium tiers run on VMs dedicated specifically for the apps associated with a single customer.

ⓘ Note

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

Because App Service supports a seamless scaling experience between tiers, the security configuration enforced for App Service apps remains the same. This configuration ensures that apps don't suddenly behave differently and fail in unexpected ways when an App Service plan switches from one tier to another.

Development frameworks

App Service pricing tiers control the amount of compute resources (CPU, disk storage, memory, and network egress) available to apps. However, the breadth of framework functionality available to apps remains the same regardless of the scaling tiers.

App Service supports various development frameworks, including ASP.NET, classic ASP, Node.js, PHP, and Python. To simplify and normalize security configuration, App Service apps typically run the development frameworks with their default settings. The frameworks and runtime components that the platform provides are updated regularly to satisfy security and compliance requirements. For this reason, we don't guarantee specific minor/patch versions. We recommend that customers target major versions as needed.

The following sections summarize the general kinds of operating system functionality available to App Service apps.

File access

Various drives exist within App Service, including local drives and network drives.

Local drives

At its core, App Service is a service running on top of the Azure platform as a service (PaaS) infrastructure. As a result, the local drives that are associated with a virtual machine are the same drive types available to any worker role running in Azure. They include:

- An operating system drive (%SystemDrive%) whose size depends on the size of the VM.
- A resource drive (%ResourceDrive%) that App Service uses internally.

A best practice is to always use the environment variables %SystemDrive% and %ResourceDrive% instead of hard-coded file paths. The root path returned from these two environment variables has shifted over time from d:\ to c:. However, older applications hard-coded with file path references to d:\ continue to work because App Service automatically remaps d:\ to point at c:. As noted earlier, we highly recommend that you always use the environment variables when building file paths and avoid confusion over platform changes to the default root file path.

It's important to monitor your disk utilization as your application grows. Reaching the disk quota can have adverse effects on your application. For example:

- The app might throw an error that indicates there's not enough space on the disk.

- You might see disk errors when browsing to the Kudu console.
- Deployment from Azure DevOps or Visual Studio might fail with `ERROR_NOT_ENOUGH_DISK_SPACE: Web deployment task failed. (Web Deploy detected insufficient space on disk).`
- Your app might have slow performance.

Network drives (UNC shares)

One of the unique aspects of App Service that make app deployment and maintenance straightforward is that all content shares are stored on a set of UNC shares. This model maps well to the common pattern of content storage used by on-premises web hosting environments that have multiple load-balanced servers.

Within App Service, UNC shares are created in each datacenter. A percentage of the user content for all customers in each datacenter is allocated to each UNC share. Each customer's subscription has a reserved directory structure on a specific UNC share in a datacenter. A customer might have multiple apps created in a specific datacenter, so all of the directories that belong to a single customer subscription are created on the same UNC share.

Because of the way that Azure services work, the specific virtual machine responsible for hosting a UNC share changes over time. UNC shares are mounted by different virtual machines as they're brought up and down during the normal course of Azure operations. For this reason, apps should never make hard-coded assumptions that the machine information in a UNC file path will remain stable over time. Instead, they should use the convenient *faux* absolute path `%HOME%\site` that App Service provides.

The faux absolute path is a portable method for referring to your own app. It's not specific to any app or user. By using `%HOME%\site`, you can transfer shared files from app to app without having to configure a new absolute path for each transfer.

Types of file access granted to an app

The `%HOME%` directory in an app maps to a content share in Azure Storage dedicated for that app. Your [pricing tier](#) defines its size. It might include directories such as those for content, error and diagnostic logs, and earlier versions of the app that source control created. These directories are available to the app's application code at runtime for read and write access. Because the files aren't stored locally, they're persistent across app restarts.

On the system drive, App Service reserves `%SystemDrive%\local` for app-specific temporary local storage. Changes to files in this directory are *not* persistent across app restarts. Although an app has full read and write access to its own temporary local storage, that storage isn't intended for direct use by the application code. Rather, the intent is to provide temporary file storage for IIS and web application frameworks.

App Service limits the amount of storage in `%SystemDrive%\local` for each app to prevent individual apps from consuming excessive amounts of local file storage. For Free, Shared, and Consumption (Azure Functions) tiers, the limit is 500 MB. The following table lists other tiers:

[+] [Expand table](#)

Tier	Local file storage
B1/S1/P1	11 GB
B2/S2/P2	15 GB
B3/S3/P3	58 GB
P0v3	11 GB
P1v2/P1v3/P1mv3/Isolated1/Isolated1v2	21 GB
P2v2/P2v3/P2mv3/Isolated2/Isolated2v2	61 GB
P3v2/P3v3/P3mv3/Isolated3/Isolated3v2	140 GB
Isolated4v2	276 GB
P4mv3	280 GB
Isolated5v2	552 GB
P5mv3	560 GB
Isolated6v2	1,104 GB

Two examples of how App Service uses temporary local storage are the directory for temporary ASP.NET files and the directory for IIS compressed files. The ASP.NET compilation system uses the `%SystemDrive%\local\Temporary ASP.NET Files` directory as a temporary compilation cache location. IIS uses the `%SystemDrive%\local\IIS Temporary Compressed Files` directory to store compressed response output. Both of these types of file usage (along with others) are remapped in App Service to per-app temporary local storage. This remapping helps ensure that functionality continues as expected.

Each app in App Service runs as a random, unique, low-privileged worker process identity called the [application pool identity](#). Application code uses this identity for basic read-only access to the operating system drive. This access means that application code can list common directory structures and read common files on the operating system drive. Although this level of access might seem to be broad, the same directories and files are accessible when you provision a worker role in an Azure-hosted service and read the drive contents.

File access across multiple instances

The content share (`%HOME%`) directory contains an app's content, and application code can write to it. If an app runs on multiple instances, the `%HOME%` directory is shared among all instances so that all instances see the same directory. For example, if an app saves uploaded files to the `%HOME%` directory, those files are immediately available to all instances.

The temporary local storage (`%SystemDrive%\local`) directory is not shared between instances. It's also not shared between the app and its [Kudu app](#).

Network access

Application code can use TCP/IP and UDP-based protocols to make outbound network connections to internet-accessible endpoints that expose external services. Apps can use these same protocols to connect to services within Azure--for example, by establishing HTTPS connections to Azure SQL Database.

There's also a limited capability for apps to establish one local loopback connection and have an app listen on that local loopback socket. This feature enables apps that listen on local loopback sockets as part of their functionality. Each app has a private loopback connection. One app can't listen to a local loopback socket that another app established.

Named pipes are also supported as a mechanism for interprocess communication between processes that collectively run an app. For example, the IIS FastCGI module relies on named pipes to coordinate the individual processes that run PHP pages.

Code execution, processes, and memory

As noted earlier, apps run inside low-privileged worker processes by using a random application pool identity. Application code has access to the memory space associated with the worker process, along with any child processes that CGI processes or other

applications might spawn. However, one app can't access the memory or data of another app, even if it's on the same virtual machine.

Apps can run scripts or pages written with supported web development frameworks. App Service doesn't configure any web framework settings to more restricted modes. For example, ASP.NET apps running in App Service run in full trust, as opposed to a more restricted trust mode. Web frameworks, including both classic ASP and ASP.NET, can call in-process COM components (like ActiveX Data Objects) that are registered by default on the Windows operating system. Web frameworks can't call out-of-process COM components.

An app can spawn and run arbitrary code, open a command shell, or run a PowerShell script. However, executable programs and scripts are still restricted to the privileges granted to the parent application pool. For example, an app can spawn an executable program that makes an outbound HTTP call, but that executable program can't try to unbind the IP address of a virtual machine from its network adapter. Making an outbound network call is allowed for low-privileged code, but trying to reconfigure network settings on a virtual machine requires administrative privileges.

Diagnostics logs and events

Log information is another set of data that some apps try to access. The types of log information available to code running in App Service include diagnostic and log information that an app generates and can easily access.

For example, app-generated W3C HTTP logs are available either:

- In a log directory in the network share location that you created for the app
- In blob storage if you set up W3C logging to storage

The latter option enables apps to gather large amounts of logs without exceeding the file storage limits associated with a network share.

Similarly, real-time diagnostics information from .NET apps can be logged through the .NET tracing and diagnostics infrastructure. You can then write the trace information to either the app's network share or a blob storage location.

Areas of diagnostics logging and tracing that aren't available to apps are Windows Event Tracing for Windows (ETW) events and common Windows event logs (for example, system, application, and security event logs). Because ETW trace information can potentially be viewable across a machine (with the right access control lists), read access and write access to ETW events are blocked. API calls to read and write ETW events and

common Windows event logs might seem to work, but in reality, the application code has no access to this event data.

Registry access

Apps have read-only access to much (though not all) of the registry of the virtual machine that they're running on. This access means that apps can access registry keys that allow read-only access to the Local Users group. One area of the registry that's currently not supported for either read or write access is the `HKEY_CURRENT_USER` hive.

Write access to the registry is blocked, including access to any per-user registry keys. From the app's perspective, it can't rely on write access to the registry in the Azure environment because apps can be migrated across virtual machines. The only persistent writeable storage that an app can depend on is the per-app content directory structure stored on the App Service UNC shares.

Remote desktop access

App Service doesn't provide remote desktop access to the VM instances.

More information

For the most up-to-date information about the execution environment of App Service, see the [Azure App Service sandbox](#). The App Service development team maintains this page.

Kudu service overview

Article • 01/29/2025

Kudu is the engine behind some features in [Azure App Service](#) that are related to source-control-based deployment and other deployment methods, like Dropbox and OneDrive sync.

Access Kudu for your app

Anytime you create an app, App Service creates a companion app for it that's secured by HTTPS. This Kudu app is accessible at these URLs:

- App not in the Isolated tier: `https://<app-name>.scm.azurewebsites.net`
- Internet-facing app in the Isolated tier (App Service Environment): `https://<app-name>.scm.<ase-name>.p.azurewebsites.net`
- Internal app in the Isolated tier (App Service Environment for internal load balancing): `https://<app-name>.scm.<ase-name>.appserviceenvironment.net`

For more information, see [Accessing the Kudu service](#).

Kudu features

Kudu gives you helpful information about your App Service app, such as:

- App settings
- Connection strings
- Environment variables
- Server variables
- HTTP headers

It also provides features like these:

- Run commands in the [Kudu console](#).
- Download IIS diagnostic dumps or Docker logs.
- Manage IIS processes and site extensions.
- Add deployment webhooks for Windows apps.
- Allow ZIP deployment UI with `/ZipDeploy`.
- Generate [custom deployment scripts](#).
- Allow access with a [REST API](#).

RBAC permissions required to access Kudu

To access Kudu in the browser by using Microsoft Entra authentication, you need to be assigned an appropriate built-in or custom role over the scope of the application. The assigned role must include permission for the `Microsoft.Web/sites/publish/Action` resource provider operation. The following table shows example built-in roles that include this permission.

[+] Expand table

Role type	Example built-in roles
Job function roles	Website Contributor Logic Apps Standard Developer (Preview)
Privileged administrator roles ¹	Owner Contributor

¹ Privileged administrator roles grant much more permission than is needed to access Kudu. If need to create a new role assignment, consider if a job function role with less access can be used instead.

See the [role-based access control overview](#) to learn more about creating role assignments.

More resources

Kudu is an [open-source project](#). It has documentation on the [Kudu wiki](#).

Feedback

Was this page helpful?

 Yes  No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

Configure gRPC on App Service

Article • 12/19/2023

This article explains how to configure your web app for gRPC.

gRPC is a Remote Procedure Call framework that can streamline messages between your client and server over HTTP/2. Using the gRPC protocol over HTTP/2 enables the use of features like:

- Multiplexing to send multiple parallel requests over the same connection.
- Bidirectional streaming to send requests and responses simultaneously.

Support for gRPC is currently available on Azure App Service for Linux. To use gRPC on your web app, you need to configure your app by selecting the HTTP version, proxy, and port.

For gRPC client and server samples for each supported language, see the [documentation on GitHub](#).

Prerequisite

Create your [web app](#) as you normally would. Choose your preferred runtime stack, and choose Linux as your operating system.

After you create your web app, configure the following details to enable gRPC before you deploy your application.

Note

If you're deploying a .NET gRPC app to App Service by using Visual Studio, skip to [step 3](#). Visual Studio sets the HTTP version and the HTTP 2.0 proxy configuration for you.

1. Configure the HTTP version

The first setting that you need to configure is the HTTP version:

1. On the left pane of your web app, under **Settings**, go to **Configuration**.
2. On the **General Settings** tab, scroll down to **Platform settings**.
3. In the **HTTP version** dropdown list, select **2.0**.
4. Select **Save**.

This setting restarts your application and configures the front end to allow clients to make HTTP/2 calls.

2. Configure the HTTP 2.0 proxy

Next, you need to configure the HTTP 2.0 proxy:

1. In the same **Platform settings** section, find the **HTTP 2.0 Proxy** setting and select **gRPC Only**.
2. Select **Save**.

This setting configures your site to receive HTTP/2 requests.

3. Configure the HTTP/2 port

App Service requires an application setting that specifically listens for HTTP/2 traffic in addition to HTTP/1.1 traffic. You define the HTTP/2 port in the app settings:

1. On the left pane of your web app, under **Settings**, go to **Environment variables**.
2. On the **App settings** tab, add the following app settings to your application:
 - **Name** = **HTTP20_ONLY_PORT**
 - **Value** = **8585**

These settings configure the port on your application that's specified to listen for HTTP/2 requests.

Now that you've configured the HTTP version, port, and proxy, you can successfully make HTTP/2 calls to your web app by using gRPC.

(Optional) Python startup command

For Python applications only, you also need to set a custom startup command:

1. On the left pane of your web app, under **Settings**, go to **Configuration**.
2. Under **General Settings**, add the following value for **Startup Command**: `python app.py`.

Common topics

The following table can answer your questions about using gRPC with App Service.

 **Note**

gRPC is not a supported feature in App Service Environment v2. Use App Service Environment v3.

 [Expand table](#)

Topic	Answer
OS support	gRPC is available on Linux. Windows support is currently in preview.
Language support	gRPC is supported for each language that supports gRPC.
Client certificates	HTTP/2 enabled on App Service doesn't currently support client certificates. Client certificates need to be ignored when you're using gRPC.
Secure calls	gRPC must make secure HTTP calls to App Service. You can't make nonsecure calls.
Activity timeout	gRPC requests on App Service have a timeout request limit. gRPC requests time out after 20 minutes of inactivity.
Custom containers	HTTP/2 and gRPC support is in addition to App Service HTTP/1.1 support. Custom containers that support HTTP/2 must also support HTTP/1.1.

Recommended services (preview)

Article • 03/05/2024

This feature suggests a set of services that are commonly deployed together for your specific app type. It offers guidance on structuring your app effectively using proven patterns.

The upcoming version of this feature will provide better smart defaults and greater customization tailored to your app, drawing from successful patterns used by other Azure developers, architects, and DevOps engineers.

Your app will be deployed as

- Secure-by-default
- On a new App Service plan

Requirements to use this feature

The following Basic tab items must be filled out:

1. Subscription
2. Resource group
3. Runtime stack
4. New app service plan

Database

The database picked for you in this list is Runtime-aware and generate a default name that can be customized. [Learn more about database types on Azure.](#) ↗

Inputs:

- **Name**: a default name is generated and can be customized
- **Database Type**: the database default matches community preferences for each Runtime

Azure Cache for Redis

Redis improves the performance and scalability of an app that uses backend data stores heavily. [Learn more about Azure Cache for Redis.](#)

Inputs:

- **Name**: a default name is generated and can be customized
- **SKU**: Dev/Test, Production, or Free

App Service, Functions, and Logic Apps on Azure Arc (Preview)

Article • 01/15/2025

You can run App Service, Functions, and Logic Apps on an Azure Arc-enabled Kubernetes cluster. The Kubernetes cluster can be on-premises or hosted in a third-party cloud. This approach lets app developers take advantage of the features of App Service. At the same time, it lets their IT administrators maintain corporate compliance by hosting the App Service apps on internal infrastructure. It also lets other IT operators safeguard their prior investments in other cloud providers by running App Service on existing Kubernetes clusters.

ⓘ Note

To learn how to set up your Kubernetes cluster for App Service, Functions, and Logic Apps, see [Create an App Service Kubernetes environment \(Preview\)](#).

In most cases, app developers need to know nothing more than how to deploy to the correct Azure region that represents the deployed Kubernetes environment. For operators who provide the environment and maintain the underlying Kubernetes infrastructure, you must be aware of the following Azure resources:

- The connected cluster, which is an Azure projection of your Kubernetes infrastructure. For more information, see [What is Azure Arc-enabled Kubernetes?](#).
- A cluster extension, which is a subresource of the connected cluster resource. The App Service extension [installs the required pods into your connected cluster](#). For more information about cluster extensions, see [Cluster extensions on Azure Arc-enabled Kubernetes](#).
- A custom location, which bundles together a group of extensions and maps them to a namespace for created resources. For more information, see [Custom locations on top of Azure Arc-enabled Kubernetes](#).
- An App Service Kubernetes environment, which enables configuration common across apps but not related to cluster operations. Conceptually, it's deployed into the custom location resource, and app developers create apps into this environment. This resource is described in greater detail in [App Service Kubernetes environment](#).

Public preview limitations

The following public preview limitations apply to App Service Kubernetes environments. This list of limitations is updated as changes and features are made available.

[+] [Expand table](#)

Limitation	Details
Supported Azure regions	East US, West Europe
Cluster networking requirement	Must support <code>LoadBalancer</code> service type
Node OS requirement	Linux only.
Cluster storage requirement	Must have cluster attached storage class available for use by the extension to support deployment and build of code-based apps where applicable
Feature: Networking	Not available (rely on cluster networking)
Feature: Managed identities	Not available
Feature: Key vault references	Not available (depends on managed identities)
Feature: Pull images from ACR with managed identity	Not available (depends on managed identities)
Feature: In-portal editing for Functions and Logic Apps	Not available
Feature: Portal listing of Functions or keys	Not available if cluster isn't publicly reachable
Feature: FTP publishing	Not available
Logs	Log Analytics must be configured with cluster extension; not per-site

Pods created by the App Service extension

When the App Service extension is installed on the Azure Arc-enabled Kubernetes cluster, several pods are created in the release namespace that was specified. These pods enable your Kubernetes cluster to be an extension of the `Microsoft.Web` resource provider in Azure and support the management and operation of your apps. Optionally, you can choose to have the extension install [KEDA](#) for event-driven scaling.

The following table describes the role of each pod that is created by default:

Pod	Description
<extensionName>-k8se-app-controller	The core operator pod that creates resources on the cluster and maintains the state of components.
<extensionName>-k8se-envoy	A front-end proxy layer for all data-plane requests. It routes the inbound traffic to the correct apps.
<extensionName>-k8se-activator	An alternative routing destination to help with apps that have scaled to zero while the system gets the first instance available.
<extensionName>-k8se-build-service	Supports deployment operations and serves the Advanced tools feature .
<extensionName>-k8se-http-scaler	Monitors inbound request volume in order to provide scaling information to KEDA .
<extensionName>-k8se-img-cacher	Pulls placeholder and app images into a local cache on the node.
<extensionName>-k8se-log-processor	Gathers logs from apps and other components and sends them to Log Analytics.
placeholder-azure-functions-*	Used to speed up cold starts for Azure Functions.

App Service Kubernetes environment

The App Service Kubernetes environment resource is required before apps can be created. It enables configuration common to apps in the custom location, such as the default DNS suffix.

Only one Kubernetes environment resource can be created in a custom location. In most cases, a developer who creates and deploys apps doesn't need to be directly aware of the resource. It can be directly inferred from the provided custom location ID. However, when defining Azure Resource Manager templates, any plan resource needs to reference the resource ID of the environment directly. The custom location values of the plan and the specified environment must match.

FAQ for App Service, Functions, and Logic Apps on Azure Arc (Preview)

- [How much does it cost?](#)

- Are both Windows and Linux apps supported?
- Can the extension be installed on Windows nodes?
- Which built-in application stacks are supported?
- Are all app deployment types supported?
- Which App Service features are supported?
- Are all networking features supported?
- Are managed identities supported?
- Are there any scaling limits?
- What logs are collected?
- What do I do if I see a provider registration error?
- Can I deploy the Application services extension on an Arm64 based cluster?
- Which Kubernetes distributions can I deploy the extension on?

How much does it cost?

App Service on Azure Arc is free during the public preview.

Are both Windows and Linux apps supported?

Only Linux-based apps are supported, both code and custom containers. Windows apps aren't supported.

Can the extension be installed on Windows nodes?

No, the extension cannot be installed on Windows nodes. The extension supports installation on **Linux nodes only**.

Which built-in application stacks are supported?

All built-in Linux stacks are supported.

Are all app deployment types supported?

FTP deployment isn't supported. Currently `az webapp up` is also not supported. Other deployment methods are supported, including Git, ZIP, CI/CD, Visual Studio, and Visual Studio Code.

Which App Service features are supported?

During the preview period, certain App Service features are being validated. When they're supported, their left navigation options in the Azure portal will be activated. Features that aren't yet supported remain grayed out.

Are all networking features supported?

No. Networking features such as hybrid connections or Virtual Network integration, aren't supported. [Access restriction](#) support was added in April 2022. Networking should be handled directly in the networking rules in the Kubernetes cluster itself.

Are managed identities supported?

No. Apps cannot be assigned managed identities when running in Azure Arc. If your app needs an identity for working with another Azure resource, consider using an [application service principal](#) instead.

Are there any scaling limits?

All applications deployed with Azure App Service on Kubernetes with Azure Arc are able to scale within the limits of the underlying Kubernetes cluster. If the underlying Kubernetes Cluster runs out of available compute resources (CPU and memory primarily), then applications will only be able to scale to the number of instances of the application that Kubernetes can schedule with available resource.

What logs are collected?

Logs for both system components and your applications are written to standard output. Both log types can be collected for analysis using standard Kubernetes tools. You can also configure the App Service cluster extension with a [Log Analytics workspace](#), and it sends all logs to that workspace.

By default, logs from system components are sent to the Azure team. Application logs aren't sent. You can prevent these logs from being transferred by setting `logProcessor.enabled=false` as an extension configuration setting. This configuration setting will also disable forwarding of application to your Log Analytics workspace. Disabling the log processor might impact time needed for any support cases, and you will be asked to collect logs from standard output through some other means.

What do I do if I see a provider registration error?

When creating a Kubernetes environment resource, some subscriptions might see a "No registered resource provider found" error. The error details might include a set of locations and API versions that are considered valid. If this error message is returned, the subscription must be re-registered with the Microsoft.Web provider, an operation that has no impact on existing applications or APIs. To re-register, use the Azure CLI to run `az provider register --namespace Microsoft.Web --wait`. Then reattempt the Kubernetes environment command.

Can I deploy the Application services extension on an Arm64 based cluster?

Arm64 based clusters aren't supported at this time.

