Tell us about your PDF experience.

App Service documentation

Azure App Service enables you to build and host web apps, mobile back ends, and RESTful APIs in the programming language of your choice without managing infrastructure. It offers 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

Python

Node.js

PHP

WordPress

Custom container

Java

Build a CRUD app

TUTORIAL

ASP.NET and Azure SQL Database

ASP.NET Core and Azure SQL Database

Python (Django) and PostgreSQL

PHP and MySQL

Node.js and MongoDB

Java Spring Boot and Azure Cosmos DB

Multi-container (preview) and MySQL

REFERENCE

REST API

Secure and deploy

CONCEPT

Security in Azure App Service

TUTORIAL

Secure with custom domain and certificate

Continuous deployment

Upload content with FTP

Getting started with Azure App Service

Article • 08/29/2023

Introduction

Azure App Service is a fully managed platform as a service (PaaS) for hosting web applications. Use the following resources to get started with .NET.

Action	Resources
Create your first .NET app	Using one of the following tools: - Visual Studio - Visual Studio Code - Command line - Azure PowerShell - Azure portal
Deploy your app	- Configure ASP.NET - Configure ASP.NET core - GitHub actions
Monitor your app	- Log stream - Diagnose and solve tool
Add domains & certificates	- Map a custom domain - Add SSL certificate
Connect to a database	- .NET with Azure SQL Database - .NET Core with Azure SQL DB
Custom containers	- Linux - Visual Studio Code - Windows - Visual Studio
Review best practices	- Scale your app - Deployment - Security - Virtual Network

Next steps

Learn about App Service

App Service overview

Article • 08/31/2023

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, PHP, and Python. Applications run and scale with ease on both Windows and Linux-based environments.

App Service adds the power of Microsoft Azure to your application, such as 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 domain, and TLS/SSL certificates.

With App Service, you pay for the Azure compute resources you use. The compute resources you use are determined by the *App Service plan* that you run your apps on. For more information, see Azure App Service plans overview.

Why use App Service?

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, Ruby, Node.js, PHP, or Python. You can also run PowerShell and other scripts or executables as background services.
- **Managed production environment** - App Service automatically patches and maintains the OS and language frameworks for you. Spend time writing great apps and let Azure worry about the platform.
- **Containerization and Docker** - Dockerize your app and host a custom Windows or Linux container in App Service. Run multi-container apps with Docker Compose. Migrate your Docker skills directly to App Service.
- **DevOps optimization** - Set up continuous integration and deployment with Azure DevOps, GitHub, BitBucket, Docker Hub, or Azure Container Registry. Promote updates through test and staging environments. Manage your apps in App Service by using Azure PowerShell or the cross-platform command-line interface (CLI).
- **Global scale with high availability** - Scale up or out manually or automatically. Host your apps anywhere in Microsoft's global datacenter infrastructure, and the App Service SLA promises high availability.

- **Connections to SaaS platforms and on-premises data** - Choose from 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 Networks.
- **Security and compliance** - App Service is ISO, SOC, and PCI compliant. Create IP address restrictions and managed service identities. Prevent subdomain takeovers.
- **Authentication** - Authenticate users using the built-in authentication component. Authenticate users with Azure Active Directory, Google, Facebook, Twitter, or Microsoft account.
- **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 microservice architecture, consider Azure Spring Apps or Service Fabric. If you need more control over the VMs on which your code runs, consider Azure Virtual Machines. For more information about how to choose between these Azure services, see Azure App Service, Virtual Machines, Service Fabric, and Cloud Services comparison.

App Service on Linux

App Service can also host web apps natively on Linux for supported application stacks. It can also run custom Linux containers (also known as Web App for Containers).

Built-in languages and frameworks

App Service on Linux supports a number of language specific built-in images. Just deploy your code. Supported languages include: Node.js, Java (Tomcat, JBoss, or with an embedded web server), PHP, Python, .NET Core, and Ruby. Run az webapp list-runtimes

--os linux to view the latest languages and supported versions. If the runtime your application requires is not supported in the built-in images, you can deploy it with a custom container.

Outdated runtimes are periodically removed from the Web Apps Create and Configuration blades in the Portal. These runtimes are hidden from the Portal when they are deprecated by the maintaining organization or found to have significant vulnerabilities. These options are hidden to guide customers to the latest runtimes where they will be the most successful.

When an outdated runtime is hidden from the Portal, any of your existing sites using that version will continue to run. If a runtime is fully removed from the App Service platform, your Azure subscription owner(s) will receive an email notice before the removal.

If you need to create another web app with an outdated runtime version that is no longer shown on the Portal see the language configuration guides for instructions on how to get the runtime version of your site. You can use the Azure CLI to create another site with the same runtime. Alternatively, you can use the **Export Template** button on the web app blade in the Portal to export an ARM template of the site. You can reuse this template to deploy a new site with the same runtime and configuration.

Limitations

- App Service on Linux is not supported on Shared pricing tier.
- The Azure portal shows only features that currently work for Linux apps. As features are enabled, they're activated on the portal.
- When deployed to built-in images, your code and content are allocated a storage volume for web content, backed by Azure Storage. The disk latency of this volume is higher and more variable than the latency of the container filesystem. Apps that require heavy read-only access to content files may benefit from the custom container option, which places files in the container filesystem instead of on the content volume.

Next steps

Create your first web app.

Getting started)

ASP.NET Core (on Windows or Linux)

ASP.NET (on Windows)

PHP (on Windows or Linux)

Ruby (on Linux)

Node.js (on Windows or Linux)

Java (on Windows or Linux)

Python (on Linux)

HTML

Custom container (Windows or Linux)

App Service Environment overview

Article • 09/19/2023

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.

> **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)

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 multi-tenant 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.

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

The number of addresses that are used by an App Service Environment v3 in its subnet will vary, 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 multi-tenant 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 multi-tenant 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 multi-tenant 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. In this case, each App Service Plan on the App Service Environment will need 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 much faster than with an App Service Environment v2. Although scaling still isn't immediate, as in the multi-tenant 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 the following:

- 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 nine instances. There's no added charge for availability zone support if you have nine or more App Service plan instances. If you have fewer than nine instances (of any size) across App Service plans in the zone redundant App Service Environment, the difference between nine and 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.

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:

Region	Single zone support	Availability zone support	Single zone support
	App Service Environment v3	App Service Environment v3	App Service Environment v1/v2
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	✅	✅	✅
Italy North	✅	✅**	
Japan East	✅	✅	✅

Region	Single zone support	Availability zone support	Single zone support
Japan West	✅		✅
Jio India West			✅
Korea Central	✅	✅	✅
Korea South	✅		✅
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	✅	✅	✅
Sweden Central	✅	✅	
Switzerland North	✅	✅	✅
Switzerland West	✅		✅
UAE Central			✅
UAE North	✅	✅	✅
UK South	✅	✅	✅
UK West	✅		✅
West Central US	✅		✅
West Europe	✅	✅	✅
West India	✅*		✅
West US	✅		✅

Region	Single zone support	Availability zone support	Single zone support
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:

Region	Single zone support	Availability zone support	Single zone support
	App Service Environment v3	App Service Environment v3	App Service Environment v1/v2
US DoD Central	✅		✅
US DoD East			✅
US Gov Arizona	✅		✅
US Gov Iowa			✅
US Gov Texas	✅		✅
US Gov Virginia	✅	✅	✅

Microsoft Azure operated by 21Vianet:

Region	Single zone support	Availability zone support	Single zone support
	App Service Environment v3	App Service Environment v3	App Service Environment v1/v2
China East 2			✅
China East 3	✅		
China North			✅

Region	Single zone support	Availability zone support	Single zone support
2			
China North 3	✅	✅	

In-region data residency

An App Service Environment will only store 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 .

App Service Environment v2

App Service Environment has three versions: App Service Environment v1, App Service Environment v2, and App Service Environment v3. The information in this article is based on App Service Environment v3. To learn more about App Service Environment v2, see App Service Environment v2 introduction.

Next steps

Whitepaper on Using App Service Environment v3 in Compliance-Oriented Industries

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.

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 managed function as a service.
- 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 container in Azure. You don't have 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 Spring Apps: A managed service designed and optimized for hosting Spring Boot apps.
- 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.

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.

> **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.

Service	Application composition	Density	Minimum number of nodes	State management	Web hosting
Azure Virtual Machines	Agnostic	Agnostic	1 [2]	Stateless or stateful	Agnostic

Service	Application composition	Density	Minimum number of nodes	State management	Web hosting
Azure App Service	Applications, containers	Multiple apps per instance by using App Service plan	1	Stateless	Built in
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 Spring Apps	Applications, microservices	Multiple apps per service instance	2	Stateless	Built in
Azure Service Fabric	Services, guest executables, containers	Multiple services per VM	5 [3]	Stateless or stateful	Agnostic
Azure Batch	Scheduled jobs	Multiple apps per	1 [4]	Stateless	No

Service	Application composition	Density	Minimum number of nodes	State management	Web hosting
		VM			

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.

Networking

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 Spring Apps	Supported	Supported
Azure Service Fabric	Supported	Supported

Service	Virtual network integration	Hybrid connectivity
Azure Batch	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

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

Service	Local debugging	Programming model	Application update
Azure Spring Apps	Visual Studio Code, Intellij, Eclipse	Spring Boot, Steeltoe	Rolling upgrade, blue-green deployment
Azure Service Fabric	Local node cluster	Guest executable, Service model, Actor model, Containers	Rolling upgrade (per service)
Azure Batch	Not supported	Command-line application	Not applicable

Notes

1. 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

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

Service	Autoscaling	Load balancer	Scale limit[3]
Azure Container Apps	Scaling rules[4]	Integrated	5 environments per region, 20 container apps per environment, 30 replicas per container app
Azure Container Instances	Not supported	No built-in support	20 container groups per subscription (default limit)
Azure Red Hat OpenShift	Pod autoscaling, cluster autoscaling	Azure Load Balancer or Azure Application Gateway	60 nodes per cluster (default limit)
Azure Spring Apps	Built-in service	Integrated	500 app instances in Standard
Azure Service Fabric	Virtual machine scale sets	Azure Load Balancer	100 nodes per virtual machine scale set
Azure Batch	Not applicable	Azure Load Balancer	20 core limit (default limit)

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.

Availability

Service	SLA	Multiregion failover
Azure Virtual Machines	SLA for Virtual Machines	Azure Traffic Manager, Azure Front Door, and cross-region Azure Load Balancer

Service	SLA	Multiregion failover
Azure App Service	SLA for App Service	Azure Traffic Manager and Azure Front Door
Azure Functions	SLA for Functions	Azure Traffic Manager and Azure Front Door
Azure Kubernetes Service	SLA for AKS	Azure Traffic Manager, Azure Front Door, and Multiregion Cluster
Azure Container Apps	SLA for Container Apps	Azure Traffic Manager and Azure Front Door
Azure Container Instances	SLA for Container Instances	Azure Traffic Manager and Azure Front Door
Azure Red Hat OpenShift	SLA for Azure Red Hat OpenShift	Azure Traffic Manager and Azure Front Door
Azure Spring Apps	SLA for Azure Spring Apps	Azure Traffic Manager, Azure Front Door, and Multiregion Cluster
Azure Service Fabric	SLA for Service Fabric	Azure Traffic Manager, Azure Front Door, and cross-region Azure Load Balancer
Azure Batch	SLA for Batch	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 Spring Apps
- Azure Service Fabric
- Azure Batch

Other criteria

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	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 Spring Apps	Supported	Azure Spring Apps pricing	Spring Boot, microservices
Azure Service Fabric	Supported	Service Fabric pricing	Microservices, event-driven architecture
Azure Batch	Supported	Batch pricing	Big compute (HPC)

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

Quickstart: Deploy an ASP.NET web app

Article • 05/08/2023

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, and 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 consisting of 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.

Prerequisites

.NET 7.0

- An Azure account with an active subscription. Create an account for free.
- Visual Studio 2022 with the **ASP.NET and web development** workload.

If you have 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**.

1. Create an ASP.NET web app

.NET 7.0

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**, then select **Next**.
3. In **Configure your new project**, name the application *MyFirstAzureWebApp*, and then select **Next**.

4. Select **.NET 7.0 (Standard-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**.

2. Publish your web app

To publish your web app, you must first create and configure a new App Service that you can publish your app to.

As part of setting up the App Service, you create:

- A new resource group to contain all of the Azure resources for the service.
- A new Hosting Plan that specifies the location, size, and features of the web server farm that hosts your 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)**. Then, 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:

Setting	Suggested value	Description
Hosting Plan	*MyFirstAzureWebAppPlan*	Name of the App Service plan.
Location	*West Europe*	The datacenter where the web app is hosted.
Size	*Free*	Pricing tier determines hosting features.

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

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

Once the wizard completes, the Azure resources are created for you, and 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.

.NET 7.0

You see the ASP.NET Core 7.0 web app displayed in the page.

3. Update the app and redeploy

Follow these steps to update and redeploy your web app:

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

2. Replace the first `<div>` element with the following code:

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

    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.

.NET 7.0

You see the updated ASP.NET Core 7.0 web app displayed in the page.

4. Manage the Azure app

To manage your web app, go to the Azure portal, and search for and select **App Services**.

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

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.

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

.NET 7.0

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

Deploy a Node.js web app in Azure

Article • 07/17/2023

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.

Set up your initial environment

- Have an Azure account with an active subscription. Create an account for free.
- Install Node.js and npm. Run the command `node --version` to verify that Node.js is installed.
- Install Visual Studio Code.
- 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
   ```

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

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

3. Start the development server with debug information.

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

4. 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, 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 **AZURE APP SERVICE** explorer.

I ran into an issue

Configure the App Service app and deploy code

1. Select the *myExpressApp* folder.

Deploy to Windows

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

3. Type a globally unique name for your web app and press **Enter**. The name must be unique across all of Azure and use only alphanumeric characters ('A-Z', 'a-z', and '0-9') and hyphens ('-').

4. Select **Create a new resource group**, 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**, then enter a name for the plan (such as *AppServiceQS-plan*), then select **F1 Free** for the pricing tier.

9. For **Select an Application Insights resource for your app**, select **Skip for now** and wait the resources to be created in Azure.

10. In the popup **Always deploy the workspace "myExpressApp" to <app-name>"**, select **Yes**. This way, 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**.

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, confirm by clicking **Deploy** again.

15. Wait for deployment to complete, 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 redeploy 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, confirm by clicking **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 create 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!

Tutorial: Node.js app with MongoDB

Configure Node.js app

Secure with custom domain and 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.

Create a PHP web app in Azure App Service

Article • 03/09/2023

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 provision 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. It will clone the sample application to your local machine, and navigate to the directory containing the sample code.

   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 will create the necessary resources and deploy 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:8.0" --os-type=linux
```

- If the `az` command isn't recognized, be sure you have Azure CLI installed.
- The `--runtime "PHP:8.0"` argument creates the web app with PHP version 8.0.
- 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 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 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 may take a few minutes to complete. While 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|8.0",
  "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've 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:8.0" --os-type=linux
```

3. Once deployment has 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

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

2. Select the name of your Azure app.

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

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 will not incur extra charges and 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 myResourceGroup
```

This command may take a minute to run.

Next steps

PHP with MySQL

Configure PHP app

Secure with custom domain and certificate

Quickstart: Create a Java app on Azure App Service

Article • 08/31/2023

In this quickstart, you'll 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 uses a Quarkus sample, which comes with a bundled web server. You can deploy your own application and server bundle in a single JAR file to App Service instead of the using the Tomcat or JBoss hosting options. If you want, you can also embed a Tomcat server in the JAR file and run that in App Service.

> **Note**
>
> For Spring apps that requires all the Spring services, try **Azure Spring Apps** instead. However, you can deploy Spring Boot apps to App Service.

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:

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 - Create a Java app

Execute the following Maven command in the Cloud Shell prompt to create a new app named `quarkus-hello-azure`:

Azure CLI

```
mvn io.quarkus.platform:quarkus-maven-plugin:3.2.2.Final:create \
  -DprojectGroupId=org.acme \
  -DprojectArtifactId=quarkus-hello-azure  \
  -Dextensions='resteasy-reactive'
```

Then change your working directory to the project folder:

Azure CLI

```
cd quarkus-hello-azure
```

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:2.12.0: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 : quarkus-hello-azure-1690375364238
ResourceGroup : quarkus-hello-azure-1690375364238-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've confirmed 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

```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

Property	Required	Description	Version
`<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>` (`quarkus-hello-azure-1690375364238` and `quarkus-hello-azure-1690375364238-rg` accordingly in the demo). They're used later.

4 - Deploy the app

With all the configuration ready in your *pom.xml* file, you can deploy your Java app to Azure with one single command.

1. Rebuild the JAR file using the following command:

   ```bash
   mvn clean package -Dquarkus.package.type=uber-jar -
   D%prod.quarkus.http.port=80
   ```

   This command adds two Quarkus properties:

- `quarkus.package.type=uber-jar` tells Maven to generate an Uber-Jar, which includes all dependencies in the JAR file.
- `%prod.quarkus.http.port=80` tells Quarkus to use port 80 for the prod environment at runtime, which is the default port that the Linux Java container uses. If you want, you can change the port number of the Java container with the `WEBSITES_PORT` app setting.

You can configure these properties by using other means, but they're added to `mvn package` command here for simplicity.

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://quarkus-hello-
azure-1690375364238.azurewebsites.net
[INFO] ------------------------------------------------------------------
--------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------
--------
[INFO] Total time:  02:20 min
[INFO] Finished at: 2023-07-26T12:47:50Z
[INFO] ------------------------------------------------------------------
--------
```

Once deployment is completed, your application is ready at `http://<appName>.azurewebsites.net/` (`http://quarkus-hello-azure-1690375364238.azurewebsites.net` in the demo). Open the url with your local web browser, you should see

Congratulations! You've 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 may 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

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

Article • 07/26/2023

In this quickstart, you'll deploy a Python web app (Django or Flask) 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.

1 - Sample application

This quickstart can be completed using either Flask or Django. 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

Cmd

```
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.

Having issues? Let us know.

2 - Create a web app in Azure

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

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, login 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 account 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.

3 - Deploy your application code to Azure

Azure App service supports multiple methods to deploy your application code to Azure including support for 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 VS Code

Since the previous step created the necessary resources and deployed your application in a single step, you can move on to 4 - Browse to the app.

Having issues? Refer first to the Troubleshooting guide, otherwise, let us know.

4 - Browse to the app

Browse to the deployed application in your web browser at 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, otherwise, let us know.

5 - Stream logs

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

Flask

Python

```
app = Flask(__name__)


@app.route('/')
def index():
   print('Request for index page received')
   return render_template('index.html')

@app.route('/favicon.ico')
def favicon():
    return send_from_directory(os.path.join(app.root_path, 'static'),
                               'favicon.ico',
mimetype='image/vnd.microsoft.icon')

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

The contents of the App Service diagnostic logs can be reviewed in the Azure portal, VS Code, or using the Azure CLI.

Azure CLI

First, you need to configure Azure App Service to output logs to the App Service filesystem using the az webapp log config command.

bash

Azure CLI

```bash
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, otherwise, 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. It will not incur extra charges and 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) web app with PostgreSQL

Configure Python app

Add user sign-in to a Python web app

Tutorial: Run Python app in custom container

Secure with custom domain and certificate

Create a Ruby on Rails App in App Service

Article • 03/29/2023

Azure App Service on Linux provides a highly scalable, self-patching web hosting service using the Linux operating system.

> **Note**
>
> The Ruby development stack only supports Ruby on Rails at this time. If you want to use a different platform, such as Sinatra, or if you want to use an unsupported Ruby version, you need to **run it in a custom container**.

If you don't have an Azure subscription, create an Azure free account before you begin.

Azure CLI

If you are running the sample code locally, you will need:

- Install Ruby 2.7 or higher
- Install Git

You will not need to install these if you are using the Cloud Shell.

Azure Cloud Shell

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

To start Azure Cloud Shell:

Option	Example/Link
Select **Try It** in the upper-right corner of a code 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.

1 - Get the sample repository

Azure CLI

This quickstart tutorial shows how to deploy a Ruby on Rails app to App Service on Linux using the Cloud Shell.

1. In a terminal window, clone the sample application to your local machine, and navigate to the directory containing the sample code.

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

2. 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`, this tutorial also shows you how to deploy a repository from `main`. For more information, see **Change deployment branch**.

Run the application locally

If you want to run the application locally to see how it works, clone the repository locally and follow these steps.

> **Note**
>
> These steps cannot be run in Azure Cloud Shell, due to version conflicts with Ruby.

1. Install the required gems. There's a `Gemfile` included in the sample, so just run the following command:

   ```bash
   bundle install
   ```

2. Once the gems are installed, start the app:

   ```bash
   bundle exec rails server
   ```

3. Using your web browser, navigate to `http://localhost:3000` to test the app locally.

2 - Deploy your application code to Azure

Azure CLI

Configure a deployment user

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

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

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

Azure CLI

```
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 on Linux in **Basic** tier, run the az appservice list-locations --sku B1 --linux-workers-enabled 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 Azure App Service plan

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

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

Azure CLI

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

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

```
{
  "freeOfferExpirationTime": null,
  "geoRegion": "West Europe",
  "hostingEnvironmentProfile": null,
  "id": "/subscriptions/0000-
0000/resourceGroups/myResourceGroup/providers/Microsoft.Web/serverfarms/
myAppServicePlan",
  "kind": "linux",
```

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

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

   In the Cloud Shell, you can use the az webapp create command. In the following example, replace `<app-name>` with a globally unique app name (valid characters are `a-z`, `0-9`, and `-`). The runtime is set to `RUBY|2.7`. To see all supported runtimes, run az webapp list-runtimes --os linux.

   Azure CLI

   ```
   az webapp create --resource-group myResourceGroup --plan
   myAppServicePlan --name <app-name> --runtime 'RUBY|2.7' --
   deployment-local-git
   ```

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

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

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

   ⓘ **Note**

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

2. 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'
```

3. Back in the local terminal window, add an Azure remote to your local Git repository. Replace <*deploymentLocalGitUrl-from-create-step*> with the `deploymentLocalGitUrl` value from app creation.

Bash

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

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

Bash

```
git push azure main
```

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

```
remote: Using turbolinks 5.2.0
remote: Using uglifier 4.1.20
remote: Using web-console 3.7.0
remote: Bundle complete! 18 Gemfile dependencies, 78 gems now
installed.
remote: Bundled gems are installed into `/tmp/bundle`
remote: Zipping up bundle contents
remote: .......
```

```
remote: ~/site/repository
remote: Finished successfully.
remote: Running post deployment command(s)...
remote: Deployment successful.
remote: App container will begin restart within 10 seconds.
To https://<app-name>.scm.azurewebsites.net/<app-name>.git
   a6e73a2..ae34be9  main -> main
```

5. Once the deployment has completed, wait about 10 seconds for the web app to restart, and then navigate to the web app and verify the results.

Bash

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

> **Note**
>
> While the app is restarting, you may observe the HTTP status code `Error 503 Server unavailable` in the browser, or the `Hey, Ruby developers!` default page. It may take a few minutes for the app to fully restart.

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

Congratulations! You've deployed your first Ruby app to App Service using the Azure portal.

3 - Update and redeploy the app

Azure CLI

1. From Azure Cloud Shell, launch a text editor and edit the file `app/controllers/application_controller.rb`. Edit the *ApplicationController* class so that it shows "Hello world from Azure App Service on Linux!" instead of "Hello from Azure App Service on Linux!".

Ruby

```ruby
class ApplicationController < ActionController::Base
    def hello
        render html: "Hello world from Azure App Service on Linux!"
    end
end
```

2. Save and close the file.

3. Commit the change to Git with the following commands:

Bash

```bash
git add .
git commit -m "Hello world"
git push azure main
```

> **Note**
>
> - Make sure your `git push` statement includes `azure main` so that you are pushing directly to the Azure Git deployment URL.
> - You may also need to set your `user.name` and `user.email` Git config values when you commit the changes. You can do that with `git config user.name "YOUR NAME"` and `git config user.email "YOUR EMAIL"`.

4 - Manage your new Azure app

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

2. Select the name of your Azure app.

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

5 - Clean up resources

Azure CLI

Clean up deployment

After the sample script has been run, the following command can be used to remove the resource group and all resources associated with it.

Azure CLI

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

Next steps

Tutorial: Ruby on Rails with Postgres

Configure Ruby app

Secure with custom domain and certificate

Create a WordPress site

Article • 05/23/2023

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.

> **Important**
>
> After November 28, 2022, **PHP will only be supported on App Service on Linux.**.
>
> For migrating WordPress to App Service, visit **Migrating to App Service**. Additional documentation can be found at **WordPress - App Service on Linux**.
>
> To submit feedback on improving the WordPress experience on App Service, visit **Web Apps Community**.

Create WordPress site using Azure portal

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

2. 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.

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.

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.

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 ☐

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 ☐

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 ☐

Storage Account: (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: (New) mydemowpsi-21f6b45092-vnet

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

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.

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.

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

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

Deploy a Go web app to Azure App Service

Article • 05/03/2023

> **Important**
>
> Go on App Service on Linux is *experimental*.

In this quickstart, you'll deploy a Go web app to Azure App Service. Azure App Service is a fully managed web hosting service that supports Go 1.19 and higher 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.
- Go 1.19 or higher installed locally.

1 - Sample application

First, create a folder for your project.

Go to the terminal window, change into the folder you created and run `go mod init <ModuleName>`. The ModuleName could just be the folder name at this point.

The `go mod init` command creates a go.mod file to track your code's dependencies. So far, the file includes only the name of your module and the Go version your code supports. But as you add dependencies, the go.mod file will list the versions your code depends on.

Create a file called main.go. We'll be doing most of our coding here.

Go

```go
package main
import (
    "fmt"
    "net/http"
)
func main() {
    http.HandleFunc("/", HelloServer)
    http.ListenAndServe(":8080", nil)
}
func HelloServer(w http.ResponseWriter, r *http.Request) {
```

```
    fmt.Fprintf(w, "Hello, %s!", r.URL.Path[1:])
}
```

This program uses the `net.http` package to handle all requests to the web root with the HelloServer function. The call to `http.ListenAndServe` tells the server to listen on the TCP network address `:8080`.

Using a terminal, go to your project's directory and run `go run main.go`. Now open a browser window and type the URL `http://localhost:8080/world`. You should see the message `Hello, world!`.

2 - Create a web app in Azure

To host your application in Azure, you need to create Azure App Service web app in Azure. You can create a web app using the 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 GO:1.19 --os linux --sku B1
```

- The `--runtime` parameter specifies what version of Go your app is running. This example uses Go 1.18. 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.