Which Kubernetes distributions can I deploy the extension on?

The extension has been validated on AKS, AKS on Azure Local, Google Kubernetes Engine, Amazon Elastic Kubernetes Service and Kubernetes Cluster API.

Extension Release Notes

Application services extension v 0.9.0 (May 2021)

- Initial public preview release of Application services extension.
- Support for code and container-based deployments of Web, Function, and Logic Applications.
- Web application runtime support --- .NET 3.1 and 5.0; Node JS 12 and 14; Python 3.6, 3.7, and 3.8; PHP 7.3 and 7.4; Ruby 2.5, 2.5.5, 2.6, and 2.6.2; Java SE 8u232, 8u242, 8u252, 11.05, 11.06 and 11.07; Tomcat 8.5, 8.5.41, 8.5.53, 8.5.57, 9.0, 9.0.20, 9.0.33, and 9.0.37.

Application services extension v 0.10.0 (November 2021)

- Removed requirement for pre-assigned Static IP Address required for assignment to the Envoy endpoint
- Upgrade Keda to v2.4.0
- Upgrade Envoy to v1.19.0
- Upgrade Azure Function runtime to v3.3.1

- Set default replica count of App Controller and Envoy Controller to 2 to add further stability

If your extension was in the stable version and auto-upgrade-minor-version is set to true, the extension upgrades automatically. To manually upgrade the extension to the latest version, you can run the command:

Azure CLI

```
az k8s-extension update --cluster-type connectedClusters -c <clustername> -g <resource group> -n <extension name> --release-train stable --version 0.10.0
```

Application services extension v 0.11.0 (December 2021)

- Added Application Insights support for Java and .NET Web Applications
- Added support for .NET 6.0 Web Applications
- Removed .NET Core 2.0
- Resolved issues that caused slot swap operations to fail
- Resolved issues customers experienced during creation of Ruby web applications

If your extension was in the stable version and auto-upgrade-minor-version is set to true, the extension upgrades automatically. To manually upgrade the extension to the latest version, you can run the command:

Azure CLI

```
az k8s-extension update --cluster-type connectedClusters -c <clustername> -g <resource group> -n <extension name> --release-train stable --version 0.11.0
```

Application services extension v 0.11.1 (December 2021)

- Minor release to resolve issue with CRD update

If your extension was in the stable version and auto-upgrade-minor-version is set to true, the extension upgrades automatically. To manually upgrade the extension to the latest version, you can run the command:

Azure CLI

```
az k8s-extension update --cluster-type connectedClusters -c <clustername> -g <resource group> -n <extension name> --release-train stable
```

```
--version 0.11.1
```

Application services extension v 0.12.0 (January 2022)

- Support for outbound proxy
- Support for parallel builds in build service
- Upgrade Envoy to 1.20.1
- Resolved issue with Application Insights support for .NET Applications

If your extension was in the stable version and auto-upgrade-minor-version is set to true, the extension upgrades automatically. To manually upgrade the extension to the latest version, you can run the command:

Azure CLI

```
az k8s-extension update --cluster-type connectedClusters -c
<clustername> -g <resource group> -n <extension name> --release-train stable
--version 0.12.0
```

Application services extension v 0.12.1 (March 2022)

- Resolved issue with outbound proxy support to enable logging to Log Analytics Workspace

If your extension was in the stable version and auto-upgrade-minor-version is set to true, the extension upgrades automatically. To manually upgrade the extension to the latest version, you can run the command:

Azure CLI

```
az k8s-extension update --cluster-type connectedClusters -c
<clustername> -g <resource group> -n <extension name> --release-train stable
--version 0.12.1
```

Application services extension v 0.12.2 (March 2022)

- Update to resolve upgrade failures when upgrading from v 0.12.0 when extension name length is over 35 characters

If your extension was in the stable version and auto-upgrade-minor-version is set to true, the extension upgrades automatically. To manually upgrade the extension to the latest version, you can run the command:

Azure CLI

```
az k8s-extension update --cluster-type connectedClusters -c <clustername> -g <resource group> -n <extension name> --release-train stable --version 0.12.2
```

Application services extension v 0.13.0 (April 2022)

- Added support for Application Insights codeless integration for Node JS applications
- Added support for [Access Restrictions](#) via CLI
- More details provided when extension fails to install, to assist with troubleshooting issues

If your extension was in the stable version and auto-upgrade-minor-version is set to true, the extension upgrades automatically. To manually upgrade the extension to the latest version, you can run the command:

Azure CLI

```
az k8s-extension update --cluster-type connectedClusters -c <clustername> -g <resource group> -n <extension name> --release-train stable --version 0.13.0
```

Application services extension v 0.13.1 (April 2022)

- Update to resolve upgrade failures seen during auto upgrade of clusters to v 0.13.0

If your extension was in the stable version and auto-upgrade-minor-version is set to true, the extension upgrades automatically. To manually upgrade the extension to the latest version, you can run the command:

Azure CLI

```
az k8s-extension update --cluster-type connectedClusters -c <clustername> -g <resource group> -n <extension name> --release-train stable --version 0.13.1
```

Application services extension v 0.13.5 (December 2023)

- Update to support Kubernetes version 1.26 and above
- Update Envoy to 1.2.1

- Update Keda to v2.10.0
- Update EasyAuth to v1.6.20
- Update base images for supported languages

If your extension was in the stable version and auto-upgrade-minor-version is set to true, the extension upgrades automatically. To manually upgrade the extension to the latest version, you can run the command:

Azure CLI

```
az k8s-extension update --cluster-type connectedClusters -c <clustername> -g <resource group> -n <extension name> --release-train stable --version 0.13.5
```

Next steps

[Create an App Service Kubernetes environment \(Preview\)](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

Set up an Azure Arc-enabled Kubernetes cluster to run App Service, Functions, and Logic Apps (Preview)

Article • 01/15/2025

If you have an [Azure Arc-enabled Kubernetes cluster](#), you can use it to create an [App Service enabled custom location](#) and deploy web apps, function apps, and logic apps to it.

Azure Arc-enabled Kubernetes lets you make your on-premises or cloud Kubernetes cluster visible to App Service, Functions, and Logic Apps in Azure. You can create an app and deploy to it just like another Azure region.

Prerequisites

If you don't have an Azure account, [sign up today](#) ↗ for a free account.

Review the [requirements and limitations](#) of the public preview. Of particular importance are the cluster requirements.

Add Azure CLI extensions

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

 [Launch Cloud Shell](#) ↗

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

```
Azure CLI

az extension add --upgrade --yes --name connectedk8s
az extension add --upgrade --yes --name k8s-extension
az extension add --upgrade --yes --name customlocation
az provider register --namespace Microsoft.ExtendedLocation --wait
az provider register --namespace Microsoft.Web --wait
az provider register --namespace Microsoft.KubernetesConfiguration --wait
az extension remove --name appservice-kube
az extension add --upgrade --yes --name appservice-kube
```

Create a connected cluster

ⓘ Note

This tutorial uses [Azure Kubernetes Service \(AKS\)](#) to provide concrete instructions for setting up an environment from scratch. However, for a production workload, you will likely not want to enable Azure Arc on an AKS cluster as it is already managed in Azure. The steps will help you get started understanding the service, but for production deployments, they should be viewed as illustrative, not prescriptive. See [Quickstart: Connect an existing Kubernetes cluster to Azure Arc](#) for general instructions on creating an Azure Arc-enabled Kubernetes cluster.

1. Create a cluster in Azure Kubernetes Service with a public IP address. Replace <group-name> with the resource group name you want.

bash

Azure CLI

```
AKS_CLUSTER_GROUP_NAME=<group-name> # Name of resource group for  
the AKS cluster  
AKS_NAME="${aksClusterGroupName}-aks" # Name of the AKS cluster  
RESOURCE_LOCATION="eastus" # "eastus" or "westeurope"  
  
az group create -g $AKS_CLUSTER_GROUP_NAME -l $RESOURCE_LOCATION  
az aks create --resource-group $AKS_CLUSTER_GROUP_NAME --name  
$AKS_NAME --enable-aad --generate-ssh-keys
```

2. Get the [kubeconfig](#) file and test your connection to the cluster. By default, the kubeconfig file is saved to `~/ .kube/config`.

Azure CLI

```
az aks get-credentials --resource-group $AKS_CLUSTER_GROUP_NAME --name  
$AKS_NAME --admin  
  
kubectl get ns
```

3. Create a resource group to contain your Azure Arc resources. Replace <group-name> with the resource group name you want.

bash

Azure CLI

```
GROUP_NAME=<group-name> # Name of resource group for the connected cluster
```

```
az group create -g $GROUP_NAME -l $RESOURCE_LOCATION
```

4. Connect the cluster you created to Azure Arc.

bash

Azure CLI

```
CLUSTER_NAME="${GROUP_NAME}-cluster" # Name of the connected cluster resource
```

```
az connectedk8s connect --resource-group $GROUP_NAME --name $CLUSTER_NAME
```

5. Validate the connection with the following command. It should show the `provisioningState` property as `Succeeded`. If not, run the command again after a minute.

Azure CLI

```
az connectedk8s show --resource-group $GROUP_NAME --name $CLUSTER_NAME
```

Create a Log Analytics workspace

While a [Log Analytic workspace](#) is not required to run App Service in Azure Arc, it's how developers can get application logs for their apps that are running in the Azure Arc-enabled Kubernetes cluster.

1. For simplicity, create the workspace now.

bash

Azure CLI

```
WORKSPACE_NAME="${GROUP_NAME}-workspace" # Name of the Log Analytics workspace
```

```
az monitor log-analytics workspace create \
--resource-group $GROUP_NAME \
--workspace-name $WORKSPACE_NAME
```

- Run the following commands to get the encoded workspace ID and shared key for an existing Log Analytics workspace. You need them in the next step.

bash

Azure CLI

```
LOG_ANALYTICS_WORKSPACE_ID=$(az monitor log-analytics workspace
show \
--resource-group $GROUP_NAME \
--workspace-name $WORKSPACE_NAME \
--query customerId \
--output tsv)
LOG_ANALYTICS_WORKSPACE_ID_ENC=$(printf %s
$LOG_ANALYTICS_WORKSPACE_ID | base64 -w0) # Needed for the next
step
LOG_ANALYTICS_KEY=$(az monitor log-analytics workspace get-shared-
keys \
--resource-group $GROUP_NAME \
--workspace-name $WORKSPACE_NAME \
--query primarySharedKey \
--output tsv)
LOG_ANALYTICS_KEY_ENC=$(printf %s $LOG_ANALYTICS_KEY | base64 -w0)
# Needed for the next step
```

Install the App Service extension

- Set the following environment variables for the desired name of the [App Service extension](#), the cluster namespace in which resources should be provisioned, and the name for the App Service Kubernetes environment. Choose a unique name for <kube-environment-name>, because it is part of the domain name for app created in the App Service Kubernetes environment.

bash

Bash

```
EXTENSION_NAME="appservice-ext" # Name of the App Service extension
NAMESPACE="appservice-ns" # Namespace in your cluster to install
the extension and provision resources
```

```
KUBE_ENVIRONMENT_NAME="<kube-environment-name>" # Name of the App  
Service Kubernetes environment resource
```

2. Install the App Service extension to your Azure Arc-connected cluster, with Log Analytics enabled. Again, while Log Analytics is not required, you can't add it to the extension later, so it's easier to do it now.

bash

Azure CLI

```
az k8s-extension create \  
    --resource-group $GROUP_NAME \  
    --name $EXTENSION_NAME \  
    --cluster-type connectedClusters \  
    --cluster-name $CLUSTER_NAME \  
    --extension-type 'Microsoft.Web.Appservice' \  
    --release-train stable \  
    --auto-upgrade-minor-version true \  
    --scope cluster \  
    --release-namespace $NAMESPACE \  
    --configuration-settings  
"Microsoft.CustomLocation.ServiceAccount=default" \  
    --configuration-settings "appsNamespace=${NAMESPACE}" \  
    --configuration-settings "clusterName=${KUBE_ENVIRONMENT_NAME}" \  
    \  
    --configuration-settings "keda.enabled=true" \  
    --configuration-settings  
"buildService.storageClassName=default" \  
    --configuration-settings  
"buildService.storageAccessMode=ReadWriteOnce" \  
    --configuration-settings "customConfigMap=${NAMESPACE}/kube-  
environment-config" \  
    --configuration-settings  
"envoy.annotations.service.beta.kubernetes.io/azure-load-balancer-  
resource-group=${aksClusterGroupName}" \  
    --configuration-settings "logProcessor.appLogs.destination=log-  
analytics" \  
    --config-protected-settings  
"logProcessor.appLogs.logAnalyticsConfig.customerId=${LOG_ANALYTICS_  
_WORKSPACE_ID_ENC}" \  
    --config-protected-settings  
"logProcessor.appLogs.logAnalyticsConfig.sharedKey=${LOG_ANALYTICS_  
KEY_ENC}"
```

ⓘ Note

To install the extension without Log Analytics integration, remove the last three `--configuration-settings` parameters from the command.

The following table describes the various `--configuration-settings` parameters when running the command:

[\[+\] Expand table](#)

Parameter	Description
<code>Microsoft.CustomLocation.ServiceAccount</code>	The service account that should be created for the custom location that is created. It is recommended that this be set to the value <code>default</code> .
<code>appsNamespace</code>	The namespace to provision the app definitions and pods. Must match that of the extension release namespace.
<code>clusterName</code>	The name of the App Service Kubernetes environment that is created against this extension.
<code>keda.enabled</code>	Whether KEDA should be installed on the Kubernetes cluster. Accepts <code>true</code> or <code>false</code> .
<code>buildService.storageClassName</code>	The name of the storage class for the build service to store build artifacts. A value like <code>default</code> specifies a class named <code>default</code> , and not any class that is marked as default . Default is a valid storage class for AKS and AKS on Azure Local but it may not be for other distributions/platforms.
<code>buildService.storageAccessMode</code>	The access mode to use with the named storage class. Accepts <code>ReadWriteOnce</code> or <code>ReadWriteMany</code> .
<code>customConfigMap</code>	The name of the config map that will be set by the App Service Kubernetes environment. Currently, it must be <code><namespace>/kube-environment-</code>

Parameter	Description
<code>envoy.annotations.service.beta.kubernetes.io/azure-load-balancer-resource-group</code>	config, replacing <code><namespace></code> with the value of <code>appsNamespace</code> .
<code>logProcessor.appLogs.destination</code>	The name of the resource group in which the Azure Kubernetes Service cluster resides. Valid and required only when the underlying cluster is Azure Kubernetes Service.
<code>logProcessor.appLogs.logAnalyticsConfig.customerId</code>	Optional. Accepts <code>log-analytics</code> or <code>none</code> , choosing none disables platform logs.
<code>logProcessor.appLogs.logAnalyticsConfig.sharedKey</code>	Required only when <code>logProcessor.appLogs.destination</code> is set to <code>log-analytics</code> . The base64-encoded Log analytics workspace ID. This parameter should be configured as a protected setting.
	Required only when <code>logProcessor.appLogs.destination</code> is set to <code>log-analytics</code> . The base64-encoded Log analytics workspace shared key. This parameter should be configured as a protected setting.

3. Save the `id` property of the App Service extension for later.

bash

Azure CLI

```
EXTENSION_ID=$(az k8s-extension show \
--cluster-type connectedClusters \
--cluster-name $CLUSTER_NAME \
--resource-group $GROUP_NAME \
--name $EXTENSION_NAME \
--query id \
--output tsv)
```

4. Wait for the extension to fully install before proceeding. You can have your terminal session wait until this complete by running the following command:

```
Azure CLI
```

```
az resource wait --ids $EXTENSION_ID --custom  
"properties.installState!='Pending'" --api-version "2020-07-01-preview"
```

You can use `kubectl` to see the pods created in your Kubernetes cluster:

```
Bash
```

```
kubectl get pods -n $NAMESPACE
```

You can learn more about these pods and their role in the system from [Pods created by the App Service extension](#).

Create a custom location

The [custom location](#) in Azure is used to assign the App Service Kubernetes environment.

1. Set the following environment variables for the desired name of the custom location and for the ID of the Azure Arc-connected cluster.

```
bash
```

```
Bash
```

```
CUSTOM_LOCATION_NAME="my-custom-location" # Name of the custom  
location  
  
CONNECTED_CLUSTER_ID=$(az connectedk8s show --resource-group  
$GROUP_NAME --name $CLUSTER_NAME --query id --output tsv)
```

2. Create the custom location:

```
bash
```

```
Azure CLI
```

```
az customlocation create \  
--resource-group $GROUP_NAME \  
--name $CUSTOM_LOCATION_NAME \  
--connected-cluster-id $CONNECTED_CLUSTER_ID
```

```
--host-resource-id $CONNECTED_CLUSTER_ID \
--namespace $NAMESPACE \
--cluster-extension-ids $EXTENSION_ID
```

⚠ Note

If you experience issues creating a custom location on your cluster, you may need to [enable the custom location feature on your cluster](#). This is required if logged into the CLI using a Service Principal or if you are logged in with a Microsoft Entra user with restricted permissions on the cluster resource.

3. Validate that the custom location is successfully created with the following command. The output should show the `provisioningState` property as `Succeeded`. If not, run it again after a minute.

Azure CLI

```
az customlocation show --resource-group $GROUP_NAME --name
$CUSTOM_LOCATION_NAME
```

4. Save the custom location ID for the next step.

bash

Azure CLI

```
CUSTOM_LOCATION_ID=$(az customlocation show \
--resource-group $GROUP_NAME \
--name $CUSTOM_LOCATION_NAME \
--query id \
--output tsv)
```

Create the App Service Kubernetes environment

Before you can start creating apps on the custom location, you need an [App Service Kubernetes environment](#).

1. Create the App Service Kubernetes environment:

bash

Azure CLI

```
az appservice kube create \
--resource-group $GROUP_NAME \
--name $KUBE_ENVIRONMENT_NAME \
--custom-location $CUSTOM_LOCATION_ID
```

2. Validate that the App Service Kubernetes environment is successfully created with the following command. The output should show the `provisioningState` property as `Succeeded`. If not, run it again after a minute.

Azure CLI

```
az appservice kube show --resource-group $GROUP_NAME --name
$KUBE_ENVIRONMENT_NAME
```

Next steps

- Quickstart: Create a web app on Azure Arc
- Create your first function on Azure Arc
- Create your first logic app on Azure Arc

Feedback

Was this page helpful?

 Yes

 No

Provide product feedback  | Get help at Microsoft Q&A

Create an App Service app on Azure Arc (Preview)

Article • 06/10/2024

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

Prerequisites

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

Add Azure CLI extensions

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

Azure CLI

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

1. Create a resource group

Run the following command.

Azure CLI

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

2. Get the custom location

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

Azure CLI

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

Get the custom location ID for the next step.

Azure CLI

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

3. Create an app

The following example creates a Node.js app. Replace `<app-name>` with a name that's unique within your cluster (valid characters are `a-z`, `0-9`, and `-`).

Supported runtimes:

[] [Expand table](#)

Description	Runtime Value for CLI
.NET Core 3.1	DOTNETCORE 3.1
.NET 5.0	DOTNETCORE 6.0
Node JS 12	NODE 12-lts
Node JS 14	NODE 14-lts
Python 3.6	PYTHON 3.6
Python 3.7	PYTHON 3.7
Python 3.8	PYTHON 3.8
PHP 7.3	PHP 7.3
PHP 7.4	PHP 7.4

Description	Runtime Value for CLI
Java 8	JAVA 8-jre8
Java 11	JAVA 11-java11
Tomcat 8.5	TOMCAT 8.5-jre8
Tomcat 8.5	TOMCAT 8.5-java11
Tomcat 9.0	TOMCAT 9.0-jre8
Tomcat 9.0	TOMCAT 9.0-java11

Azure CLI

```
az webapp create \
--resource-group myResourceGroup \
--name <app-name> \
--custom-location $customLocationId \
--runtime 'NODE|14-lts'
```

4. Deploy some code

ⓘ Note

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

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

Azure CLI

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

5. Get diagnostic logs using Log Analytics

ⓘ Note

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

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

```
Kusto

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

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

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

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

(Optional) Deploy a custom container

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

For example, try:

```
Azure CLI

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

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

Next steps

- Configure an ASP.NET Core app
 - Configure a Node.js app
 - Configure a PHP app
 - Configure a Linux Python app
 - Configure a Java app
 - Configure a custom container
-

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#)

Best practices for Azure App Service

Article • 01/08/2024

This article summarizes best practices for using [Azure App Service](#).

Colocation

An Azure App Service solution consists of a web app and a database or storage account for holding content or data. When these resources are in different regions, the situation can have the following effects:

- Increased latency in communication between resources
- Monetary charges for outbound data transfer across regions, as noted on the [Azure pricing page](#) ↗

Colocation is best for Azure resources that compose a solution. When you create resources, make sure they're in the same Azure region unless you have specific business or design reasons for them not to be. You can move an App Service app to the same region as your database by using the [App Service cloning feature](#) available in Premium App Service plans.

Certificate pinning

Certificate pinning is a practice in which an application allows only a specific list of acceptable certificate authorities (CAs), public keys, thumbprints, or any part of the certificate hierarchy.

Applications should never have a hard dependency or pin to the default wildcard (*.azurewebsites.net) TLS certificate. App Service is a platform as a service (PaaS), so this certificate could be rotated anytime. If the service rotates the default wildcard TLS certificate, certificate-pinned applications will break and disrupt the connectivity for applications that are hardcoded to a specific set of certificate attributes. The periodicity with which the certificate is rotated is also not guaranteed because the rotation frequency can change at any time.

Applications that rely on certificate pinning also shouldn't have a hard dependency on an App Service managed certificate. App Service managed certificates could be rotated anytime, leading to similar problems for applications that rely on stable certificate properties. It's a best practice to provide a custom TLS certificate for applications that rely on certificate pinning.

If your application needs to rely on certificate pinning behavior, we recommend that you add a custom domain to a web app and provide a custom TLS certificate for the domain. The application can then rely on the custom TLS certificate for certificate pinning.

Memory resources

When monitoring or service recommendations indicate that an app consumes more memory than you expected, consider the [App Service auto-healing feature](#). You can configure auto-healing by using `web.config`.

One of the options for the auto-healing feature is taking custom actions based on a memory threshold. Actions range from email notifications to investigation via memory dump to on-the-spot mitigation by recycling the worker process.

CPU resources

When monitoring or service recommendations indicate that an app consumes more CPU than you expected or it experiences repeated CPU spikes, consider scaling up or scaling out the App Service plan. If your application is stateful, scaling up is the only option. If your application is stateless, scaling out gives you more flexibility and higher scale potential.

For more information about App Service scaling and autoscaling options, see [Scale up an app in Azure App Service](#).

Socket resources

A common reason for exhausting outbound TCP connections is the use of client libraries that don't reuse TCP connections or that don't use a higher-level protocol such as HTTP keep-alive.

Review the documentation for each library that the apps in your App Service plan reference. Ensure that the libraries are configured or accessed in your code for efficient reuse of outbound connections. Also follow the library documentation guidance for proper creation and release or cleanup to avoid leaking connections. While such investigations into client libraries are in progress, you can mitigate impact by scaling out to multiple instances.

Node.js and outgoing HTTP requests

When you're working with Node.js and many outgoing HTTP requests, dealing with HTTP keep-alive is important. You can use the [agentkeepalive](#) npm package to make it easier in your code.

Always handle the `http` response, even if you do nothing in the handler. If you don't handle the response properly, your application eventually gets stuck because no more sockets are available.

Here's an example of handling the response when you're working with the `http` or `https` package:

```
JavaScript
```

```
const request = https.request(options, function(response) {
  response.on('data', function() { /* do nothing */ });
});
```

If you're running your App Service app on a Linux machine that has multiple cores, another best practice is to use PM2 to start multiple Node.js processes to run your application. You can do it by specifying a startup command to your container.

For example, use this command to start four instances:

```
pm2 start /home/site/wwwroot/app.js --no-daemon -i 4
```

App backup

Backups typically run on a schedule and require access to storage (for outputting the backed-up files) and databases (for copying and reading contents to be included in the backup). The result of failing to access either of these resources is consistent backup failure.

The two most common reasons why app backup fails are invalid storage settings and invalid database configuration. These failures typically happen after changes to storage or database resources, or after changes to credentials for accessing those resources. For example, credentials might be updated for the database that you selected in the backup settings.

When backup failures happen, review the most recent results to understand which type of failure is happening. For storage access failures, review and update the storage settings in your backup configuration. For database access failures, review and update

your connection strings as part of app settings. Then proceed to update your backup configuration to properly include the required databases.

For more information on app backups, see [Back up and restore your app in Azure App Service](#).

Node.js apps

The Azure App Service default configuration for Node.js apps is intended to best suit the needs of most common apps. If you want to personalize the default configuration for your Node.js app to improve performance or optimize resource usage for CPU, memory, or network resources, see [Best practices and troubleshooting guide for Node applications on Azure App Service](#). That article describes the `iisnode` settings that you might need to configure for your Node.js app. It also explains how to address scenarios or problems with your app.

IoT devices

You can improve your environment when you're running Internet of Things (IoT) devices that are connected to App Service.

One common practice with IoT devices is certificate pinning. To avoid any unforeseen downtime due to changes in the service's managed certificates, you should never pin certificates to the default `*.azurewebsites.net` certificate or to an App Service managed certificate. If your system needs to rely on certificate pinning behavior, we recommend that you add a custom domain to a web app and provide a custom TLS certificate for the domain. The application can then rely on the custom TLS certificate for certificate pinning. For more information, see the [certificate pinning](#) section of this article.

To increase resiliency in your environment, don't rely on a single endpoint for all your devices. Host your web apps in at least two regions to avoid a single point of failure, and be ready to fail over traffic.

In App Service, you can add identical custom domains to multiple web apps, as long as these web apps are hosted in different regions. This capability ensures that if you need to pin certificates, you can also pin on the custom TLS certificate that you provided.

Another option is to use a load balancer in front of the web apps, such as Azure Front Door or Azure Traffic Manager, to ensure high availability for your web apps. For more information, see [Quickstart: Create a Front Door instance for a highly available global web application](#) or [Controlling Azure App Service traffic with Azure Traffic Manager](#).

Next steps

To get actionable best practices that are specific to your resource, use [App Service diagnostics](#):

1. Go to your web app in the [Azure portal](#).
2. Open App Service diagnostics by selecting **Diagnose and solve problems** on the left pane.
3. Select the **Best Practices** tile.
4. Select **Best Practices for Availability & Performance** or **Best Practices for Optimal Configuration** to view the current state of your app in regard to these best practices.

You can also use this link to directly open App Service diagnostics for your resource:

`https://portal.azure.com/?websitesextension_ext=asd.featurePath%3Ddetectors%2FParentAvailabilityAndPerformance@#microsoft.onmicrosoft.com/resource/subscriptions/{subscriptionId}/resourceGroups/{resourceGroupName}/providers/Microsoft.Web/sites/{siteName}/troubleshoot`

```
websitesextension_ext=asd.featurePath%3Ddetectors%2FParentAvailabilityAndPerformance@#microsoft.onmicrosoft.com/resource/subscriptions/{subscriptionId}/resourceGroups/{resourceGroupName}/providers/Microsoft.Web/sites/{siteName}/troubleshoot.
```

At-scale assessment of .NET web apps

Article • 06/29/2023

Once you've discovered ASP.NET web apps you should proceed to the next step of assessing these web apps. Assessment provides you with migration readiness and sizing recommendations based on properties defined by you. Below is the list of key assessment capabilities:

- Modify assessment properties as per your requirements like target Azure region, application isolation requirements, and reserved instance pricing.
- Provide App Service SKU recommendation and display monthly cost estimates
- Provide per web app migration readiness information and provide detailed information on blockers and errors.

You can create multiple assessments for the same web apps with different sets of assessment properties

For more information on web apps assessment, see:

- [At scale discovery and assessment for ASP.NET app migration with Azure Migrate ↗](#)
- [Create an Azure App Service assessment](#)
- [Tutorial to assess web apps for migration to Azure App Service](#)
- [Azure App Service assessments in Azure Migrate Discovery and assessment tool](#)
- [Assessment best practices in Azure Migrate Discovery and assessment tool](#)

Next steps: [At-scale migration of .NET web apps](#)

At-scale discovery of .NET web apps

Article • 09/20/2022

For ASP. Net web apps discovery you need to either install a new Azure Migrate appliance or upgrade an existing Azure Migrate appliance.

Once the appliance is configured, Azure Migrate initiates the discovery of web apps deployed on IIS web servers hosted within your on-premises VMware environment. Discovery of ASP.NET web apps provide the following key capabilities:

- Agentless discovery of up to 20,000 web apps with a single Azure Migrate appliance
- Provide a rich & interactive dashboard with a list of IIS web servers and underlying VM infra details. Web apps discovery surfaces information such as:
 - web app name
 - web server type and version
 - URLs
 - binding port
 - application pool
- If web app discovery has failed then the discovery dashboard allows easy navigation to review relevant error messages, possible causes of failure and suggested remediation actions

For more information about web apps discovery please refer to:

- [At scale discovery and assessment for ASP.NET app migration with Azure Migrate](#)
- [Discover and assess ASP.NET apps at-scale with Azure Migrate](#)
- [At scale discovery and assessment for ASP.NET app migration with Azure Migrate](#)
- [Discover software inventory on on-premises servers with Azure Migrate](#)
- [Discover web apps and SQL Server instances](#)

Next steps: [At-scale assessment of .NET web apps](#)

.NET migration cases for Azure App Service

Article • 01/23/2024

Azure App Service provides easy-to-use tools to quickly discover on-premises .NET web apps, assess for readiness, and migrate both the content & supported configurations to App Service.

These tools are developed to support different kinds of scenarios, focused on discovery, assessment, and migration. Following is list of .NET migration tools and use cases.

Migrate from multiple servers at-scale

Note

Learn how to migrate .NET apps to App Service using the [.NET migration tutorial](#).

Azure Migrate recently announced at-scale, agentless discovery, and assessment of ASP.NET web apps. You can now easily discover ASP.NET web apps running on Internet Information Services (IIS) servers in a VMware environment and assess them for migration to Azure App Service. Assessments will help you determine the web app migration readiness, migration blockers, remediation guidance, recommended SKU, and hosting costs. At-scale migration resources for are found below.

Once you have successfully assessed readiness, you should proceed with migration of ASP.NET web apps to Azure App Services.

There are existing tools which enable migration of a standalone ASP.NET web app or multiple ASP.NET web apps hosted on a single IIS server as explained in [Migrate .NET apps to Azure App Service](#). With introduction of At-Scale or bulk migration feature integrated with Azure Migrate we are now opening up the possibilities to migrate multiple ASP.NET application hosted on multiple on-premises IIS servers.

Azure Migrate provides at-scale, agentless discovery, and assessment of ASP.NET web apps. You can discover ASP.NET web apps running on Internet Information Services (IIS) servers in a VMware environment and assess them for migration to Azure App Service. Assessments will help you determine the web app migration readiness, migration blockers, remediation guidance, recommended SKU, and hosting costs. At-scale migration resources for are found below.

Bulk migration provides the following key capabilities:

- Bulk Migration of ASP.NET web apps to Azure App Services multitenant or App services environment
- Migrate ASP.NET web apps assessed as "Ready" & "Ready with Conditions"
- Migrate up to five App Service Plans (and associated web apps) as part of a single E2E migration flow
- Ability to change suggested SKU for the target App Service Plan (Ex: Change suggested Pv3 SKU to Standard PV2 SKU)
- Ability to change web apps suggested web apps packing density for target app service plan (Add or Remove web apps associated with an App Service Plan)
- Change target name for app service plans and\or web apps
- Bulk edit migration settings\attributes
- Download CSV with details of target web app and app service plan name
- Track progress of migration using ARM template deployment experience

Move .NET apps to Azure App Service

Azure App Service is a cloud platform that offers a fast, easy, and cost-effective way to migrate your .NET web apps from on-premises to the cloud. Start learning today about how Azure empowers you to modernize your .NET apps with the following resources.

Select one of the following options to get started with a migration assessment:

- [Self-service assessment ↗](#)
- [Partner assessment ↗](#)

App Service migration tools and resources

App Service Migration Assistant tool and App Service migration assistant for PowerShell scripts are governed by the terms and conditions in the EULA.pdf packaged with the respective tools.

[+] Expand table

Migration Tools	Description	Documentation
App Service Migration Assistant ↗	Migrate .NET web apps from Windows OS to App Service.	App Service Migration Assistant Documentation ↗
App Service migration assistant for Java on	Download prerelease software for migrating Java web applications on Tomcat web server running on Windows servers.	App Service Migration Assistant Documentation ↗

Migration Tools	Description	Documentation
Apache Tomcat (Windows—preview) ↗		
App Service Migration Assistant PowerShell scripts ↗	Download PowerShell scripts for discovering and assessing all Microsoft Internet Information Services (IIS) web apps on a single server in bulk and migrating .NET web apps from Windows OS to App Service.	App Service Migration Assistant Powershell Documentation ↗ SHA256 Identifier ↗

[\[+\] Expand table](#)

More resources to migrate .NET apps to the cloud
Video
.NET on Azure for Beginners ↗
Start Your Cloud Journey with Azure App Service ↗
Blog
Reliable web app pattern for .NET ↗
Start your cloud journey with Azure App Service ↗
Start your cloud journey with Azure App Service - Move your code ↗
Learn how to modernize your .NET apps from the pros ↗
On-demand event
Azure Developers - .NET Day
Learning path
Migrate ASP.NET Apps to Azure
Host a web application with Azure App Service
Publish a web app to Azure with Visual Studio

At-scale migration resources

[\[+\] Expand table](#)

How-tos
Discover web apps and SQL Server instances

How-tos
Create an Azure App Service assessment
Tutorial to assess web apps for migration to Azure App Service
Discover software inventory on on-premises servers with Azure Migrate
Migrate .NET apps to App Service
Blog
Discover and assess ASP.NET apps at-scale with Azure Migrate
FAQ
Azure App Service assessments in Azure Migrate Discovery and assessment tool
Best practices
Assessment best practices in Azure Migrate Discovery and assessment tool
Video
At scale discovery and assessment for ASP.NET app migration with Azure Migrate

Migrate from an IIS server

You can migrate ASP.NET web apps from single IIS server discovered through Azure Migrate's at-scale discovery experience using [PowerShell scripts](#) (download). Watch the video for [updates on migrating to Azure App Service](#).

ASP.NET web app migration

Using App Service Migration Assistant, you can [migrate your standalone on-premises ASP.NET web app onto Azure App Service](#). App Service Migration Assistant is designed to simplify your journey to the cloud through a free, simple, and fast solution to migrate applications from on-premises to the cloud. For more information about the migration assistant tool, see the [FAQ](#).

Containerize an ASP.NET web app

Some .NET Framework web applications may have dependencies to libraries and other capabilities not available in Azure App Service. These apps may rely on other components in the Global Assembly Cache. Previously, you could only run these

applications on virtual machines. However, now you can run them in Azure App Service Windows Containers.

The [app containerization tool](#) can repackage applications as containers with minimal changes. The tool currently supports containerizing ASP.NET applications and Apache Tomcat Java applications. For more information about containerization and migration, see the [how-to](#).

Next steps

[Migrate an on-premises web application to Azure App Service](#)

ASP.NET app containerization and migration to Azure App Service

Article • 09/19/2024

In this article, you'll learn how to containerize ASP.NET applications and migrate them to [Azure App Service](#) by using the Azure Migrate App Containerization tool. The containerization process doesn't require access to your codebase and provides an easy way to containerize existing applications. The tool works by using the running state of the applications on a server to determine the application components. It then helps you package them in a container image. You can then deploy the containerized application on Azure App Service.

The Azure Migrate App Containerization tool currently supports:

- Containerizing ASP.NET apps and deploying them on Windows containers on App Service.
- Containerizing ASP.NET apps and deploying them on Windows containers on Azure Kubernetes Service (AKS). [Learn more about this containerization scenario.](#)
- Containerizing Java web apps on Apache Tomcat (on Linux servers) and deploying them on Linux containers on AKS. [Learn more about this containerization scenario.](#)
- Containerizing Java web apps on Apache Tomcat (on Linux servers) and deploying them on Linux containers on App Service. [Learn more about this containerization scenario.](#)

The App Containerization tool helps you:

- **Discover your application components.** The tool remotely connects to the application servers that run your ASP.NET application and discovers the application components. It creates a Dockerfile that you can use to create a container image for the application.
- **Build the container image.** You can inspect and further customize the Dockerfile based on your application requirements and use it to build your application container image. The application container image is pushed to an Azure container registry that you specify.
- **Deploy to Azure App Service.** The tool then generates the deployment files needed to deploy the containerized application to Azure App Service.

Note

The Azure Migrate App Containerization tool helps you discover specific application types (ASP.NET and Java web apps on Apache Tomcat) and their components on an application server. To discover servers and the inventory of apps, roles, and features running on on-premises computers, use the [Azure Migrate Discovery and assessment tool](#).

Not all applications will benefit from a straight shift to containers without significant rearchitecting. But some of the benefits of moving existing apps to containers without rewriting include:

- **Improved infrastructure utilization.** When you use containers, multiple applications can share resources and be hosted on the same infrastructure. This can help you consolidate infrastructure and improve utilization.
- **Simplified management.** By hosting your applications on modern managed platforms like AKS and App Service, you can simplify your management practices. You can achieve this simplification by retiring or reducing the infrastructure maintenance and management processes that you'd traditionally perform with owned infrastructure.
- **Application portability.** With increased adoption and standardization of container specification formats and platforms, application portability is no longer a concern.
- **Adopt modern management by using DevOps.** Using containers helps you adopt and standardize on modern practices for management and security and transition to DevOps.

In this tutorial, you'll learn how to:

- ✓ Set up an Azure account.
- ✓ Install the Azure Migrate App Containerization tool.
- ✓ Discover your ASP.NET application.
- ✓ Build the container image.
- ✓ Deploy the containerized application on App Service.

Note

Tutorials provide the simplest deployment path for a scenario so that you can quickly set up a proof of concept. Tutorials use default options when possible and don't show all settings and paths.

Prerequisites

Before you start this tutorial, you should:

[+] Expand table

Requirement	Details
Identify a machine on which to install the tool	<p>You need a Windows machine on which to install and run the Azure Migrate App Containerization tool. The Windows machine could run a server (Windows Server 2016 or later) or client (Windows 10) operating system. (The tool can run on your desktop.)</p> <p>The Windows machine running the tool should have network connectivity to the servers or virtual machines hosting the ASP.NET applications that you'll containerize.</p> <p>Ensure that 6 GB is available on the Windows machine running the Azure Migrate App Containerization tool. This space is for storing application artifacts.</p> <p>The Windows machine should have internet access, directly or via a proxy.</p> <p>If the Microsoft Web Deployment tool isn't already installed on the machine running the App Containerization tool and the application server, install it. You can download the tool.</p>
Application servers	<p>Enable PowerShell remoting on the application servers: sign in to the application server and follow these instructions to turn on PowerShell remoting.</p> <p>Ensure that PowerShell 5.1 is installed on the application server. Follow the instructions in Install and Configure WMF 5.1 on the application server.</p> <p>If the Microsoft Web Deployment tool isn't already installed on the machine running the App Containerization tool and the application server, install it. You can download the tool.</p>
ASP.NET application	<p>The tool currently supports:</p> <ul style="list-style-type: none">ASP.NET applications that use .NET Framework 3.5 or later.Application servers that run Windows Server 2012 R2 or later. (Application servers should be running PowerShell 5.1.)Applications that run on Internet Information Services 7.5 or later. <p>The tool currently doesn't support:</p> <ul style="list-style-type: none">Applications that require Windows authentication. (AKS doesn't currently support gMSA.)

Requirement	Details
	<ul style="list-style-type: none"> Applications that depend on other Windows services hosted outside of Internet Information Services.

Prepare an Azure user account

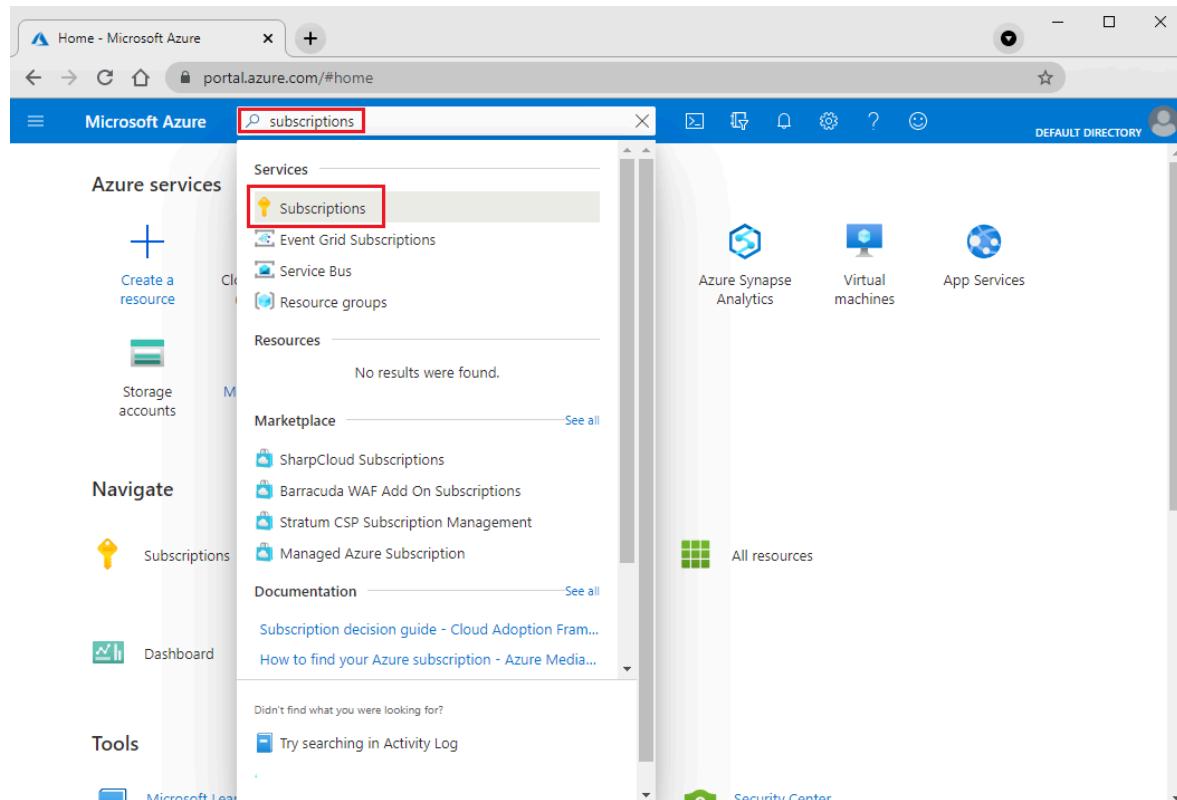
If you don't have an Azure subscription, create a [free account](#) before you start.

After your subscription is set up, you'll need an Azure user account with:

- Owner permissions on the Azure subscription.
- Permissions to register Microsoft Entra apps.

If you just created a free Azure account, you're the owner of your subscription. If you're not the subscription owner, work with the owner to assign the permissions as follows:

- In the Azure portal, search for "subscriptions." Under **Services**, select **Subscriptions**:



- On the **Subscriptions** page, select the subscription in which you want to create an Azure Migrate project.
- In the subscription, on the left pane, select **Access control (IAM)**.
- On the **Check access** tab, search for the relevant user account.

5. Under Add a role assignment, select Add:

Azure Migrate Demo Subscription | Access control (IAM)

Subscription

Search (Ctrl+ /)

Add Download role assignments Edit columns Refresh Remove ...

Overview Activity log Access control (IAM) Tags Diagnose and solve problems Security Events Cost Management

Check access Role assignments Roles Deny assignments Classic administrators

Check access Review the level of access a user, group, service principal, or managed identity has to this resource. Learn more

Find Search by name or email address

Add a role assignment Grant access to resources at this scope by assigning role to a user, group, service principal, or managed identity.

Add Learn more

6. On the Add role assignment page, select the Owner role, and then select the account (azmigrateuser in our example). Then select Save.

Add role assignment

Role ⓘ

Contributor ⓘ

Assign access to ⓘ

Select ⓘ

azmigrateuser

AZ azmigrateuser (Guest)
azmigrateuser@contoso.com

Your Azure account also needs permissions to register Microsoft Entra apps.

7. In the Azure portal, go to **Microsoft Entra ID > Users > User Settings**.
8. In **User settings**, verify that Microsoft Entra users can register applications. (This option is set to **Yes** by default.)

 **Users | User settings** ...

«  Save  Discard

 All users (Preview)	Enterprise applications Manage how end users launch and view their applications
 Deleted users (Preview)	
 Password reset	
 User settings	App registrations Users can register applications 
 Diagnose and solve problems	<input checked="" type="button"/> Yes <input type="button"/> No
<hr/>	
Activity	Administration portal
 Sign-ins	<input type="button"/> Yes <input checked="" type="button"/> No
 Audit logs	
 Bulk operation results	

 **Important**

Microsoft recommends that you use roles with the fewest permissions. This helps improve security for your organization. Global Administrator is a highly privileged role that should be limited to emergency scenarios when you can't use an existing role.

9. If the **App registrations** option is set to **No**, ask the tenant/global admin to assign the required permission. Alternatively, the tenant/global admin can assign the Application developer role to an account to allow the registration of Microsoft Entra apps. For more information, see [Assign roles to users](#).