- 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 account 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>' ...
Creating zip with contents of dir /home/tulika/myGoApp ...
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": "go|1.19",
  "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.

3 - Browse to the app

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

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

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

4 - Clean up resources

When no longer needed, you can use the az group delete command to remove the resource group, and all related resources:

Azure CLI

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

Next steps

Configure an App Service app

Tutorial: Deploy from Azure Container Registry

Secure with custom domain and certificate

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", "/defaulthome/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 `/defaulthome/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.

Deployment Best Practices

Article • 09/04/2023

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

Deployment Components

Deployment Source

A deployment source is the location of your application code. For production apps, the deployment source is usually a repository hosted by version control software such as GitHub, BitBucket, or Azure Repos. For development and test scenarios, the deployment source may be a project on your local machine. App Service also supports OneDrive and Dropbox folders as deployment sources. While cloud folders can make it easy to get started with App Service, it's not typically recommended to use this source for enterprise-level production applications.

Build Pipeline

Once you decide on a deployment source, your next step is to choose a build pipeline. A build pipeline reads your source code from the deployment source and executes a series of steps (such as compiling code, minifying HTML and JavaScript, running tests, and packaging components) to get the application in a runnable state. The specific commands executed by the build pipeline depend on your language stack. These operations can be executed on a build server such as Azure Pipelines, or executed locally.

Deployment Mechanism

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

- Kudu endpoints: Kudu is the open-source developer productivity tool that runs as a separate process in Windows App Service, and as a second container in Linux App Service. Kudu handles continuous deployments and provides HTTP endpoints for deployment, such as zipdeploy/.
- FTP and WebDeploy: Using your site or user credentials, you can upload files via FTP or WebDeploy. These mechanisms 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, use deployment slots when deploying a new production build. When using a Standard App Service Plan tier or better, you can deploy your app to a staging environment, validate your changes, and do smoke tests. When you're ready, you can swap your staging and production slots. The swap operation warms up the necessary worker instances to match your production scale, thus eliminating downtime.

Continuously deploy code

If your project has designated branches for testing, QA, and staging, then each of those branches should be continuously deployed to a staging slot. (This is known as the Gitflow design.) This 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 because 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 the following 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 far more difficult.
2. **Push the tagged image**. Once the image is built and tagged, the pipeline pushes the image to our container registry. In the next step, the deployment slot will pull the tagged image from the container registry.
3. **Update the deployment slot with the new image tag**. When this property is updated, the site will automatically restart and pull the new container image.

There are examples below 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 and select **Deployment Center** under **Deployment**. Follow the instructions to select your repository and branch. This will configure 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 below will build and tag the container with the commit ID, push it to a container registry, and update the specified site slot with the new image tag.

YAML

```yaml
name: Build and deploy a container image to Azure Web Apps

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

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

```yaml
    steps:
    - uses: actions/checkout@main

    -name: Authenticate using a Service Principal
      uses: azure/actions/login@v1
      with:
        creds: ${{ secrets.AZURE_SP }}

    - uses: azure/container-actions/docker-login@v1
      with:
        username: ${{ secrets.DOCKER_USERNAME }}
        password: ${{ secrets.DOCKER_PASSWORD }}

    - name: Build and push the image tagged with the git commit hash
      run: |
        docker build . -t contoso/demo:${{ github.sha }}
        docker push contoso/demo:${{ github.sha }}

    - name: Update image tag on the Azure Web App
      uses: azure/webapps-container-deploy@v1
      with:
        app-name: '<your-webapp-name>'
        slot-name: '<your-slot-name>'
        images: 'contoso/demo:${{ 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

```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's information. You can then use `az webapp config container set` to set the container name, tag, registry URL, and registry password. Below are some helpful links for you to construct your container CI process.

- How to sign in to the Azure CLI on Circle CI

Language-Specific Considerations

Java

Use the Kudu zipdeploy/ API for deploying JAR applications, and wardeploy/ for WAR apps. If you're using Jenkins, you can use those APIs directly in your deployment phase. For more information, see this article.

Node

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

.NET

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

Other Deployment Considerations

Local Cache

Azure App Service content is stored on Azure Storage and is surfaced up in a durable manner as a content share. However, some apps just need a high-performance, read-only content store that they can run with high availability. These apps can benefit from using local cache. Local cache isn't recommended for content management sites such as WordPress.

Always use local cache in conjunction with deployment slots to prevent downtime. See this section for information on using these features together.

High CPU or Memory

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

For more information on best practices, visit App Service Diagnostics to find out actionable best practices specific to your resource.

- Navigate to your Web App in the Azure portal.
- Select on **Diagnose and solve problems** in the left navigation, which opens App Service Diagnostics.
- Choose **Best Practices** homepage tile.
- Select **Best Practices for Availability & Performance** or **Best Practices for Optimal Configuration** to view the current state of your app in regards to these best practices.

You can also use this link to directly open App Service Diagnostics for your resource: `https://portal.azure.com/?websitesextension_ext=asd.featurePath%3Ddetectors%2FParentAvailabilityAndPerformance#@microsoft.onmicrosoft.com/resource/subscriptions/{subscriptionId}/resourceGroups/{resourceGroupName}/providers/Microsoft.Web/sites/{siteName}/troubleshoot`.

More resources

Environment variables and app settings reference

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

Article • 03/03/2023

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

In this tutorial, you learn how to:

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

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

If you don't have an Azure subscription, create an Azure free account before you begin.

Prerequisites

To complete this tutorial:

- Install Git
- Install the latest .NET Core 3.1 SDK

Create local ASP.NET Core app

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

Clone the sample application

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

2. Clone the sample repository and change 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 following tutorial: ASP.NET Core Web API help pages using Swagger. It uses a Swagger generator to serve the Swagger UI and the Swagger JSON endpoint.

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 also 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 play with the Swagger UI.

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

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

5. To stop ASP.NET Core at any time, press `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:

Option	Example/Link

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 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.

Azure CLI

```
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 a deployment user**, not the credentials you use to sign in to the Azure portal.

   Bash

   ```
   git push azure main
   ```

This command may 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 play with 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 sample app

1. In your local repository, open *wwwroot/index.html*.

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

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, the fact that your REST API the App Service app is not 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 way to enable CORS to a localhost URL is the same as a public URL.

Enable CORS

In the Cloud Shell, enable CORS to your client's URL by using the az 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-separate 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 may want to specify different allowed origins for different routes or methods. Since App Service CORS lets you specify one set of accepted origins for all API routes and methods, you would want to use your own CORS code. See how ASP.NET Core does it at Enabling Cross-Origin Requests (CORS).

The built-in App Service CORS feature does not 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. When used 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 `*`. However, 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 it using the 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 may 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 is not allowed when allowed origins include the wildcard origin `'*'`. Specifying `AllowAnyOrigin` and `AllowCredentials` is an insecure configuration and 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

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

Tutorial: Authenticate and authorize users end-to-end

Configure an App Service app

Article • 04/21/2023

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

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
- Ruby
- Custom containers

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

> **Note**
>
> App settings can also be resolved from **Key Vault** using **Key Vault references**.

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** > **Application settings**.

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 **New application setting**. To edit a setting, select the **Edit** button on the right side.

4. In the dialog, you can stick the setting to the current slot.

   App setting names can't contain periods (`.`). If an app setting contains a period, the period is replaced with an underscore in the container.

> **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).

5. When finished, select **Update**. Don't forget to select **Save** back in the **Configuration** page.

Edit app settings in bulk

Azure portal

Select the **Advanced edit** button. Edit the settings in the text area. When finished, select **Update**. Don't forget to select **Save** back in the **Configuration** 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

In the Azure portal, search for and select **App Services**, and then select your app. In the app's left menu, select **Configuration** > **Application settings**.

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_`

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

- ASP.NET Core
- Node.js
- PHP
- Python
- Java
- Ruby
- Custom containers

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

> **Note**
>
> Connection strings can also be resolved from **Key Vault** using **Key Vault references**.

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** > **Application settings**.

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 **New connection string**. To edit a connection string, select the **Edit** button on the right side.

4. In the dialog, you can stick the connection string to the current slot.

5. When finished, select **Update**. Don't forget to select **Save** back in the **Configuration** page.

Edit connection strings in bulk

Azure portal

Select the **Advanced edit** button. Edit the connection strings in the text area. When finished, select **Update**. Don't forget to select **Save** back in the **Configuration** 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
- Ruby

Configure general settings

Azure portal

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**.

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.

- **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.
  - **ARR affinity**: In a multi-instance deployment, ensure that the client is routed to the same instance for the life of the session. You can set this option to **Off** for stateless applications.
  - **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.

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**.

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**.

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

   - **Extension**. The file extension you want to handle, such as **.php* or *handler.fcgi*.
   - **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**.

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

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 string parts	Description
Endpoint=*endpoint*;	**Endpoint** is the required part of the reference string. The value for **Endpoint** should have the url of your App Configuration resource.
Key=*keyName*;	**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.
Label=*label*	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.azconfig.io;
Key=myAppConfigKey; Label=myKeysLabel)
```

Alternatively without any `Label`:

```
@Microsoft.AppConfiguration(Endpoint=https://myAppConfigStore.azconfig.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": [
                        "
[resourceId('Microsoft.AppConfiguration/configurationStores',
variables('appConfigStoreName'))]"

                    ],
                    "properties": {
                        "value": "Calibri",
                        "contentType": "application/json"
                    }
                },
                {
                    "type": "keyValues",
                    "name": "[variables('FontColorKey')]",
                    "apiVersion": "2021-10-01-preview",
                    //...
                    "dependsOn": [
                        "
[resourceId('Microsoft.AppConfiguration/configurationStores',
variables('appConfigStoreName'))]"

                    ],
                    "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

Configure an ASP.NET app for Azure App Service

Article • 01/25/2023

> **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.

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

Configure an ASP.NET Core app for Azure App Service

Article • 02/15/2023

> **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 3.1:

Azure CLI

```
az webapp config set --name <app-name> --resource-group <resource-group-name> --linux-fx-version "DOTNETCORE|3.1"
```

Customize build automation

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 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 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**
>
> 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 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** error or **An error occurred while processing your request.** message. To display the detailed exception page in App Service, Add the `ASPNETCORE_ENVIRONMENT` app setting to your app by running the following command in the Cloud Shell.

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.

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.

> **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 additional resources:

Environment variables and app settings reference

Configure a Node.js app for Azure App Service

Article • 10/18/2022

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 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 you do it in a simple Express app:

JavaScript

```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 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 `npm install` without any flags, which includes npm `preinstall` and `postinstall` scripts and also installs `devDependencies`.
3. Run `npm run build` if a build script is specified in your *package.json*.
4. Run `npm run build:azure` if a build:azure script is specified in your *package.json*.

5. Run custom script if specified by `POST_BUILD_SCRIPT_PATH`.

> **Note**
>
> As described in **npm docs**, scripts named `prebuild` and `postbuild` run before and after `build`, respectively, if specified. `preinstall` and `postinstall` run before and after `install`, respectively.

`PRE_BUILD_COMMAND` and `POST_BUILD_COMMAND` are environment variables that are empty by default. To run pre-build commands, define `PRE_BUILD_COMMAND`. To run post-build commands, define `POST_BUILD_COMMAND`.

The following example specifies the two variables to a series of commands, separated by commas.

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 Node.js apps in Linux, see Oryx documentation: How Node.js apps are detected and built.

Configure Node.js server

The Node.js containers come with PM2, a production process manager. You can configure your app to start with PM2, or with NPM, or with a custom command.

Tool	Purpose
Run with PM2	**Recommended** - Production or staging use. PM2 provides a full-service app management platform.
Run npm start	Development use only.
Run 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* and *ecosystem.config.js*

You can also configure a custom start file with the following extensions:

- A *.js* file
- A PM2 file with the extension *.json*, *.config.js*, *.yaml*, or *.yml*

> **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 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 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

> **Note**
>
> Remote debugging is currently in Preview.

You can debug your Node.js app remotely in Visual Studio Code if you configure it to run with PM2, except when you run it using a *.config.js*,.yml, or *.yaml*.

In most cases, no extra configuration is required for your app. If your app is run with a *process.json* file (default or custom), it must have a `script` property in the JSON root. For example:

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**. Click **Yes** to enable it for your app. App Service starts a tunnel proxy for you and attaches the debugger. You can then make requests to the app and see the debugger pausing at break points.

Once finished with debugging, stop the debugger by selecting **Disconnect**. When prompted, you should click **Yes** to disable remote debugging. To disable it later, right-click your app again in the Azure explorer and select **Disable Remote Debugging**.

Access environment variables

In App Service, you can set app settings outside of your app code. Then you can access them using the standard Node.js pattern. For example, to access an app setting called `NODE_ENV`, use the following code:

JavaScript

```
process.env.NODE_ENV
```

Run Grunt/Bower/Gulp

By default, App Service build automation runs `npm install --production` when it recognizes 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 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
  exitWithMessageOnError "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
  exitWithMessageOnError "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
  exitWithMessageOnError "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 or not, inspect the `X-Forwarded-Proto` header.

Popular web frameworks let you access the `X-Forwarded-*` information in your standard app pattern. In Express, you can use trust proxies. For example:

```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`.

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 App Insights agent, go to your web app in the Portal and select **Application Insights** under **Settings**, then select **Turn on Application Insights**. Next, select an existing App Insights resource or create a new one. Finally, select **Apply** at the bottom. To instrument your web app using PowerShell, please 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 may be installed for production mode (`dependencies` vs. `devDependencies`).
  - Certain web frameworks may deploy static files differently in production mode.
  - Certain web frameworks may use custom startup scripts when running in production mode.
- Run your app in App Service in development mode. For example, in MEAN.js, you can set your app to development mode in runtime by setting the NODE_ENV app setting.

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

App Service Linux FAQ

Or, see additional resources:

Environment variables and app settings reference

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 • 12/02/2022

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 needed. Python apps must be deployed with all the required pip modules.

The App Service deployment engine automatically activates a virtual environment and runs `pip install -r requirements.txt` for you when you deploy a Git repository, or a zip package with build 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 on 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 on 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 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 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 happens during deployment. The setting is true when deploying 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 it 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 the 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 on 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 needed, 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 (djangoproject.com).

The following table describes the production settings that are relevant to Azure. These settings are defined in the app's *setting.py* file.

Django setting	Instructions for Azure
`SECRET_KEY`	Store the value in an App Service setting as described on Access app settings as environment variables. You can alternately store the value as a "secret" in Azure Key Vault.
`DEBUG`	Create a `DEBUG` setting on App Service with the value 0 (false), then load the value as an environment variable. In your development environment, create a `DEBUG` environment variable with the value 1 (true).
`ALLOWED_HOSTS`	In production, Django requires that you include app's URL in the `ALLOWED_HOSTS` array of *settings.py*. You can retrieve this URL at runtime with the code, `os.environ['WEBSITE_HOSTNAME']`. App Service automatically sets the `WEBSITE_HOSTNAME` environment variable to the app's URL.
`DATABASES`	Define settings in App Service for the database connection and load them as environment variables to populate the DATABASES dictionary. You can alternately 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

   ```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

   ```python
   FRONTEND_DIR = "path-to-frontend-folder"
   STATICFILES_DIRS = [os.path.join(FRONTEND_DIR, 'build', 'static')]
   ```

   Here, `FRONTEND_DIR`, 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

   ```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 quickstart 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 on Deploying Gunicorn (docs.gunicorn.org).

- 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 provided.
2. Check for the existence of a Django app, and launch Gunicorn for it if detected.
3. Check for the existence of a Flask app, and launch Gunicorn for it if detected.
4. If no other app is found, start a default app that's built into the container.

The following sections provide 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, or 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.

Again, if you expect to see a deployed app instead of the default app, see Troubleshooting - App doesn't appear.

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` 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 (docs.gunicorn.org). 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 (docs.gunicorn.org).

- **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 on 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 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 (wikipedia.org) 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

```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

```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

```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 exactly 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.

> **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 may be preventing the app container from starting. See Troubleshooting for steps to investigate possible issues.

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 may take longer to start up when using free tiers, for example, and becomes responsive after you refresh the browser.
    - Check that your app is structured as App Service expects for Django or Flask, or use a custom startup command.
    - 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 site 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, please see this article on virtual environment portability.

Database is locked

When attempting to run database migrations with a Django app, you may 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 PostgreSQL for Azure.

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 press **Enter** when done.
- **Commands in the SSH session appear to be cut off**: The editor may 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.

More resources

- Tutorial: Python app with PostgreSQL
- Tutorial: Deploy from private container repository
- App Service Linux FAQ
- Environment variables and app settings reference

Configure a Java app for Azure App Service

Article • 03/10/2023

Azure App Service lets Java developers to quickly build, deploy, and scale their Java SE, Tomcat, and JBoss EAP web applications on a fully managed service. Deploy applications with Maven plugins, from the command line, or in editors like IntelliJ, Eclipse, or Visual Studio Code.

This guide provides key concepts and instructions for Java developers using App Service. If you've never used Azure App Service, you should read through the Java quickstart first. General questions about using App Service that aren't specific to Java development are answered in the App Service FAQ.

Show Java version

To show the current Java 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 Java versions, run the following command in the Cloud Shell:

Azure CLI

```
az webapp list-runtimes --os linux | grep "JAVA\|TOMCAT\|JBOSSEAP"
```

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:2.11.0:config
```

This command adds a `azure-webapp-maven-plugin` plugin and related configuration by prompting you to select an existing Azure Web App or create a new one. Then you can deploy your Java app to Azure using the following command:

shell

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

Here is a sample configuration in `pom.xml`:

XML

```xml
<plugin>
  <groupId>com.microsoft.azure</groupId>
  <artifactId>azure-webapp-maven-plugin</artifactId>
  <version>2.11.0</version>
  <configuration>
    <subscriptionId>111111-11111-11111-1111111</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 11</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 "1.7.1"
}
```

2. Configure your Web App details, corresponding Azure resources will be created if not exist. Here is 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 9.0' // or 'Java SE' if you want to run an
executable jar
      javaVersion = 'Java 8'
    }
    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

Java SE

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 (mentioned above) will automatically rename your application for you during deployment. If you do not wish to rename your JAR to *app.jar*, you can upload a shell script with the command to run your .jar app. Paste the absolute path to this script in the **Startup File** textbox in the Configuration section of the portal. The startup script does not run from the directory into which it is placed. Therefore, always use absolute paths to reference files in your startup script (for example: `java -jar /home/myapp/myapp.jar`).

Tomcat

To deploy .war files to Tomcat, use the `/api/wardeploy/` endpoint to POST your archive file. For more information on this API, see this documentation.

JBoss EAP

To deploy .war files to JBoss, use the `/api/wardeploy/` endpoint to POST your archive file. For more information on this API, see this documentation.

To deploy .ear files, use FTP. Your .ear application will be deployed to the context root defined in your application's configuration. For example, if the context root of your app is `<context-root>myapp</context-root>`, then you can browse the site at the `/myapp` path: `http://my-app-name.azurewebsites.net/myapp`. If you want your web app to be served in the root path, ensure that your app sets the context root to the root path: `<context-root>/</context-root>`. For more information, see Setting the context root of a web application.

Do not deploy your .war or .jar using FTP. The FTP tool is designed to upload startup scripts, dependencies, or other runtime files. It is not the optimal choice for deploying web apps.

Logging and debugging apps

Performance reports, traffic visualizations, and health checkups are available for each app through the Azure portal. For more information, see Azure App Service diagnostics overview.

Stream 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`.

For more information, see Stream logs in Cloud Shell.

SSH console access

To make open a direct SSH session with your container, your app should be running.

Paste the following URL into your browser and replace `<app-name>` with your app name:

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

If you're not yet authenticated, you're required to authenticate with your Azure subscription to connect. Once authenticated, you see an in-browser shell, where you can run commands inside your container.

> **Note**
>
> Any changes you make outside the */home* directory are stored in the container itself and don't persist beyond an app restart.

To open a remote SSH session from your local machine, see Open SSH session from remote shell.

Troubleshooting 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 this 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.

App logging

Enable application logging through the Azure portal or Azure CLI to configure App Service to write your application's standard console output and standard console error streams to the local filesystem or Azure Blob Storage. 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.

Azure Blob Storage logging for Linux based App Services can only be configured using Azure Monitor

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 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` for Java SE or `CATALINA_OPTS` for Tomcat 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

```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'll need to restart the application for the setting to take effect.

Turn on web socket support using the Azure CLI with the following command:

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>
```

Pre-Compile JSP files

To improve performance of Tomcat applications, you can compile your JSP files before deploying to App Service. You can use the Maven plugin provided by Apache Sling, or using this Ant build file.

Secure applications

Java applications running in App Service have the same set of security best practices as other applications.

Authenticate users (Easy Auth)

Set up app authentication in the Azure portal with the **Authentication and Authorization** option. From there, you can enable authentication using Azure Active Directory or social logins like Facebook, Google, or GitHub. Azure portal configuration only works when configuring a single authentication provider. For more information, see Configure your App Service app to use Azure Active Directory login and the related articles for other identity providers. If you need to enable multiple sign-in providers, follow the instructions in the customize sign-ins and sign-outs article.

Java SE

Spring Boot developers can use the Azure Active Directory Spring Boot starter to secure applications using familiar Spring Security annotations and APIs. Be sure to increase the maximum header size in your *application.properties* file. We suggest a value of `16384`.

Tomcat

Your Tomcat application can access the user's claims directly from the servlet by casting the Principal object to a Map object. The Map object will map each claim type to a collection of the claims for that type. In the code below, `request` is an instance of `HttpServletRequest`.

```Java
Map<String, Collection<String>> map = (Map<String, Collection<String>>)
request.getUserPrincipal();
```

Now you can inspect the `Map` object for any specific claim. For example, the following code snippet iterates through all the claim types and prints the contents of each collection.

```Java
for (Object key : map.keySet()) {
        Object value = map.get(key);
        if (value != null && value instanceof Collection {
            Collection claims = (Collection) value;
            for (Object claim : claims) {
                System.out.println(claims);
            }
        }
    }
```

To sign out users, use the `/.auth/ext/logout` path. To perform other actions, see the documentation on Customize sign-ins and sign-outs. There is also official documentation on the Tomcat HttpServletRequest interface and its methods. The following servlet methods are also hydrated based on your App Service configuration:

```Java
public boolean isSecure()
public String getRemoteAddr()
public String getRemoteHost()
public String getScheme()
public int getServerPort()
```

To disable this feature, create an Application Setting named `WEBSITE_AUTH_SKIP_PRINCIPAL` with a value of `1`. To disable all servlet filters added by App Service, create a setting named `WEBSITE_SKIP_FILTERS` with a value of `1`.