Download and install the Azure Migrate App Containerization tool

1. [Download the Azure Migrate App Containerization installer](#) on a Windows machine.
2. Open PowerShell in administrator mode and change the PowerShell directory to the folder that contains the installer.
3. Run the installation script by using this command:

```
PowerShell
```

```
.\AppContainerizationInstaller.ps1
```

ⓘ Note

For Windows Server 2022, edit line 135 and remove `PowerShell-ISE` from the feature list, as it's no longer supported.

Open the App Containerization tool

1. Open a browser on any machine that can connect to the Windows machine that's running the App Containerization tool. Go to the tool URL: **<https://machine name or IP address: 44369>**.

Alternatively, you can open the app from your desktop by using the app shortcut.

2. If you see a warning that says your connection isn't private, select **Advanced** and continue to the website. This warning appears because the web interface uses a self-signed TLS/SSL certificate.
3. On the sign-in screen, use the machine's local administrator account to sign in.
4. Select **ASP.NET web apps** as the type of application you want to containerize.
5. In the **Target Azure service** list, select **Containers on Azure App Service**:

Azure Migrate: App Containerization helper (Preview)

The screenshot shows a user interface for the 'Azure Migrate: App Containerization helper (Preview)'. At the top, there is a header with the title. Below it is a section titled 'Application type and target' with a sub-instruction 'Select the type of application to containerize and the target Azure service'. Underneath this, there are two dropdown menus: 'Application type' and 'Target Azure service', both currently set to 'Select'. In the bottom right corner of the form area, there is a 'Continue' button.

Complete the tool prerequisites

1. Accept the license terms and read the third-party information.
2. In the tool web app **Set up prerequisites**, complete these steps:

- **Connectivity.** The tool checks whether the Windows machine has internet access. If the machine uses a proxy:
 - a. Select **Set up proxy** to specify the proxy address (in the form IP address or FQDN) and listening port.
 - b. Specify credentials if the proxy needs authentication.
 - c. If you've added proxy details or disabled the proxy or authentication, select **Save** to trigger the connectivity check again.

Only HTTP proxy is supported.

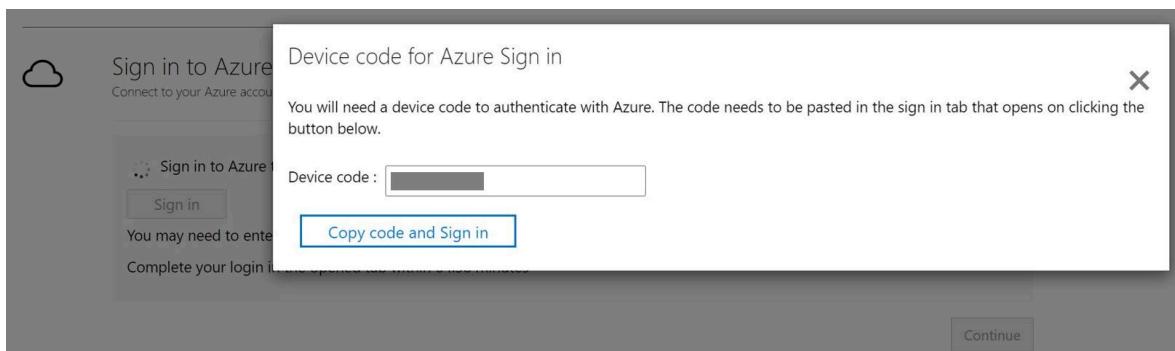
- **Install updates.** The tool automatically checks for the latest updates and installs them. You can also [manually install the latest version of the tool ↗](#).
- **Install Microsoft Web Deploy tool.** The tool checks whether the Microsoft Web Deployment tool is installed on the Windows machine that's running the Azure Migrate App Containerization tool.
- **Enable PowerShell remoting.** The tool prompts you to ensure that PowerShell remoting is enabled on the application servers running the ASP.NET applications that you want to containerize.

Sign in to Azure

1. Select **Sign in** to sign in to your Azure account.

You need a device code to authenticate with Azure. Selecting **Sign in** should open a window that contains the device code. If the window doesn't appear, make sure you've disabled the pop-up blocker in the browser.

2. Select **Copy code and Sign in** to copy the device code and open an Azure sign-in prompt in a new browser tab:



3. On the new tab, paste in the device code and complete sign-in by using your Azure account credentials. After you're signed in, you can close the browser tab and return to the App Containerization tool's web interface.
4. Select the **Azure tenant** that you want to use.
5. Specify the **Azure subscription** that you want to use.

Discover ASP.NET applications

The App Containerization tool connects remotely to the application servers by using the provided credentials and attempts to discover ASP.NET applications hosted on the application servers.

1. Specify the **Server IP address / FQDN** and the credentials of the server that's running the ASP.NET application that should be used to remotely connect to the server for application discovery.
 - The credentials provided must be for a local administrator (Windows) on the application server.
 - For domain accounts (the user must be an administrator on the application server), prefix the user name with the domain name in this format: `<domain\user name>`.
 - For local accounts (the user must be an administrator on the application server), prefix the user name with the host name in this format: `<host name\user name>`.
 - You can run application discovery for as many as five servers at a time.

2. Select **Validate** to verify that the application server is reachable from the machine running the tool and that the credentials are valid. Upon successful validation, the **Status** column will show the status as **Mapped**:

The screenshot shows the 'Discover applications' interface. At the top, there's a search icon and the text 'Discover applications' followed by 'Provide server details and run application discovery'. Below this is a 'Provide server details' step with a checked checkbox. The instructions say 'Provide server credentials with local administrator/root privileges to discover applications.' A table is present with columns: Server IP / FQDN, Username, Password, and Status. The first row contains '127.0.0.1', 'webvm\demouser', '*****', and a green checkmark with the word 'Mapped'. Below the table are 'Add server' and 'Validate' buttons. To the right is a 'Review' button. The bottom part of the screenshot shows a 'Review discovered applications' step with a 'Review' button.

Server IP / FQDN	Username	Password	Status
127.0.0.1	webvm\demouser	*****	✓ Mapped

3. Select **Continue** to start application discovery on the selected application servers.
4. When application discovery is finished, select the applications that you want to containerize:

Review discovered applications

1 application(s) discovered

Select applications to containerize, configurations to parameterize, and folders to migrate.

<input type="checkbox"/> Name	Server IP/ FQDN	Target container	Application configurations	Application folders
<input type="checkbox"/> localapp	127.0.0.1		1 app configuration(s)	Edit

Continue

5. Specify a name for the target container for each selected application. Specify the container name as `<name:tag>`, where `tag` is used for the container image. For example, you can specify the target container name as `appname:v1`.

Parameterize application configurations

Parameterizing the configuration makes it available as a deploy-time parameter. Parameterization allows you to configure a setting when you deploy the application as opposed to having it hard coded to a specific value in the container image. For example, this option is useful for parameters like database connection strings.

1. Select **app configurations** to review detected configurations.
2. Select the parameters that you want to parameterize, and then select **Apply**:

Discover applications

Provide server details and run discovery

Review discovered applications

1 application(s) discovered

Select applications to containerize

Parameterize application configurations

Review the identified application configurations that can be parameterized. The selected application configurations will be converted as-is into deployment settings. You can edit the values in deployment configurations step later.

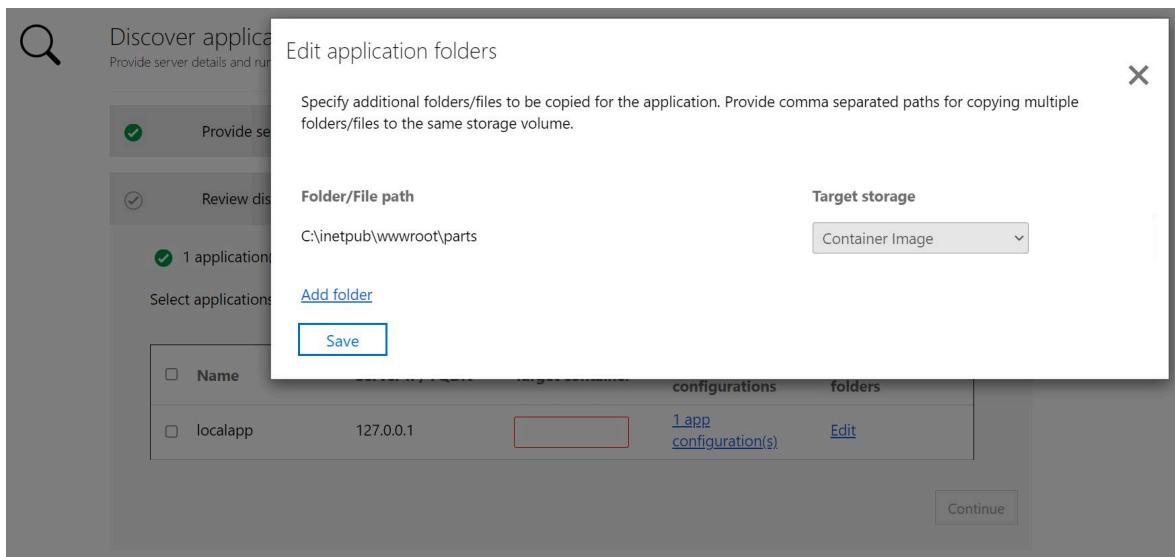
<input type="checkbox"/> File name	Section tag	Attribute name
<input checked="" type="checkbox"/> abc.config	/configuration/add[@name='StringDefault']	StringDefault

Apply

Externalize file system dependencies

You can add other folders that your application uses. Specify if they should be part of the container image or should be externalized to persistent storage via Azure file share. Using external persistent storage works great for stateful applications that store state outside the container or have other static content stored on the file system.

1. Select **Edit** under **Application folders** to review the detected application folders.
These folders have been identified as mandatory artifacts needed by the application. They'll be copied into the container image.
2. Select **Add folder** and specify the folder paths that you want to add.
3. To add multiple folders to the same volume, separate the values with commas.
4. Select **Azure file share** as the storage option if you want the folders to be stored outside the container on persistent storage.
5. Select **Save** after you review the application folders:



6. Select **Continue** to proceed to the container image build phase.

Build container image

1. In the dropdown list, select an **Azure container registry** that will be used to build and store the container images for the apps. You can use an existing Azure container registry or create a new one by selecting **Create new registry**:

Azure Container Registry subscription
Azure Migrate Demo Subscription

Azure Container Registry
Select
[Create new registry](#)

Click **Build** to start building the container image after reviewing the Dockerfile.

<input type="checkbox"/>	Container	App	Dockerfile	Build status
<input type="checkbox"/>	localapp:v1	localapp	Review	⚠ Not Built

Build **Continue**

! Note

Only Azure container registries with the admin user account enabled are displayed. The admin user account is currently required for deploying an image from an Azure container registry to Azure App Service. For more information, see [Authenticate with an Azure container registry](#).

2. The Dockerfiles needed to build the container images for each selected application are generated at the beginning of the build step. Select **Review** to review the Dockerfile. You can also add any necessary customizations to the Dockerfile in the review step and save the changes before you start the build process.
3. Select the applications that you want to build images for, and then select **Build**. Selecting **Build** will start the container image build for each application. The tool monitors the build status and will let you continue to the next step when the build finishes.
4. You can monitor the progress of the build by selecting **Build in Progress** under the status column. The link will become active a couple minutes after you trigger the build process.
5. After the build is complete, select **Continue** to specify deployment settings:

Review the generated Dockerfile. Select applications and click **Build** to start building the container image.

<input type="checkbox"/> Container	App	Dockerfile	Build status
<input checked="" type="checkbox"/> localapp:v1	localapp	Review	Successful

[Build](#)

[Continue](#)

Deploy the containerized app on Azure App Service

After the container image is built, the next step is to deploy the application as a container on [Azure App Service](#).

1. Select the Azure App Service plan that the application should use.

If you don't have an App Service plan or want to create a new App Service plan to use, you can create one by selecting [Create new App Service plan](#).

2. Select **Continue** after you select the App Service plan.

3. If you parameterized application configurations, specify the secret store to use for the application. You can choose Azure Key Vault or App Service application settings to manage your application secrets. For more information, see [Configure connection strings](#).

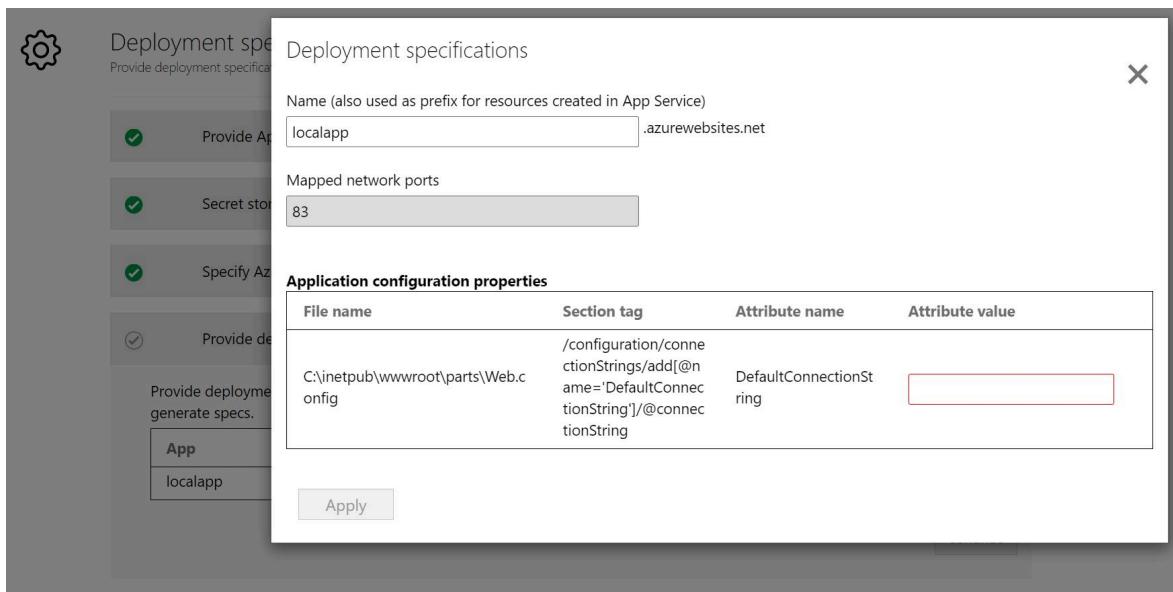
- If you selected App Service application settings to manage your secrets, select **Continue**.
- If you want to use an Azure key vault to manage your application secrets, specify the key vault that you want to use.
 - If you don't have an Azure key vault or want to create a new key vault, you can create one by selecting [Create new Azure Key Vault](#).
 - The tool will automatically assign the necessary permissions for managing secrets via the key vault.

4. If you added more folders and selected the Azure file share option for persistent storage, specify the Azure file share to be used by the App Containerization tool during deployment. The tool will copy over the application folders that you configured for Azure Files and mount them on the application container during deployment.

If you don't have an Azure file share or want to create a new Azure file share, you can create one by selecting [Create new Storage Account and file share](#).

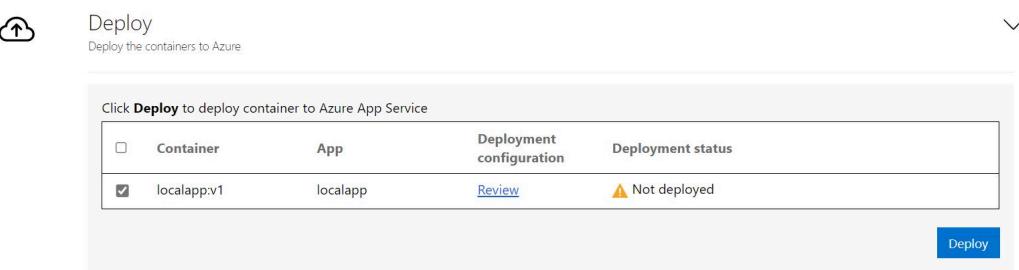
5. You now need to specify the deployment configuration for the application. Select **Configure** to customize the deployment for the application. In the configure step, you can provide these customizations:

- **Name.** Specify a unique app name for the application. This name will be used to generate the application URL. It will also be used as a prefix for other resources created as part of the deployment.
- **Application configuration.** For any application configurations that are parameterized, provide the values to use for the current deployment.
- **Storage configuration.** Review the information for any application folders that are configured for persistent storage.



6. After you save the deployment configuration for the application, the tool will generate the Kubernetes deployment YAML for the application.

- Select **Review** to review the deployment configuration for the applications.
- Select the applications that you want to deploy.
- Select **Deploy** to start deployment for the selected applications.



- After the application is deployed, you can select the **Deployment status** column to track the resources that were deployed for the application.

Troubleshoot problems

To troubleshoot problems with the App Containerization tool, you can look at the log files on the Windows machine that's running the tool. Log files for the tool are located at `C:\ProgramData\Microsoft Azure Migrate App Containerization\Logs`.

Next steps

- Containerizing ASP.NET web apps and deploying them on Windows containers on AKS
 - Containerizing Java web apps on Apache Tomcat (on Linux servers) and deploying them on Linux containers on AKS
 - Containerizing Java web apps on Apache Tomcat (on Linux servers) and deploying them on Linux containers on App Service
-

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

Java migration resources for Azure App Service

Article • 06/13/2022

Azure App Service provides tools to discover web apps deployed to on-premises web servers. You can assess these apps for readiness, then migrate them to App Service. Both the web app content and supported configuration can be migrated to App Service. These tools are developed to support a wide variety of scenarios focused on discovery, assessment, and migration.

Java Tomcat migration (Linux)

[Download the assistant](#) to migrate a Java app running on Apache Tomcat web server. You can also use Azure Container Registry to migrate on-premises Linux Docker containers to App Service.

Resources
How-tos
Java App-Service-Migration-Assistant Wiki
Azure/App-Service-Migration-Assistant Wiki
Videos
Point and Migrate Java Apps to Azure App Service Using the Migration System

Standalone Tomcat Web App Migration (Windows OS)

Download this [preview tool](#) to migrate a Java web app on Apache Tomcat to App Service on Windows. For more information, see the [video](#) and [how-to](#).

Containerize standalone Tomcat Web App

App Containerization offers a simple approach to repackage applications as containers with minimal code changes. The tool currently supports containerizing ASP.NET and Apache Tomcat Java web applications. For more information, see the [blog](#) and [how-to](#).

Next steps

[Migrate Tomcat applications to Tomcat on Azure App Service](#)

Migrate Tomcat applications to Tomcat on Azure App Service

Article • 11/25/2024

This guide describes what you should be aware of when you want to migrate an existing Tomcat application to run on Azure App Service using Tomcat 9.0.

Pre-migration

To ensure a successful migration, before you start, complete the assessment and inventory steps described in the following sections.

If you can't meet any of these pre-migration requirements, see the following companion migration guides:

- [Migrate Tomcat applications to containers on Azure Kubernetes Service](#)
- Migrate Tomcat Applications to Azure Virtual Machines (guidance planned)

Switch to a supported platform

App Service offers specific versions of Tomcat on specific versions of Java. To ensure compatibility, migrate your application to one of the supported versions of Tomcat and Java in its current environment before you continue with any of the remaining steps. Be sure to fully test the resulting configuration. Use the latest stable release of your Linux distribution in such tests.

ⓘ Note

This validation is especially important if your current server is running on an unsupported JDK (such as Oracle JDK or IBM OpenJ9).

To obtain your current Java version, sign in to your production server and run the following command:

Bash

```
java -version
```

On Azure App Service, the binaries for Java 8 are provided from Eclipse Temurin. For Java 11, 17, and all future LTS releases of Java, App Service provides the [Microsoft Build of OpenJDK](#). These binaries are available for free download at the following sites:

- [Download Java 8 ↗](#)
- [Download Java 11, 17, and all future LTS versions](#)

To obtain your current Tomcat version, sign in to your production server and run the following command:

Bash

```
 ${CATALINA_HOME}/bin/version.sh
```

To obtain the current version used by Azure App Service, download [Tomcat 9 ↗](#), depending on which version you plan to use in Azure App Service.

Inventory external resources

External resources, such as data sources, JMS message brokers, and others are injected via Java Naming and Directory Interface (JNDI). Some such resources may require migration or reconfiguration.

Inside your application

Inspect the `META-INF/context.xml` file. Look for `<Resource>` elements inside the `<Context>` element.

On the application server(s)

Inspect the `$CATALINA_BASE/conf/context.xml` and `$CATALINA_BASE/conf/server.xml` files as well as the `.xml` files found in `$CATALINA_BASE/conf/<engine-name>/<host-name>` directories.

In `context.xml` files, JNDI resources will be described by the `<Resource>` elements inside the top-level `<Context>` element.

In `server.xml` files, JNDI resources will be described by the `<Resource>` elements inside the `<GlobalNamingResources>` element.

Datasources

Datasources are JNDI resources with the `type` attribute set to `javax.sql.DataSource`. For each datasource, document the following information:

- What is the datasource name?
- What is the connection pool configuration?
- Where can I find the JDBC driver JAR file?

For more information, see [JNDI Datasource HOW-TO](#) in the Tomcat documentation.

All other external resources

It isn't feasible to document every possible external dependency in this guide. It's your team's responsibility to verify that you can satisfy every external dependency of your application after the migration.

Inventory secrets

Passwords and secure strings

Check all properties and configuration files on the production server(s) for any secret strings and passwords. Be sure to check `server.xml` and `context.xml` in `$CATALINA_BASE/conf`. You may also find configuration files containing passwords or credentials inside your application. These may include `META-INF/context.xml`, and, for Spring Boot applications, `application.properties` or `application.yml` files.

Inventory certificates

Document all the certificates used for public SSL endpoints or communication with backend databases and other systems. You can view all certificates on the production server(s) by running the following command:

```
Bash
```

```
keytool -list -v -keystore <path to keystore>
```

Determine whether and how the file system is used

Any usage of the file system on the application server will require reconfiguration or, in rare cases, architectural changes. You may identify some or all of the following scenarios.

Read-only static content

If your application currently serves static content, you'll need an alternate location for it. You may wish to consider moving static content to Azure Blob Storage and adding Azure CDN for lightning-fast downloads globally. For more information, see [Static website hosting in Azure Storage](#) and [Quickstart: Integrate an Azure storage account with Azure CDN](#).

Dynamically published static content

If your application allows for static content that is uploaded/produced by your application but is immutable after its creation, you can use Azure Blob Storage and Azure CDN as described above, with an Azure Function to handle uploads and CDN refresh. We've provided a sample implementation for your use at [Uploading and CDN-preloading static content with Azure Functions](#).

Dynamic or internal content

For files that are frequently written and read by your application (such as temporary data files), or static files that are visible only to your application, you can mount Azure Storage into the App Service file system. For more information, see [Mount Azure Storage as a local share in App Service](#).

Identify session persistence mechanism

To identify the session persistence manager in use, inspect the `context.xml` files in your application and Tomcat configuration. Look for the `<Manager>` element, and then note the value of the `className` attribute.

Tomcat's built-in [PersistentManager](#) implementations, such as [StandardManager](#) or [FileStore](#) aren't designed for use with a distributed, scaled platform such as App Service. Because App Service may load balance among several instances and transparently restart any instance at any time, persisting mutable state to a file system isn't recommended.

If session persistence is required, you'll need to use an alternate `PersistentManager` implementation that will write to an external data store, such as VMware Tanzu Session Manager with Redis Cache. For more information, see [Use Redis as a session cache with Tomcat](#).

Identify all outside processes and daemons running on the production servers

If you have any processes running outside the application server, such as monitoring daemons, you'll need to eliminate them or migrate them elsewhere.

Special cases

Certain production scenarios may require additional changes or impose additional limitations. While such scenarios can be infrequent, it's important to ensure that they're either inapplicable to your application or correctly resolved.

Determine whether application relies on scheduled jobs

Scheduled jobs, such as Quartz Scheduler tasks or cron jobs, can't be used with App Service. App Service won't prevent you from deploying an application containing scheduled tasks internally. However, if your application is scaled out, the same scheduled job may run more than once per scheduled period. This situation can lead to unintended consequences.

Inventory any scheduled jobs, inside or outside the application server.

Determine whether your application contains OS-specific code

If your application contains any code with dependencies on the host OS, then you need to refactor it to remove those dependencies. For example, you may need to replace any use of `/` or `\` in file system paths with [File.Separator](#) or [Paths.get](#) if your application is running on Windows.

Determine whether Tomcat clustering is used

[Tomcat clustering](#) isn't supported on Azure App Service. Instead, you can configure and manage scaling and load balancing through Azure App Service without Tomcat-specific functionality. You can persist session state to an alternate location to make it available across replicas. For more information, see [Identify session persistence mechanism](#).

To determine whether your application uses clustering, look for the `<Cluster>` element inside the `<Host>` or `<Engine>` elements in the `server.xml` file.

Determine whether non-HTTP connectors are used

App Service supports only a single HTTP connector. If your application requires additional connectors, such as the AJP connector, don't use App Service.

To identify HTTP connectors used by your application, look for `<Connector>` elements inside the `server.xml` file in your Tomcat configuration.

Determine whether MemoryRealm is used

[MemoryRealm](#) requires a persisted XML file. On Azure AppService, you'll need to upload this file to the `/home` directory or one of its subdirectories, or to mounted storage. You'll then need to modify the `pathName` parameter accordingly.

To determine whether `MemoryRealm` is currently used, inspect your `server.xml` and `context.xml` files and search for `<Realm>` elements where the `className` attribute is set to `org.apache.catalina.realm.MemoryRealm`.

Determine whether SSL session tracking is used

App Service performs session offloading outside of the Tomcat runtime, so you can't use [SSL session tracking](#). Use a different session tracking mode instead (`COOKIE` or `URL`). If you need SSL session tracking, don't use App Service.

Determine whether AccessLogValve is used

If you use [AccessLogValve](#), you should set the `directory` parameter to `/home/LogFiles` or one of its subdirectories.

Migration

Parameterize the configuration

In the pre-migration steps, you likely identified some secrets and external dependencies, such as datasources, in `server.xml` and `context.xml` files. For each item you identified, replace any username, password, connection string, or URL with an environment variable.

Note

Microsoft recommends using the most secure authentication flow available. The authentication flow described in this procedure, such as for databases, caches, messaging, or AI services, requires a very high degree of trust in the application and carries risks not present in other flows. Use this flow only when more secure options, like managed identities for passwordless or keyless connections, are not viable. For local machine operations, prefer user identities for passwordless or keyless connections.

For example, suppose the `context.xml` file contains the following element:

XML

```
<Resource
    name="jdbc/dbconnection"
    type="javax.sql.DataSource"
    url="jdbc:postgresql://postgresdb.contoso.com/wickedsecret?ssl=true"
    driverClassName="org.postgresql.Driver"
    username="postgres"
    password="{password}"
/>
```

In this case, you could change it as shown in the following example:

XML

```
<Resource
    name="jdbc/dbconnection"
    type="javax.sql.DataSource"
    url="${postgresdb.connectionString}"
    driverClassName="org.postgresql.Driver"
    username="${postgresdb.username}"
    password="${postgresdb.password}"
/>
```

To ensure that parameter substitution occurs for any `context.xml` file within the **META-INF** folder inside a deployed `.war` file, be sure to set the `CATALINA_OPTS` environment variable as shown in the following example:

Bash

```
export CATALINA_OPTS="-
Dorg.apache.tomcat.util.digester.PROPERTY_SOURCE=org.apache.tomcat.util.dige
ster.EnvironmentPropertySource"
```

Provision an App Service plan

From the list of available service plans at [App Service pricing](#), select the plan whose specifications meet or exceed those of the current production hardware.

ⓘ Note

If you plan to run staging/canary deployments or use deployment slots, the App Service plan must include that additional capacity. We recommend using Premium or higher plans for Java applications. For more information, see [Set up staging environments in Azure App Service](#).

Then, create the App Service plan. For more information, see [Manage an App Service plan in Azure](#).

Create and deploy Web App(s)

You'll need to create a Web App on your App Service Plan (choosing a version of Tomcat as the runtime stack) for every WAR file deployed to your Tomcat server.

ⓘ Note

While it's possible to deploy multiple WAR files to a single web app, this is highly undesirable. Deploying multiple WAR files to a single web app prevents each application from scaling according to its own usage demands. It also adds complexity to subsequent deployment pipelines. If multiple applications need to be available on a single URL, consider using a routing solution such as [Azure Application Gateway](#).

Maven applications

If your application is built from a Maven POM file, [use the Webapp plugin for Maven](#) to create the Web App and deploy your application.

Non-Maven applications

If you can't use the Maven plugin, you'll need to provision the Web App through other mechanisms, such as:

- [Azure portal](#)
- [Azure CLI](#)
- [Azure PowerShell](#)

Once the Web App has been created, use one of the [available deployment mechanisms](#) to deploy your application.

Migrate JVM runtime options

If your application requires specific runtime options, [use the most appropriate mechanism to specify them](#).

Populate secrets

Use Application Settings to store any secrets specific to your application. If you intend to use the same secret(s) among multiple applications or require fine-grained access policies and audit capabilities, [use Azure Key Vault](#) instead.

Configure custom domain and SSL

If your application will be visible on a custom domain, you'll need to map your web application to it. For more information, see [Tutorial: Map an existing custom DNS name to Azure App Service](#).

Then, you'll need to bind the SSL certificate for that domain to your App Service Web App. For more information, see [Secure a custom DNS name with an SSL binding in Azure App Service](#).

Import backend certificates

All certificates for communicating with backend systems, such as databases, need to be made available to App Service. For more information, see [Add an SSL certificate in App Service](#).

Migrate data sources, libraries, and JNDI resources

For data source configuration steps, see the [Data sources](#) section of [Configure a Linux Java app for Azure App Service](#).

For additional data source instructions, see the following sections of the [JNDI Datasource How-To](#) in the Tomcat documentation:

- [MySQL](#)
- [PostgreSQL](#)
- [SQL Server](#)

Migrate any additional server-level classpath dependencies by following [the same steps as for data source JAR files](#).

Migrate any additional [Shared server-level JDNI resources](#).

 **Note**

If you're following the recommended architecture of one WAR per webapp, consider migrating server-level classpath libraries and JNDI resources into your application. This will significantly simplify component governance and change management.

Migrate remaining configuration

Upon completing the preceding section, you should have your customizable server configuration in `/home/tomcat/conf`.

Complete the migration by copying any additional configuration (such as [realms](#) ↗ and [JASPICTM](#) ↗)

Migrate scheduled jobs

To execute scheduled jobs on Azure, consider using a [Timer trigger for Azure Functions](#). You don't need to migrate the job code itself into a function. The function can simply invoke a URL in your application to trigger the job. If such job executions have to be dynamically invoked and/or centrally tracked, consider using [Spring Batch](#) ↗.

Alternatively, you can create a Logic app with a Recurrence trigger to invoke the URL without writing any code outside your application. For more information, see [Overview - What is Azure Logic Apps?](#) and [Create, schedule, and run recurring tasks and workflows with the Recurrence trigger in Azure Logic Apps](#).

 **Note**

To prevent malicious use, you'll likely need to ensure that the job invocation endpoint requires credentials. In this case, the trigger function will need to provide the credentials.

Restart and smoke-test

Finally, you'll need to restart your Web App to apply all configuration changes. Upon completion of the restart, verify that your application is running correctly.

Post-migration

Now that you have your application migrated to Azure App Service you should verify that it works as you expect. Once you've done that we have some recommendations for you that can make your application more Cloud native.

Recommendations

- If you opted to use the `/home` directory for file storage, consider [replacing it with Azure Storage](#).
- If you have configuration in the `/home` directory that contains connection strings, SSL keys, and other secret information, consider using a combination of [Azure Key Vault](#) and/or [parameter injection with application settings](#) where possible.

Note

Microsoft recommends using the most secure authentication flow available. The authentication flow described in this procedure, such as for databases, caches, messaging, or AI services, requires a very high degree of trust in the application and carries risks not present in other flows. Use this flow only when more secure options, like managed identities for passwordless or keyless connections, are not viable. For local machine operations, prefer user identities for passwordless or keyless connections.

- Consider [using Deployment Slots](#) for reliable deployments with zero downtime.
- Design and implement a DevOps strategy. To maintain reliability while increasing your development velocity, consider [automating deployments and testing with Azure Pipelines](#). When you use Deployment Slots, you can [automate deployment to a slot](#) followed by the slot swap.
- Design and implement a business continuity and disaster recovery strategy. For mission-critical applications, consider a [multi-region deployment architecture](#).

Feedback

Was this page helpful?

 Yes

 No

Get help at Microsoft Q&A

Migrate JBoss EAP applications to JBoss EAP on Azure App Service

Article • 10/21/2024

This guide describes what you should be aware of when you want to migrate an existing JBoss EAP application to run on JBoss EAP in an Azure App Service instance.

Pre-migration

To ensure a successful migration, before you start, complete the assessment and inventory steps described in the following sections.

Inventory server capacity

Document the hardware (memory, CPU, disk) of the current production server(s) and the average and peak request counts and resource utilization. You'll need this information regardless of the migration path you choose. It's useful, for example, to help guide selection of the size of the VMs in your node pool, the amount of memory to be used by the container, and how many CPU shares the container needs.

It's possible to resize node pools in AKS. To learn how, see [Resize node pools in Azure Kubernetes Service \(AKS\)](#).

Inventory all secrets

Check all properties and configuration files on the production server(s) for any secrets and passwords. Be sure to check `jboss-web.xml` in your WARs. Configuration files that contain passwords or credentials may also be found inside your application.

Consider storing those secrets in Azure KeyVault. For more information, see [Azure Key Vault basic concepts](#).

You can use Key Vault secrets in your App Service instance with Key Vault references. Key Vault references allow you to use the secrets in your application while keeping them secured and encrypted at rest. For more information, see [Use Key Vault references for App Service and Azure Functions](#).

Inventory all certificates

Document all the certificates used for public SSL endpoints. You can view all certificates on the production server(s) by running the following command:

Bash

```
keytool -list -v -keystore <path to keystore>
```

Validate that the supported Java version works correctly

JBoss EAP on Azure VMs requires a supported version of Java. For guidance on which version of the JDK to use, see [Supported Configurations](#) in the Red Hat documentation.

Note

This validation is especially important if your current server is running on an unsupported JDK (such as Oracle JDK or IBM OpenJ9).

To obtain your current Java version, sign in to your production server and run the following command:

Bash

```
java -version
```

Inventory external resources

External resources, such as data sources, JMS message brokers, and others are injected via Java Naming and Directory Interface (JNDI). Some such resources may require migration or reconfiguration.

Inside your application

Inspect the `WEB-INF/jboss-web.xml` and/or `WEB-INF/web.xml` files. Look for `<Resource>` elements inside the `<Context>` element.

Datasources

Datasources are JNDI resources with the `type` attribute set to `javax.sql.DataSource`. For each datasource, document the following information:

- What is the datasource name?
- What is the connection pool configuration?
- Where can I find the JDBC driver JAR file?

For more information, see [About JBoss EAP Datasources](#) in the JBoss EAP documentation.

All other external resources

It isn't feasible to document every possible external dependency in this guide. It's your team's responsibility to verify that you can satisfy every external dependency of your application after the migration.

Determine whether session replication is used

If your application relies on session replication, you'll have to change your application to remove this dependency. App Service does not allow instances to communicate directly with one another.

Determine whether and how the file system is used

Any usage of the file system on the application server will require reconfiguration or, in rare cases, architectural changes. The file system may be used by JBoss EAP modules or by your application code. You may identify some or all of the scenarios described in the following sections.

Read-only static content

If your application currently serves static content, you'll need an alternate location for it. You may wish to consider moving static content to Azure Blob Storage and adding Azure CDN for lightning-fast downloads globally. For more information, see [Static website hosting in Azure Storage](#) and [Quickstart: Integrate an Azure storage account with Azure CDN](#).

Dynamically published static content

If your application allows for static content that is uploaded/produced by your application but is immutable after its creation, you can use Azure Blob Storage and Azure CDN as described above, with an Azure Function to handle uploads and CDN refresh. We've provided a sample implementation for your use at [Uploading and CDN-preloading static content with Azure Functions](#).

Dynamic or internal content

For files that are frequently written and read by your application (such as temporary data files), or static files that are visible only to your application, you can use local file storage associated with your app service plan. For more information, see [Operating system functionality on Azure App Service](#) and [Understanding the Azure App Service file system](#).

Determine whether your application relies on scheduled jobs

Scheduled jobs, such as Quartz Scheduler tasks or Unix cron jobs, should NOT be used with Azure App Service. Azure App Service will not prevent you from deploying an application containing scheduled tasks internally. However, if your application is scaled out, the same scheduled job may run more than once per scheduled period. This situation can lead to unintended consequences.

Inventory any scheduled tasks running on the production server(s), inside or outside your application code.

Determine whether a connection to on-premises is needed

If your application needs to access any of your on-premises services, you'll need to provision one of Azure's connectivity services. For more information, see [Connect an on-premises network to Azure](#). Alternatively, you'll need to refactor your application to use publicly available APIs that your on-premises resources expose.

Determine whether Java Message Service (JMS) Queues or Topics are in use

If your application is using JMS Queues or Topics, you'll need to migrate them to an externally hosted JMS server. Azure Service Bus and the Advanced Message Queuing Protocol (AMQP) can be a great migration strategy for those using JMS. For more information, see [Use Java Message Service 1.1 with Azure Service Bus standard and AMQP 1.0](#).

If JMS persistent stores have been configured, you must capture their configuration and apply it after the migration.

Determine whether JCA connectors are in use

If your application uses JCA connectors, validate that you can use the JCA connector on JBoss EAP. If you can use the JCA connector on JBoss EAP, then for it to be available, you must add the JARs to the server classpath and put the necessary configuration files in the correct location in the JBoss EAP server directories.

Determine whether JAAS is in use

If your application is using JAAS, you'll need to capture how JAAS is configured. If it's using a database, you can convert it to a JAAS domain on JBoss EAP. If it's a custom implementation, you'll need to validate that it can be used on JBoss EAP.

Determine whether your application uses a Resource Adapter

If your application needs a Resource Adapter (RA), it needs to be compatible with JBoss EAP. Determine whether the RA works fine on a standalone instance of JBoss EAP by deploying it to the server and properly configuring it. If the RA works properly, you'll need to add the JARs to the server classpath of the App Service and put the necessary configuration files in the correct location in the JBoss EAP server directories for it to be available.

Determine whether your application is composed of multiple WARs

If your application is composed of multiple WARs, you should treat each of those WARs as separate applications and go through this guide for each of them.

Determine whether your application is packaged as an EAR

If your application is packaged as an EAR file, be sure to examine the `application.xml` file and capture the configuration.

 **Note**

If you want to be able to scale each of your web applications independently for better use of your App Service resources you should break up the EAR into separate web applications.

Identify all outside processes and daemons running on the production servers

If you have any processes running outside the application server, such as monitoring daemons, you'll need to eliminate them or migrate them elsewhere.

Perform in-place testing

Before creating your container images, migrate your application to the JDK and JBoss EAP versions that you intend to use on App Service. Test the application thoroughly to ensure compatibility and performance.

JBoss EAP on App Service feature notes

When using JBoss EAP on App Service, be sure to take the following notes into consideration.

- **JBoss EAP management console:** The JBoss web console isn't exposed on App Service. Instead, the Azure portal provides the management APIs for your application, and you should deploy using the Azure CLI, Azure Maven Plugin, or other Azure developer tools. Further configuration of JBoss resources can be achieved using the JBoss CLI during the application startup.
- **Transactions:** The Transactions API is supported and there is support for automatic transaction recovery. For more information, see [Managing transactions on JBoss EAP](#) in the Red Hat documentation.
- **Managed domain mode:** In a multi-server production environment, Managed Domain mode in JBoss EAP offers centralized managed capabilities. However with JBoss EAP on App Service, the App Service platform assumes the responsibility for configuration and management of your server instances. App Service eliminates the need for JBoss EAP's managed domain mode. Domain mode is a good choice for virtual machine-based multi-server deployments. For more information, see [About managed domains](#) in the Red Hat documentation.
- **Server-to-server clustering:** As of September 28th, 2023, clustered deployment of JBoss EAP is Generally Available. This support means that you no longer have to

- remove the following features from your applications before you can deploy them to App Service:
- Stateful session beans.
 - Distributed transactions.
 - Similar features that require instance-to-instance communication or high availability.

For more information, see the [release announcement](#) and the [Clustering](#) section of [Configure a Java app for Azure App Service](#).

Migration

Red Hat Migration Toolkit for Apps

The [Red Hat Migration Toolkit for Applications](#) is a free extension for Visual Studio Code. This extension analyzes your application code and configuration to provide recommendations for migrating to the cloud from on-premises. For more information, see [Migration Toolkit for Applications overview](#).

The contents of this guide will help you address the other components of the migration journey, such as choosing the correct App Service Plan type, externalizing your session state, and using Azure to manage your EAP instances instead of the JBoss Management interface.

Provision Azure App Service for JBoss EAP runtime

Use the following commands to create a resource group and an Azure App Service Plan. After the App Service Plan is created, a Linux web app plan is created using the JBoss EAP runtime. You can create JBoss EAP sites only on PremiumV3 and IsolatedV2 App Service Plan tiers.

Be sure the specified environment variables have appropriate values.

 **Note**

PremiumV3 and IsolatedV2 are both eligible for Reserved Instance pricing, which can reduce your costs. For more information on App Service Plan tiers and Reserved Instance pricing, see [App Service pricing](#).

```
az group create --resource-group $resourceGroup --location eastus
az acr create --resource-group $resourceGroup --name $acrName --sku Standard
az appservice plan create \
    --resource-group $resourceGroup \
    --name $jbossAppService \
    --is-linux \
    --sku P1V2
az webapp create \
    --resource-group $resourceGroup \
    --name $jbossWebApp \
    --plan $jbossAppServicePlan \
    --runtime "JBOSSEAP|7-java8"
# Or use "JBOSSEAP|7-java11" if you're using Java 11
```

Build the application

Build the application using the following Maven command.

Bash

```
mvn clean install -DskipTests
```

Deploy the application

If your application is built from a Maven POM file, use the Webapp plugin for Maven to create the Web App and deploy your application. For more information, see [Quickstart: Create a Java app on Azure App Service](#).

To automate the deployment of JBoss EAP applications, you can use [Azure Pipelines task for Web App](#) or [GitHub Action for deploying to Azure WebApp](#).

Set up data sources

There are three core steps when registering a data source with JBoss EAP: uploading the JDBC driver, adding the JDBC driver as a module, and registering the module. For more information, see [Datasource Management](#) in the JBoss EAP documentation. App Service is a stateless hosting service, so the configuration commands for adding and registering the data source module must be scripted and applied as the container starts.

To set up data sources, use the following steps.

1. Obtain your database's JDBC driver.

2. Create an XML module definition file for the JDBC driver. The example shown below is a module definition for PostgreSQL. Be sure to replace the `resource-root path` value with the path to the JDBC driver you use.

XML

```
<?xml version="1.0" ?>
<module xmlns="urn:jboss:module:1.1" name="org.postgres">
    <resources>
        <!-- ***** IMPORTANT: REPLACE THIS PLACEHOLDER *****-->
        <resource-root path="/home/site/deployments/tools/postgresql-42.2.12.jar" />
    </resources>
    <dependencies>
        <module name="javax.api"/>
        <module name="javax.transaction.api"/>
    </dependencies>
</module>
```

3. Put your JBoss CLI commands into a file named `jboss-cli-commands.cli`. The JBoss commands must add the module and register it as a data source. The example below shows the JBoss CLI commands for PostgreSQL.

ⓘ Note

Microsoft recommends using the most secure authentication flow available. The authentication flow described in this procedure, such as for databases, caches, messaging, or AI services, requires a very high degree of trust in the application and carries risks not present in other flows. Use this flow only when more secure options, like managed identities for passwordless or keyless connections, are not viable. For local machine operations, prefer user identities for passwordless or keyless connections.

Bash

```
module add --name=org.postgres --
resources=/home/site/deployments/tools/postgresql-42.2.12.jar --module-
xml=/home/site/deployments/tools/postgres-module.xml

/subsystem=datasources/jdbc-driver=postgres:add(driver-
name="postgres",driver-module-name="org.postgres",driver-class-
name=org.postgresql.Driver,driver-xa-datasource-class-
name=org.postgresql.xa.PGXDataSource)

data-source add --name=postgresDS --driver-name=postgres --jndi-
name=java:jboss/datasources/postgresDS --connection-
url=${POSTGRES_CONNECTION_URL},env.POSTGRES_CONNECTION_URL:jdbc:postgres
```

```
ql://db:5432/postgres} --user-  
name=${POSTGRES_SERVER_ADMIN_FULL_NAME},env.POSTGRES_SERVER_ADMIN_FULL_N  
AME:postgres} --  
password=${POSTGRES_SERVER_ADMIN_PASSWORD},env.POSTGRES_SERVER_ADMIN_PAS  
WORD:example} --use-ccm=true --max-pool-size=5 --blocking-timeout-  
wait-millis=5000 --enabled=true --driver-class=org.postgresql.Driver --  
exception-sorter-class-  
name=org.jboss.jca.adapters.jdbc.extensions.postgres.PostgreSQLExceptionSorter --jta=true --use-java-context=true --valid-connection-checker-  
class-  
name=org.jboss.jca.adapters.jdbc.extensions.postgres.PostgreSQLValidConnectionChecker
```

4. Create a startup script called `startup_script.sh` that calls the JBoss CLI commands. The example below shows how to call your `jboss-cli-commands.cli` file. Later you will configure App Service to run this script when the instance starts.