Configure TLS/SSL

Follow the instructions in the Secure a custom DNS name with an TLS/SSL binding in Azure App Service to upload an existing TLS/SSL certificate and bind it to your application's domain name. By default your application will still allow HTTP connections-follow the specific steps in the tutorial to enforce 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 Key Vault and making a KeyVault reference to your secret in an Application Setting. You can validate that the

reference resolves to the secret by printing the environment variable while remotely accessing the App Service terminal.

To inject these secrets in your Spring or Tomcat configuration file, use environment variable injection syntax (`${MY_ENV_VAR}`). For Spring configuration files, see this documentation on externalized configurations.

Use the Java Key Store

By default, any public or private certificates uploaded to App Service Linux will be loaded into the respective Java Key Stores as the container starts. After uploading your certificate, you will need to restart your App Service for it to be loaded into the Java Key Store. Public certificates are loaded into the Key Store at `$JRE_HOME/lib/security/cacerts`, and private certificates are stored in `$JRE_HOME/lib/security/client.jks`.

More configuration may 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
- MySQL
- MongoDB
- Cassandra

Initialize the Java Key Store

To initialize the `import java.security.KeyStore` object, load the keystore file with the password. The default password for both key stores is `changeit`.

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

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, refer to the official documentation.

Configure APM platforms

This section shows how to connect Java applications deployed on Azure App Service with Azure Monitor Application Insights, NewRelic, and AppDynamics application performance monitoring (APM) platforms.

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, auto-collecting logs, metrics, and distributed traces, eliminating the need for including an SDK in your app. See the Application Insights documentation for more information about the available app settings for configuring the agent.

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 will be used. You can choose to use an existing application insights resource, or change the name. Click **Apply** at the bottom

Azure CLI

To enable via the Azure CLI, you will need to create an Application Insights resource and set a couple app settings on the Azure portal to connect Application Insights to your web app.

1. Enable the Applications Insights extension

   Azure CLI

   ```
   az extension add -n application-insights
   ```

2. Create an Application Insights resource using the CLI command below. Replace the placeholders with your desired resource name and group.

   Azure CLI

   ```
   az monitor app-insights component create --app <resource-name> -g
   <resource-group> --location westus2  --kind web --application-type web
   ```

   Note the values for `connectionString` and `instrumentationKey`, you will need these values in the next step.

   > To retrieve a list of other locations, run `az account list-locations`.

3. Set the instrumentation key, connection string, and monitoring agent version as app settings on the web app. Replace `<instrumentationKey>` and `<connectionString>` with the values from the previous step.

   Azure CLI

   ```
   az webapp config appsettings set -n <webapp-name> -g <resource-group> -
   -settings "APPINSIGHTS_INSTRUMENTATIONKEY=<instrumentationKey>"
   "APPLICATIONINSIGHTS_CONNECTION_STRING=<connectionString>"
   "ApplicationInsightsAgent_EXTENSION_VERSION=~3"
   "XDT_MicrosoftApplicationInsights_Mode=default"
   ```

Configure New Relic

1. Create a NewRelic account at NewRelic.com

2. Download the Java agent from NewRelic, it will have a file name similar to *newrelic-java-x.x.x.zip*.

3. Copy your license key, you'll need it to configure the agent later.

4. SSH into your App Service instance and create a new directory */home/site/wwwroot/apm*.

5. Upload the unpacked NewRelic Java agent files into a directory under */home/site/wwwroot/apm*. The files for your agent should be in */home/site/wwwroot/apm/newrelic*.

6. Modify the YAML file at */home/site/wwwroot/apm/newrelic/newrelic.yml* and replace the placeholder license value with your own license key.

7. In the Azure portal, browse to your application in App Service and create a new Application Setting.

   - For **Java SE** apps, create an environment variable named `JAVA_OPTS` with the value `-javaagent:/home/site/wwwroot/apm/newrelic/newrelic.jar`.
   - For **Tomcat**, create an environment variable named `CATALINA_OPTS` with the value `-javaagent:/home/site/wwwroot/apm/newrelic/newrelic.jar`.

> If you already have an environment variable for `JAVA_OPTS` or `CATALINA_OPTS`, append the `-javaagent:/...` option to the end of the current value.

Configure AppDynamics

1. Create an AppDynamics account at AppDynamics.com

2. Download the Java agent from the AppDynamics website, the file name will be similar to *AppServerAgent-x.x.x.xxxxx.zip*

3. SSH into your App Service instance and create a new directory */home/site/wwwroot/apm*.

4. Upload the Java agent files into a directory under */home/site/wwwroot/apm*. The files for your agent should be in */home/site/wwwroot/apm/appdynamics*.

5. In the Azure portal, browse to your application in App Service and create a new Application Setting.

   - For **Java SE** apps, 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.
   - For **Tomcat** apps, create an environment variable named `CATALINA_OPTS` with the value `-javaagent:/home/site/wwwroot/apm/appdynamics/javaagent.jar -`

`Dappdynamics.agent.applicationName=<app-name>` where `<app-name>` is your App Service name.

> **Note**
>
> If you already have an environment variable for `JAVA_OPTS` or `CATALINA_OPTS`, append the `-javaagent:/...` option to the end of the current value.

Configure data sources

Java SE

To connect to data sources in Spring Boot applications, we suggest creating connection strings and injecting them into your *application.properties* file.

1. In the "Configuration" section of the App Service page, set a name for the string, paste your JDBC connection string into the value field, and set the type to "Custom". You can optionally set this connection string as slot setting.

   This connection string is accessible to our application as an environment variable named `CUSTOMCONNSTR_<your-string-name>`. For example, the connection string we created above will be named `CUSTOMCONNSTR_exampledb`.

2. In your *application.properties* file, reference this connection string with the environment variable name. For our example, we would use the following.

   ```yml
   app.datasource.url=${CUSTOMCONNSTR_exampledb}
   ```

See the Spring Boot documentation on data access and externalized configurations for more information on this topic.

Tomcat

These instructions apply to all database connections. You will need to fill placeholders with your chosen database's driver class name and JAR file. Provided is a table with class names and driver downloads for common databases.

Database	Driver Class Name	JDBC Driver

Database	Driver Class Name	JDBC Driver
PostgreSQL	`org.postgresql.Driver`	Download
MySQL	`com.mysql.jdbc.Driver`	Download (Select "Platform Independent")
SQL Server	`com.microsoft.sqlserver.jdbc.SQLServerDriver`	Download

To configure Tomcat to use Java Database Connectivity (JDBC) or the Java Persistence API (JPA), first customize the `CATALINA_OPTS` environment variable that is read in by Tomcat at start-up. Set these values through an app setting in the App Service Maven plugin:

```xml
<appSettings>
    <property>
        <name>CATALINA_OPTS</name>
        <value>"$CATALINA_OPTS -Ddbuser=${DBUSER} -Ddbpassword=${DBPASSWORD}
-DconnURL=${CONNURL}"</value>
    </property>
</appSettings>
```

Or set the environment variables in the **Configuration** > **Application Settings** page in the Azure portal.

Next, determine if the data source should be available to one application or to all applications running on the Tomcat servlet.

Application-level data sources

1. Create a *context.xml* file in the *META-INF/* directory of your project. Create the *META-INF/* directory if it does not exist.

2. In *context.xml*, add a `Context` element to link the data source to a JNDI address. Replace the `driverClassName` placeholder with your driver's class name from the table above.

   ```xml
   <Context>
       <Resource
           name="jdbc/dbconnection"
           type="javax.sql.DataSource"
           url="${connURL}"
   ```

```
        driverClassName="<insert your driver class name>"
        username="${dbuser}"
        password="${dbpassword}"
    />
</Context>
```

3. Update your application's *web.xml* to use the data source in your application.

```XML
<resource-env-ref>
    <resource-env-ref-name>jdbc/dbconnection</resource-env-ref-name>
    <resource-env-ref-type>javax.sql.DataSource</resource-env-ref-type>
</resource-env-ref>
```

Shared server-level resources

Adding a shared, server-level data source will require you to edit Tomcat's server.xml. First, upload a startup script and set the path to the script in **Configuration** > **Startup Command**. You can upload the startup script using FTP.

Your startup script will make an xsl transform to the server.xml file and output the resulting xml file to `/usr/local/tomcat/conf/server.xml`. The startup script should install libxslt via apk. Your xsl file and startup script can be uploaded via FTP. Below is an example startup script.

```sh
# Install libxslt. Also copy the transform file to /home/tomcat/conf/
apk add --update libxslt

# Usage: xsltproc --output output.xml style.xsl input.xml
xsltproc --output /home/tomcat/conf/server.xml
/home/tomcat/conf/transform.xsl /usr/local/tomcat/conf/server.xml
```

An example xsl file is provided below. The example xsl file adds a new connector node to the Tomcat server.xml.

```XML
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:output method="xml" indent="yes"/>

  <xsl:template match="@* | node()" name="Copy">
    <xsl:copy>
```

```
      <xsl:apply-templates select="@* | node()"/>
    </xsl:copy>
  </xsl:template>

  <xsl:template match="@* | node()" mode="insertConnector">
    <xsl:call-template name="Copy" />
  </xsl:template>

  <xsl:template match="comment()[not(../Connector[@scheme = 'https']) and
                                 contains(., '&lt;Connector') and
                                 (contains(., 'scheme=&quot;https&quot;') or
                                  contains(.,
&quot;scheme='https'&quot;))]">
    <xsl:value-of select="." disable-output-escaping="yes" />
  </xsl:template>

  <xsl:template match="Service[not(Connector[@scheme = 'https'] or
                                   comment()[contains(., '&lt;Connector')
and
                                             (contains(.,
'scheme=&quot;https&quot;') or
                                              contains(.,
&quot;scheme='https'&quot;))]
                                 )]
                      ">
    <xsl:copy>
      <xsl:apply-templates select="@* | node()" mode="insertConnector" />
    </xsl:copy>
  </xsl:template>

  <!-- Add the new connector after the last existing Connnector if there is
one -->
  <xsl:template match="Connector[last()]" mode="insertConnector">
    <xsl:call-template name="Copy" />

    <xsl:call-template name="AddConnector" />
  </xsl:template>

  <!-- ... or before the first Engine if there is no existing Connector -->
  <xsl:template match="Engine[1][not(preceding-sibling::Connector)]"
                mode="insertConnector">
    <xsl:call-template name="AddConnector" />

    <xsl:call-template name="Copy" />
  </xsl:template>

  <xsl:template name="AddConnector">
    <!-- Add new line -->
    <xsl:text>&#xa;</xsl:text>
    <!-- This is the new connector -->
    <Connector port="8443" protocol="HTTP/1.1" SSLEnabled="true"
               maxThreads="150" scheme="https" secure="true"
               keystroreFile="${{user.home}}/.keystore"
keystorePass="changeit"
               clientAuth="false" sslProtocol="TLS" />
```

```
  </xsl:template>

</xsl:stylesheet>
```

Finalize configuration

Finally, place the driver JARs in the Tomcat classpath and restart your App Service.

1. Ensure that the JDBC driver files are available to the Tomcat classloader by placing them in the */home/tomcat/lib* directory. (Create this directory if it does not already exist.) To upload these files to your App Service instance, perform the following steps:
   a. In the Cloud Shell, install the webapp extension:

   Azure CLI

   ```
   az extension add --name webapp
   ```

   b. Run the following CLI command to create an SSH tunnel from your local system to App Service:

   Azure CLI

   ```
   az webapp remote-connection create --resource-group <resource-group-name> --name <app-name> --port <port-on-local-machine>
   ```

   c. Connect to the local tunneling port with your SFTP client and upload the files to the */home/tomcat/lib* folder.

   Alternatively, you can use an FTP client to upload the JDBC driver. Follow these instructions for getting your FTP credentials.

2. If you created a server-level data source, restart the App Service Linux application. Tomcat will reset `CATALINA_BASE` to `/home/tomcat` and use the updated configuration.

JBoss EAP 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. 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.

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.

XML

```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.

Bash

```bash
#!/usr/bin/env 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.PGXADataSource)

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
SWORD: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.PostgreSQLException
nSorter --jta=true --use-java-context=true --valid-connection-checker-
class-
name=org.jboss.jca.adapters.jdbc.extensions.postgres.PostgreSQLValidCon
nectionChecker
```

4. Create a startup script, `startup_script.sh` that calls the JBoss CLI commands. The example below shows how to call your `jboss-cli-commands.cli`. Later you will configure App Service to run this script when the container 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`. **Save** your changes.

To confirm that the datasource was added to the JBoss server, SSH into your webapp and run `$JBOSS_HOME/bin/jboss-cli.sh --connect`. Once you are connected to JBoss run the `/subsystem=datasources:read-resource` to print a list of the data sources.

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 sites should use pinned patch JVM versions. This will prevent unanticipated outages during a patch version auto-update. All Java web apps use 64-bit JVMs, this is not configurable.

If you are using Tomcat, you can choose to pin the patch version of Tomcat. On Windows, you can pin the patch versions of the JVM and Tomcat independently. On Linux, you can pin the patch version of Tomcat; the patch version of the JVM will also be pinned but is not separately configurable.

If you choose to pin the minor version, you will need to periodically update the JVM minor version on the site. To ensure that your application runs on the newer minor version, create a staging slot and increment the minor version on the staging site. Once you have confirmed the application runs correctly on the new minor version, you can swap the staging and production slots.

JBoss EAP

Clustering in JBoss EAP

App Service supports clustering for JBoss EAP versions 7.4.1 and greater. To enable clustering, your web app must be integrated with a virtual network. When the web app is integrated with a virtual network, the web app will restart and JBoss EAP will automatically start up with a clustered configuration. The JBoss EAP instances will communicate over the subnet specified in the virtual network integration, using the ports shown in the `WEBSITES_PRIVATE_PORTS` environment variable at runtime. You can disable clustering by creating an app setting named `WEBSITE_DISABLE_CLUSTERING` with any value.

> ⓘ **Note**
>
> If you are enabling your virtual network integration with an ARM template, you will need to manually set the property `vnetPrivatePorts` to a value of `2`. If you enable virtual network integration from the CLI or Portal, this property will be set for you automatically.

When clustering is enabled, the JBoss EAP instances use the FILE_PING JGroups discovery protocol to discover new instances and persist the cluster information like the cluster members, their identifiers, and their IP addresses. On App Service, these files are under `/home/clusterinfo/`. The first EAP instance to start will obtain read/write permissions on the cluster membership file. Other instances will read the file, find the

primary node, and coordinate with that node to be included in the cluster and added to the file.

The Premium V3 and Isolated V2 App Service Plan types can optionally be distributed across Availability Zones to improve resiliency and reliability for your business-critical workloads. This architecture is also known as zone redundancy. The JBoss EAP clustering feature is compatible with the zone redundancy feature.

Auto-Scale Rules

When configuring auto-scale rules for horizontal scaling it is important to remove instances incrementally (one at a time) to ensure each removed instance can transfer its activity (such as handling a database transaction) to another member of the cluster. When configuring your autoscale rules in the Portal to scale down, use the following options:

- **Operation**: "Decrease count by"
- **Cool down**: "5 minutes" or greater
- **Instance count**: 1

You do not need to incrementally add instances (scaling out), you can add multiple instances to the cluster at a time.

JBoss EAP App Service Plans

JBoss EAP is only available on the Premium v3 and Isolated v2 App Service Plan types. Customers that created a JBoss EAP site on a different tier during the public preview should scale up to Premium or Isolated hardware tier to avoid unexpected behavior.

Java runtime statement of support

JDK versions and maintenance

Microsoft and Adoptium builds of OpenJDK are provided and supported on App Service for Java 8, 11, and 17. 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. The table below describes the new Java versions included in the January 2022 App Service platform release:

Java Version	Linux	Windows
Java 8	1.8.0_312 (Adoptium) *	1.8.0_312 (Adoptium)
Java 11	11.0.13 (Microsoft)	11.0.13 (Microsoft)
Java 17	17.0.1 (Microsoft)	17.0.1 (Microsoft)

* In following releases, Java 8 on Linux will be distributed from Adoptium builds of the OpenJDK.

If you are pinned to an older minor version of Java, your site may be using the deprecated Azul Zulu for Azure binaries provided through Azul Systems. You can continue to use these binaries for your site, but any security patches or improvements will only be available 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 will be 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 will be released as soon as they become available in Microsoft builds of the OpenJDK. A "major" vulnerability is defined by a base score of 9.0 or higher on the NIST Common Vulnerability Scoring System, version 2.

Tomcat 8.0 has reached End of Life (EOL) as of September 30, 2018. While the runtime is still available on Azure App Service, Azure will not apply security updates to Tomcat 8.0. If possible, migrate your applications to Tomcat 8.5 or 9.0. Both Tomcat 8.5 and 9.0 are available on Azure App Service. See the official Tomcat site for more information.

Community support for Java 7 will terminate on July 29th, 2022 and Java 7 will be retired from App Service at that time. If you have a web app running on Java 7, please upgrade to Java 8 or 11 before July 29th.

Deprecation and retirement

If a supported Java runtime will be retired, Azure developers using the affected runtime will be given a deprecation notice at least six months before the runtime is retired.

- 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.

Development support

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.

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

Configure a Linux Ruby app for Azure App Service

Article • 03/03/2022

This article describes how Azure App Service runs Ruby apps in a Linux container, and how you can customize the behavior of App Service when needed. Ruby apps must be deployed with all the required gems.

This guide provides key concepts and instructions for Ruby developers who use a built-in Linux container in App Service. If you've never used Azure App Service, you should follow the Ruby quickstart and Ruby with PostgreSQL tutorial first.

Show Ruby version

To show the current Ruby version, run the following command in the Cloud Shell:

Azure CLI

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

To show all supported Ruby versions, run the following command in the Cloud Shell:

Azure CLI

```
az webapp list-runtimes --os linux | grep RUBY
```

You can run an unsupported version of Ruby by building your own container image instead. For more information, see use a custom Docker image.

Set Ruby version

Run the following command in the Cloud Shell to set the Ruby version to 2.7:

Azure CLI

```
az webapp config set --resource-group <resource-group-name> --name <app-name> --linux-fx-version "RUBY:2.7"
```

Note

If you see errors similar to the following during deployment time:

```
Your Ruby version is 2.7.3, but your Gemfile specified 2.3.1
```

or

```
rbenv: version `2.3.1' is not installed
```

It means that the Ruby version configured in your project is different than the version that's installed in the container you're running (`2.7.3` in the example above). In the example above, check both *Gemfile* and *.ruby-version* and verify that the Ruby version is not set, or is set to the version that's installed in the container you're running (`2.7.3` in the example above).

Access environment variables

In App Service, you can set app settings outside of your app code. Then you can access them using the standard ENV['<path-name>'] pattern. For example, to access an app setting called `WEBSITE_SITE_NAME`, use the following code:

Ruby

```ruby
ENV['WEBSITE_SITE_NAME']
```

Customize deployment

When you deploy a Git repository, or a Zip package with build automation enabled, the deployment engine (Kudu) automatically runs the following post-deployment steps by default:

1. Check if a *Gemfile* exists.
2. Run `bundle clean`.
3. Run `bundle install --path "vendor/bundle"`.
4. Run `bundle package` to package gems into vendor/cache folder.

Use --without flag

To run `bundle install` with the --without flag, set the `BUNDLE_WITHOUT` app setting to a comma-separated list of groups. For example, the following command sets it to `development,test`.

Azure CLI

```
az webapp config appsettings set --name <app-name> --resource-group
<resource-group-name> --settings BUNDLE_WITHOUT="development,test"
```

If this setting is defined, then the deployment engine runs `bundle install` with `--without $BUNDLE_WITHOUT`.

Precompile assets

The post-deployment steps don't precompile assets by default. To turn on asset precompilation, set the `ASSETS_PRECOMPILE` app setting to `true`. Then the command `bundle exec rake --trace assets:precompile` is run at the end of the post-deployment steps. For example:

Azure CLI

```
az webapp config appsettings set --name <app-name> --resource-group
<resource-group-name> --settings ASSETS_PRECOMPILE=true
```

For more information, see Serve static assets.

Customize start-up

By default, the Ruby container starts the Rails server in the following sequence (for more information, see the start-up script):

1. Generate a secret_key_base value, if one doesn't exist already. This value is required for the app to run in production mode.
2. Set the `RAILS_ENV` environment variable to `production`.
3. Delete any *.pid* file in the *tmp/pids* directory that's left by a previously running Rails server.
4. Check if all dependencies are installed. If not, try installing gems from the local *vendor/cache* directory.
5. Run `rails server -e $RAILS_ENV`.

You can customize the start-up process in the following ways:

- Serve static assets
- Run in non-production mode
- Set secret_key_base manually

Serve static assets

The Rails server in the Ruby container runs in production mode by default, and assumes that assets are precompiled and are served by your web server. To serve static assets from the Rails server, you need to do two things:

- **Precompile the assets** - Precompile the static assets locally and deploy them manually. Or, let the deployment engine handle it instead (see Precompile assets.
- **Enable serving static files** - To serve static assets from the Ruby container, set the RAILS_SERVE_STATIC_FILES app setting to `true`. For example:

Azure CLI

```
az webapp config appsettings set --name <app-name> --resource-group
<resource-group-name> --settings RAILS_SERVE_STATIC_FILES=true
```

Run in non-production mode

The Rails server runs in production mode by default. To run in development mode, for example, set the `RAILS_ENV` app setting to `development`.

Azure CLI

```
az webapp config appsettings set --name <app-name> --resource-group
<resource-group-name> --settings RAILS_ENV="development"
```

However, this setting alone causes the Rails server to start in development mode, which accepts localhost requests only and isn't accessible outside of the container. To accept remote client requests, set the `APP_COMMAND_LINE` app setting to `rails server -b 0.0.0.0`. This app setting lets you run a custom command in the Ruby container. For example:

Azure CLI

```
az webapp config appsettings set --name <app-name> --resource-group
<resource-group-name> --settings APP_COMMAND_LINE="rails server -b 0.0.0.0"
```

Set secret_key_base manually

To use your own `secret_key_base` value instead of letting App Service generate one for you, set the `SECRET_KEY_BASE` app setting with the value you want. For example:

Azure CLI

```
az webapp config appsettings set --name <app-name> --resource-group
<resource-group-name> --settings SECRET_KEY_BASE="<key-base-value>"
```

Access diagnostic logs

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

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`.

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.

> **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.

More resources

- Tutorial: Rails app with PostgreSQL
- App Service Linux FAQ
- Environment variables and app settings reference

Deploy files to App Service

Article • 07/21/2023

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

> **Note**
>
> If you downloaded the files in a ZIP package, extract the files first. For example, if you downloaded a ZIP package from GitHub, you cannot deploy that file as-is. GitHub adds additional nested directories, which do not work with App Service.

In a local terminal window, navigate to the root directory of your app project.

This directory should contain the entry file to your web app, such as *index.html*, *index.php*, and *app.js*. It can also contain package management files like *project.json*, *composer.json*, *package.json*, *bower.json*, and *requirements.txt*.

Unless you want App Service to run deployment automation for you, run all the build tasks (for example, `npm`, `bower`, `gulp`, `composer`, and `pip`) and make sure that you have all the files you need to run the app. This step is required if you want to run your package directly.

Create a ZIP archive of everything in your project. For `dotnet` projects, this folder is the output folder of the `dotnet publish` command. The following command uses the default tool in your terminal:

```
# Bash
zip -r <file-name>.zip .

# PowerShell
Compress-Archive -Path * -DestinationPath <file-name>.zip
```

Deploy 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.

For more information, see Kudu documentation.

> **Note**
>
> Files in the ZIP package are copied only if their timestamps don't match what is already deployed. Generating a zip using a build process that caches outputs can result in faster deployments. See **Deploying from a zip file or url**, for more information.

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.

Depending on your web apps's networking configuration, direct access to the site from your local environment may be blocked. To deploy your code in this scenario, you can publish your ZIP to a storage system accessible from the web app and trigger the app to *pull* the ZIP from the storage location, instead of *pushing* the ZIP to the web app. See this article on deploying to network secured web apps for more information.

The following example uses the `--src-url` parameter to specify the URL of an Azure Storage account that the site should pull the ZIP from.

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=slk22f3UrS823n4kSh8Skjpa7Naj4CG3
```

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.

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.

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 places the package on the shared file drive correctly (see Kudu publish API reference). For that reason, deploying WAR/JAR/EAR packages using FTP or WebDeploy is not recommended.

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
```

Depending on your web apps's networking configuration, direct access to the site from your local environment may be blocked. To deploy your code in this scenario, you can publish your ZIP to a storage system accessible from the web app and trigger the app to *pull* the ZIP from the storage

location, instead of *pushing* the ZIP to the web app. See this article on deploying to network secured web apps for more information.

The following example uses the `--src-url` parameter to specify the URL of an Azure Storage account that the web app should pull the WAR from.

Azure CLI

```
az webapp deploy --resource-group <group-name> --name <app-name> --src-url
"https://storagesample.blob.core.windows.net/sample-container/myapp.war?sv=2021-10-
01&sb&sig=slk22f3UrS823n4kSh8Skjpa7Naj4CG3 --type 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
```

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 API, you can use basic authentication with your app's deployment credentials.

The table below shows the available query parameters, their allowed values, and descriptions.

Key	Allowed values	Description	Required	Type
`type`	`war`|`jar`|`ear`|`lib`|`startup`|`static`|`zip`	The type of the artifact being deployed, this sets the default target path and informs the web app how the deployment should be handled. - `type=zip`: Deploy a ZIP package by unzipping the content to `/home/site/wwwroot`. `path` parameter is optional. - `type=war`: Deploy a WAR package. By default, the WAR package is deployed to `/home/site/wwwroot/app.war`. The target path can be specified with `path`. - `type=jar`: Deploy a JAR package to `/home/site/wwwroot/app.jar`. The `path` parameter is ignored - `type=ear`: Deploy an EAR package to `/home/site/wwwroot/app.ear`. The `path` parameter is ignored - `type=lib`: Deploy a JAR library file. By default, the file is deployed to `/home/site/libs`. The target path can be specified with `path`. - `type=static`: Deploy a static file (e.g. a script). By default, the file is deployed to `/home/site/wwwroot`. - `type=startup`: Deploy a script that App Service automatically uses as the startup script for your app. By default, the script is deployed to `D:\home\site\scripts\<name-of-source>` for Windows and `home/site/wwwroot/startup.sh` for Linux. The target path can be specified with `path`.	Yes	String
`restart`	`true`|`false`	By default, the API restarts the app following the deployment operation (`restart=true`). To deploy multiple artifacts, prevent restarts on the all but the final deployment by setting `restart=false`.	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 `WEBSITE_STACK` environment variable to choose safe	No	Boolean

Key	Allowed values	Description	Required	Type
		defaults depending on your site's language stack. Setting this parameter to `false` disables any language-specific defaults.		
`path`	`"<absolute-path>"`	The absolute path to deploy the artifact to. For example, `"/home/site/deployments/tools/driver.jar"`, `"/home/site/scripts/helper.sh"`.	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
- Azure App Service Deployment Credentials
- Environment variables and app settings reference

Deploy your app to Azure App Service using FTP/S

Article • 03/09/2023

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**
>
> The **Development Center (Classic)** page in the Azure portal, which is the old deployment experience, will be deprecated in March, 2021. This change will not affect any existing deployment settings in your app, and you can continue to manage app deployment in the **Deployment Center** page.

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:

Application-scope	User-scope
`<app-name>\$<app-name>`	`<app-name>\<deployment-user>`

   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, click **Save**. If using **FTPS Only**, you must enforce TLS 1.2 or higher by navigating to the **TLS/SSL**

settings blade of your web app. TLS 1.0 and 1.1 are not supported with **FTPS Only**.

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?

How can I troubleshoot FTP deployment?

The first step for troubleshooting FTP deployment is isolating a deployment issue from a runtime application issue.

A deployment issue typically results in no files or wrong files deployed to your app. You can troubleshoot by investigating your FTP deployment or selecting an alternate deployment path (such as source control).

A runtime application issue typically results in the right set of files deployed to your app but incorrect app behavior. You can troubleshoot by focusing on code behavior at runtime and investigating specific failure paths.

To determine a deployment or runtime issue, see Deployment vs. runtime issues.

I'm not able to FTP and publish my code. How can I resolve the issue?

Check that you've entered the correct hostname and credentials. Check also that the following FTP ports on your machine are not blocked by a firewall:

- FTP control connection port: 21, 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.

How can I determine the method that was used to deploy my Azure App Service?

Let us say you take over owning an app and you wish to find out how the Azure App Service was deployed so you can make changes and deploy them. You can determine how an Azure App Service was deployed by checking the application settings. If the app was deployed using an external package URL, you will see the WEBSITE_RUN_FROM_PACKAGE setting in the application settings with a URL value. Or if it was deployed using zip deploy, you will see the WEBSITE_RUN_FROM_PACKAGE setting with a value of 1. If the app was deployed using Azure DevOps, you will see the deployment history in the Azure DevOps portal. If Azure Functions Core Tools was used, you will 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 • 03/24/2022

Azure App Service enables continuous deployment from GitHub, Bitbucket, and Azure Repos repositories by pulling in the latest updates.

> **Note**
>
> The **Development Center (Classic)** page in the Azure portal, an earlier version of the deployment functionality, was deprecated in March 2021. This change doesn't affect existing deployment settings in your app, and you can continue to manage app deployment from the **Deployment Center** page in the portal.

Prepare your repository

To get automated builds from Azure App Service build server, make sure that your repository root has the correct files in your project.

Runtime	Root directory files
ASP.NET (Windows only)	**.sln*, **.csproj*, or *default.aspx*
ASP.NET Core	**.sln* or **.csproj*
PHP	*index.php*
Ruby (Linux only)	*Gemfile*
Node.js	*server.js*, *app.js*, or *package.json* with a start script
Python	**.py*, *requirements.txt*, or *runtime.txt*
HTML	*default.htm*, *default.html*, *default.asp*, *index.htm*, *index.html*, or *iisstart.htm*
WebJobs	*<job_name>/run.<extension>* under *App_Data/jobs/continuous* for continuous WebJobs, or *App_Data/jobs/triggered* for triggered WebJobs. For more information, see Kudu WebJobs documentation.

Runtime	Root directory files
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.

GitHub

4. GitHub Actions is the default build provider. To change the provider, select **Change provider** > **App Service Build Service** (Kudu) > **OK**.

> **Note**

To use Azure Pipelines as the build provider for your App Service app, configure CI/CD directly from Azure Pipelines. Don't configure it in App Service. The **Azure Pipelines** option just points you in the right direction.

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** 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.

7. When GitHub Actions is selected as the build provider, you can select the workflow file you want by using the **Runtime stack** and **Version** dropdown lists. Azure commits this workflow file into your selected GitHub repository to handle build and deploy tasks. To see the file before saving your changes, select **Preview file**.

   > **Note**
   >
   > App Service detects the **language stack setting** of your app and selects the most appropriate workflow template. If you choose a different template, it might deploy an app that doesn't run properly. For more information, see **How the GitHub Actions build provider works**.

8. 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 will continue to trigger deployment to your app. To delete the file from your repository, select **Delete workflow file**.

4. Select **OK**.

What happens to my app during deployment?

All the officially supported deployment methods make changes to the files in the */home/site/wwwroot* folder of your app. These files are used to run your app. 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.

How the GitHub Actions build provider works

The GitHub Actions build provider is an option for CI/CD from GitHub. 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.
- Adds the publishing profile for your app as a GitHub secret. The workflow file uses this secret to authenticate with App Service.
- Captures information from the workflow run logs and displays it on the **Logs** tab in your app's 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 a publishing profile, deploy by using a service principal in Azure Active Directory.

Authenticate by using a service principal

This optional configuration replaces the default authentication with publishing profiles in the generated workflow file.

1. Generate a service principal by using 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:

   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> \
                              --sdk-auth
   ```

   > **Important**
   >
   > For security, grant the minimum required access to the service principal. The scope in the previous example is limited to the specific App Service app and not the entire resource group.

2. Save the entire JSON output for the next step, including the top-level `{}`.

3. In GitHub, in your repository, select **Settings** > **Secrets** > **Add a new secret**.

4. Paste the entire JSON output from the Azure CLI command into the secret's value field. Give the secret a name like `AZURE_CREDENTIALS`.

5. In the workflow file generated by the Deployment Center, revise the `azure/webapps-deploy` step to look like the following example (which is modified from a Node.js workflow file):

YAML

```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'
    package: .
- name: Sign out of Azure.
  run: |
    az logout
```

Deploy from other repositories

For Windows apps, you can manually configure continuous deployment from a cloud Git or Mercurial repository that the portal doesn't directly support, like GitLab. You do that by selecting **External Git** in the **Source** dropdown list. For more information, see Set up continuous deployment using manual steps.

More resources

- Deploy from Azure Pipelines to Azure App Services
- Investigate common problems with continuous deployment
- Use Azure PowerShell
- Project Kudu

Local Git deployment to Azure App Service

Article • 04/24/2023

This how-to guide shows you how to deploy your app to Azure App Service from a Git repository on your local computer.

Prerequisites

To follow the steps in this how-to guide:

- If you don't have an Azure subscription, create 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.

Runtime	Root directory files
ASP.NET (Windows only)	*.sln*, *.csproj*, or *default.aspx*
ASP.NET Core	*.sln* or *.csproj*
PHP	*index.php*
Ruby (Linux only)	*Gemfile*

Runtime	Root directory files
Node.js	*server.js*, *app.js*, or *package.json* with a start script
Python	**.py*, *requirements.txt*, or *runtime.txt*
HTML	*default.htm*, *default.html*, *default.asp*, *index.htm*, *index.html*, or *iisstart.htm*
WebJobs	*<job_name>/run.<extension>* under *App_Data/jobs/continuous* for continuous WebJobs, or *App_Data/jobs/triggered* 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 haven't created 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 may see runtime-specific automation, such as MSBuild for ASP.NET, `npm install` for Node.js, and `pip install` for Python.

5. Browse to your app in the Azure portal to verify that the content is deployed.

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 may see the following common error messages when you use Git to publish to an App Service app in Azure:

Message	Cause	Resolution
`Unable to access '[siteURL]': Failed to connect to [scmAddress]`	The app isn't up and running.	Start the app in the Azure portal. Git deployment isn't available when the web app is stopped.
`Couldn't resolve host 'hostname'`	The address information for the 'azure' remote is incorrect.	Use the `git remote -v` command to list all remotes, along with the associated URL. Verify that the URL for the 'azure' remote is correct. If needed, remove and recreate this remote using the correct URL.
`No refs in common and none specified; doing nothing. Perhaps you should specify a branch such as 'main'.`	You didn't specify a branch during `git push`, or you haven't set the `push.default` value in `.gitconfig`.	Run `git push` again, specifying the main branch: `git push azure main`.
`Error - Changes committed to remote repository but deployment to website failed.`	You pushed a local branch that doesn't match the app deployment branch on 'azure'.	Verify that current branch is `master`. To change the default branch, use `DEPLOYMENT_BRANCH` application setting (see Change deployment branch).
`src refspec [branchname] does not match any.`	You tried to push to a branch other than main on the 'azure' remote.	Run `git push` again, specifying the main branch: `git push azure main`.
`RPC failed; result=22, HTTP code = 5xx.`	This error can happen if you try to push a large git repository over HTTPS.	Change the git configuration on the local machine to make the `postBuffer` bigger. For example: `git config --global http.postBuffer 524288000`.

Message	Cause	Resolution
`Error - Changes committed to remote repository but your web app not updated.`	You deployed a Node.js app with a *package.json* file that specifies additional required modules.	Review the `npm ERR!` error messages before this error for more context on the failure. The following are the known causes of this error, and the corresponding `npm ERR!` messages: **Malformed package.json file**: `npm ERR! Couldn't read dependencies.` **Native module doesn't have a binary distribution for Windows**: `npm ERR! \cmd "/c" "node-gyp rebuild"\ failed with 1` or `npm ERR! [modulename@version] preinstall: \make || gmake\`

Additional resources

- 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 • 10/18/2022

Azure DevOps Services | Azure DevOps Server 2020 | Azure DevOps Server 2019

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 (pre-packaged script). Learn about the key concepts and components that make up a pipeline.

You'll use the Azure Web App task 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.

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

Create your pipeline

The code examples in this section assume you are 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**.

Add the Azure Web App task

1. Use the Task assistant to add the Azure Web App task.

2. Select **Azure Resource Manager** for the **Connection type** and choose your **Azure subscription**. Make sure to **Authorize** your connection.
3. Select **Web App on Linux** and enter your `azureSubscription`, `appName`, and `package`. Your complete YAML should look like this.

```yaml
variables:
  buildConfiguration: 'Release'
```

```yaml
steps:
- script: dotnet build --configuration $(buildConfiguration)
  displayName: 'dotnet build $(buildConfiguration)'
- task: DotNetCoreCLI@2
  inputs:
    command: 'publish'
    publishWebProjects: true
- task: AzureWebApp@1
  inputs:
    azureSubscription: '<Azure service connection>'
    appType: 'webAppLinux'
    appName: '<Name of web app>'
    package: '$(System.DefaultWorkingDirectory)/**/*.zip'
```

- **azureSubscription**: your Azure subscription.
- **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.

Now you're ready to read through the rest of this topic to learn some of the more common changes that people make to customize an Azure Web App deployment.

Use the Azure Web App task

YAML

The Azure Web App Deploy task 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 allows you to modify configuration settings inside web packages and XML parameters files.

Deploy a Web Deploy package

To deploy a .zip Web Deploy package (for example, from an ASP.NET web app) to an Azure Web App, add the following snippet to your *azure-pipelines.yml* file:

YAML

```yaml
- task: AzureWebApp@1
  inputs:
    azureSubscription: '<Azure service connection>'
```

```
    appName: '<Name of web app>'
    package: $(System.DefaultWorkingDirectory)/**/*.zip
```

- **azureSubscription**: your Azure subscription.
- **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.

The snippet assumes that the build steps in your YAML file produce the zip archive in the `$(System.DefaultWorkingDirectory)` folder on your agent.

For information on Azure service connections, see the following section.

Deploy a .NET app

if you're building a .NET Core app, use the following snipped to deploy the build to an app.

YAML

```yaml
variables:
  buildConfiguration: 'Release'

steps:
- script: dotnet build --configuration $(buildConfiguration)
  displayName: 'dotnet build $(buildConfiguration)'
- task: DotNetCoreCLI@2
  inputs:
    command: 'publish'
    publishWebProjects: true
- task: AzureWebApp@1
  inputs:
    azureSubscription: '<Azure service connection>'
    appType: 'webAppLinux'
    appName: '<Name of web app>'
    package: '$(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.

Use a service connection

To deploy to Azure App Service, you'll need to use an Azure Resource Manager service connection. The Azure service connection stores the credentials to connect from Azure Pipelines or Azure DevOps Server to Azure.

Learn more about Azure Resource Manager service connections. If your service connection is not working as expected, see Troubleshooting service connections.

YAML

You'll need an Azure service connection for the `AzureWebApp` task. The Azure service connection stores the credentials to connect from Azure Pipelines to Azure. See Create an Azure service connection.

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 `AzureRmWebAppDeployment` task:

YAML

```yaml
- task: AzureRmWebAppDeployment@4
  inputs:
    VirtualApplication: '<name of virtual application>'
```

- **VirtualApplication**: the name of the Virtual Application that has been configured in the Azure portal. See Configure an App Service app in the Azure portal for more details.

Deploy to a slot

YAML

You can configure the Azure Web App to have multiple slots. Slots allow you to safely deploy your app and test it before making it available to your customers.

The following example shows how to deploy to a staging slot, and then swap to a production slot:

YAML

```yaml
- task: AzureWebApp@1
  inputs:
    azureSubscription: '<Azure service connection>'
    appType: webAppLinux
    appName: '<name of web app>'
    deployToSlotOrASE: true
    resourceGroupName: '<name of resource group>'
    slotName: staging
    package: '$(Build.ArtifactStagingDirectory)/**/*.zip'

- task: AzureAppServiceManage@0
  inputs:
    azureSubscription: '<Azure service connection>'
    appType: webAppLinux
    WebAppName: '<name of web app>'
    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, 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.

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

```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: '<Azure service connection>'
      appType: <app type>
      appName: '<name of test stage web app>'
      deployToSlotOrASE: true
      resourceGroupName: <resource 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: '<Azure service connection>'
      appType: <app type>
      appName: '<name of test stage web app>'
      resourceGroupName: <resource group name>
      package: '$(Pipeline.Workspace)/**/*.zip'
```

Make configuration changes

For most language stacks, app settings and connection strings can be set as environment variables at runtime.

App settings can also be resolved from Key Vault using Key Vault references.

For ASP.NET and ASP.NET Core developers, setting app settings in App Service are like setting them in `<appSettings>` in Web.config. You might want to apply a specific configuration for your web app target before deploying to it. This is useful when you deploy the same build to multiple web apps in a pipeline. For example, if your Web.config file contains a connection string named `connectionString`, you can change its value before deploying to each web app. You can do this either by applying a Web.config transformation or by substituting variables in your Web.config file.

Azure App Service Deploy task allows users to modify configuration settings in configuration files (*.config files) inside web packages and XML parameters files (parameters.xml), based on the stage name specified.

> **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.

Variable substitution

YAML

The following snippet shows an example of variable substitution:

YAML

```yaml
jobs:
- job: test
  variables:
    connectionString: <test-stage connection string>
  steps:
  - task: AzureRmWebAppDeployment@4
    inputs:
      azureSubscription: '<Test stage Azure service connection>'
      WebAppName: '<name of test stage web app>'
```

```
      enableXmlVariableSubstitution: true

  - job: prod
    dependsOn: test
    variables:
      connectionString: <prod-stage connection string>
    steps:
    - task: AzureRmWebAppDeployment@4
      inputs:
        azureSubscription: '<Prod stage Azure service connection>'
        WebAppName: '<name of prod stage web app>'
        enableXmlVariableSubstitution: true
```

Deploying conditionally

YAML

To do this in YAML, you can use one of these 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: '<Azure service connection>'
    appName: '<name of web app>'
```

To learn more about conditions, see Specify conditions.

(Classic) Deploy a release pipeline

You can use a release pipeline to pick up the artifacts published by your build and then deploy them to your Azure web site.

1. Do one of the following to start creating a release pipeline:

- If you've just completed a CI build, choose the link (for example, *Build 20170815.1*) to open the build summary. Then choose **Release** to start a new release pipeline that's automatically linked to the build pipeline.

- Open the **Releases** tab in **Azure Pipelines**, open the + drop-down in the list of release pipelines, and choose **Create release pipeline**.

2. The easiest way to create a release pipeline is to use a template. If you are deploying a Node.js app, select the **Deploy Node.js App to Azure App Service** template. Otherwise, select the **Azure App Service Deployment** template. Then choose **Apply**.

> ⓘ **Note**
>
> The only difference between these templates is that Node.js template configures the task to generate a **web.config** file containing a parameter that starts the **iisnode** service.

3. If you created your new release pipeline from a build summary, check that the build pipeline and artifact is shown in the **Artifacts** section on the **Pipeline** tab. If you created a new release pipeline from the **Releases** tab, choose the **+ Add** link and select your build artifact.

4. Choose the **Continuous deployment** icon in the **Artifacts** section, check that the continuous deployment trigger is enabled, and add a filter to include the **main** branch.

> ⓘ **Note**
>
> Continuous deployment is not enabled by default when you create a new release pipeline from the **Releases** tab.

5. Open the **Tasks** tab and, with **Stage 1** selected, configure the task property variables as follows:

   - **Azure Subscription:** Select a connection from the list under **Available Azure Service Connections** or create a more restricted permissions connection to your Azure subscription. If you are using Azure Pipelines and if you see an **Authorize** button next to the input, click on it to authorize Azure Pipelines to connect to your Azure subscription. If you are using TFS or if you do not see the desired Azure subscription in the list of subscriptions, see Azure Resource Manager service connection to manually set up the connection.

- **App Service Name**: Select the name of the web app from your subscription.

> ⓘ **Note**
>
> Some settings for the tasks may have been automatically defined as **stage variables** when you created a release pipeline from a template. These settings cannot be modified in the task settings; instead you must select the parent stage item in order to edit these settings.

6. Save the release pipeline.

Create a release to deploy your app

You're now ready to create a release, which means to run the release pipeline with the artifacts produced by a specific build. This will result in deploying the build:

1. Choose + **Release** and select **Create a release**.
2. In the **Create a new release** panel, check that the artifact version you want to use is selected and choose **Create**.
3. Choose the release link in the information bar message. For example: "Release **Release-1** has been created".
4. In the pipeline view, choose the status link in the stages of the pipeline to see the logs and agent output.
5. After the release is complete, navigate to your site running in Azure using the Web App URL `http://{web_app_name}.azurewebsites.net`, and verify its contents.

Next steps

- Customize your Azure DevOps pipeline.

Tutorial: Use GitHub Actions to deploy to App Service and connect to a database

Article • 01/09/2023

Learn how to set up a GitHub Actions workflow to deploy a ASP.NET Core application with an Azure SQL Database backend. When you're finished, you have an ASP.NET app running in Azure and connected to SQL Database. You'll first use an ARM template to create resources.

This tutorial does not use containers. If you want to deploy to a containerized ASP.NET Core application, see Use GitHub Actions to deploy to App Service for Containers and connect to a database.

In this tutorial, you learn how to:

- ✓ Use a GitHub Actions workflow to add resources to Azure with a Azure Resource Manager template (ARM template)
- ✓ Use a GitHub Actions workflow to build an ASP.NET Core application

If you don't have an Azure subscription, create an Azure free account before you begin.

Prerequisites

To complete this tutorial, you'll need:

- 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.
  - A GitHub repository to store your Resource Manager templates and your workflow files. To create one, see Creating a new repository.

Download the sample

Fork the sample project in the Azure Samples repo.