Bash

```
$JBoss_HOME/bin/jboss-cli.sh --connect --  
file=/home/site/deployments/tools/jboss-cli-commands.cli
```

5. Using an FTP client of your choice, upload your JDBC driver, `jboss-cli-commands.cli`, `startup_script.sh`, and the module definition to `/site/deployments/tools/`.
6. Configure your site to run `startup_script.sh` when the container starts. In the Azure portal, navigate to **Configuration > General Settings > Startup Command**. Set the startup command field to `/home/site/deployments/tools/startup_script.sh`, then select **Save**.
7. Restart the web app, which will cause it to run the configuration script.
8. Update the JTA datasource configuration for your application. Open the `src/main/resources/META-INF/persistence.xml` file for your app and find the `<jta-data-source>` element. Replace its contents as shown here:

Bash

```
<jta-data-source>java:jboss/datasources/postgresDS</jta-data-source>
```

Build the application

Build the application using the following Maven command.

```
Bash
```

```
mvn clean install -DskipTests
```

Deploy the application

If your application is built from a Maven POM file, use the Webapp plugin for Maven to create the Web App and deploy your application. For more information, see [Quickstart: Create a Java app on Azure App Service](#).

To automate the deployment of JBoss EAP applications, you can use [Azure Pipelines task for Web App](#) or [GitHub Action for deploying to Azure WebApp ↗](#).

Post-migration

Now that you've migrated your application to Azure App Service, you should verify that it works as you expect. After you've done that, we have some recommendations for you that can make your application more cloud-native.

Recommendations

- If you opted to use the `/home` directory for file storage, consider replacing it with Azure Storage. For more information, see [Access Azure Storage as a network share from a container in App Service](#).
- If you have configuration in the `/home` directory that contains connection strings, SSL keys, and other secret information, consider using Azure Key Vault and/or parameter injection with application settings where possible. For more information, see [Use Key Vault references for App Service and Azure Functions](#) and [Configure an App Service app in the Azure portal](#).
- Consider using deployment slots for reliable deployments with zero downtime. For more information, see [Set up staging environments in Azure App Service](#).
- Design and implement a DevOps strategy. To maintain reliability while increasing your development velocity, consider automating deployments and testing with Azure Pipelines. For more information, see [Build and deploy to a Java web app](#). When you use deployment slots, you can [automate deployment to a slot](#) followed by the slot swap. For more information, see the [Deploy to a slot](#) section of [Deploy an Azure Web App \(Linux\)](#).

- Design and implement a business continuity and disaster recovery strategy. For mission-critical applications, consider a multi-region deployment architecture. For more information, see [Highly available multi-region web application](#).
-

Feedback

Was this page helpful?



Yes



No

[Get help at Microsoft Q&A](#)

Migrate WebLogic Server applications to JBoss EAP on Azure App Service

Article • 09/09/2024

This guide describes what you should be aware of when you want to migrate an existing WebLogic Server application to run on Azure App Service using JBoss EAP.

Pre-migration

To ensure a successful migration, before you start, complete the assessment and inventory steps described in the following sections.

If you can't meet any of these pre-migration requirements, see the companion migration guide to migrate your applications to Virtual Machines instead: [Migrate WebLogic Server applications to Azure Virtual Machines](#)

Inventory server capacity

Document the hardware (memory, CPU, disk) of the current production server(s) and the average and peak request counts and resource utilization. You'll need this information regardless of the migration path you choose. It's useful, for example, to help guide selection of the App Service Plan.

The [list of available App Service Plan tiers](#) shows the memory, CPU cores, storage, and pricing information. Note that JBoss EAP on App Service is only available on the **Premium V3** and **Isolated V2** App Service Plan tiers.

Inventory all secrets

Before the advent of "configuration as a service" technologies such as Azure Key Vault, there wasn't a well-defined concept of "secrets". Instead, you had a disparate set of configuration settings that effectively functioned as what we now call "secrets". With app servers such as WebLogic Server, these secrets are in many different config files and configuration stores. Check all properties and configuration files on the production server(s) for any secrets and passwords. Be sure to check `weblogic.xml` in your WARs. Configuration files containing passwords or credentials may also be found inside your application. For more information, see [Azure Key Vault basic concepts](#).

Inventory all certificates

Document all the certificates used for public SSL endpoints. You can view all certificates on the production server(s) by running the following command:

Bash

```
keytool -list -v -keystore <path to keystore>
```

Inventory JNDI resources

Inventory all JNDI resources. For example, datasources such as databases may have an associated JNDI name that allows JPA to correctly bind instances of `EntityManager` to a particular database. For more information on JNDI resources and databases, see [WebLogic Server Data Sources](#) in the Oracle documentation. Other JNDI-related resources, such as JMS message brokers, may require migration or reconfiguration. For more information on JMS configuration see [Oracle WebLogic Server 12.2.1.4.0](#).

Inspect your domain configuration

The main configuration unit in WebLogic Server is the domain. As such, the `config.xml` file contains a wealth of configuration that you must carefully consider for migration. The file includes references to additional XML files that are stored in subdirectories. Oracle advises that you should normally use the **Administration Console** to configure WebLogic Server's manageable objects and services and allow WebLogic Server to maintain the `config.xml` file. For more information, see [Domain Configuration Files](#).

Inside your application

Inspect the `WEB-INF/weblogic.xml` file and/or the `WEB-INF/web.xml` file.

Determine whether session replication is used

If your application relies on session replication, with or without Oracle Coherence*Web, you have two options:

- Refactor your application to use a database for session management.
- Refactor your application to externalize the session to Azure Redis Service. For more information, see [Azure Cache for Redis](#).

For all of these options, it's a good idea to master how WebLogic does HTTP Session State Replication. For more information, see [HTTP Session State Replication](#) in the Oracle documentation.

Document datasources

If your application uses any databases, you need to capture the following information:

- What is the datasource name?
- What is the connection pool configuration?
- Where can I find the JDBC driver JAR file?

For more information on JDBC drivers in WebLogic, see [Using JDBC Drivers with WebLogic Server](#).

Determine whether WebLogic has been customized

Determine which of the following customizations have been made, and capture what's been done.

- Have the startup scripts been changed? Such scripts include *setDomainEnv*, *commEnv*, *startWebLogic*, and *stopWebLogic*.
- Are there any specific parameters passed to the JVM?
- Are there JARs added to the server classpath?

Determine whether a connection to on-premises is needed

If your application needs to access any of your on-premises services, you'll need to provision one of Azure's connectivity services. For more information, see [Connect an on-premises network to Azure](#). Alternatively, you'll need to refactor your application to use publicly available APIs that your on-premises resources expose.

Determine whether Java Message Service (JMS) Queues or Topics are in use

If your application is using JMS Queues or Topics, you'll need to migrate them to an externally hosted JMS server. Azure Service Bus and the Advanced Message Queuing Protocol (AMQP) can be a great migration strategy for those using JMS. For more information, see [Use Java Message Service 1.1 with Azure Service Bus standard and AMQP 1.0](#).

If JMS persistent stores have been configured, you must capture their configuration and apply it after the migration.

Determine whether you are using your own custom created Shared Java EE Libraries

If you're using the Shared Java EE library feature, you have two options:

- Refactor your application code to remove all dependencies on your libraries, and instead incorporate the functionality directly into your application.
- Add the libraries to the server classpath.

Determine whether OSGi bundles are used

If you used OSGi bundles added to the WebLogic server, you'll need to add the equivalent JAR files directly to your web application.

Determine whether your application contains OS-specific code

If your application contains any code with dependencies on the host OS, then you'll need to refactor it to remove those dependencies. For example, you may need to replace any use of `/` or `\` in file system paths with [File.Separator](#) or [Paths.get](#).

Determine whether Oracle Service Bus is in use

If your application is using Oracle Service Bus (OSB), you'll need to capture how OSB is configured. For more information, see [About the Oracle Service Bus Installation](#).

Determine whether your application is composed of multiple WARs

If your application is composed of multiple WARs, you should treat each of those WARs as separate applications and go through this guide for each of them.

Determine whether your application is packaged as an EAR

If your application is packaged as an EAR file, be sure to examine the *application.xml* and *weblogic-application.xml* files and capture their configurations.

Identify all outside processes and daemons running on the production servers

If you have any processes running outside the application server, such as monitoring daemons, you'll need to eliminate them or migrate them elsewhere.

Validate that the supported Java version works correctly

JBoss EAP on Azure App Service supports Java 8 and 11. Therefore, you'll need to validate that your application is able to run correctly using that supported version. This validation is especially important if your current server is using a supported JDK (such as Oracle JDK or IBM OpenJ9).

To obtain your current Java version, sign in to your production server and run the following command:

```
Bash
```

```
java -version
```

Determine whether your application relies on scheduled jobs

Scheduled jobs, such as Quartz Scheduler tasks or Unix cron jobs, should NOT be used with Azure App Service. Azure App Service will not prevent you from deploying an application containing scheduled tasks internally. However, if your application is scaled out, the same scheduled job may run more than once per scheduled period. This situation can lead to unintended consequences.

To execute scheduled jobs on Azure, consider using Azure Functions with a Timer Trigger. For more information, see [Timer trigger for Azure Functions](#). You don't need to migrate the job code itself into a function. The function can simply invoke a URL in your application to trigger the job.

Note

To prevent malicious use, you'll likely need to ensure that the job invocation endpoint requires credentials. In this case, the trigger function will need to provide the credentials.

Determine whether WebLogic Scripting Tool (WLST) is used

If you currently use WLST to perform the deployment, you will need to assess what it is doing. If WLST is changing any (runtime) parameters of your application as part of the deployment, you will need to make sure those parameters conform to one of the following options:

- They are externalized as app settings.
- They are embedded in your application.
- They are using the JBoss CLI during deployment.

If WLST is doing more than what is mentioned above, you will have some additional work to do during migration.

Determine whether your application uses WebLogic-specific APIs

If your application uses WebLogic-specific APIs, you will need to refactor your application to NOT use them. For example, if you have used a class mentioned in the [Java API Reference for Oracle WebLogic Server](#), you have used a WebLogic-specific API in your application. The [Red Hat Migration Toolkit for Apps](#) can assist with removing and refactoring these dependencies.

Determine whether your application uses Entity Beans or EJB 2.x-style CMP Beans

If your application uses Entity Beans or EJB 2.x style CMP beans, you will need to refactor your application to NOT use them.

Determine whether the Java EE Application Client feature is used

If you have client applications that connect to your (server) application using the Java EE Application Client feature, you will need to refactor both your client applications and your (server) application to use HTTP APIs.

Determine whether a deployment plan was used

If a deployment plan was used to perform the deployment, you'll need to assess what the deployment plan is doing. If the deployment plan is a straight deploy, then you'll be able to deploy your web application without any changes. If the deployment plan is more elaborate, you'll need to determine whether you can use the JBoss CLI to properly configure your application as part of the deployment. If it isn't possible to use the JBoss CLI, you'll need to refactor your application in such a way that a deployment plan is no longer needed.

Determine whether EJB timers are in use

If your application uses EJB timers, you'll need to validate that the EJB timer code can be triggered by each JBoss EAP instance independently. This validation is needed because when your App Service is scaled out horizontally, each EJB timer will be triggered on its own JBoss EAP instance.

Validate if and how the file system is used

Any usage of the file system on the application server will require reconfiguration or, in rare cases, architectural changes. File system may be used by WebLogic shared modules or by your application code. You may identify some or all of the following scenarios.

Read-only static content

If your application currently serves static content, an alternate location for that static content will be required. You may wish to consider moving [static content to Azure Blob Storage](#) and [adding Azure CDN](#) for lightning-fast downloads globally.

Dynamically published static content

If your application allows for static content that is uploaded/produced by your application but is immutable after its creation, you can use Azure Blob Storage and Azure CDN as described above, with an Azure Function to handle uploads and CDN refresh. We have provided [a sample implementation for your use ↗](#).

Dynamic or internal content

For files that are frequently written and read by your application (such as temporary data files), or static files that are visible only to your application, Azure Storage can be [mounted into the App Service file system](#).

Determine whether JCA connectors are used

If your application uses JCA connectors you'll have to validate the JCA connector can be used on JBoss EAP. If the JCA implementation is tied to WebLogic, you'll have to refactor your application to NOT use the JCA connector. If it can be used, then you'll need to add the JARs to the server classpath and put the necessary configuration files in the correct location in the JBoss EAP server directories for it to be available.

Determine whether your application uses a Resource Adapter

If your application needs a Resource Adapter (RA), it needs to be compatible with JBoss EAP. Determine whether the RA works fine on a standalone instance of JBoss EAP by deploying it to the server and properly configuring it. If the RA works properly, you'll need to add the JARs to the server classpath of the App Service instance and put the necessary configuration files in the correct location in the JBoss EAP server directories for it to be available.

Determine whether JAAS is used

If your application is using JAAS, then you'll need to capture how JAAS is configured. If it's using a database, you can convert it to a JAAS domain on JBoss EAP. If it's a custom implementation, you'll need to validate that it can be used on JBoss EAP.

Determine whether WebLogic clustering is used

Most likely, you've deployed your application on multiple WebLogic servers to achieve high availability. Azure App Service is capable of scaling, but if you've used the WebLogic Cluster API, you'll need to refactor your code to eliminate the use of that API.

Migration

Red Hat Migration Toolkit for Apps

The [Red Hat Migration Toolkit for Applications](#) is a free extension for Visual Studio Code. This extension analyzes your application code and configuration to provide recommendations for migrating your Jakarta EE applications to JBoss EAP from other app servers, such as removing dependencies on proprietary APIs. The extension will also provide recommendations if you're migrating to the cloud from on-premises. For more information, see [Migration Toolkit for Applications overview](#).

The contents of this guide will help you address the other components of the migration journey, such as choosing the correct App Service Plan type, externalizing your session state, and using Azure to manage your EAP instances instead of the JBoss Management interface.

Provision an App Service plan

From the [list of available service plans](#), select the plan whose specifications meet or exceed the specifications of the current production hardware.

ⓘ Note

If you plan to run staging/canary deployments or use [deployment slots](#), the App Service plan must include that additional capacity. We recommend using Premium or higher plans for Java applications.

[Create the App Service plan.](#)

Create and Deploy Web App(s)

You'll need to create a Web App on your App Service Plan for every WAR file deployed to your JBoss EAP server.

ⓘ Note

While it's possible to deploy multiple WAR files to a single web app, this is highly undesirable. Deploying multiple WAR files to a single web app prevents each application from scaling according to its own usage demands. It also adds complexity to subsequent deployment pipelines. If multiple applications need to be available on a single URL, consider using a routing solution such as [Azure Application Gateway](#).

Maven applications

If your application is built from a Maven POM file, use the Webapp plugin for Maven to create the Web App and deploy your application. For more information, see the [Configure the Maven plugin](#) section of [Quickstart: Create a Java app on Azure App Service](#).

Non-Maven applications

If you can't use the Maven plugin, you'll need to provision the Web App through other mechanisms, such as:

- [Azure portal ↗](#)
- [Azure CLI](#)
- [Azure PowerShell](#)

After you've created the web app, use one of the available deployment mechanisms to deploy your application. For more information, see [Deploy files to App Service](#).

Migrate JVM runtime options

If your application requires specific runtime options, use the most appropriate mechanism to specify them. For more information, see the [Set Java runtime options](#) section of [Configure a Java app for Azure App Service](#).

Migrate externalized parameters

If you need to use external parameters, you'll need to set them as app settings. For more information, see [Configure app settings](#).

Migrate startup scripts

If the original application used a custom startup script, you'll need to migrate it to a Bash script. For more information, see [Customize application server configuration](#).

Populate secrets

Use Application Settings to store any secrets specific to your application. If you intend to use the same secret or secrets among multiple applications, or you require fine-grained access policies and audit capabilities, use Azure Key Vault references instead. For more information, see the [Use KeyVault References](#) section of [Configure a Java app for Azure App Service](#).

Configure Custom Domain and SSL

If your application will be visible on a custom domain, you'll need to map your web application to it. For more information, see [Tutorial: Map an existing custom DNS name to Azure App Service](#).

You'll then need to bind the TLS/SSL certificate for that domain to your App Service Web App. For more information, see [Secure a custom DNS name with a TLS/SSL binding in Azure App Service](#).

Migrate data sources, libraries, and JNDI resources

To migrate data sources, follow the steps in the [Configure data sources](#) section of [Configure a Java app for Azure App Service](#).

Migrate any additional server-level classpath dependencies by following the instructions in the [JBoss EAP](#) section of [Configure a Java app for Azure App Service](#).

Migrate any additional shared server-level JNDI resources. For more information, see the [JBoss EAP](#) section of [Configure a Java app for Azure App Service](#).

Migrate JCA connectors and JAAS modules

Migrate any JCA connectors and JAAS modules by following the instructions at [Install modules and dependencies](#).

Note

If you're following the recommended architecture of one WAR per application, consider migrating server-level classpath libraries and JNDI resources into your application. Doing so will significantly simplify component governance and change management. If you want to deploy more than one WAR per application, you should review one of our companion guides mentioned at the beginning of this guide.

Migrate scheduled jobs

At a minimum, you should move your scheduled jobs to an Azure VM so they're no longer part of your application. Alternately, you can opt to modernize them into event driven Java using Azure services such as Azure Functions, SQL Database, and Event Hubs.

Restart and smoke-test

Finally, you'll need to restart your Web App to apply all configuration changes. Upon completion of the restart, verify that your application is running correctly.

Post-migration

Now that you've migrated your application to Azure App Service, you should verify that it works as you expect. Once you've done that, we have some recommendations for you that can make your application more cloud-native.

Recommendations

- If you opted to use the `/home` directory for file storage, consider replacing it with Azure Storage. For more information, see [Mount Azure Storage as a local share in a custom container in App Service](#).
- If you have configuration in the `/home` directory that contains connection strings, SSL keys, and other secret information, consider using a combination of Azure Key Vault and parameter injection with application settings where possible. For more information, see [Use Key Vault references for App Service and Azure Functions](#) and [Configure an App Service app](#).
- Consider using deployment slots for reliable deployments with zero downtime. For more information, see [Set up staging environments in Azure App Service](#).
- Design and implement a DevOps strategy. In order to maintain reliability while increasing your development velocity, consider automating deployments and testing with Azure Pipelines. For more information, see [Build & deploy to Java web app](#). If you're using deployment slots, you can automate deployment to a slot and the subsequent slot swap. For more information, see the [Example: Deploy to a slot](#) section of [Deploy to App Service using Azure Pipelines](#).
- Design and implement a business continuity and disaster recovery strategy. For mission-critical applications, consider a multi-region deployment architecture. For more information, see [Highly available multi-region web application](#).

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

Migrate WebSphere applications to JBoss EAP on Azure App Service

Article • 09/20/2024

This guide describes what you should be aware of when you want to migrate an existing WebSphere application to run on Azure App Service using JBoss EAP.

Pre-migration

To ensure a successful migration, before you start, complete the assessment and inventory steps described in the following sections.

Inventory server capacity

Document the hardware (memory, CPU, disk) of the current production server(s) and the average and peak request counts and resource utilization. You'll need this information regardless of the migration path you choose. It's useful, for example, to help guide selection of the App Service Plan.

The [list of available App Service Plan tiers](#) shows the memory, CPU cores, storage, and pricing information. Note that JBoss EAP on App Service is only available on the **Premium V3** and **Isolated V2** App Service Plan tiers.

Inventory all secrets

Check all properties and configuration files on the production server or servers for any secrets and passwords. Be sure to check `ibm-web-bnd.xml` in your WARs. Configuration files that contain passwords or credentials may also be found inside your application. These files may include, for Spring Boot applications, the `application.properties` or `application.yml` files.

Inventory all certificates

Document all the certificates used for public SSL endpoints. You can view all certificates on the production server(s) by running the following command:

Bash

```
keytool -list -v -keystore <path to keystore>
```

Validate that the supported Java version works correctly

JBoss EAP on Azure App Service supports Java 8 and 11. Therefore, you'll need to validate that your application is able to run correctly using that supported version. This validation is especially important if your current server is using a supported JDK (such as Oracle JDK or IBM OpenJ9).

To obtain your current Java version, sign in to your production server and run the following command:

```
Bash
```

```
java -version
```

Inventory JNDI resources

Inventory all JNDI resources. Some resources, such as JMS message brokers, may require migration or reconfiguration.

Inside your application

Inspect the `WEB-INF/ibm-web-bnd.xml` file and/or the `WEB-INF/web.xml` file.

Determine whether databases are used

If your application uses any databases, you need to capture the following information:

- The datasource name.
- The connection pool configuration.
- The location of the JDBC driver JAR file.

Determine whether and how the file system is used

Any usage of the file system on the application server will require reconfiguration or, in rare cases, architectural changes. File system may be used by WebSphere shared modules or by your application code. You may identify some or all of the following scenarios.

Read-only static content

If your application currently serves static content, you'll need an alternate location for it. You may wish to consider moving static content to Azure Blob Storage and adding Azure CDN for lightning-fast downloads globally. For more information, see [Static website hosting in Azure Storage](#) and [Quickstart: Integrate an Azure storage account with Azure CDN](#).

Dynamically published static content

If your application allows for static content that is uploaded/produced by your application but is immutable after its creation, you can use Azure Blob Storage and Azure CDN as described above, with an Azure Function to handle uploads and CDN refresh. We've provided a sample implementation for your use at [Uploading and CDN-preloading static content with Azure Functions](#).

Dynamic or internal content

For files that are frequently written and read by your application (such as temporary data files), or static files that are visible only to your application, you can mount Azure Storage into your App Service file system. For more information, see [Mount Azure Storage as a local share in App Service](#).

Determine whether your application relies on scheduled jobs

Scheduled jobs, such as Quartz Scheduler tasks or Unix cron jobs, should NOT be used with Azure App Service. Azure App Service will not prevent you from deploying an application containing scheduled tasks internally. However, if your application is scaled out, the same scheduled job may run more than once per scheduled period. This situation can lead to unintended consequences.

To execute scheduled jobs on Azure, consider using Azure Functions with a Timer Trigger. For more information, see [Timer trigger for Azure Functions](#). You don't need to migrate the job code itself into a function. The function can simply invoke a URL in your application to trigger the job.

Note

To prevent malicious use, you'll likely need to ensure that the job invocation endpoint requires credentials. In this case, the trigger function will need to provide the credentials.

Determine whether a connection to on-premises is needed

If your application needs to access any of your on-premises services, you'll need to provision one of Azure's connectivity services. For more information, see [Connect an on-premises network to Azure](#). Alternatively, you'll need to refactor your application to use publicly available APIs that your on-premises resources expose.

Determine whether Java Message Service (JMS) Queues or Topics are in use

If your application is using JMS Queues or Topics, you'll need to migrate them to an externally hosted JMS server. Azure Service Bus and the Advanced Message Queuing Protocol (AMQP) can be a great migration strategy for those using JMS. For more information, see [Use Java Message Service 1.1 with Azure Service Bus standard and AMQP 1.0](#).

If JMS persistent stores have been configured, you must capture their configuration and apply it after the migration.