```
https://github.com/Azure-Samples/dotnetcore-sqldb-ghactions
```

Create the resource group

Open the Azure Cloud Shell at https://shell.azure.com. You can alternately use the Azure CLI if you've installed it locally. (For more information on Cloud Shell, see the Cloud Shell Overview.)

Azure CLI

```
az group create --name {resource-group-name} --location {resource-group-location}
```

Generate deployment credentials

OpenID Connect is an authentication method that uses short-lived tokens. Setting up OpenID Connect with GitHub Actions is more complex process that offers hardened security.

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 will output JSON with an `appId` that is your `client-id`. Save the value to use as the `AZURE_CLIENT_ID` GitHub secret later.

   You'll 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` and will be used 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 will be tied to your default subscription. Replace `$subscriptionId` with your subscription ID, `$resourceGroupName` with your resource group name, and `$assigneeObjectId` with the generated `assignee-object-id`. Learn how to manage Azure subscriptions with the Azure CLI.

   Azure CLI

   ```
   az role assignment create --role contributor --scope
   /subscriptions/$subscriptionId/resourceGroups/$resourceGroupName --
   subscription $subscriptionId --assignee-object-id  $assigneeObjectId --
   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 **objectId (generated while creating app)** for your Active Directory application.
   - Set a value for `CREDENTIAL-NAME` to reference later.
   - Set the `subject`. The value of this is defined by GitHub depending on your workflow:
     - 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 rest --method POST --uri
   'https://graph.microsoft.com/beta/applications/<APPLICATION-OBJECT-
   ID>/federatedIdentityCredentials' --body '{"name":"<CREDENTIAL-
   NAME>","issuer":"https://token.actions.githubusercontent.com","subject"
   :"repo:organization/repository:ref:refs/heads/main","description":"Test
   ing","audiences":["api://AzureADTokenExchange"]}'
   ```

To learn how to create a Create an active directory application, service principal, and federated credentials in Azure portal, see Connect GitHub and Azure.

Configure the GitHub secret for authentication

You need to provide your application's **Client ID**, **Tenant ID**, and **Subscription ID** to the 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. In GitHub, go to your repository.
2. Go to **Settings** in the navigation menu.
3. Select **Security** > **Secrets and variables** > **Actions**.

4. Select **New repository secret**.
5. 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:

GitHub Secret	Active Directory Application
AZURE_CLIENT_ID	Application (client) ID
AZURE_TENANT_ID	Directory (tenant) ID
AZURE_SUBSCRIPTION_ID	Subscription ID

6. Save each secret by selecting **Add secret**.

Add GitHub secrets for your build

1. Create two new secrets in your GitHub repository for `SQLADMIN_PASS` and `SQLADMIN_LOGIN`. Make sure you choose a complex password, otherwise the create step for the SQL database server will fail. You won't be able to access this password again so save it separately.
2. Create an `AZURE_SUBSCRIPTION_ID` secret for your Azure subscription ID. If you do not know your subscription ID, use this command in the Azure Shell to find it. Copy the value in the `SubscriptionId` column.

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

Create Azure resources

The create Azure resources workflow runs an ARM template to deploy resources to Azure. The workflow:

- Checks out source code with the Checkout action.
- Logs into Azure with the Azure Login action and gathers environment and Azure resource information.
- Deploys resources with the Azure Resource Manager Deploy action.

To run the create Azure resources workflow:

1. Open the `infraworkflow.yml` file in `.github/workflows` within your repository.

2. Update the value of `AZURE_RESOURCE_GROUP` to your resource group name.

3. Set the input for `region` in your ARM Deploy actions to your region.
   a. Open `templates/azuredeploy.resourcegroup.parameters.json` and update the `rgLocation` property to your region.

4. Go to **Actions** and select **Run workflow**.

5. Verify that your action ran successfully by checking for a green checkmark on the **Actions** page.

6. After you've created your resources, go to **Actions**, select Create Azure Resources, disable the workflow.

Create a publish profile secret

1. In the Azure portal, open your new staging App Service (Slot) created with the `Create Azure Resources` workflow.
2. Select **Get Publish Profile**.
3. Open the publish profile file in a text editor and copy its contents.
4. Create a new GitHub secret for `AZURE_WEBAPP_PUBLISH_PROFILE`.

Build and deploy your app

To run the build and deploy workflow:

1. Open your `workflow.yaml` file in `.github/workflows` within your repository.
2. Verify that the environment variables for `AZURE_RESOURCE_GROUP`, `AZURE_WEBAPP_NAME`, `SQLSERVER_NAME`, and `DATABASE_NAME` match the ones in `infraworkflow.yml`.
3. Verify that your app deployed by visiting the URL in the Swap to production slot output. You should see a sample app, My TodoList App.

Clean up resources

If you no longer need your sample project, delete your resource group in the Azure portal and delete your repository on GitHub.

Next steps

Learn about Azure and GitHub integration

Deploy to App Service using GitHub Actions

Article • 02/21/2023

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.
- A working Azure App Service app.
  - .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

Workflow file overview

A workflow is defined by a YAML (.yml) file in the `/.github/workflows/` path in your repository. This definition contains the various steps and parameters that make up the workflow.

The file has three sections:

Section	Tasks
Authentication	1. Define a service principal or publish profile. 2. Create a GitHub secret.
Build	1. Set up the environment. 2. Build the web app.
Deploy	1. Deploy the web app.

Use the Deployment Center

You can quickly get started with GitHub Actions by using the App Service Deployment Center. This will automatically generate a workflow file based on your application stack

and commit it to your GitHub repository in the correct directory.

1. Navigate to your webapp in the Azure portal
2. On the left side, click **Deployment Center**
3. Under **Continuous Deployment (CI / CD)**, select **GitHub**
4. Next, select **GitHub Actions**
5. Use the dropdowns to select your GitHub repository, branch, and application stack
    - If the selected branch is protected, you can still continue to add the workflow file. Be sure to review your branch protections before continuing.
6. On the final screen, you can review your selections and preview the workflow file that will be committed to the repository. If the selections are correct, click **Finish**

This will commit the workflow file to the repository. The workflow to build and deploy your app will start immediately.

Set up a workflow manually

You can also deploy a workflow without using the Deployment Center. To do so, you will need to first generate deployment credentials.

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**.
3. Save the downloaded file. You'll use the contents of the file to create a GitHub secret.

> **Note**
>
> As of October 2020, Linux web apps will need the app setting `WEBSITE_WEBDEPLOY_USE_SCM` set to `true` **before downloading the publish profile**. This requirement will be removed in the future.

Configure the GitHub secret

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

```yaml
- uses: azure/webapps-deploy@v2
  with:
    publish-profile: ${{ secrets.AZURE_WEBAPP_PUBLISH_PROFILE }}
```

Set up the environment

Setting up the environment can be done using one of the setup actions.

Language	Setup Action
.NET	`actions/setup-dotnet`
ASP.NET	`actions/setup-dotnet`
Java	`actions/setup-java`
JavaScript	`actions/setup-node`
Python	`actions/setup-python`

The following examples show how to set up the environment for the different supported languages:

.NET

YAML

```yaml
    - name: Setup Dotnet 3.3.x
      uses: actions/setup-dotnet@v1
      with:
        dotnet-version: '3.3.x'
```

ASP.NET

YAML

```yaml
    - name: Install Nuget
      uses: nuget/setup-nuget@v1
      with:
        nuget-version: ${{ env.NUGET_VERSION}}
```

Java

YAML

```yaml
    - name: Setup Java 1.8.x
      uses: actions/setup-java@v1
      with:
        # If your pom.xml <maven.compiler.source> version is not in 1.8.x,
        # change the Java version to match the version in pom.xml
<maven.compiler.source>
        java-version: '1.8.x'
```

JavaScript

YAML

```yaml
env:
  NODE_VERSION: '14.x'                # set this to the node version to use

jobs:
  build-and-deploy:
    name: Build and Deploy
    runs-on: ubuntu-latest
    steps:
    - uses: actions/checkout@main
    - name: Use Node.js ${{ env.NODE_VERSION }}
      uses: actions/setup-node@v1
```

```
      with:
        node-version: ${{ env.NODE_VERSION }}
```

Python

YAML

```
    - name: Setup Python 3.x
      uses: actions/setup-python@v1
      with:
        python-version: 3.x
```

Build the web app

The process of building a web app and deploying to Azure App Service changes depending on the language.

The following examples show the part of the workflow that builds the web app, in different supported languages.

For all languages, you can set the web app root directory with `working-directory`.

.NET

The environment variable `AZURE_WEBAPP_PACKAGE_PATH` sets the path to your web app project.

YAML

```
- name: dotnet build and publish
  run: |
    dotnet restore
    dotnet build --configuration Release
    dotnet publish -c Release --property:PublishDir='${{
env.AZURE_WEBAPP_PACKAGE_PATH }}/myapp'
```

ASP.NET

You can restore NuGet dependencies and run msbuild with `run`.

YAML

```
- name: NuGet to restore dependencies as well as project-specific tools that
are specified in the project file
  run: nuget restore
```

```yaml
- name: Add msbuild to PATH
  uses: microsoft/setup-msbuild@v1.0.2

- name: Run msbuild
  run: msbuild .\SampleWebApplication.sln
```

Java

YAML

```yaml
- name: Build with Maven
  run: mvn package --file pom.xml
```

JavaScript

For Node.js, you can set `working-directory` or change for npm directory in `pushd`.

YAML

```yaml
- name: npm install, build, and test
  run: |
    npm install
    npm run build --if-present
    npm run test --if-present
  working-directory: my-app-folder # set to the folder with your app if it
is not the root directory
```

Python

YAML

```yaml
- name: Install dependencies
  run: |
    python -m pip install --upgrade pip
    pip install -r requirements.txt
```

Deploy to App Service

To deploy your code to an App Service app, use the `azure/webapps-deploy@v2` action. This action has four parameters:

Parameter	Explanation
app-name	(Required) Name of the App Service app

Parameter	Explanation
publish-profile	(Optional) Publish profile file contents with Web Deploy secrets
package	(Optional) Path to package or folder. The path can include *.zip, *.war, *.jar, or a folder to deploy
slot-name	(Optional) Enter an existing slot other than the production slot

Publish profile

.NET Core

Build and deploy a .NET Core app to Azure using an Azure publish profile. The `publish-profile` input references the `AZURE_WEBAPP_PUBLISH_PROFILE` secret that you created earlier.

YAML

```yaml
name: .NET Core CI

on: [push]

env:
  AZURE_WEBAPP_NAME: my-app-name    # 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: '3.1.x'           # set this to the dot net version to use

jobs:
  build:
    runs-on: ubuntu-latest

    steps:
      # Checkout the repo
      - uses: actions/checkout@main

      # Setup .NET Core SDK
      - name: Setup .NET Core
        uses: actions/setup-dotnet@v1
        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@v2
        with:
          app-name: ${{ env.AZURE_WEBAPP_NAME }} # Replace with your app
name
          publish-profile: ${{ secrets.AZURE_WEBAPP_PUBLISH_PROFILE  }}
# Define secret variable in repository settings as per action
documentation
          package: '${{ env.AZURE_WEBAPP_PACKAGE_PATH }}/myapp'
```

ASP.NET

Build and deploy an ASP.NET MVC app that uses NuGet and `publish-profile` for authentication.

YAML

```yaml
name: Deploy ASP.NET MVC App deploy to Azure Web App

on: [push]

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
  NUGET_VERSION: '5.3.x'           # set this to the dot net version to
use

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

    - uses: actions/checkout@main

    - name: Install Nuget
      uses: nuget/setup-nuget@v1
      with:
        nuget-version: ${{ env.NUGET_VERSION}}
    - name: NuGet to restore dependencies as well as project-specific
tools that are specified in the project file
      run: nuget restore

    - name: Add msbuild to PATH
```

```
      uses: microsoft/setup-msbuild@v1.0.2

    - name: Run MSBuild
      run: msbuild .\SampleWebApplication.sln

    - name: 'Run Azure webapp deploy action using publish profile
credentials'
      uses: azure/webapps-deploy@v2
      with:
        app-name: ${{ env.AZURE_WEBAPP_NAME }} # Replace with your app
name
        publish-profile: ${{ secrets.AZURE_WEBAPP_PUBLISH_PROFILE  }} #
Define secret variable in repository settings as per action
documentation
        package: '${{ env.AZURE_WEBAPP_PACKAGE_PATH
}}/SampleWebApplication/'
```

Java

Build and deploy a Java Spring app to Azure using an Azure publish profile. The `publish-profile` input references the `AZURE_WEBAPP_PUBLISH_PROFILE` secret that you created earlier.

YAML

```yaml
name: Java CI with Maven

on: [push]

jobs:
  build:

    runs-on: ubuntu-latest

    steps:
    - uses: actions/checkout@v2
    - name: Set up JDK 1.8
      uses: actions/setup-java@v1
      with:
        java-version: 1.8
    - name: Build with Maven
      run: mvn -B package --file pom.xml
      working-directory: my-app-path
    - name: Azure WebApp
      uses: Azure/webapps-deploy@v2
      with:
        app-name: my-app-name
        publish-profile: ${{ secrets.AZURE_WEBAPP_PUBLISH_PROFILE }}
        package: my/target/*.jar
```

To deploy a `war` instead of a `jar`, change the `package` value.

YAML

```yaml
    - name: Azure WebApp
      uses: Azure/webapps-deploy@v2
      with:
        app-name: my-app-name
        publish-profile: ${{ secrets.AZURE_WEBAPP_PUBLISH_PROFILE }}
        package: my/target/*.war
```

JavaScript

Build and deploy a Node.js app to Azure using the app's publish profile. The `publish-profile` input references the `AZURE_WEBAPP_PUBLISH_PROFILE` secret that you created earlier.

YAML

```yaml
# File: .github/workflows/workflow.yml
name: JavaScript CI

on: [push]

env:
  AZURE_WEBAPP_NAME: my-app-name   # set this to your application's name
  AZURE_WEBAPP_PACKAGE_PATH: 'my-app-path'      # set this to the path
to your web app project, defaults to the repository root
  NODE_VERSION: '14.x'                # set this to the node version to
use

jobs:
  build-and-deploy:
    name: Build and Deploy
    runs-on: ubuntu-latest
    steps:
    - uses: actions/checkout@main
    - name: Use Node.js ${{ env.NODE_VERSION }}
      uses: actions/setup-node@v1
      with:
        node-version: ${{ env.NODE_VERSION }}
    - name: npm install, build, and test
      run: |
        # Build and test the project, then
        # deploy to Azure Web App.
        npm install
        npm run build --if-present
        npm run test --if-present
      working-directory: my-app-path
    - name: 'Deploy to Azure WebApp'
```

```
      uses: azure/webapps-deploy@v2
      with:
        app-name: ${{ env.AZURE_WEBAPP_NAME }}
        publish-profile: ${{ secrets.AZURE_WEBAPP_PUBLISH_PROFILE }}
        package: ${{ env.AZURE_WEBAPP_PACKAGE_PATH }}
```

Python

Build and deploy a Python app to Azure using the app's publish profile. Note how the `publish-profile` input references the `AZURE_WEBAPP_PUBLISH_PROFILE` secret that you created earlier.

YAML

```yaml
name: Python CI

on:
  [push]

env:
  AZURE_WEBAPP_NAME: my-web-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

jobs:
  build:
    runs-on: ubuntu-latest
    steps:
    - uses: actions/checkout@v2
    - name: Set up Python 3.x
      uses: actions/setup-python@v2
      with:
        python-version: 3.x
    - name: Install dependencies
      run: |
        python -m pip install --upgrade pip
        pip install -r requirements.txt
    - name: Building web app
      uses: azure/appservice-build@v2
    - name: Deploy web App using GH Action azure/webapps-deploy
      uses: azure/webapps-deploy@v2
      with:
        app-name: ${{ env.AZURE_WEBAPP_NAME }}
        publish-profile: ${{ secrets.AZURE_WEBAPP_PUBLISH_PROFILE }}
        package: ${{ env.AZURE_WEBAPP_PACKAGE_PATH }}
```

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 login
- Azure WebApp
- Azure WebApp for containers
- Docker login/logout
- Events that trigger workflows
- K8s deploy
- Starter Workflows

Run your app in Azure App Service directly from a ZIP package

Article • 01/27/2023

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

> **Note**
>
> If you downloaded the files in a ZIP package, extract the files first. For example, if you downloaded a ZIP package from GitHub, you cannot deploy that file as-is. GitHub adds additional nested directories, which do not work with App Service.

In a local terminal window, navigate to the root directory of your app project.

This directory should contain the entry file to your web app, such as *index.html*, *index.php*, and *app.js*. It can also contain package management files like *project.json*, *composer.json*, *package.json*, *bower.json*, and *requirements.txt*.

Unless you want App Service to run deployment automation for you, run all the build tasks (for example, `npm`, `bower`, `gulp`, `composer`, and `pip`) and make sure that you have all the files you need to run the app. This step is required if you want to run your package directly.

Create a ZIP archive of everything in your project. For `dotnet` projects, this folder is the output folder of the `dotnet publish` command. The following command uses the default tool in your terminal:

```
# Bash
zip -r <file-name>.zip .

# PowerShell
Compress-Archive -Path * -DestinationPath <file-name>.zip
```

Enable running from package

The `WEBSITE_RUN_FROM_PACKAGE` app setting enables running from a package. To set it, run the following command with Azure CLI.

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 deployment source config-zip --resource-group <group-name> --name <app-name> --src <filename>.zip
```

Because the `WEBSITE_RUN_FROM_PACKAGE` app setting is set, this command doesn't extract the package content to the *D:\home\site\wwwroot* directory of your app. Instead, it uploads the ZIP file as-is to *D:\home\data\SitePackages*, and creates a *packagename.txt* in the same directory, that contains the name of the ZIP package to load at runtime. If you upload your ZIP package in a different way (such as FTP), you need to create the *D:\home\data\SitePackages* directory and the *packagename.txt* file manually.

The command also restarts the app. Because `WEBSITE_RUN_FROM_PACKAGE` is set, App Service mounts the uploaded package as the read-only *wwwroot* directory and runs the app directly from that mounted directory.

Run from external URL instead

You can also run a package from an external URL, such as Azure Blob Storage. You can use the Azure Storage Explorer to upload package files to your Blob storage account. You should use a private storage container with a Shared Access Signature (SAS) or use a managed identity to enable the App Service runtime to access the package securely.

Once you upload your file to Blob storage and have an SAS URL for the file, set the `WEBSITE_RUN_FROM_PACKAGE` app setting to the URL. The following example does it by using Azure CLI:

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=rl&sv=2017-04-
17&sr=b&sig=bNrVrEFzRHQB17GFJ7boEanetyJ9DGwBSV8OM3Mdh%2FM%3D"
```

If you publish an updated package with the same name to Blob storage, you need to restart your app so that the updated package is loaded into App Service.

Access a package in Azure Blob Storage using a managed identity

Azure Blob Storage can be configured to authorize requests with Azure AD. This 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 will be 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, although this is the same as omitting the setting altogether.

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 will likely be 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

Sync content from a cloud folder to Azure App Service

Article • 10/18/2022

This article shows you how to sync your content to Azure App Service from Dropbox and OneDrive.

With the content sync approach, your work with your app code and content in a designated cloud folder to make sure it's in a ready-to-deploy state, and then sync to App Service with the click of a button.

Because of underlying differences in the APIs, **OneDrive for Business** is not supported at this time.

> **Note**
>
> The **Development Center (Classic)** page in the Azure portal, which is the old deployment experience, will be deprecated in March, 2021. This change will not affect any existing deployment settings in your app, and you can continue to manage app deployment in the **Deployment Center** page.

Enable content sync deployment

1. In the Azure portal, navigate to the management page for your App Service app.
2. From the left menu, click **Deployment Center** > **Settings**.
3. In **Source**, select **OneDrive** or **Dropbox**.
4. Click **Authorize** and follow the authorization prompts.

You only need to authorize with OneDrive or Dropbox once for your Azure account. To authorize a different OneDrive or Dropbox account for an app, click **Change account**.

5. In **Folder**, select the folder to synchronize. This folder is created under the following designated content path in OneDrive or Dropbox.

   - **OneDrive**: `Apps\Azure Web Apps`
   - **Dropbox**: `Apps\Azure`

6. Click **Save**.

Synchronize content

Azure portal

1. In the Azure portal, navigate to the management page for your App Service app.

2. From the left menu, click **Deployment Center** > **Redeploy/Sync**.

3. Click **OK** to confirm the sync.

Disable content sync deployment

1. In the Azure portal, navigate to the management page for your App Service app.

2. From the left menu, click **Deployment Center** > **Settings** > **Disconnect**.

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.

OneDrive and Dropbox integration retirements

On September 30th, 2023 the integrations for Microsoft OneDrive and Dropbox for Azure App Service and Azure Functions will be retired. If you are using OneDrive or Dropbox, you should disable content sync deployments from OneDrive and Dropbox. Then, you can set up deployments from any of the following alternatives

- GitHub Actions
- Azure DevOps Pipelines
- Azure CLI
- VS Code
- Local Git Repository

Next steps

Deploy from local Git repo

Authentication types by deployment methods in Azure App Service

Article • 07/31/2023

Azure App Service lets you deploy your web application code and configuration by using multiple options. These deployment options may 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 **Configure deployment credentials**.

Deployment method	Authentication	Reference Documents
Azure CLI	Azure AD authentication	In Azure CLI, version 2.48.1 or higher, the following commands have been modified to use Azure AD authentication if basic authentication is turned off for your web app or function app: - az webapp up - az webapp deploy - az webapp deployment source config-zip - 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	Azure AD authentication	In Azure PowerShell, version 9.7.1 or above, Azure AD authentication is available for App Service. For more information, see PowerShell samples for Azure App Service.
SCM/Kudu/OneDeploy REST endpoint	Basic authentication, Azure AD authentication	Deploy files to App Service
Kudu UI	Basic authentication, Azure AD authentication	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	Azure AD authentication	Quickstart: Deploy an ASP.NET web app Working with GitHub in VS Code
GitHub with GitHub Actions	Publish profile, service principal, OpenID Connect	Deploy to App Service using GitHub Actions
GitHub with App Service build service as build engine	Publish profile	Continuous deployment to Azure App Service
GitHub with Azure Pipelines as build engine	Publish profile, Azure DevOps service connection	Deploy to App Service using Azure Pipelines
Azure Repos with App Service build service as build engine	Publish profile	Continuous deployment to Azure App Service

Deployment method	Authentication	Reference Documents
Azure Repos with Azure Pipelines as build engine	Publish profile, Azure DevOps service connection	Deploy to App Service using GitHub Actions
Bitbucket	Publish profile	Continuous deployment to Azure App Service
Local Git	Publish profile	Local Git deployment to Azure App Service
External Git repository	Publish profile	Setting up continuous deployment using manual steps
Run directly from an uploaded ZIP file	Azure AD authentication	Run your app in Azure App Service directly from a ZIP package
Run directly from external URL	Storage account key, managed identity	Run from external URL instead
Azure Web app plugin for Maven (Java)	Azure AD authentication	Quickstart: Create a Java app on Azure App Service
Azure WebApp Plugin for Gradle (Java)	Azure AD authentication	Configure a Java app for Azure App Service
Webhooks	Publish profile	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	Azure AD authentication	Tutorial: Assess ASP.NET web apps for migration to Azure App Service Modernize ASP.NET web apps to Azure App Service code

Configure deployment credentials for Azure App Service

Article • 03/09/2023

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**
>
> The **Development Center (Classic)** page in the Azure portal, which is the old deployment experience, will be deprecated in March, 2021. This change will not affect any existing deployment settings in your app, and you can continue to manage app deployment in the **Deployment Center** page.

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
```

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

Some organizations need to meet security requirements and would rather disable access via FTP or WebDeploy. This way, the organization's members can only access its App Services through APIs that are controlled by Azure Active Directory (Azure AD).

FTP

To disable FTP access to the site, run the following CLI command. Replace the placeholders with your resource group and site name.

Azure CLI

```
az resource update --resource-group <resource-group> --name ftp --namespace Microsoft.Web --resource-type basicPublishingCredentialsPolicies --parent sites/<site-name> --set properties.allow=false
```

To confirm that FTP access is blocked, you can try to authenticate using an FTP client such as FileZilla. To retrieve the publishing credentials, go to the overview blade of your site and click Download Publish Profile. Use the file's FTP hostname, username, and password to authenticate, and you will get a 401 error response, indicating that you are not authorized.

WebDeploy and SCM

To disable basic auth access to the WebDeploy port and SCM site, run the following CLI command. Replace the placeholders with your resource group and site name.

Azure CLI

```
az resource update --resource-group <resource-group> --name scm --namespace
Microsoft.Web --resource-type basicPublishingCredentialsPolicies --parent
sites/<site-name> --set properties.allow=false
```

To confirm that the publish profile credentials are blocked on WebDeploy, try publishing a web app using Visual Studio 2019.

Disable access to the API

The API in the previous section is backed Azure role-based access control (Azure RBAC), which means you can create a custom role and assign lower-priveldged users to the role so they cannot enable basic auth on any sites. To configure the custom role, follow these instructions.

You can also use Azure Monitor to audit any successful authentication requests and use Azure Policy to enforce this configuration for all sites in your subscription.

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 • 07/30/2023

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.

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.

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.

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 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 into the production slot), App Service does the following to ensure that the target slot doesn't experience downtime:

1. Apply the following settings from the target slot (for example, the production slot) to all instances of the source slot:

   - Slot-specific app settings and connection strings, if applicable.
   - Continuous deployment settings, if enabled.
   - App Service authentication settings, if enabled.

   Any of these cases trigger all instances in the source slot to restart. During swap with preview, this marks the end of the first phase. The swap operation is paused, and you can validate that the source slot works correctly with the target slot's settings.

2. Wait for every instance in the source slot to complete its restart. If any instance fails to restart, the swap operation reverts all changes to the source slot and stops the operation.

3. If local cache is enabled, trigger local cache initialization by making an HTTP request to the application root ("/") on each instance of the source slot. Wait until each instance returns any HTTP response. Local cache initialization causes another restart on each instance.

4. If auto swap is enabled with custom warm-up, trigger Application Initiation by making an HTTP request to the application root ("/") on each instance of the source slot.

   If `applicationInitialization` isn't specified, trigger an HTTP request to the application root of the source slot on each instance.

   If an instance returns any HTTP response, it's considered to be warmed up.

5. If all instances on the source slot are warmed up successfully, swap the two slots by switching the routing rules for the two slots. After this step, the target slot (for example, the production slot) has the app that's previously warmed up in the source slot.

6. Now that the source slot has the pre-swap app previously in the target slot, perform the same operation by applying all settings and restarting the instances.

At any point of the swap operation, all work of initializing the swapped apps happens on the source slot. The target slot remains online while the source slot is being prepared

and warmed up, regardless of where the swap succeeds or fails. To swap a staging slot with the production slot, make sure that the production slot is always the target slot. This way, the swap operation doesn't affect your production app.

> **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:

- General settings, such as framework version, 32/64-bit, web sockets
- App settings (can be configured to stick to a slot)
- Connection strings (can be configured to stick to a slot)
- Handler mappings
- Public certificates
- WebJobs content
- Hybrid connections *
- Service endpoints *
- Azure Content Delivery Network *
- Path mappings

Features marked with an asterisk (*) are planned to be unswapped.

Settings that aren't swapped:

- Publishing endpoints
- Custom domain names
- Non-public certificates and TLS/SSL settings
- Scale settings
- WebJobs schedulers
- IP restrictions

- Always On
- Diagnostic settings
- Cross-origin resource sharing (CORS)
- Virtual network integration
- Managed identities and related settings
- Settings that end with the suffix _EXTENSION_VERSION

> **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.

> **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 **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**.

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**.

The dialog box shows you how the configuration in the source slot changes in phase 1, and how the source and target slot change in phase 2.

2. When you're ready to start the swap, select **Start Swap**.

When phase 1 finishes, you're notified in the dialog box. Preview the swap in the source slot by going to `https://<app_name>-<source-slot-name>.azurewebsites.net`.

3. When you're ready to complete the pending swap, select **Complete Swap** in **Swap action** and select **Complete Swap**.

   To cancel a pending swap, select **Cancel Swap** instead, 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 start-up, whereas the two warm-up behavior app settings apply only to slot swaps.

If you have any problems, see Troubleshoot swaps.

Monitor a swap

If the swap operation takes a long time to complete, you can get information on the swap operation in the activity log.

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**.

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.

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 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

  ```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

  ```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

Mount Azure Storage as a local share in App Service

Article • 09/11/2023

> **Note**
>
> Azure Key Vault support for Azure Storage is in preview.

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. See the video how to mount Azure Storage as a local share. For using Azure Storage in an ARM template, see Bring your own storage. Azure Storage is non-default storage for App Service and billed separately.

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.

The following features are supported for Linux containers:

- Azure Files (read/write).
- Azure Blobs (read-only).
- Up to five mount points per app.

This article covers three options for mounting your storage account:

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.

If you plan to mount storage using the Azure CLI, you need to obtain an access key.

Prerequisites

Basic

- An existing App Service on Linux app.
- An Azure Storage account.
- An Azure file share and directory.

Limitations

- 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.
- Azure Files NFS is currently unsupported for App Service on Linux. Only Azure Files SMB are supported.
- With VNET integration on your app, the mounted drive uses an RFC1918 IP address and not an IP address from your VNET.

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 CLI

Using Azure CLI to mount storage requires you to provide the storage access key.

> **Note**
>
> Adding, editing, or deleting a storage mount causes the app to be restarted.

Test 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

- 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 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.

- 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**.

- If you delete an Azure Storage account, container, or share, remove the corresponding storage mount configuration in the app to avoid possible error scenarios.

- 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:
  - For Files
  - For 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.

- 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 readded.

Next steps

- Configure a custom container.
- Video: How to mount Azure Storage as a local share.

Quickstart: Create App Service app using an ARM template

Article • 04/09/2023

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. Because you use a free App Service tier, you incur no costs to complete this quickstart.

A resource manager template is a JavaScript Object Notation (JSON) file that defines the infrastructure and configuration for your project. The template uses declarative syntax. In declarative syntax, you describe your intended deployment without writing the sequence of programming commands to create the deployment.

If your environment meets the prerequisites and you're familiar with using ARM templates, select the **Deploy to Azure** button. The template will open in the Azure portal.

Use the following button to deploy on **Linux**:

Deploy to Azure

Use the following button to deploy on **Windows**:

Deploy to Azure

Prerequisites

If you don't have an Azure subscription, create an Azure free account before you begin.

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')]"
      },
      "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 below for parameter defaults and their descriptions:

Parameters	Type	Default value	Description
webAppName	string	"webApp-<uniqueString>"	App name
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 code below 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|3.7" \
--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 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

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"
Ruby	linuxFxVersion="RUBY|2.6"

> **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.

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

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	linuxFxVersion="DOTNETCORE|3.0"