Determine whether your application uses WebSphere-specific APIs

If your application uses WebSphere-specific APIs, you'll need to refactor your application to NOT use them. The [Red Hat Migration Toolkit for Apps](#) can assist with removing and refactoring these dependencies.

Determine whether your application uses Entity Beans or EJB 2.x-style CMP Beans

If your application uses Entity Beans or EJB 2.x style CMP beans, you'll need to refactor your application to remove these dependencies.

Determine whether the JavaEE Application Client feature is used

If you have client applications that connect to your (server) application using the JavaEE Application Client feature, you'll need to refactor both your client applications and your (server) application to use HTTP APIs.

Determine whether your application contains OS-specific code

If your application contains any code with dependencies on the host OS, then you need to refactor it to remove those dependencies. For example, you may need to replace any use of `/` or `\` in file system paths with [File.Separator](#) or [Paths.get](#) if your application is running on Windows.

Determine whether EJB timers are in use

If your application uses EJB timers, you'll need to validate that the EJB timer code can be triggered by each JBoss EAP instance independently. This validation is needed because when your App Service is scaled our horizontally, each EJB timer will be triggered on its own JBoss EAP instance.

Determine whether JCA connectors are in use

If your application uses JCA connectors, you'll need to validate that the JCA connector can be used on JBoss EAP. If the JCA implementation is tied to WebSphere, you'll need to refactor your application remove the dependency on the JCA connector. If the JCA connector can be used, then you'll need to add the JARs to the server classpath. You'll also need to put the necessary configuration files in the correct location in the JBoss EAP server directories for it to be available.

Determine whether JAAS is in use

If your application uses JAAS, you'll need to capture how JAAS is configured. If it's using a database, you can convert it to a JAAS domain on JBoss EAP. If it's a custom implementation, you'll need to validate that it can be used on JBoss EAP.

Determine whether your application uses a Resource Adapter

If your application needs a Resource Adapter (RA), it needs to be compatible with JBoss EAP. Determine whether the RA works fine on a standalone instance of JBoss EAP by deploying it to the server and properly configuring it. If the RA works properly, you'll need to add the JARs to the server classpath of the App Service instance and put the necessary configuration files in the correct location in the JBoss EAP server directories for it to be available.

Determine whether your application is composed of multiple WARs

If your application is composed of multiple WARs, you should treat each of those WARs as separate applications and go through this guide for each of them.

Determine whether your application is packaged as an EAR

If your application is packaged as an EAR file, be sure to examine the `application.xml` and `ibm-application-bnd.xml` files and capture their configurations.

Identify all outside processes and daemons running on the production servers

If you have any processes running outside the application server, such as monitoring daemons, you'll need to eliminate them or migrate them elsewhere.

Migration

Red Hat Migration Toolkit for Apps

The [Red Hat Migration Toolkit for Applications](#) is a free extension for Visual Studio Code. This extension analyzes your application code and configuration to provide recommendations for migrating your Jakarta EE applications to JBoss EAP from other app servers, such as removing dependencies on proprietary APIs. The extension will also provide recommendations if you're migrating to the cloud from on-premises. For more information, see [Migration Toolkit for Applications overview](#).

The contents of this guide will help you address the other components of the migration journey, such as choosing the correct App Service Plan type, externalizing your session state, and using Azure to manage your EAP instances instead of the JBoss Management interface.

Provision an App Service plan

From the [list of available service plans](#), select the plan whose specifications meet or exceed the specifications of the current production hardware.

Note

If you plan to run staging/canary deployments or use [deployment slots](#), the App Service plan must include that additional capacity. We recommend using Premium or higher plans for Java applications.

[Create that app service plan.](#)

Create and deploy web app(s)

You'll need to create a Web App on your App Service Plan for every WAR file deployed to your JBoss EAP server.

Note

While it's possible to deploy multiple WAR files to a single web app, this is highly undesirable. Deploying multiple WAR files to a single web app prevents each application from scaling according to its own usage demands. It also adds complexity to subsequent deployment pipelines. If multiple applications need to be available on a single URL, consider using a routing solution such as [Azure Application Gateway](#).

Maven applications

If your application is built from a Maven POM file, use the Webapp plugin for Maven to create the Web App and deploy your application. For more information, see the [Configure the Maven plugin](#) section of [Quickstart: Create a Java app on Azure App Service](#).

Non-Maven applications

If you can't use the Maven plugin, you'll need to provision the Web App through other mechanisms, such as:

- [Azure portal ↗](#)
- [Azure CLI](#)
- [Azure PowerShell](#)

After you've created the web app, use one of the available deployment mechanisms to deploy your application. For more information, see [Deploy files to App Service](#).

Migrate JVM runtime options

If your application requires specific runtime options, use the most appropriate mechanism to specify them. For more information, see the [Set Java runtime options](#) section of [Deploy and configure a Tomcat, JBoss, or Java SE app in Azure App Service](#).

Populate secrets

Use Application Settings to store any secrets specific to your application. If you intend to use the same secret or secrets among multiple applications, or you require fine-grained access policies and audit capabilities, use Azure Key Vault references instead. For more information, see [Use Key Vault references as app settings in Azure App Service and Azure Functions](#).

Configure custom domain and SSL

If your application will be visible on a custom domain, you'll need to map your web application to it. For more information, see [Tutorial: Map an existing custom DNS name to Azure App Service](#).

You'll then need to bind the TLS/SSL certificate for that domain to your App Service Web App. For more information, see [Secure a custom DNS name with a TLS/SSL binding in Azure App Service](#).

Migrate data sources, libraries, and JNDI resources

To migrate data sources, follow the steps in the [Configure data sources for a Tomcat, JBoss, or Java SE app in Azure App Service](#).

Migrate any additional server-level classpath dependencies. For more information, see [Configure data sources for a Tomcat, JBoss, or Java SE app in Azure App Service](#).

Migrate any additional shared server-level JNDI resources. For more information, see [Configure data sources for a Tomcat, JBoss, or Java SE app in Azure App Service](#).

Note

If you're following the recommended architecture of one WAR per application, consider migrating server-level classpath libraries and JNDI resources into your application. Doing so will significantly simplify component governance and change management. If you want to deploy more than one WAR per application, you

should review one of our companion guides mentioned at the beginning of this guide.

Migrate scheduled jobs

At a minimum, you should move your scheduled jobs to an Azure VM so they're no longer part of your application. Alternately, you can opt to modernize them into event driven Java using Azure services such as Azure Functions, SQL Database, and Event Hubs.

Restart and smoke-test

Finally, you'll need to restart your Web App to apply all configuration changes. Upon completion of the restart, verify that your application is running correctly.

Post-migration

Now that you've migrated your application to Azure App Service, you should verify that it works as you expect. Once you've done that, we have some recommendations for you that can make your application more Cloud native.

Recommendations

- If you opted to use the `/home` directory for file storage, consider replacing it with Azure Storage. For more information, see [Mount Azure Storage as a local share in a custom container in App Service](#).
- If you have configuration in the `/home` directory that contains connection strings, SSL keys, and other secret information, consider using a combination of Azure Key Vault and parameter injection with application settings where possible. For more information, see [Use Key Vault references for App Service and Azure Functions](#) and [Configure an App Service app](#).
- Consider using deployment slots for reliable deployments with zero downtime. For more information, see [Set up staging environments in Azure App Service](#).
- Design and implement a DevOps strategy. In order to maintain reliability while increasing your development velocity, consider automating deployments and testing with Azure Pipelines. For more information, see [Build & deploy to Java web app](#). If you're using deployment slots, you can automate deployment to a slot and

the subsequent slot swap. For more information, see the [Example: Deploy to a slot](#) section of [Deploy to App Service using Azure Pipelines](#).

- Design and implement a business continuity and disaster recovery strategy. For mission-critical applications, consider a multi-region deployment architecture. For more information, see [Highly available multi-region web application](#).
-

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

Basic web application

Azure App Service Azure Monitor Azure SQL Database

This article provides a basic architecture intended for learning about running web applications on Azure App Service in a single region.

Important

This architecture isn't meant to be used for production applications. It's intended to be an introductory architecture you can use for learning and proof of concept (POC) purposes. When designing your production Azure App Service application, see the [Baseline highly available zone-redundant web application](#).

Important



The guidance is backed by an [example implementation](#) which showcases this basic App Service implementation on Azure. This implementation can be used as a basis for your POC to experience working with Azure App Service.

Architecture

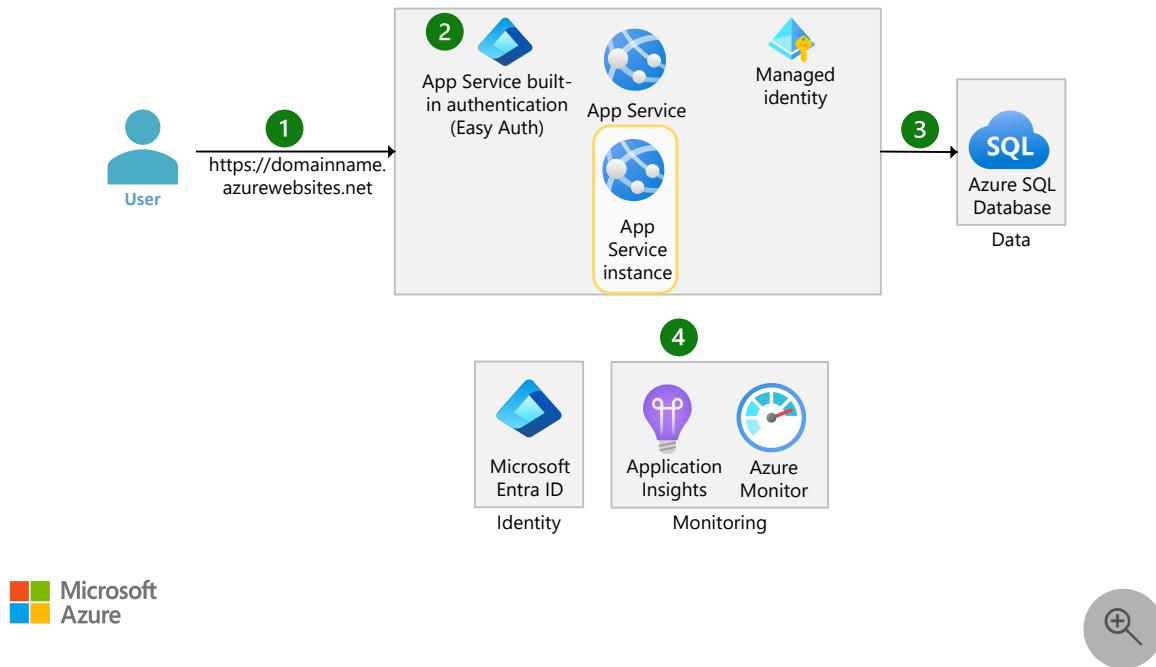


Figure 1: Basic Azure App Service architecture

Download a [Visio file](#) of this architecture.

Workflow

1. A user issues an HTTPS request to the App Service's default domain on azurewebsites.net. This domain automatically points to your App Service's built-in public IP. The TLS connection is established from the client directly to app service. The certificate is managed completely by Azure.
2. Easy Auth, a feature of Azure App Service, ensures that the user accessing the site is authenticated with Microsoft Entra ID.
3. Your application code deployed to App Service handles the request. For example, that code might connect to an Azure SQL Database instance, using a connection string configured in the App Service configured as an app setting.
4. The information about original request to App Service and the call to Azure SQL Database are logged in Application Insights.

Components

- **Microsoft Entra ID** is a cloud-based identity and access management service. It provides a single identity control plane to manage permissions and roles for users

accessing your web application. It integrates with App Service and simplifies authentication and authorization for web apps.

- [App Service](#) is a fully managed platform for building, deploying, and scaling web applications.
- [Azure Monitor](#) is a monitoring service that collects, analyzes, and acts on telemetry data across your deployment.
- [Azure SQL Database](#) is a managed relational database service for relational data.

Considerations

These considerations implement the pillars of the Azure Well-Architected Framework, which is a set of guiding tenets that can be used to improve the quality of a workload. For more information, see [Microsoft Azure Well-Architected Framework](#).

The [components](#) listed in this architecture link to Azure Well-Architected service guides. Service guides detail recommendations and considerations for specific services. This section extends that guidance by highlighting key Azure Well-Architected Framework recommendations and considerations that apply to this architecture. For more information, see [Microsoft Azure Well-Architected Framework](#).

This *basic architecture* isn't intended for production deployments. The architecture favors simplicity and cost efficiency over functionality to allow you to evaluate and learn Azure App Service. The following sections outline some deficiencies of this basic architecture, along with recommendations and considerations.

Reliability

Reliability ensures your application can meet the commitments you make to your customers. For more information, see [Design review checklist for Reliability](#).

Because this architecture isn't designed for production deployments, the following outlines some of the critical reliability features that are omitted in this architecture:

- The App Service Plan is configured for the `Standard` tier, which doesn't have [Azure availability zone](#) support. The App Service becomes unavailable in the event of any issue with the instance, the rack, or the datacenter hosting the instance.
- The Azure SQL Database is configured for the `Basic` tier, which doesn't support [zone-redundancy](#). This means that data isn't replicated across Azure availability zones, risking loss of committed data in the event of an outage.
- Deployments to this architecture might result in downtime with application deployments, as most deployment techniques require all running instances to be

restarted. Users may experience 503 errors during this process. This deployment downtime is addressed in the baseline architecture through [deployment slots](#). Careful application design, schema management, and application configuration handling are necessary to support concurrent slot deployment. Use this POC to design and validate your slot-based production deployment approach.

- Autoscaling isn't enabled in this basic architecture. To prevent reliability issues due to lack of available compute resources, you'd need to overprovision to always run with enough compute to handle max concurrent capacity.

See how to overcome these reliability concerns in the [reliability section in the Baseline highly available zone-redundant web application](#).

If this workload will eventually require a multi-region active-active or active-passive architecture, see the following resources:

- [Multi-region App Service app approaches for disaster recovery](#) for guidance on deploying your App Service-hosted workload across multiple regions.
- [Highly available multi-region web application](#) for a reference architecture that follows an active-passive approach.

Security

Security provides assurances against deliberate attacks and the abuse of your valuable data and systems. For more information, see [Design review checklist for Security](#).

Because this architecture isn't designed for production deployments, the following outlines some of the critical security features that were omitted in this architecture, along with other reliability recommendations and considerations:

- This basic architecture doesn't implement network privacy. The data and management planes for the resources, such as the Azure App Service and Azure SQL Server, are reachable over the public internet. Omitting private networking significantly increases the attack surface of your architecture. To see how implementing private networking ensures the following security features, see the [networking section of the Baseline highly available zone-redundant web application](#):
 - A single secure entry point for client traffic
 - Network traffic is filtered both at the packet level and at the DDoS level.
 - Data exfiltration is minimized by keeping traffic in Azure by using Private Link
 - Network resources are logically grouped and isolated from each other through network segmentation.

- This basic architecture doesn't include a deployment of the [Azure Web Application Firewall](#). The web application isn't protected against common exploits and vulnerabilities. See the [baseline implementation](#) to see how the Web Application Firewall can be implemented with Azure Application Gateway in an Azure App Services architecture.
- This basic architecture stores secrets such as the Azure SQL Server connection string in App Settings. While app settings are encrypted, when moving to production, consider storing secrets in Azure Key Vault for increased governance. An even better solution is to use managed identity for authentication and not have secrets stored in the connection string.
- Leaving remote debugging and Kudu endpoints enabled while in development or the proof of concept phase is fine. When you move to production, you should disable unnecessary control plane, deployment, or remote access.
- Leaving local authentication methods for FTP and SCM site deployments enabled is fine while in the development or proof of concept phase. When you move to production, you should disable local authentication to those endpoints.
- You don't need to enable [Microsoft Defender for App Service](#) in the proof of concept phase. When moving to production, you should enable Defender for App Service to generate security recommendations you should implement to increase your security posture and to detect multiple threats to your App Service.
- Azure App Service includes an SSL endpoint on a subdomain of `azurewebsites.net` at no extra cost. HTTP requests are redirected to the HTTPS endpoint by default. For production deployments, you'll typically use a custom domain associated with application gateway or API management in front of your App Service deployment.
- Use the [integrated authentication mechanism for App Service \("EasyAuth"\)](#). EasyAuth simplifies the process of integrating identity providers into your web app. It handles authentication outside your web app, so you don't have to make significant code changes.
- Use managed identity for workload identities. Managed identity eliminates the need for developers to manage authentication credentials. The basic architecture authenticates to SQL Server via password in a connection string. Consider using [managed identity to authenticate to Azure SQL Server](#).

For some other security considerations, see [Secure an app in Azure App Service](#).

Cost Optimization

Cost Optimization is about looking at ways to reduce unnecessary expenses and improve operational efficiencies. For more information, see [Design review checklist for Cost Optimization](#).

This architecture optimizes for cost through the many trade-offs against the other pillars of the Well-Architected Framework specifically to align with the learning and proof-of-concept goals of this architecture. The cost savings compared to a more production-ready architecture, such as the [Baseline highly available zone-redundant web application](#), mainly result from the following choices.

- Single App Service instance, with no autoscaling enabled
- Standard pricing tier for Azure App Service
- No custom TLS certificate or static IP
- No web application firewall (WAF)
- No dedicated storage account for application deployment
- Basic pricing tier for Azure SQL Database, with no backup retention policies
- No Microsoft Defender for Cloud components
- No network traffic egress control through a firewall
- No private endpoints
- Minimal logs and log retention period in Log Analytics

To view the estimated cost of this architecture, see the [Pricing calculator estimate](#) using this architecture's components. The cost of this architecture can usually be further reduced by using an [Azure Dev/Test subscription](#), which would be an ideal subscription type for proof of concepts like this.

Operational Excellence

Operational Excellence covers the operations processes that deploy an application and keep it running in production. For more information, see [Design review checklist for Operational Excellence](#).

The following sections provide guidance around configuration, monitoring, and deployment of your App Service application.

App configurations

Because the basic architecture isn't intended for production, it uses [App Service configuration](#) to store configuration values and secrets. Storing secrets in App Service configuration is fine for the PoC phase. You aren't using real secrets and don't require secrets governance that production workloads require.

The following are configuration recommendations and considerations:

- Start by using App Service configuration to store configuration values and connection strings in proof of concept deployments. App settings and connection strings are encrypted and decrypted just before being injected into your app when it starts.
- When you move into production phase, store your secrets in Azure Key Vault. The use of Azure Key Vault improves the governance of secrets in two ways:
 - Externalizing your storage of secrets to Azure Key Vault allows you to centralize your storage of secrets. You have one place to manage secrets.
 - Using Azure Key Vault, you're able to log every interaction with secrets, including every time a secret is accessed.
- When you move into production, you can maintain your use of both Azure Key Vault and App Service configuration by [using Key Vault references](#).

Containers

The basic architecture can be used to deploy supported code directly to Windows or Linux instances. Alternatively, App Service is also a container hosting platform to run your containerized web application. App Service offers various built-in containers. If you're using custom or multi-container apps to further fine-tune your runtime environment or to support a code language not natively supported, you'll need to introduce a container registry.

Control plane

During the POC phase, get comfortable with Azure App Service's control plane as exposed through the Kudu service. This service exposes common deployment APIs, such as ZIP deployments, exposes raw logs and environment variables.

If using containers, be sure to understand Kudu's ability to Open an SSH session to a container to support advanced debugging capabilities.

Diagnostics and monitoring

During the proof of concept phase, it's important to get an understanding of what logs and metrics are available to be captured. The following are recommendations and considerations for monitoring in the proof of concept phase:

- Enable [diagnostics logging](#) for all items log sources. Configuring the use of all diagnostic settings helps you understand what logs and metrics are provided for

you out of the box and any gaps you'll need to close using a logging framework in your application code. When you move to production, you should eliminate log sources that aren't adding value and are adding noise and cost to your workload's log sink.

- Configure logging to use Azure Log Analytics. Azure Log Analytics provides you with a scalable platform to centralize logging that is easy to query.
- Use [Application Insights](#) or another Application Performance Management (APM) tool to emit telemetry and logs to monitor application performance.

Deployment

The following lists guidance around deploying your App Service application.

- Follow the guidance in [CI/CD for Azure Web Apps with Azure Pipelines](#) to automate the deployment of your application. Start building your deployment logic in the PoC phase. Implementing CI/CD early in the development process allows you to quickly and safely iterate on your application as you move toward production.
- Use [ARM templates](#) to deploy Azure resources and their dependencies. It's important to start this process in the PoC phase. As you move toward production, you want the ability to automatically deploy your infrastructure.
- Use different ARM Templates and integrate them with Azure DevOps services. This setup lets you create different environments. For example, you can replicate production-like scenarios or load testing environments only when needed and save on cost.

For more information, see the DevOps section in [Azure Well-Architected Framework](#).

Performance Efficiency

Performance Efficiency is the ability of your workload to meet the demands placed on it by users in an efficient manner. For more information, see [Design review checklist for Performance Efficiency](#).

Because this architecture isn't designed for production deployments, the following outlines some of the critical performance efficiency features that were omitted in this architecture, along with other recommendations and considerations.

An outcome of your proof of concept should be SKU selection that you estimate is suitable for your workload. Your workload should be designed to efficiently meet demand through horizontal scaling by adjusting the number of compute instances

deployed in the App Service Plan. Do not design the system to depend on changing the compute SKU to align with demand.

- The App Service in this basic architecture doesn't have automatic scaling implemented. The service doesn't dynamically scale out or in to efficiently keep aligned with demand.
 - The Standard tier does support [auto scale settings](#) to allow you to configure rule-based autoscaling. Part of your POC process should be to arrive at efficient autoscaling settings based on your application code's resource needs and expected usage characteristics.
 - For production deployments, consider Premium tiers that support [automatic autoscaling](#) where the platform automatically handles scaling decisions.
- Follow the [guidance to scale up individual databases with no application downtime](#) if you need a higher service tier or performance level for SQL Database.

Deploy this scenario

The guidance is backed by an [example implementation](#) ↗ that showcases a basic App Service implementation on Azure.

Next steps

[Baseline highly available zone-redundant web application](#)

Related resources

- [Baseline zone-redundant web application](#)
- [Highly available multi-region web application](#)
- [Multi-region App Service app approaches for disaster recovery](#)

Product documentation:

- [App Service overview](#)
- [Azure Monitor overview](#)
- [Azure App Service plan overview](#)
- [Overview of Log Analytics in Azure Monitor](#)
- [What is Microsoft Entra ID?](#)
- [What is Azure SQL Database?](#)

Microsoft Learn modules:

- Configure and manage Azure Monitor
 - Configure Microsoft Entra ID
 - Configure Azure Monitor
 - Deploy and configure servers, instances, and databases for Azure SQL
 - Explore Azure App Service
 - Host a web application with Azure App Service
 - Host your domain on Azure DNS
 - Implement Azure Key Vault
 - Manage users and groups in Microsoft Entra ID
-

Feedback

Was this page helpful?

 Yes

 No

Use existing runbooks and modules

Article • 09/09/2024

Rather than creating your own runbooks and modules in Azure Automation, you can access scenarios that have already been built by Microsoft and the community. You can get Azure-related PowerShell and Python runbooks from the Runbook Gallery in the Azure portal, and [modules](#) and [runbooks](#) (which may or may not be specific to Azure) from the PowerShell Gallery. You can also contribute to the community by sharing [scenarios that you develop](#).

ⓘ Note

The TechNet Script Center is retiring. All of the runbooks from Script Center in the Runbook gallery have been moved to our [Automation GitHub organization](#). For more information, see [Azure Automation Runbooks moving to GitHub](#).

Import runbooks from GitHub with the Azure portal

ⓘ Note

- The **Browse gallery** option in the Azure portal has an enhanced user experience.
- In **Process Automation > Runbook** blade, you can import runbooks either by **Import a runbook** or **Browse gallery** option and the **Runbooks** page displays two new columns - **Runtime version** and **Runbook type**.

1. In the Azure portal, open your Automation account.
2. Select **Runbooks** blade under **Process Automation**.
3. Click **Import a runbook** in the **Runbooks** page.

The screenshot shows the Azure Automation Runbooks list page. The left sidebar includes sections like Overview, Activity log, Access control (IAM), Tags, Diagnose and solve problems, Configuration Management, Update management, Process Automation (with 'Runbooks' highlighted by a red box), Jobs, Hybrid worker groups, and Watcher tasks. The main content area displays a table of runbooks with columns for Name, Authoring status, Runbook type, Runtime version, Last modified, and Tags. A search bar and filter buttons for Runbook type (All) and Authoring Status (All) are at the top. A red box highlights the 'Import a runbook' button.

Name	Authoring status	Runbook type	Runtime version	Last modified	Tags
TestAccessTokenRunbook	Published	Python	3.10 (preview)	10/17/2022, 5:29 PM	
AzureAutomationTutorialWithiden...	Published	PowerShell	5.1	10/17/2022, 4:17 PM	
AzureAutomationTutorialWithiden...	Published	Graphical PowerShell	5.1	10/17/2022, 4:17 PM	
test_abc	New	PowerShell	7.1 (preview)	10/25/2022, 5:54 PM	
test-xyz	Published	PowerShell	7.2 (preview)	10/17/2022, 4:27 PM	
python310	Published	Python	3.10 (preview)	10/17/2022, 4:26 PM	

4. In the **Import a runbook** page, you can either import a file stored on your local machine or from GitHub using **Browse for file** or **Browse from gallery** respectively.
5. Select the file.
6. Enter the **Name**, **Runtime version**, and **Description**.
7. Click **Import**.

Home > Automation Accounts > inframon-auto-2 >

Import a runbook

Upload a runbook file * ⓘ

Browse for file
 Browse from gallery

Runbook file ⓘ

Scheduled Virtual Machine Shutdown/Startup
[Change](#)

Name * ⓘ test-1

Runbook type ⓘ PowerShell

Runtime version * ⓘ 7.1 (preview)

Description

Automates the scheduled startup and shutdown of Azure virtual machines. Schedules are implemented by tagging VMs or resource groups with individual simple schedules. Schedules can define multiple

Info During runbook execution, PowerShell modules targeting 7.1 runtime version will be used. Please make sure the required PowerShell modules are present in 7.1 runtime version.

Import ⌂ Cancel

8. Alternatively, Select **Browse Gallery** in the Runbooks page to browse the available runbooks.

Dashboard > previewAaTara | Runbooks ⓘ ...

Automation Account

Search

+ Create a runbook Import a runbook **Browse gallery** Learn more Refresh

Showing 1 to 6 of 6 records.

Name	Authoring status	Runbook type	Runtime version	Last modified	Tags
TestAccessTokenRunbook	Published	Python	3.10 (preview)	10/17/2022, 5:29 PM	
AzureAutomationTutorialWithiden...	Published	PowerShell	5.1	10/17/2022, 4:17 PM	
AzureAutomationTutorialWithiden...	Published	Graphical PowerShell	5.1	10/17/2022, 4:17 PM	
test_abc	New	PowerShell	7.1 (preview)	10/25/2022, 5:54 PM	
test-xyz	Published	PowerShell	7.2 (preview)	10/17/2022, 4:27 PM	
python310	Published	Python	3.10 (preview)	10/17/2022, 4:26 PM	

Overview Activity log Access control (IAM) Tags Diagnose and solve problems Configuration Management Inventory Change tracking State configuration (DSC) Update management Update management Process Automation **Runbooks** Jobs Hybrid worker groups Watcher tasks Shared Resources

9. You can use the filters above the list to narrow the display by publisher, type, and sort. Locate the gallery item you want and select it to view its details.

The screenshot shows the 'Browse Gallery' page of the Azure Runbook Gallery. At the top, there's a message about TechNet Script Center retiring. Below that are search and filter options: 'Source: GitHub', 'Type: All', 'Publisher: All', and 'Sort: Popularity'. A large list of runbooks is displayed, each with a thumbnail icon, name, description, creator information, and download statistics. The runbooks listed are:

- Start Azure V2 VMs**: Graphical Runbook. Created by Azure Automation Product Team. Ratings: 4.5 of 5. 160,206 downloads. Last updated: 10/23/2016.
- Stop-Start-AzureVM (Scheduled VM Shutdown/Startup)**: PowerShell Workflow Runbook. Created by Pradeeban Raja. Ratings: 5 of 5. 72,400 downloads. Last updated: 8/21/2018.
- Stop Azure V2 VMs**: Graphical Runbook. Created by Azure Automation Product Team. Ratings: 4.3 of 5. 106,652 downloads. Last updated: 10/23/2016.
- Shutdown/Start VMs by tag**: PowerShell Workflow Runbook. Created by Gisela Torres. Ratings: 4.5 of 5. 23,883 downloads. Last updated: 7/19/2016.
- Create Automation Windows HybridWorker**: PowerShell Runbook. Created by Kranti Kumar.

10. Click **Select** to select a chosen runbook.

11. In the **Import a runbook** page, enter the **Name** and select the **Runtime versions**.

12. The **Runbook type** and **Description** are auto populated.

13. Click **Import**.

The screenshot shows the 'Import a runbook' dialog box. It has the following fields:

- Upload a runbook file**: Radio buttons for 'Browse for file' (unchecked) and 'Browse from gallery' (checked).
- Runbook file**: A dropdown menu showing 'Scheduled Virtual Machine Shutdown/Startup' with a 'Change' link.
- Name**: An input field containing 'test-1'.
- Runbook type**: A dropdown menu showing 'PowerShell'.
- Runtime version**: A dropdown menu showing '7.1 (preview)'.
- Description**: A rich text area containing the text: 'Automates the scheduled startup and shutdown of Azure virtual machines. Schedules are implemented by tagging VMs or resource groups with individual simple schedules. Schedules can define multiple'.
- Note**: A callout box stating: 'During runbook execution, PowerShell modules targeting 7.1 runtime version will be used. Please make sure the required PowerShell modules are present in 7.1 runtime version.'
- Buttons**: 'Import' (highlighted with a red box) and 'Cancel'.

14. The runbook appears on the **Runbooks** tab for the Automation account.

Runbooks in the PowerShell Gallery

ⓘ Important

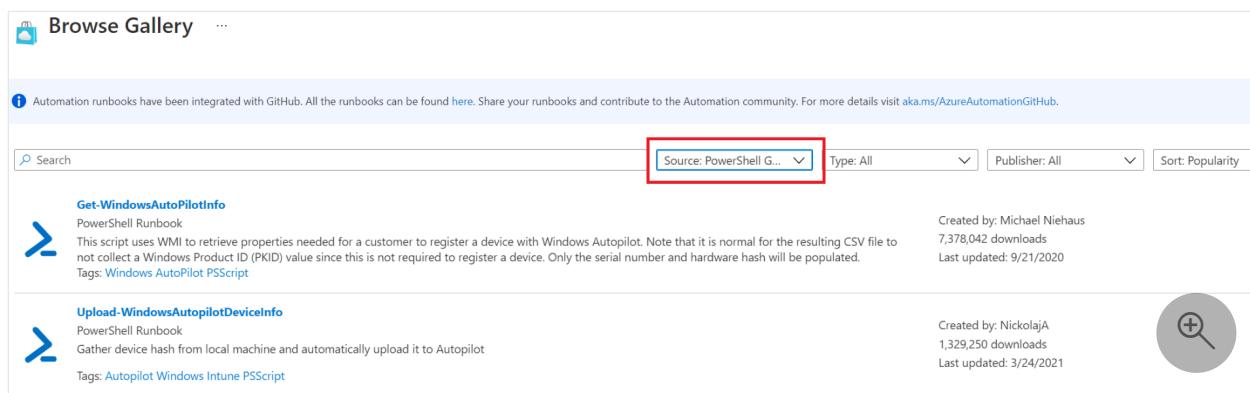
You should validate the contents of any runbooks that you get from the PowerShell Gallery. Use extreme caution in installing and running them in a production environment.

The [PowerShell Gallery](#) provides various runbooks from Microsoft and the community that you can import into Azure Automation. To use one, download a runbook from the gallery, or you can directly import runbooks from the gallery, or from your Automation account in the Azure portal.

ⓘ Note

Graphical runbooks are not supported in PowerShell Gallery.

You can only import directly from the PowerShell Gallery using the Azure portal. You cannot perform this function using PowerShell. The procedure is the same as shown in [Import runbooks from GitHub with the Azure portal](#), except that the **Source** will be **PowerShell Gallery**.



The screenshot shows the PowerShell Gallery interface. At the top, there's a navigation bar with 'Browse Gallery' and other options. Below it is a message about runbooks being integrated with GitHub. A search bar is followed by a dropdown menu labeled 'Source: PowerShell G...' which is highlighted with a red box. Other dropdowns show 'Type: All', 'Publisher: All', and 'Sort: Popularity'. The main area displays two runbook entries:

- Get-WindowsAutoPilotInfo**: PowerShell Runbook. Description: This script uses WMI to retrieve properties needed for a customer to register a device with Windows Autopilot. Note that it is normal for the resulting CSV file to not collect a Windows Product ID (PKID) value since this is not required to register a device. Only the serial number and hardware hash will be populated. Tags: Windows AutoPilot PSScript. Created by: Michael Niehaus, 7,378,042 downloads, Last updated: 9/21/2020.
- Upload-WindowsAutopilotDeviceInfo**: PowerShell Runbook. Description: Gather device hash from local machine and automatically upload it to Autopilot. Tags: Autopilot Windows Intune PSScript. Created by: NickolajA, 1,329,250 downloads, Last updated: 3/24/2021.

A magnifying glass icon is located in the bottom right corner of the search area.

Modules in the PowerShell Gallery

PowerShell modules contain cmdlets that you can use in your runbooks. Existing modules that you can install in Azure Automation are available in the [PowerShell Gallery](#). You can launch this gallery from the Azure portal and install the modules directly into Azure Automation, or you can manually download and install them.

You can also find modules to import in the Azure portal. They're listed for your Automation Account in the **Modules** under **Shared resources**.

Important

Do not include the keyword "AzureRm" in any script designed to be executed with the Az module. Inclusion of the keyword, even in a comment, may cause the AzureRm to load and then conflict with the Az module.

Common scenarios available in the PowerShell Gallery

The list below contains a few runbooks that support common scenarios. For a full list of runbooks created by the Azure Automation team, see [AzureAutomationTeam profile](#).

- [Update-ModulesInAutomationToLatestVersion](#) - Imports the latest version of all modules in an Automation account from PowerShell Gallery.
- [Enable-AzureDiagnostics](#) - Configures Azure Diagnostics and Log Analytics to receive Azure Automation logs containing job status and job streams.
- [Copy-ItemFromAzureVM](#) - Copies a remote file from a Windows Azure virtual machine.
- [Copy-ItemToAzureVM](#) - Copies a local file to an Azure virtual machine.

Contribute to the community

We strongly encourage you to contribute and help grow the Azure Automation community. Share the amazing runbooks you've built with the community. Your contributions will be appreciated!

Add a runbook to the GitHub Runbook gallery

You can add new PowerShell or Python runbooks to the Runbook gallery with this GitHub workflow.

1. Create a public repository on GitHub, and add the runbook and any other necessary files (like readme.md, description, and so on).
2. Add the topic `azureautomationrunbookgallery` to make sure the repository is discovered by our service, so it can be displayed in the Automation Runbook gallery.
3. If the runbook that you created is a PowerShell workflow, add the topic `PowerShellWorkflow`. If it's a Python 3 runbook, add `Python3`. No other specific

topics are required for Azure runbooks, but we encourage you to add other topics that can be used for categorization and search in the Runbook Gallery.

① Note

Check out existing runbooks in the gallery for things like formatting, headers, and existing tags that you might use (like `Azure Automation` or `Linux Azure Virtual Machines`).

To suggest changes to an existing runbook, file a pull request against it.

If you decide to clone and edit an existing runbook, best practice is to give it a different name. If you re-use the old name, it will show up twice in the Runbook gallery listing.

① Note

Please allow at least 12 hours for synchronization between GitHub and the Automation Runbook Gallery, for both updated and new runbooks.

Add a PowerShell runbook to the PowerShell gallery

Microsoft encourages you to add runbooks to the PowerShell Gallery that you think would be useful to other customers. The PowerShell Gallery accepts PowerShell modules and PowerShell scripts. You can add a runbook by [uploading it to the PowerShell Gallery](#).

Import a module from the Modules gallery in the Azure portal

1. In the Azure portal, open your Automation account.
2. Under **Shared Resources**, select **Modules**.
3. In **Modules** page, select **Browse gallery** to open the list of modules.

The screenshot shows the 'Browse Gallery' page of the PowerShell Gallery. At the top, there's a search bar and a dropdown menu set to 'Sort: Popularity'. Below the header, a list of modules is displayed in a table format. Each module entry includes a thumbnail icon, the module name, a brief description, and metadata such as 'Created by', 'Downloads', and 'Last updated'. A magnifying glass icon with a plus sign is located in the bottom right corner of the list area.

Module Name	Description	Created by	Downloads	Last updated
SpeculationControl	This module provides the ability to query the speculation control settings for the system.	PowerShellTeam,msftsecresponse	507736261	5/15/2019
PSWindowsUpdate	This module contains cmdlets to manage Windows Update Client.	MichalGajda	9182531	4/20/2020
DellBIOSProvider	The 'Dell Command PowerShell Provider' provides native configuration capability of Dell Optiplex, Latitude, Precision, XPS Notebook and Venue 11 systems within PowerShell.	dcpp.dell	85446343	9/28/2021
NetworkingDsc	DSC resources for configuring settings related to networking.	PowerShellTeam,gaelcolas,dsccommunity	77647429	10/16/2020
PackageManagement	PackageManagement (a.k.a. OneGet) is a new way to discover and install software packages from around the web. It is a manager or multiplexer of existing package managers (also called package providers) that unifies Windows package management with a single Windows PowerShell interface. With PackageManagement, you can do the following.	PowerShellTeam,alerickson,NateLehman,krishnayalavart	66918111	4/24/2020
AzureRM.profile	Microsoft Azure PowerShell - Profile credential management cmdlets for Azure Resource Manager	azure-sdk	63481568	3/23/2021
PowerShellGet	PowerShell module with commands for discovering, installing, updating and publishing the PowerShell artifacts like Modules, DSC Resources, Role Capabilities and Scripts.	PowerShellTeam,alerickson,anamnavi	54453566	9/22/2020

4. On the Browse gallery page, you can search by the following fields:

- Module Name
- Tags
- Author
- Cmdlet/DSC resource name

5. Locate a module that you're interested in and select it to view its details.

When you drill into a specific module, you can view more information. This information includes a link back to the PowerShell Gallery, any required dependencies, and all of the cmdlets or DSC resources that the module contains.

The screenshot shows the details page for the 'Posh-SSH' module. At the top, there's a back navigation link to 'Home > AzureAutomation > Browse Gallery >'. The module title is 'Posh-SSH' with a 'PowerShell Module' badge. A red box highlights the 'Import' button. Below the button, a description states: 'Provide SSH and SCP functionality for executing commands against remote hosts.' To the right, the 'Version' is listed as '2.3.0', 'Downloads' as '4,125,729', and 'Last updated' as '10/4/2020'. Below the description, there are links to 'Learn more', 'View in PowerShell Gallery', and 'Licensing Information (PowerShell Gallery Default)'. A section titled 'Content' follows, featuring a search bar and a table of cmdlets: Get-SCPFile, Get-SCPFolder, and Get-SCPItem.

Type	Name
Cmdlet	Get-SCPFile
Cmdlet	Get-SCPFolder
Cmdlet	Get-SCPItem

6. To install the module directly into Azure Automation, click **Import**.

7. On the Import pane, you can see the name of the module to import. If all the dependencies are installed, the **OK** button is activated. If you're missing dependencies, you need to import those dependencies before you can import this module.
8. On the Import pane, click **OK** to import the module. While Azure Automation imports a module to your account, it extracts metadata about the module and the cmdlets. This action might take a couple of minutes since each activity needs to be extracted.
9. You receive an initial notification that the module is being deployed and another notification when it has completed.
10. After the module is imported, you can see the available activities. You can use module resources in your runbooks and DSC resources.

 **Note**

Modules that only support PowerShell core are not supported in Azure Automation and are unable to be imported in the Azure portal, or deployed directly from the PowerShell Gallery.

Request a runbook or module

You can send requests to [User Voice](#). If you need help with writing a runbook or have a question about PowerShell, post a question to our [Microsoft Q&A question page](#).

Next steps

- To get started with PowerShell runbooks, see [Tutorial: Create a PowerShell runbook](#).
- To work with runbooks, see [Manage runbooks in Azure Automation](#).
- For more info on PowerShell scripting, see [PowerShell Docs](#).
- For a PowerShell cmdlet reference, see [Az.Automation](#).

Feedback

Was this page helpful?

 Yes

 No

How to prepare for an inbound IP address change

Article • 03/03/2023

If you received a notification that the inbound IP address of your Azure App Service app is changing, follow the instructions in this article.

Determine if you have to do anything

- Option 1: If your App Service app does not have a Custom Domain, no action is required.
- Option 2: If only a CNAME record (DNS record pointing to a URI) is configured in your Domain Registration Portal (third party DNS Provider or Azure DNS), no action is required.
- Option 3: If an A record (DNS record pointing directly to your IP address) is configured in your Domain Registration Portal (third party DNS Provider or Azure DNS), replace the existing IP address with the new one. You can find the new IP address by following the instructions in the next section.
- Option 4: If your application is behind a load balancer, IP Filter, or any other IP mechanism that requires your app's IP address, replace the existing IP address with the new one. You can find the new IP address by following the instructions in the next section.

Find the new inbound IP Address in the Azure portal

The new inbound IP address that is being given to your app is in the portal in the **Virtual IP address** field. Both this new IP address and the old one are connected to your app now, and later the old one will be disconnected.

1. Open the [Azure portal](#).
2. In the left-hand navigation menu, select **App Services**.
3. Select your App Service app from the list.
4. If the app is a function app, see [Function app inbound IP address](#).

5. Under the **Settings** header, click **Properties** in the left navigation, and find the section labeled **Virtual IP address**.

6. Copy the IP address and reconfigure your domain record or IP mechanism.

Next steps

This article explained how to prepare for an IP address change that was initiated by Azure. For more information about IP addresses in Azure App Service, see [Inbound and outbound IP addresses in Azure App Service](#).

How to prepare for an outbound IP address change

Article • 03/03/2023

If you received a notification that the outbound IP addresses of your Azure App Service app are changing, follow the instructions in this article.

Determine if you have to do anything

- Option 1: If your App Service app does not use IP filtering, an explicit inclusion list, or special handling of outbound traffic such as routing or firewall, no action is required.
- Option 2: If your app does have special handling for the outbound IP addresses (see examples below), add the new outbound IP addresses wherever the existing ones appear. Don't replace the existing IP addresses. You can find the new outbound IP addresses by following the instructions in the next section.

For example, an outbound IP address may be explicitly included in a firewall outside your app, or an external payment service may have an allowed list that contains the outbound IP address for your app. If your outbound address is configured in a list anywhere outside your app, that needs to change.

Find the outbound IP addresses in the Azure portal

The new outbound IP addresses are shown in the portal before they take effect. When Azure starts using the new ones, the old ones will no longer be used. Only one set at a time is used, so entries in inclusion lists must have both old and new IP addresses to prevent an outage when the switch happens.

1. Open the [Azure portal](#).
2. In the left-hand navigation menu, select **App Services**.
3. Select your App Service app from the list.
4. If the app is a function app, see [Function app outbound IP addresses](#).

5. Under the **Settings** header, click **Properties** in the left navigation, and find the section labeled **Outbound IP addresses**.
6. Copy the IP addresses, and add them to your special handling of outbound traffic such as a filter or allowed list. Don't delete the existing IP addresses in the list.

Next steps

This article explained how to prepare for an IP address change that was initiated by Azure. For more information about IP addresses in Azure App Service, see [Inbound and outbound IP addresses in Azure App Service](#).

How to prepare for a TLS/SSL IP address change

Article • 11/19/2024

If you received a notification that the TLS/SSL IP address of your Azure App Service app is changing, follow the instructions in this article to release existing TLS/SSL IP address and assign a new one.

ⓘ Note

Service Endpoint is not currently supported when enabling IP Based SSL on App Service TLS/SSL bindings.

Release TLS/SSL IP addresses and assign new ones

1. Open the [Azure portal](#).
2. In the left-hand navigation menu, select **App Services**.
3. Select your App Service app from the list.
4. Under the **Settings** header, click **Custom domain** in the left navigation.
5. In the **Custom domains** section, look for the domain with the Binding type of IP Based SSL. Click the three ellipses for that domain and select **Update binding**. In the editor that opens, choose **SNI SSL** for the TLS/SSL type and click **Update**.
6. In the **Custom domains** section, select the same host name record. In the editor that opens, this time choose **IP Based SSL** for the TLS/SSL type and click **Update**. When you see the operation success message, you've acquired a new IP address.
7. If an A record (DNS record pointing directly to your IP address) is configured in your Domain Registration Portal (third party DNS Provider or Azure DNS), replace the existing IP address with the newly generated one. You can find the new IP address by following the instructions in the next section.

Find the new SSL IP address in the Azure Portal

1. Wait a few minutes, and then open the [Azure portal](#).
2. In the left-hand navigation menu, select **App Services**.
3. Select your App Service app from the list.
4. Under the **Settings** header, click **Properties** in the left navigation, and find the section labeled **Virtual IP address**.
5. Copy the IP address and reconfigure your domain record or IP mechanism.

Next steps

This article explained how to prepare for an IP address change that was initiated by Azure. For more information about IP addresses in Azure App Service, see [Inbound and outbound IP addresses in Azure App Service](#).

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)