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

   ```bash
   mkdir appservice_tf_quickstart
   cd appservice_tf_quickstart
   ```

2. Create a file named `main.tf` and insert the above code.

   Bash

   ```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 • 09/01/2023

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

Types	Versions
certificates	2022-09-01 2022-03-01 2021-03-01 2021-02-01 2021-01-15 2021-01-01 2020-12-01 2020-10-01
connectionGateways	2016-06-01
connections	2016-06-01
containerApps	2022-09-01 2022-03-01 2021-03-01
containerApps/revisions	2022-09-01 2022-03-01

Types	Versions
	2021-03-01
customApis	2016-06-01
deletedSites	2022-09-01 2022-03-01 2021-03-01 2021-02-01 2021-01-15 2021-01-01 2020-12-01
hostingEnvironments	2022-09-01 2022-03-01 2021-03-01 2021-02-01 2021-01-15 2021-01-01 2020-12-01 2020-10-01
hostingEnvironments/capacities	2022-09-01 2022-03-01 2021-03-01 2021-02-01 2021-01-15

Types	Versions
	2021-01-01 2020-12-01 2020-10-01
hostingEnvironments/configurations	2021-03-01 2021-02-01 2021-01-15 2021-01-01 2020-12-01
hostingEnvironments/configurations-customdnssuffix	2022-09-01 2022-03-01
hostingEnvironments/configurations-networking	2022-09-01 2022-03-01
hostingEnvironments/detectors	2022-09-01 2022-03-01 2021-03-01 2021-02-01 2021-01-15 2021-01-01 2020-12-01 2020-10-01
hostingEnvironments/multiRolePools	2022-09-01 2022-03-01

Types	Versions
	2021-03-01 2021-02-01 2021-01-15 2021-01-01 2020-12-01 2020-10-01
hostingEnvironments/privateEndpointConnections	2022-09-01 2022-03-01 2021-03-01 2021-02-01 2021-01-15 2021-01-01 2020-12-01
hostingEnvironments/recommendations	2022-09-01 2022-03-01 2021-03-01 2021-02-01 2021-01-15 2021-01-01 2020-12-01 2020-10-01
hostingEnvironments/workerPools	2022-09-01 2022-03-

Types	Versions
	01 2021-03-01 2021-02-01 2021-01-15 2021-01-01 2020-12-01 2020-10-01
kubeEnvironments	2022-09-01 2022-03-01 2021-03-01 2021-02-01 2021-01-15 2021-01-01
locations/connectionGatewayInstallations	2016-06-01
locations/deletedSites	2022-09-01 2022-03-01 2021-03-01 2021-02-01 2021-01-15 2021-01-01 2020-12-01 2020-10-01

Types	Versions
locations/managedApis	2016-06-01
publishingUsers	2022-09-01 2022-03-01 2021-03-01 2021-02-01 2021-01-15 2021-01-01 2020-12-01 2020-10-01
serverfarms	2022-09-01 2022-03-01 2021-03-01 2021-02-01 2021-01-15 2021-01-01 2020-12-01 2020-10-01
serverfarms/hybridConnectionNamespaces/relays	2022-09-01 2022-03-01 2021-03-01 2021-02-01 2021-01-15 2021-01-

Types	Versions
	01 2020-12-01 2020-10-01
serverfarms/hybridConnectionPlanLimits	2022-09-01 2022-03-01 2021-03-01 2021-02-01 2021-01-15 2021-01-01 2020-12-01 2020-10-01
serverfarms/virtualNetworkConnections	2022-09-01 2022-03-01 2021-03-01 2021-02-01 2021-01-15 2021-01-01 2020-12-01 2020-10-01
serverfarms/virtualNetworkConnections/gateways	2022-09-01 2022-03-01 2021-03-01 2021-02-01

Types	Versions
	2021-01-15 2021-01-01 2020-12-01 2020-10-01
serverfarms/virtualNetworkConnections/routes	2022-09-01 2022-03-01 2021-03-01 2021-02-01 2021-01-15 2021-01-01 2020-12-01 2020-10-01
sites	2022-09-01 2022-03-01 2021-03-01 2021-02-01 2021-01-15 2021-01-01 2020-12-01 2020-10-01
sites/backups	2022-09-01 2022-03-01 2021-03-

Types	Versions
	01 2021-02-01 2021-01-15 2021-01-01 2020-12-01 2020-10-01
sites/basicPublishingCredentialsPolicies-ftp	2022-09-01 2022-03-01 2021-03-01 2021-02-01 2021-01-15 2021-01-01 2020-12-01 2020-10-01
sites/basicPublishingCredentialsPolicies-scm	2022-09-01 2022-03-01 2021-03-01 2021-02-01 2021-01-15 2021-01-01 2020-12-01 2020-10-01
sites/config-appsettings	2022-09-01

Types	Versions
	2022-03-01 2021-03-01 2021-02-01 2021-01-15 2021-01-01 2020-12-01 2020-10-01
sites/config-authsettings	2022-09-01 2022-03-01 2021-03-01 2021-02-01 2021-01-15 2021-01-01 2020-12-01 2020-10-01
sites/config-authsettingsV2	2022-09-01 2022-03-01 2021-03-01 2021-02-01 2021-01-15 2021-01-01 2020-12-01 2020-10-01

Types	Versions
sites/config-azurestorageaccounts	2022-09-01 2022-03-01 2021-03-01 2021-02-01 2021-01-15 2021-01-01 2020-12-01 2020-10-01
sites/config-backup	2022-09-01 2022-03-01 2021-03-01 2021-02-01 2021-01-15 2021-01-01 2020-12-01 2020-10-01
sites/config-connectionstrings	2022-09-01 2022-03-01 2021-03-01 2021-02-01 2021-01-15 2021-01-01 2020-12-01

Types	Versions
	2020-10-01
sites/config-logs	2022-09-01 2022-03-01 2021-03-01 2021-02-01 2021-01-15 2021-01-01 2020-12-01 2020-10-01
sites/config-metadata	2022-09-01 2022-03-01 2021-03-01 2021-02-01 2021-01-15 2021-01-01 2020-12-01 2020-10-01
sites/config-pushsettings	2022-09-01 2022-03-01 2021-03-01 2021-02-01 2021-01-15 2021-01-

Types	Versions
	01 2020-12-01 2020-10-01
sites/config-slotConfigNames	2022-09-01 2022-03-01 2021-03-01 2021-02-01 2021-01-15 2021-01-01 2020-12-01 2020-10-01
sites/config-web	2022-09-01 2022-03-01 2021-03-01 2021-02-01 2021-01-15 2021-01-01 2020-12-01 2020-10-01
sites/config/appsettings	2022-09-01 2022-03-01 2021-03-01 2021-02-01

Types	Versions
	2021-01-15 2021-01-01 2020-12-01
sites/config/connectionstrings	2022-09-01 2022-03-01 2021-03-01 2021-02-01 2021-01-15 2021-01-01 2020-12-01
sites/config/snapshots	2022-09-01 2022-03-01 2021-03-01 2021-02-01 2021-01-15 2021-01-01 2020-12-01 2020-10-01
sites/continuouswebjobs	2022-09-01 2022-03-01 2021-03-01 2021-02-01 2021-01-

Types	Versions
	15 2021-01-01 2020-12-01 2020-10-01
sites/deployments	2022-09-01 2022-03-01 2021-03-01 2021-02-01 2021-01-15 2021-01-01 2020-12-01 2020-10-01
sites/deploymentStatus	2022-09-01 2022-03-01
sites/detectors	2022-09-01 2022-03-01 2021-03-01 2021-02-01 2021-01-15 2021-01-01 2020-12-01 2020-10-01

Types	Versions
sites/diagnostics	2022-09-01 2022-03-01 2021-03-01 2021-02-01 2021-01-15 2021-01-01 2020-12-01 2020-10-01
sites/diagnostics/analyses	2022-09-01 2022-03-01 2021-03-01 2021-02-01 2021-01-15 2021-01-01 2020-12-01 2020-10-01
sites/diagnostics/detectors	2022-09-01 2022-03-01 2021-03-01 2021-02-01 2021-01-15 2021-01-01 2020-12-01

Types	Versions
	2020-10-01
sites/domainOwnershipIdentifiers	2022-09-01 2022-03-01 2021-03-01 2021-02-01 2021-01-15 2021-01-01 2020-12-01 2020-10-01
sites/extensions	2021-02-01 2021-01-15 2021-01-01 2020-12-01 2020-10-01
sites/extensions-MSDeploy	2022-09-01 2022-03-01 2021-03-01
sites/extensions-onedeploy	2022-09-01 2022-03-01 2021-03-01
sites/functions	2022-09-01 2022-03-01

Types	Versions
	2021-03-01 2021-02-01 2021-01-15 2021-01-01 2020-12-01 2020-10-01
sites/functions/keys	2022-09-01 2022-03-01 2021-03-01 2021-02-01 2021-01-15 2021-01-01 2020-12-01 2020-10-01
sites/hostNameBindings	2022-09-01 2022-03-01 2021-03-01 2021-02-01 2021-01-15 2021-01-01 2020-12-01 2020-10-01

Types	Versions
sites/hostruntime/webhooks/api/workflows/runs	2022-09-01 2022-03-01
sites/hostruntime/webhooks/api/workflows/runs/actions	2022-09-01 2022-03-01
sites/hostruntime/webhooks/api/workflows/runs/actions/repetitions	2022-09-01 2022-03-01
sites/hostruntime/webhooks/api/workflows/runs/actions/repetitions/requestHistories	2022-09-01 2022-03-01
sites/hostruntime/webhooks/api/workflows/runs/actions/scopeRepetitions	2022-09-01 2022-03-01
sites/hostruntime/webhooks/api/workflows/triggers	2022-09-01 2022-03-01
sites/hostruntime/webhooks/api/workflows/triggers/histories	2022-09-01 2022-03-01
sites/hostruntime/webhooks/api/workflows/versions	2022-09-01 2022-03-01
sites/hybridconnection	2022-09-01 2022-03-01 2021-03-01 2021-02-01 2021-01-

Types	Versions
	15 2021-01-01 2020-12-01 2020-10-01
sites/hybridConnectionNamespaces/relays	2022-09-01 2022-03-01 2021-03-01 2021-02-01 2021-01-15 2021-01-01 2020-12-01 2020-10-01
sites/instances	2022-09-01 2022-03-01 2021-03-01 2021-02-01 2021-01-15 2021-01-01 2020-12-01 2020-10-01
sites/instances/extensions	2022-09-01 2022-03-01 2021-03-01

Types	Versions
	2021-02-01 2021-01-15 2021-01-01 2020-12-01 2020-10-01
sites/instances/processes	2022-09-01 2022-03-01 2021-03-01 2021-02-01 2021-01-15 2021-01-01 2020-12-01 2020-10-01
sites/instances/processes/modules	2022-09-01 2022-03-01 2021-03-01 2021-02-01 2021-01-15 2021-01-01 2020-12-01 2020-10-01
sites/migratemysql	2022-09-01 2022-03-

Types	Versions
	01 2021-03-01 2021-02-01 2021-01-15 2021-01-01 2020-12-01 2020-10-01
sites/networkConfig	2022-09-01 2022-03-01 2021-03-01 2021-02-01 2021-01-15 2021-01-01 2020-12-01 2020-10-01
sites/networkFeatures	2022-09-01 2022-03-01 2021-03-01 2021-02-01 2021-01-15 2021-01-01 2020-12-01 2020-10-01

Types	Versions
sites/premieraddons	2022-09-01 2022-03-01 2021-03-01 2021-02-01 2021-01-15 2021-01-01 2020-12-01 2020-10-01
sites/privateAccess	2022-09-01 2022-03-01 2021-03-01 2021-02-01 2021-01-15 2021-01-01 2020-12-01 2020-10-01
sites/privateEndpointConnections	2022-09-01 2022-03-01 2021-03-01 2021-02-01 2021-01-15 2021-01-01 2020-12-01

Types	Versions
	2020-10-01
sites/processes	2022-09-01 2022-03-01 2021-03-01 2021-02-01 2021-01-15 2021-01-01 2020-12-01 2020-10-01
sites/processes/modules	2022-09-01 2022-03-01 2021-03-01 2021-02-01 2021-01-15 2021-01-01 2020-12-01 2020-10-01
sites/publicCertificates	2022-09-01 2022-03-01 2021-03-01 2021-02-01 2021-01-15 2021-01-

Types	Versions
	01 2020-12-01 2020-10-01
sites/recommendations	2022-09-01 2022-03-01 2021-03-01 2021-02-01 2021-01-15 2021-01-01 2020-12-01 2020-10-01
sites/resourceHealthMetadata	2022-09-01 2022-03-01 2021-03-01 2021-02-01 2021-01-15 2021-01-01 2020-12-01 2020-10-01
sites/siteextensions	2022-09-01 2022-03-01 2021-03-01 2021-02-01

Types	Versions
	2021-01-15 2021-01-01 2020-12-01 2020-10-01
sites/slots	2022-09-01 2022-03-01 2021-03-01 2021-02-01 2021-01-15 2021-01-01 2020-12-01 2020-10-01
sites/slots/backups	2022-09-01 2022-03-01 2021-03-01 2021-02-01 2021-01-15 2021-01-01 2020-12-01 2020-10-01
sites/slots/basicPublishingCredentialsPolicies-ftp	2022-09-01 2022-03-01 2021-03-

Types	Versions
	01 2021-02-01 2021-01-15 2021-01-01 2020-12-01
sites/slots/basicPublishingCredentialsPolicies-scm	2022-09-01 2022-03-01 2021-03-01 2021-02-01 2021-01-15 2021-01-01 2020-12-01
sites/slots/config-appsettings	2022-09-01 2022-03-01 2021-03-01 2021-02-01 2021-01-15 2021-01-01 2020-12-01 2020-10-01
sites/slots/config-authsettings	2022-09-01 2022-03-01 2021-03-01

Types	Versions
	2021-02-01 2021-01-15 2021-01-01 2020-12-01 2020-10-01
sites/slots/config-authsettingsV2	2022-09-01 2022-03-01 2021-03-01 2021-02-01 2021-01-15 2021-01-01 2020-12-01 2020-10-01
sites/slots/config-azurestorageaccounts	2022-09-01 2022-03-01 2021-03-01 2021-02-01 2021-01-15 2021-01-01 2020-12-01 2020-10-01
sites/slots/config-backup	2022-09-01 2022-03-

Types	Versions
	01 2021-03-01 2021-02-01 2021-01-15 2021-01-01 2020-12-01 2020-10-01
sites/slots/config-connectionstrings	2022-09-01 2022-03-01 2021-03-01 2021-02-01 2021-01-15 2021-01-01 2020-12-01 2020-10-01
sites/slots/config-logs	2022-09-01 2022-03-01 2021-03-01 2021-02-01 2021-01-15 2021-01-01 2020-12-01 2020-10-01

Types	Versions
sites/slots/config-metadata	2022-09-01 2022-03-01 2021-03-01 2021-02-01 2021-01-15 2021-01-01 2020-12-01 2020-10-01
sites/slots/config-pushsettings	2022-09-01 2022-03-01 2021-03-01 2021-02-01 2021-01-15 2021-01-01 2020-12-01 2020-10-01
sites/slots/config-web	2022-09-01 2022-03-01 2021-03-01 2021-02-01 2021-01-15 2021-01-01 2020-12-01

Types	Versions
	2020-10-01
sites/slots/config/appsettings	2022-09-01 2022-03-01 2021-03-01 2021-02-01 2021-01-15 2021-01-01 2020-12-01
sites/slots/config/connectionstrings	2022-09-01 2022-03-01 2021-03-01 2021-02-01 2021-01-15 2021-01-01 2020-12-01
sites/slots/config/snapshots	2022-09-01 2022-03-01 2021-03-01 2021-02-01 2021-01-15 2021-01-01 2020-12-01

Types	Versions
	2020-10-01
sites/slots/continuouswebjobs	2022-09-01 2022-03-01 2021-03-01 2021-02-01 2021-01-15 2021-01-01 2020-12-01 2020-10-01
sites/slots/deployments	2022-09-01 2022-03-01 2021-03-01 2021-02-01 2021-01-15 2021-01-01 2020-12-01 2020-10-01
sites/slots/deploymentStatus	2022-09-01 2022-03-01
sites/slots/detectors	2022-09-01 2022-03-01 2021-03-01

Types	Versions
	2021-02-01 2021-01-15 2021-01-01 2020-12-01 2020-10-01
sites/slots/diagnostics	2022-09-01 2022-03-01 2021-03-01 2021-02-01 2021-01-15 2021-01-01 2020-12-01 2020-10-01
sites/slots/diagnostics/analyses	2022-09-01 2022-03-01 2021-03-01 2021-02-01 2021-01-15 2021-01-01 2020-12-01 2020-10-01
sites/slots/diagnostics/detectors	2022-09-01 2022-03-

Types	Versions
	01 2021-03-01 2021-02-01 2021-01-15 2021-01-01 2020-12-01 2020-10-01
sites/slots/domainOwnershipIdentifiers	2022-09-01 2022-03-01 2021-03-01 2021-02-01 2021-01-15 2021-01-01 2020-12-01 2020-10-01
sites/slots/extensions	2022-09-01 2022-03-01 2021-03-01 2021-02-01 2021-01-15 2021-01-01 2020-12-01 2020-10-01

Types	Versions
sites/slots/functions	2022-09-01 2022-03-01 2021-03-01 2021-02-01 2021-01-15 2021-01-01 2020-12-01 2020-10-01
sites/slots/functions/keys	2022-09-01 2022-03-01 2021-03-01 2021-02-01 2021-01-15 2021-01-01 2020-12-01 2020-10-01
sites/slots/hostNameBindings	2022-09-01 2022-03-01 2021-03-01 2021-02-01 2021-01-15 2021-01-01 2020-12-01

Types	Versions
	2020-10-01
sites/slots/hybridconnection	2022-09-01 2022-03-01 2021-03-01 2021-02-01 2021-01-15 2021-01-01 2020-12-01 2020-10-01
sites/slots/hybridConnectionNamespaces/relays	2022-09-01 2022-03-01 2021-03-01 2021-02-01 2021-01-15 2021-01-01 2020-12-01 2020-10-01
sites/slots/instances	2022-09-01 2022-03-01 2021-03-01 2021-02-01 2021-01-15 2021-01-

Types	Versions
	01 2020-12-01 2020-10-01
sites/slots/instances/extensions	2022-09-01 2022-03-01 2021-03-01 2021-02-01 2021-01-15 2021-01-01 2020-12-01 2020-10-01
sites/slots/instances/processes	2022-09-01 2022-03-01 2021-03-01 2021-02-01 2021-01-15 2021-01-01 2020-12-01 2020-10-01
sites/slots/instances/processes/modules	2022-09-01 2022-03-01 2021-03-01 2021-02-01

Types	Versions
	2021-01-15 2021-01-01 2020-12-01 2020-10-01
sites/slots/migratemysql	2022-09-01 2022-03-01 2021-03-01 2021-02-01 2021-01-15 2021-01-01 2020-12-01 2020-10-01
sites/slots/networkConfig	2022-09-01 2022-03-01 2021-03-01 2021-02-01 2021-01-15 2021-01-01 2020-12-01 2020-10-01
sites/slots/networkFeatures	2022-09-01 2022-03-01 2021-03-

Types	Versions
	01 2021-02-01 2021-01-15 2021-01-01 2020-12-01 2020-10-01
sites/slots/premieraddons	2022-09-01 2022-03-01 2021-03-01 2021-02-01 2021-01-15 2021-01-01 2020-12-01 2020-10-01
sites/slots/privateAccess	2022-09-01 2022-03-01 2021-03-01 2021-02-01 2021-01-15 2021-01-01 2020-12-01 2020-10-01
sites/slots/privateEndpointConnections	2022-09-01

Types	Versions
	2022-03-01 2021-03-01 2021-02-01 2021-01-15 2021-01-01 2020-12-01
sites/slots/processes	2022-09-01 2022-03-01 2021-03-01 2021-02-01 2021-01-15 2021-01-01 2020-12-01 2020-10-01
sites/slots/processes/modules	2022-09-01 2022-03-01 2021-03-01 2021-02-01 2021-01-15 2021-01-01 2020-12-01 2020-10-01

Types	Versions
sites/slots/publicCertificates	2022-09-01 2022-03-01 2021-03-01 2021-02-01 2021-01-15 2021-01-01 2020-12-01 2020-10-01
sites/slots/resourceHealthMetadata	2022-09-01 2022-03-01 2021-03-01 2021-02-01 2021-01-15 2021-01-01 2020-12-01 2020-10-01
sites/slots/siteextensions	2022-09-01 2022-03-01 2021-03-01 2021-02-01 2021-01-15 2021-01-01 2020-12-01

Types	Versions
	2020-10-01
sites/slots/sourcecontrols	2022-09-01 2022-03-01 2021-03-01 2021-02-01 2021-01-15 2021-01-01 2020-12-01 2020-10-01
sites/slots/triggeredwebjobs	2022-09-01 2022-03-01 2021-03-01 2021-02-01 2021-01-15 2021-01-01 2020-12-01 2020-10-01
sites/slots/triggeredwebjobs/history	2022-09-01 2022-03-01 2021-03-01 2021-02-01 2021-01-15 2021-01-

Types	Versions
	01 2020-12-01 2020-10-01
sites/slots/virtualNetworkConnections	2022-09-01 2022-03-01 2021-03-01 2021-02-01 2021-01-15 2021-01-01 2020-12-01 2020-10-01
sites/slots/virtualNetworkConnections/gateways	2022-09-01 2022-03-01 2021-03-01 2021-02-01 2021-01-15 2021-01-01 2020-12-01 2020-10-01
sites/slots/webjobs	2022-09-01 2022-03-01 2021-03-01 2021-02-01

Types	Versions
	2021-01-15 2021-01-01 2020-12-01 2020-10-01
sites/sourcecontrols	2022-09-01 2022-03-01 2021-03-01 2021-02-01 2021-01-15 2021-01-01 2020-12-01 2020-10-01
sites/triggeredwebjobs	2022-09-01 2022-03-01 2021-03-01 2021-02-01 2021-01-15 2021-01-01 2020-12-01 2020-10-01
sites/triggeredwebjobs/history	2022-09-01 2022-03-01 2021-03-

Types	Versions
	01 2021-02-01 2021-01-15 2021-01-01 2020-12-01 2020-10-01
sites/virtualNetworkConnections	2022-09-01 2022-03-01 2021-03-01 2021-02-01 2021-01-15 2021-01-01 2020-12-01 2020-10-01
sites/virtualNetworkConnections/gateways	2022-09-01 2022-03-01 2021-03-01 2021-02-01 2021-01-15 2021-01-01 2020-12-01 2020-10-01
sites/webjobs	2022-09-01

Types	Versions
	2022-03-01 2021-03-01 2021-02-01 2021-01-15 2021-01-01 2020-12-01 2020-10-01
sourcecontrols	2022-09-01 2022-03-01 2021-03-01 2021-02-01 2021-01-15 2021-01-01 2020-12-01 2020-10-01
staticSites	2022-09-01 2022-03-01 2021-03-01 2021-02-01 2021-01-15 2021-01-01 2020-12-01 2020-10-01

Types	Versions
staticSites/basicAuth	2022-09-01
staticSites/builds	2022-09-01 2022-03-01 2021-03-01 2021-02-01 2021-01-15 2021-01-01 2020-12-01 2020-10-01
staticSites/builds/config	2020-10-01
staticSites/builds/config-appsettings	2022-09-01 2022-03-01 2021-03-01 2021-02-01 2021-01-15 2021-01-01 2020-12-01
staticSites/builds/config-functionappsettings	2022-09-01 2022-03-01 2021-03-01 2021-02-01 2021-01-15

Types	Versions
	2021-01-01 2020-12-01
staticSites/builds/databaseConnections	2022-09-01
staticSites/builds/linkedBackends	2022-09-01 2022-03-01
staticSites/builds/userProvidedFunctionApps	2022-09-01 2022-03-01 2021-03-01 2021-02-01 2021-01-15 2021-01-01 2020-12-01
staticSites/config	2020-10-01
staticSites/config-appsettings	2022-09-01 2022-03-01 2021-03-01 2021-02-01 2021-01-15 2021-01-01 2020-12-01
staticSites/config-functionappsettings	2022-09-01 2022-03-

Types	Versions
	01 2021-03-01 2021-02-01 2021-01-15 2021-01-01 2020-12-01
staticSites/customDomains	2022-09-01 2022-03-01 2021-03-01 2021-02-01 2021-01-15 2021-01-01 2020-12-01 2020-10-01
staticSites/databaseConnections	2022-09-01
staticSites/linkedBackends	2022-09-01 2022-03-01
staticSites/privateEndpointConnections	2022-09-01 2022-03-01 2021-03-01 2021-02-01 2021-01-15 2021-01-

Types	Versions
	01 2020-12-01
staticSites/userProvidedFunctionApps	2022-09-01 2022-03-01 2021-03-01 2021-02-01 2021-01-15 2021-01-01 2020-12-01

Environment variables and app settings in Azure App Service

Article • 09/14/2023

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.

Setting name	Description	Example
`WEBSITE_SITE_NAME`	Read-only. App name.	
`WEBSITE_RESOURCE_GROUP`	Read-only. Azure resource group name that contains the app resource.	
`WEBSITE_OWNER_NAME`	Read-only. Contains the Azure subscription ID that owns the app, the resource group, and the webspace.	
`REGION_NAME`	Read-only. Region name of the app.	
`WEBSITE_PLATFORM_VERSION`	Read-only. App Service platform version.	
`HOME`	Read-only. Path to the home directory (for example, `D:\home` for Windows).	
`SERVER_PORT`	Read-only. The port the app should listen to.	
`WEBSITE_WARMUP_PATH`	A relative path to ping to warm up the app, beginning with a slash. The default is `/`, 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.)	
`WEBSITE_COMPUTE_MODE`	Read-only. Specifies whether app runs on dedicated (`Dedicated`) or shared (`Shared`) VM/s.	
`WEBSITE_SKU`	Read-only. SKU of the app. Possible values are `Free`, `Shared`, `Basic`, and `Standard`.	
`SITE_BITNESS`	Read-only. Shows whether the app is 32-bit (`x86`) or 64-bit (`AMD64`).	
`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 **Standard** tier and	

Setting name	Description	Example
	above, the value is `-1`, but there's still a per VM limit based on your VM size (see Cross VM Numerical Limits).	
`WEBSITE_PRIVATE_EXTENSIONS`	Set to `0` 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 TimeZone. 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 `1` on all slots (default is `0`). 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 `false` to disable this recovery behavior. The default is `true`. 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 crash is logged in your App Service's logs. Set to `false` to disable this automatic monitoring behavior. The default is `true`. For more information, see Proactive Crash Monitoring.	
`WEBSITE_DAAS_STORAGE_SASURI`	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.	
`WEBSITE_CRASHMONITORING_ENABLED`	Set to `true` to enable crash monitoring manually. You must also set `WEBSITE_DAAS_STORAGE_SASURI` and `WEBSITE_CRASHMONITORING_SETTINGS`. The	

Setting name	Description	Example
	default is `false`. This setting has no effect if remote debugging is enabled. Also, if this setting is set to `true`, proactive crash monitoring is disabled.	
`WEBSITE_CRASHMONITORING_SETTINGS`	A JSON with the following format: `{"StartTimeUtc": "2020-02-10T08:21","MaxHours": "<elapsed-hours-from-StartTimeUtc>","MaxDumpCount": "<max-number-of-crash-dumps>"}`. Required to configure crash monitoring if `WEBSITE_CRASHMONITORING_ENABLED` is specified. To only log the call stack without saving the crash dump in the storage account, add `,"UseStorageAccount":"false"` in the JSON.	
`REMOTEDEBUGGINGVERSION`	Remote debugging version.	
`WEBSITE_CONTENTAZUREFILECONNECTIONSTRING`	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.	`DefaultEndpointsProtocol=https;AccountName=<name>;AccountKey=<key>`
`WEBSITE_CONTENTSHARE`	When you use specify a custom storage account with `WEBSITE_CONTENTAZUREFILECONNECTIONSTRING`, 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.	`myapp123`
`WEBSITE_SCM_ALWAYS_ON_ENABLED`	Read-only. Shows whether Always On is enabled (`1`) or not (`0`).	
`WEBSITE_SCM_SEPARATE_STATUS`	Read-only. Shows whether the Kudu app is running in a separate process (`1`) or not (`0`).	
`WEBSITE_DNS_ATTEMPTS`	Number of times to try name resolve.	
`WEBSITE_DNS_TIMEOUT`	Number of seconds to wait for name resolve	

Variable prefixes

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

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.

Setting name	Description
`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.
`AZUREFILESSTORAGE_`	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.

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 `master`.
`WEBSITE_RUN_FROM_PACKAGE`	Set to `1` 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 `WEBSITE_RUN_FROM_PACKAGE`.
`WEBSITE_RUN_FROM_ZIP`	Deprecated. Use `WEBSITE_RUN_FROM_PACKAGE`.
`WEBSITE_WEBDEPLOY_USE_SCM`	Set to `false` for WebDeploy to stop using the Kudu deployment engine. The default is `true`. To deploy to Linux apps using Visual Studio (WebDeploy/MSDeploy), set it to `false`.
`MSDEPLOY_RENAME_LOCKED_FILES`	Set to `1` to attempt to rename DLLs if they can't be copied during a WebDeploy deployment. This setting isn't applicable if `WEBSITE_WEBDEPLOY_USE_SCM` is set to `false`.
`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 `true`) is a legacy mode that's no longer fully supported.
`WEBSITE_ENABLE_SYNC_UPDATE_SITE`	Set to `1` ensure that REST API calls to update `site` and `siteconfig` are completely applied to all instances before returning. The default is `1` 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 `1` in the ARM template to pre-start the Kudu app as part of app creation.

Setting name	Description
`WEBSITE_START_SCM_WITH_PRELOAD`	For Linux apps, set to `true` to force preloading the Kudu app when Always On is enabled by pinging its URL. The default is `false`. 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.

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: `-t:Clean;Compile`
`SCM_SCRIPT_GENERATOR_ARGS`	Kudu uses the `azure site deploymentscript` 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: `--basic -p <folder-to-deploy>`
`SCM_TRACE_LEVEL`	Build trace level. The default is `1`. Set to higher values, up to 4, for more tracing.	`4`
`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 `60` (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 `1800` (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 `0` to use git.exe instead of libgit2sharp for git operations.	
`WEBSITE_LOAD_USER_PROFILE`	In case of the error `The specified user does not have a valid profile.` during ASP.NET build automation (such as during Git deployment), set this variable to `1` to load a full user profile in the build environment. This setting is only applicable when `WEBSITE_COMPUTE_MODE` is `Dedicated`.	
`WEBSITE_SCM_IDLE_TIMEOUT_IN_MINUTES`	Time out in minutes for the SCM (Kudu) site. The default is `20`.	
`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 `true`.	

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

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 (`/home`).
`DUMP_DIR`	Read-only. Directory for the crash dumps (`/home/logs/dumps`).
`APP_SVC_RUN_FROM_COPY`	Linux apps only. By default, the app is run from `/home/site/wwwroot`, a shared directory for all scaled-out instances. Set this variable to `true` 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 `/home/site/wwwroot`. Instead, use a path relative to `/home/site/wwwroot`.
`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 `decryption` value, configure it as an App Service app setting, or set it directly in the `machineKey` element of the *Web.config* 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 `decryptionKey` value, configure it as an App Service app setting, or set it directly in the `machineKey` element of the *Web.config* 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 `validation` value, configure it as an App Service app setting, or set it directly in the `machineKey` element of the *Web.config* 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 `validationKey` value, configure it as an App Service app setting, or set it directly in the `machineKey` element of the *Web.config* file.

WordPress

Application Setting	Scope	Value	Max	Description
WEBSITES_ENABLE_APP_SERVICE_STORAGE	Web App	true	-	When set to TRUE, file contents are preserved during restarts.
WP_MEMORY_LIMIT	WordPress	128M	512M	Frontend or general wordpress PHP memory limit (per script). Can't be more than PHP_MEMORY_LIMIT
WP_MAX_MEMORY_LIMIT	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 PHP_MEMORY_LIMIT.
PHP_MEMORY_LIMIT	PHP	512M	512M	Memory limits for general PHP script. It can only be decreased.
FILE_UPLOADS	PHP	On	-	Can be either On or Off. Note that values are case sensitive. Enables or disables file uploads.
UPLOAD_MAX_FILESIZE	PHP	50M	256M Max file upload size limit. Can be increased up to 256M.	

Application Setting	Scope	Value	Max	Description
POST_MAX_SIZE	PHP	128M	256M	Can be increased up to 256M. Generally should be more than UPLOAD_MAX_FILESIZE.
MAX_EXECUTION_TIME	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.
MAX_INPUT_TIME	PHP	120	120	Max time limit for parsing the input requests. Can only be decreased.
MAX_INPUT_VARS	PHP	10000	10000	-
DATABASE_HOST	Database	-	-	Database host used to connect to WordPress.
DATABASE_NAME	Database	-	-	Database name used to connect to WordPress.
DATABASE_USERNAME	Database	-	-	Database username used to connect to WordPress.
DATABASE_PASSWORD	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.
WORDPRESS_ADMIN_EMAIL	Deployment only	-	-	WordPress admin email.
WORDPRESS_ADMIN_PASSWORD	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.
WORDPRESS_ADMIN_USER	Deployment only	-	-	WordPress admin username
WORDPRESS_ADMIN_LOCALE_CODE	Deployment only	-	-	Database username used to connect to WordPress.

Domain and DNS

Setting name	Description	Example
`WEBSITE_DNS_SERVER`	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 `168.63.129.16`. If your app uses VNet integration or is in an App Service environment, it inherits the DNS server configuration from the VNet by default.	`10.0.0.1`
`WEBSITE_DNS_ALT_SERVER`	IP address of fallback DNS server for outgoing connections. See `WEBSITE_DNS_SERVER`.	
`WEBSITE_ENABLE_DNS_CACHE`	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.

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

Setting name	Description	Example
`WEBSITE_OVERRIDE_STICKY_EXTENSION_VERSIONS`	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 `0` on *all slots*.	
`WEBSITE_OVERRIDE_PRESERVE_DEFAULT_STICKY_SLOT_SETTINGS`	Designates certain settings as sticky or not swappable by default. Default is `true`. Set this setting to `false` or `0` for *all deployment slots* to make them swappable instead. There's no fine-grain control for specific setting types.	
`WEBSITE_SWAP_WARMUP_PING_PATH`	Path to ping to warm up the target slot in a swap, beginning with a slash. The default is `/`, which pings the root path over HTTP.	`/statuscheck`
`WEBSITE_SWAP_WARMUP_PING_STATUSES`	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.	`200,202`
`WEBSITE_SLOT_NUMBER_OF_TIMEOUTS_BEFORE_RESTART`	During a slot swap, maximum number of timeouts after which we force restart the site on a specific VM instance. The default is `3`.	
`WEBSITE_SLOT_MAX_NUMBER_OF_TIMEOUTS`	During a slot swap, maximum number of timeout requests for a single URL to make before giving up. The default is `5`.	
`WEBSITE_SKIP_ALL_BINDINGS_IN_APPHOST_CONFIG`	Set to `true` or `1` to skip all bindings in `applicationHost.config`. The default is `false`. If your app triggers a restart because `applicationHost.config` is updated with the swapped hostnames of th slots, set this variable to `true` 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.

Setting name	Description	Example
`WEBSITES_ENABLE_APP_SERVICE_STORAGE`	Set to `true` to enable the `/home` directory to be shared across scaled instances. The default is `true` for custom containers.	
`WEBSITES_CONTAINER_START_TIME_LIMIT`	Amount of time in seconds to wait for the container to complete start-up before restarting the container. Default is `230`. You can increase it up to	

Setting name	Description	Example
	the maximum of `1800`.	
`WEBSITES_CONTAINER_STOP_TIME_LIMIT`	Amount of time in seconds to wait for the container to terminate gracefully. Deafult is `5`. You can increase to a maximum of `120`	
`DOCKER_REGISTRY_SERVER_URL`	URL of the registry server, when running a custom container in App Service. For security, this variable isn't passed on to the container.	`https://<server-name>.azurecr.io`
`DOCKER_REGISTRY_SERVER_USERNAME`	Username to authenticate with the registry server at `DOCKER_REGISTRY_SERVER_URL`. For security, this variable isn't passed on to the container.	
`DOCKER_REGISTRY_SERVER_PASSWORD`	Password to authenticate with the registry server at `DOCKER_REGISTRY_SERVER_URL`. For security, this variable isn't passed on to the container.	
`DOCKER_ENABLE_CI`	Set to `true` to enable the continuous deployment for custom containers. The default is `false` for custom containers.	
`WEBSITE_PULL_IMAGE_OVER_VNET`	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.	
`WEBSITES_WEB_CONTAINER_NAME`	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.	
`WEBSITES_PORT`	For a custom container, the custom port number on the container for App Service to route requests to. By default, App Service attempts automatic 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

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 `WEBSITE_INSTANCE_ID`.
`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 `1`.
`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 `1` or `true` to disable cross-stamp scaling within the app's region. The default is `0`. Custom Docker containers that set `WEBSITES_ENABLE_APP_SERVICE_STORAGE` to `true` or `1` can't scale cross-stamps because their content isn't completely encapsulated in the Docker container.

Logging

Setting name	Description	Example
`WEBSITE_HTTPLOGGING_ENABLED`	Read-only. Shows whether the web server logging for Windows native apps is enabled (`1`) or not (`0`).	
`WEBSITE_HTTPLOGGING_RETENTION_DAYS`	Retention period in days of web server logs for Windows native apps, if web server logs are enabled.	`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.	
`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 `logging-errors.txt`.	
`DIAGNOSTICS_LOGGINGSETTINGSFILE`	Path to the log settings file, relative to `D:\home` or `/home`. The default is `site\diagnostics\settings.json`.	
`DIAGNOSTICS_TEXTTRACELOGDIRECTORY`	The log folder, relative to the app root (`D:\home\site\wwwroot` or `/home/site/wwwroot`).	`..\..\LogFiles\Application`
`DIAGNOSTICS_TEXTTRACEMAXLOGFILESIZEBYTES`	Maximum size of the log file in bytes. The default is `131072` (128 KB).	
`DIAGNOSTICS_TEXTTRACEMAXLOGFOLDERSIZEBYTES`	Maximum size of the log folder in bytes. The default is `1048576` (1 MB).	
`DIAGNOSTICS_TEXTTRACEMAXNUMLOGFILES`	Maximum number of log files to keep. The default is `20`.	
`DIAGNOSTICS_TEXTTRACETURNOFFPERIOD`	Time out in milliseconds to keep application logging enabled. The default is `43200000` (12 hours).	
`WEBSITE_LOG_BUFFERING`	By default, log buffering is enabled. Set to `0` to disable it.	
`WEBSITE_ENABLE_PERF_MODE`	For native Windows apps, set to `TRUE` 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.

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.

Setting name	Description
`WEBSITE_COUNTERS_ALL`	A JSON object containing the combination of the other three variables.

Caching

Setting name	Description
`WEBSITE_LOCAL_CACHE_OPTION`	Whether local cache is enabled. Available options are: - `Default`: Inherit the stamp-level global setting. - `Always`: Enable for the app. - OnStorageUnavailability - `Disabled`: Disabled for the app.
`WEBSITE_LOCAL_CACHE_READWRITE_OPTION`	Read-write options of the local cache. Available options are: - `ReadOnly`: Cache is read-only. - `WriteButDiscardChanges`: Allow writes to local cache but discard changes made locally.
`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). By default, full content caching is enabled (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`.
`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.

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.

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 VNet. To avoid errors at create or update time, set this variable to `1`.
`WEBSITE_DELAY_CERT_DELETION`	This env var can be set to 1 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.

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

Authentication & Authorization

The following environment variables are related to App Service authentication.

Setting name	Description
`WEBSITE_AUTH_DISABLE_IDENTITY_FLOW`	When set to `true`, 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 `false`.
`WEBSITE_AUTH_HIDE_DEPRECATED_SID`	`true` or `false`. The default value is `false`. This is a setting for the legacy Azure Mobile Apps integration for Azure App Service. Setting this to `true` 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.
`WEBSITE_AUTH_NONCE_DURATION`	A *timespan* value in the form `_hours_:_minutes_:_seconds_`. The default value is `00:05:00`, 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 `login.nonce.nonceExpirationInterval` configuration value.
`WEBSITE_AUTH_PRESERVE_URL_FRAGMENT`	When set to `true` 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.
`WEBSITE_AUTH_USE_LEGACY_CLAIMS`	To maintain backward compatibility across upgrades, the authentication module uses the legacy claims mapping of short to long names in the `/.auth/me` API, so certain mappings are excluded (e.g. "roles"). To get the more modern version of the claims mappings, set this variable to `False`. In the "roles" example, it would be mapped to the long claim name "http://schemas.microsoft.com/ws/2008/06/identity/claims/role".

Setting name	Description
`WEBSITE_AUTH_DISABLE_WWWAUTHENTICATE`	`true` or `false`. The default value is `false`. When set to `true`, 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 `identityProviders.azureActiveDirectory.login.disableWwwAuthenticate` configuration value.
`WEBSITE_AUTH_STATE_DIRECTORY`	A local file system directory path where tokens are stored when the file-based token store is enabled. The default value is `%HOME%\Data\.auth`. 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 `login.tokenStore.fileSystem.directory` configuration value.
`WEBSITE_AUTH_TOKEN_CONTAINER_SASURL`	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.
`WEBSITE_AUTH_TOKEN_REFRESH_HOURS`	Any positive decimal number. The default value is `72` (hours). This setting controls the amount of time after a session token expires that the `/.auth/refresh` 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 `login.tokenStore.tokenRefreshExtensionHours` configuration value.
`WEBSITE_AUTH_TRACE_LEVEL`	Controls the verbosity of authentication traces written to Application Logging. Valid values are `Off`, `Error`, `Warning`, `Information`, and `Verbose`. The default value is `Verbose`.
`WEBSITE_AUTH_VALIDATE_NONCE`	`true` or `false`. The default value is `true`. This value should never be set to `false` 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 `login.nonce.validateNonce` configuration value.
`WEBSITE_AUTH_V2_CONFIG_JSON`	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.
`WEBSITE_AUTH_ENABLED`	Read-only. Injected into a Windows or Linux app to indicate whether App Service authentication is enabled.
`WEBSITE_AUTH_ENCRYPTION_KEY`	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 `MACHINEKEY_DecryptionKey` setting.
`WEBSITE_AUTH_SIGNING_KEY`	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 `MACHINEKEY_ValidationKey` setting.

Managed identity

The following environment variables are related to managed identities.

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

Health check

The following environment variables are related to health checks.

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.

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.

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 (`D:/home/site/wwwroot/` or `/home/site/wwwroot/`).
`WEBJOBS_LOG_TRIGGERED_JOBS_TO_APP_LOGS`	Set to true to send output from triggered WebJobs to the application logs pipeline (which supports file system, blobs, and tables).
`WEBJOBS_SHUTDOWN_FILE`	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

Setting name	Description
	WebJobs SDK, this part is handled automatically.
`WEBJOBS_PATH`	Read-only. Root path of currently running job (will be under some temporary directory).
`WEBJOBS_NAME`	Read-only. Current job name.
`WEBJOBS_TYPE`	Read-only. Current job type (`triggered` or `continuous`).
`WEBJOBS_DATA_PATH`	Read-only. Current job metadata path to contain the job's logs, history, and any artifact of the job.
`WEBJOBS_RUN_ID`	Read-only. For triggered jobs, current run ID of the job.

Functions

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

Azure App Service monitoring overview

Article • 06/29/2023

Azure App Service provides several monitoring options for monitoring resources for availability, performance, and operation. Options such as Diagnostic Settings, Application Insights, Log stream, Metrics, Quotas and alerts, and Activity logs. This article aims to bring clarity on monitoring options on App Service and provide scenarios when each should be used.

Diagnostic Settings (via Azure Monitor)

Azure Monitor is a monitoring service that provides a complete set of features to monitor your Azure resources in addition to resources in other clouds and on-premises. The Azure Monitor data platform collects data into logs and metrics where they can be analyzed. App Service monitoring data can be shipped to Azure Monitor through Diagnostic Settings.

Diagnostic settings lets you export logs to other services, such as Log Analytics, Storage account, and Event Hubs. Large amounts of data using SQL-like Kusto can be queried with Log Analytics. You can capture platform logs in Azure Monitor Logs as configured via Diagnostic Settings, and instrument your app further with the dedicated application performance management feature (Application Insights) for additional telemetry and logs.

For an end-to-end tutorial on Diagnostic Settings, see the article Troubleshoot an App Service app with Azure Monitor.

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.

Metrics

Build visualizations of metrics on Azure resources (web apps and App Service Plans). 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.

Activity logs

View a historical log of events changing your resource. Resource events help you understand any changes that were made to your underlying web app resources and take action as necessary. Event examples include scaling of instances, updates to application settings, restarting of the web app, and many more.

Application Insights (via Azure Monitor)

Application Insights, a feature of Azure Monitor, is an extensible Application Performance Management (APM) service for developers and DevOps professionals. Use it to monitor your live applications. It will automatically detect performance anomalies, and includes powerful analytics tools to help you diagnose issues and to understand what users actually do with your app. The logs in Application Insights are generated by application code.

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.

Monitoring scenarios

The table below lists monitoring methods to use for different scenarios.

Scenario	Monitoring method
I want to monitor platform metrics and logs	(Azure Monitor) Diagnostic Settings
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

Next steps

- Query logs with Azure Monitor
- How to Monitor Azure App Service
- Troubleshooting Azure App Service with Azure Monitor
- Monitor App Service with Azure Monitor

Enable diagnostics logging for apps in Azure App Service

Article • 06/29/2023

Overview

Azure provides built-in diagnostics to assist with debugging an App Service app. In this article, you learn how to enable diagnostic logging and add instrumentation to your application, as well as how to access the information logged by Azure.

This article uses the Azure portal and Azure CLI to work with diagnostic logs. For information on working with diagnostic logs using Visual Studio, see Troubleshooting Azure in Visual Studio.

> **Note**
>
> In addition to the logging instructions in this article, there's new, integrated logging capability with Azure Monitoring. You'll find more on this capability in the **Send logs to Azure Monitor** section.

Type	Platform	Location	Description
Application logging	Windows, Linux	App Service file system and/or Azure Storage blobs	Logs messages generated by your application code. The messages can be generated by the web framework you choose, or from your application code directly using the standard logging pattern of your language. Each message is assigned one of the following categories: **Critical**, **Error**, **Warning**, **Info**, **Debug**, and **Trace**. You can select how verbose you want the logging to be by setting the severity level when you enable application logging.
Web server logging	Windows	App Service file system or Azure Storage blobs	Raw HTTP request data in the W3C extended log file format. Each log message includes data such as the HTTP method, resource URI, client IP, client port, user agent, response code, and so on.

Type	Platform	Location	Description
Detailed Error Messages	Windows	App Service file system	Copies of the *.htm* error pages that would have been sent to the client browser. For security reasons, detailed error pages shouldn't be sent to clients in production, but App Service can save the error page each time an application error occurs that has HTTP code 400 or greater. The page may contain information that can help determine why the server returns the error code.
Failed request tracing	Windows	App Service file system	Detailed tracing information on failed requests, including a trace of the IIS components used to process the request and the time taken in each component. It's useful if you want to improve site performance or isolate a specific HTTP error. One folder is generated for each failed request, which contains the XML log file, and the XSL stylesheet to view the log file with.
Deployment logging	Windows, Linux	App Service file system	Logs for when you publish content to an app. Deployment logging happens automatically and there are no configurable settings for deployment logging. It helps you determine why a deployment failed. For example, if you use a custom deployment script, you might use deployment logging to determine why the script is failing.

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**
>
> If your Azure Storage account is secured by firewall rules, see **Networking considerations**.

> ⓘ **Note**
>
> Currently only .NET application logs can be written to the blob storage. Java, PHP, Node.js, Python application logs can only be stored on the App Service file system (without code modifications to write logs to external storage).
>
> Also, if you **regenerate your storage account's access keys**, you must reset the respective logging configuration to use the updated access keys. To do this:
>
> 1. In the **Configure** tab, set the respective logging feature to **Off**. Save your setting.
> 2. Enable logging to the storage account blob again. Save your setting.

Select the **Level**, or the level of details to log. The following table shows the log categories included in each level:

Level	Included categories
Disabled	None
Error	Error, Critical
Warning	Warning, Error, Critical
Information	Info, Warning, Error, Critical
Verbose	Trace, Debug, Info, Warning, Error, Critical (all categories)

When finished, select **Save**.

> ⓘ **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.

> **Note**
>
> If your Azure Storage account is secured by firewall rules, see **Networking considerations**.

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**, then select **Save**.

Both types of logs are stored in the App Service file system. Up to 50 errors (files/folders) are retained. When the number of HTML files 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 out of order events in the stream. For example, an application log entry that occurs when a user visits a page may be displayed in the stream before the corresponding HTTP log entry for the page request.

In Azure portal

To stream logs in the Azure portal, navigate to your app and select **Log stream**.

In Cloud Shell

To stream logs live in Cloud Shell, use the following command:

> **Important**
>
> This command may not work with web apps hosted in a Linux app service plan.

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 local terminal

To stream logs in the local console, install Azure CLI and sign in to your account. Once signed in, followed the instructions for Cloud Shell

Access log files

If you configure the Azure Storage blobs option for a log type, you need a client tool that works with Azure Storage. For more information, see Azure Storage Client Tools.

For logs stored in the App Service file system, the easiest way is to download the ZIP file in the browser at:

- Linux/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.

For Windows apps, the ZIP file contains the contents of the *D:\Home\LogFiles* directory in the App Service file system. It has the following structure:

Log type	Directory	Description
Application logs	*/LogFiles/Application/*	Contains one or more text files. The format of the log messages depends on the logging provider you use.
Failed Request Traces	*/LogFiles/W3SVC########/*	Contains XML files, and an XSL file. You can view the formatted XML files in the browser.
Detailed Error Logs	*/LogFiles/DetailedErrors/*	Contains HTM error files. You can view the HTM files in the browser. Another way to view the failed request traces is to navigate to your app page in the portal. From the left menu, select **Diagnose and solve problems**, then search for **Failed Request Tracing Logs**, then click the icon to browse and view the trace you want.

Log type	Directory	Description
Web Server Logs	*/LogFiles/http/RawLogs/*	Contains text files formatted using the W3C extended log file format. This information can be read using a text editor or a utility like Log Parser. App Service doesn't support the `s-computername`, `s-ip`, or `cs-version` fields.
Deployment logs	*/LogFiles/Git/* and */deployments/*	Contain logs generated by the internal deployment processes, as well as logs for Git deployments.

Send logs to Azure Monitor

With the new Azure Monitor integration, you can create Diagnostic Settings to send logs to Storage Accounts, Event Hubs and Log Analytics.

Supported log types

The following table shows the supported log types and descriptions:

Log type	Windows	Windows Container	Linux	Linux Container	Description
AppServiceConsoleLogs	Java SE & Tomcat	Yes	Yes	Yes	Standard output and standard error
AppServiceHTTPLogs	Yes	Yes	Yes	Yes	Web server logs
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, .NET Core, & Tomcat [1]	ASP.NET & Tomcat [1]	.NET Core, Java, SE & Tomcat Blessed Images [2]	Java SE & Tomcat Blessed Images [2]	Application logs
AppServiceIPSecAuditLogs	Yes	Yes	Yes	Yes	Requests from IP Rules
AppServicePlatformLogs	TBA	Yes	Yes	Yes	Container operation logs

Log type	Windows	Windows Container	Linux	Linux Container	Description
AppServiceAntivirusScanAuditLogs [3]	Yes	Yes	Yes	Yes	Anti-virus scan logs using Microsoft Defender for Cloud; **only available for Premium tier**

[1] For Tomcat apps, add `TOMCAT_USE_STARTUP_BAT` to the app settings and set it to `false` or `0`. Need to be on the *latest* Tomcat version and use *java.util.logging*.

[2] For Java SE apps, add `WEBSITE_AZMON_PREVIEW_ENABLED` to the app settings and set it to `true` or to `1`.

[3] AppServiceAntivirusScanAuditLogs log type is still currently in Preview

Networking considerations

If you secure your Azure Storage account by only allowing selected networks, it can receive logs from App Service only if both of the following are true:

- The Azure Storage account is in a different Azure region from the App Service app.
- All outbound addresses of the App Service app are added to the Storage account's firewall rules. To find the outbound addresses for your app, see Find outbound IPs.

Next steps

- Query logs with Azure Monitor
- How to Monitor Azure App Service
- Troubleshooting Azure App Service in Visual Studio
- Analyze app Logs in HDInsight
- Tutorial: Run a load test to identify performance bottlenecks in a web app

Open an SSH session to a Linux container in Azure App Service

Article • 11/18/2022

Secure Shell (SSH) is commonly used to execute administrative commands remotely from a command-line terminal. App Service on Linux provides SSH support into the app container.

You can also connect to the container directly from your local development machine using SSH and SFTP.

Open SSH session in browser

To make open a direct SSH session with your container, your app should be running.

Paste the following URL into your browser and replace `<app-name>` with your app name:

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

If you're not yet authenticated, you're required to authenticate with your Azure subscription to connect. Once authenticated, you see an in-browser shell, where you can run commands inside your container.

Use SSH support with custom Docker images

See Configure SSH in a custom container.

Open SSH session from remote shell

> **Note**
>
> This feature is currently in Preview.

Using TCP tunneling you can create a network connection between your development machine and Web App for Containers over an authenticated WebSocket connection. It enables you to open an SSH session with your container running in App Service from the client of your choice.

To get started, you need to install Azure CLI. To see how it works without installing Azure CLI, open Azure Cloud Shell.

Open a remote connection to your app using the az webapp 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

```
Port 21382 is open
SSH is available { username: root, password: Docker! }
Start your favorite client and connect to port 21382
```

Open an SSH session with your container with the client of your choice, using the local port. The following example uses the default ssh command:

Bash

```
ssh root@127.0.0.1 -p <port>
```

When being prompted, type `yes` to continue connecting. You are then prompted for the password. Use `Docker!`, which was shown to you earlier.

```
Warning: Permanently added '[127.0.0.1]:21382' (ECDSA) to the list of known
hosts.
root@127.0.0.1's password:
```

Once you're authenticated, you should see the session welcome screen.

```
  _____
  /  _  \ __________ _________   ____
 /  /_\  \___    /  |  \_  __ \_/ __ \
/    |    \/    /|  |  /|  | \/\  ___/
\____|__  /_____ \____/ |__|    \___  >
        \/      \/                  \/
A P P   S E R V I C E   O N   L I N U X

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
- Using Ruby in Azure App Service on Linux
- Azure App Service Web App for Containers FAQ

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.

  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 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.

Break the app

Now that you've verified the app by converting two images successfully, we try to convert the first five images.

This action fails and produces a `HTTP 500` error that wasn't tested during development.

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`.

3. Click `Run`.

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
<?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

```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

Monitoring App Service

Article • 09/12/2023

When you have critical applications and business processes relying on Azure resources, you want to monitor those resources for their availability, performance, and operation. This article describes the monitoring data generated by App Service and shipped to Azure Monitor. You can also use built-in diagnostics to monitor resources to assist with debugging an App Service app. If you're unfamiliar with the features of Azure Monitor common to all Azure services, read Monitoring Azure resources with Azure Monitor.

Monitoring data

App Service collects the same kinds of monitoring data as other Azure resources that are described in Monitoring data from Azure resources.

See Monitoring *App Service* data reference for detailed information on App Service metrics and logs.

App Service also provides built-in diagnostics to assist with debugging apps. See Enable diagnostics logging for more information on enabling the built-in logs. To monitor App Service instances, see Monitor App Service instances using Health check.

Application Insights

Application Insights monitors the availability, performance, and usage of your web applications. It leverages the powerful data analysis platform in Azure Monitor to provide you with deep insights into your application's operations. It enables you to diagnose errors without waiting for a user to report them. Application Insights includes connection points to a variety of development tools and integrates with Visual Studio to support your DevOps processes. Learn more about Application Insights in the Application Monitoring for App Service Overview.

Collection and routing

Platform metrics and the Activity log are collected and stored automatically, but can be routed to other locations by using a diagnostic setting.

Resource Logs aren't collected and stored until you create a diagnostic setting and route them to one or more locations.

See Create diagnostic setting to collect platform logs and metrics in Azure for the detailed process for creating a diagnostic setting using the Azure portal, CLI, or PowerShell. When you create a diagnostic setting, you specify which categories of logs to collect. The categories for *App Service* are listed in App Service monitoring data reference.

The metrics and logs you can collect are discussed in the following sections.

> **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**.

Analyzing metrics

You can analyze metrics for *App Service* with metrics from other Azure services using metrics explorer by opening **Metrics** from the **Azure Monitor** menu. See Getting started with Azure Metrics Explorer for details on using this tool.

For a list of platform metrics collected for App Service, see Monitoring App Service data reference metrics

For reference, you can see a list of all resource metrics supported in Azure Monitor.

Analyzing logs

Data in Azure Monitor Logs is stored in tables where each table has its own set of unique properties.

All resource logs in Azure Monitor have the same fields followed by service-specific fields. The common schema is outlined in Azure Monitor resource log schema.

The Activity log is a type of platform log that provides insight into subscription-level events. You can view it independently or route to Azure Monitor Logs. Routing to Azure Monitor Logs gives the benefit of using Log Analytics to run complex queries.

For a list of types of resource logs collected for App Service, see Monitoring App Service data reference

For a list of queryable tables used by Azure Monitor Logs and Log Analytics, see Monitoring App Service data reference.

Sample Kusto queries

> **Important**
>
> When you select **Logs** from the App Service menu, Log Analytics is opened with the query scope set to the current resource. This means that log queries will only include data from that resource. If you want to run a query that includes data from other [resource] or 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.

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 important conditions are found in your monitoring data. They allow you to identify and address issues in your system before your customers notice them. You can set alerts on metrics, logs, and the activity log.

If you're running an application on App Service Azure Monitor Application Insights offers more types of alerts.

The following table lists common and recommended alert rules for App Service.

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

Next steps

- See Monitoring App Service data reference for a reference of metrics, logs, and other important values created by App Service.
- See Monitoring Azure resources with Azure Monitor for details on monitoring Azure resources.

Monitoring App Service data reference

Article • 06/29/2023

This reference applies to the use of Azure Monitor for monitoring App Service. See Monitoring App Service for details on collecting and analyzing monitoring data for App Service.

Metrics

This section lists all the automatically collected platform metrics collected for App Service.

Metric Type	Resource Provider / Type Namespace and link to individual metrics
App Service Plans	Microsoft.Web/serverfarms
Web apps	Microsoft.Web/sites
Staging slots	Microsoft.Web/sites/slots
App Service Environment	Microsoft.Web/hostingEnvironments
App Service Environment Front-end	Microsoft.Web/hostingEnvironments/multiRolePools

For more information, see a list of all platform metrics supported in Azure Monitor.

Metric Dimensions

For more information on what metric dimensions are, see Multi-dimensional metrics.

App Service doesn't have any metrics that contain dimensions.

Resource logs

This section lists the types of resource logs you can collect for App Service.

Log type	Windows	Windows Container	Linux	Linux Container	Description

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 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

Log type	Windows	Windows Container	Linux	Linux Container	Description
AppServiceAntivirusScanAuditLogs	Yes	Yes	Yes	Yes	Anti-virus scan logs 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.

For reference, see a list of all resource logs category types supported in Azure Monitor.

Azure Monitor Logs tables

Azure App Service uses Kusto tables from Azure Monitor Logs. You can query these tables with Log analytics. For a list of App Service tables used by Kusto, see the Azure Monitor Logs table reference - App Service tables.

Activity log

The following table lists common operations related to App Service that may be created in the Activity log. This is not an exhaustive list.

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

Operation	Description
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

For more information on the schema of Activity Log entries, see Activity Log schema.

See Also

- See Monitoring Azure App Service for a description of monitoring Azure App Service.
- See Monitoring Azure resources with Azure Monitor for details on monitoring Azure resources.

Monitor App Service instances using Health check

Article • 08/14/2023

This article uses 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 every minute a path of your web application of your choice.

Note that */api/health* is just an example added for illustration purposes. We do not create a Health Check path by default. You should make sure that the path you are selecting is a valid path that exists within your application

What App Service does with Health checks

- When given a path on your app, Health check pings this path on all instances of your App Service app at 1-minute intervals.
- If an instance doesn't respond with a status code between 200-299 (inclusive) after 10 requests, App Service determines it's unhealthy and removes it from the load balancer for this Web App. The required number of failed requests for an instance to be deemed unhealthy is configurable to a minimum of two requests.
- After removal, Health check continues to ping the unhealthy instance. If the instance begins to respond with a healthy status code (200-299), then the instance is returned to the load balancer.

- If an instance remains unhealthy for one hour, it's replaced with a new instance.
- 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 your health check is giving 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 on 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 does not return a response within 1 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 your Function App, you must use a **premium or dedicated hosting plan**.

⊗ 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:

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 set to `2`, your instances are removed after `2` failed pings. (Default value is `10`)
`WEBSITE_HEALTHCHECK_MAXUNHEALTHYWORKERPERCENT`	1 - 100	By default, no more than half of the instances will be excluded from the load balancer at one time to avoid overwhelming the remaining healthy instances. For example, if

App setting name	Allowed values	Description
		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 the worst-case scenario where all instances are unhealthy, none are excluded. To override this behavior, set app setting to a value between `1` and `100`. A higher value means more unhealthy instances are removed (default value is `50`).

Authentication and security

Health check integrates with App Service's 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 secure 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.

.NET

```
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 Windows App Service.

Instances

Once Health Check is enabled, you can restart and monitor the status of your application instances through the instances tab. The instances tab shows your instance's name, the status of that instance and gives you the option to manually restart the application instance.

If the status of your instance is unhealthy, you can restart the instance manually 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 are listed on the opening blade from the restart button.

If you restart the instance and the restart process fails, you will then be given the option to replace the worker (only 1 instance can be replaced per hour). This will also affect any applications using the same App Service Plan.

Windows applications will also have the option to 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 in the Health Check tab. Enabling diagnostic collection adds an auto-heal rule that creates memory dumps for unhealthy instances and saves it 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 or choose an existing storage account 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 the **Metrics** in the top toolbar. This will open a new blade where you can see the site's historical health status and option to create a new alert rule. Health check metrics aggregate the successful pings & display failures only when the instance was deemed unhealthy based on the health check configuration. For more information on monitoring your sites, see the guide on Azure Monitor.

Limitations

- Health check can be enabled for **Free** and **Shared** App Service Plans so you can have metrics on the site's health and setup alerts, but because **Free** and **Shared** sites can't scale out, any unhealthy instances won't be replaced. You should scale up to the **Basic** tier or higher so you can scale out to 2 or more instances and utilize the full benefit of Health check. This is recommended for production-facing applications as it will increase your app's availability and performance.
- The App Service plan can have a maximum of one unhealthy instance replaced per hour and, at most, three instances per day.
- There's a non-configurable 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 will not 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 Health check's 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. This also means the request will have an origin of `127.0.0.1` since the request is being sent internally. You can add log statements in your Health check code to keep logs of when your Health check path is pinged.

Are the Health check requests sent over HTTP or HTTPS?

On Windows App Service, the Health check requests are sent via HTTPS when HTTPS Only is enabled on the site. Otherwise, they're sent over HTTP. On Linux App Service, the health check requests are only sent over HTTP and can't be sent over HTTP**S** at this time.

Is Health check following the application code configured redirects between the default domain and the custom domain?

No, the Health check feature is pinging the path of the default domain of the web application. If there is a redirect from the default domain to a custom domain, then the status code that Health check is returning is not going to be a 200 but a redirect (301), which is going to mark 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 over one hour, the instance will only be replaced if all other apps with Health check 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, called App A and App B. They are on the same App Service Plan and that 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 over an hour on those two instances, those instances are only replaced if App B is **also** unhealthy on those instances. If App B is healthy, the instance isn't replaced.

> **Note**
>
> If there were another site or slot on the Plan (Site C) without Health check enabled, it would not be taken into consideration for the instance replacement.

What if all my instances are unhealthy?

In the scenario where all instances of your application are unhealthy, App Service will not 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 instances replacement will still be honored.

Does Health check work on App Service Environments?

Yes, health check is available for the App Service Environment v3, but not for versions 1 or 2. If you are using the older versions of the 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

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.

> **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.

Risk Alerts

The App Service diagnostics homepage performs a series of configuration checks and offers recommendations based on your unique application's configuration.

Recommendations and checks performed can be reviewed by clicking "View more details" link.

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**

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**

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:

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.

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.

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.

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.

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.

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

Monitor apps in Azure App Service

Article • 06/29/2023

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 to be used only for development and testing purposes.

If the app is hosted in a *Free* or *Shared* plan, the limits on the resources that the app can use are defined by quotas.

If the app is hosted in a *Basic*, *Standard*, or *Premium* plan, the limits on the resources that they can use are set by the *size* (Small, Medium, Large) and *instance count* (1, 2, 3, and so on) of the App Service plan.

Quotas for Free or Shared apps are:

Quota	Description
CPU (Short)	The amount of CPU allowed for this app in a 5-minute interval. This quota resets every five minutes.
CPU (Day)	The total amount of CPU allowed for this app in a day. This quota resets every 24 hours at midnight UTC.
Memory	The total amount of memory allowed for this app.
Bandwidth	The total amount of outgoing bandwidth allowed for this app in a day. This quota resets every 24 hours at midnight UTC.

Quota	Description
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 an app, the available metrics are:

Metric	Description
Response Time	The time taken for the app to serve requests, in seconds.
Average Response Time (deprecated)	The average time taken for the app to serve requests, in seconds.
Average memory working set	The average amount of memory used by the app, in megabytes (MiB).
Connections	The number of bound sockets existing in the sandbox (w3wp.exe and its child processes). A bound socket is created by calling bind()/connect() APIs and remains until said socket is closed with CloseHandle()/closesocket().
CPU Time	The amount of CPU consumed by the app, in seconds. For more information about this metric, see CPU time vs CPU percentage.
Current Assemblies	The current number of Assemblies loaded across all AppDomains in this application.
Data In	The amount of incoming bandwidth consumed by the app, in MiB.
Data Out	The amount of outgoing bandwidth consumed by the app, in MiB.

Metric	Description
File System Usage	The amount of usage in bytes by storage share.
Gen 0 Garbage Collections	The number of times the generation 0 objects are garbage collected since the start of the app process. Higher generation GCs include all lower generation GCs.
Gen 1 Garbage Collections	The number of times the generation 1 objects are garbage collected since the start of the app process. Higher generation GCs include all lower generation GCs.
Gen 2 Garbage Collections	The number of times the generation 2 objects are garbage collected since the start of the app process.
Handle Count	The total number of handles currently open by the app process.
Health Check Status	The average health status across the application's instances in the App Service Plan.
Http 2xx	The count of requests resulting in an HTTP status code ≥ 200 but < 300.
Http 3xx	The count of requests resulting in an HTTP status code ≥ 300 but < 400.
Http 401	The count of requests resulting in HTTP 401 status code.
Http 403	The count of requests resulting in HTTP 403 status code.
Http 404	The count of requests resulting in HTTP 404 status code.
Http 406	The count of requests resulting in HTTP 406 status code.
Http 4xx	The count of requests resulting in an HTTP status code ≥ 400 but < 500.
Http Server Errors	The count of requests resulting in an HTTP status code ≥ 500 but < 600.
IO Other Bytes Per Second	The rate at which the app process is issuing bytes to I/O operations that don't involve data, such as control operations.
IO Other Operations Per Second	The rate at which the app process is issuing I/O operations that aren't read or write operations.
IO Read Bytes Per Second	The rate at which the app process is reading bytes from I/O operations.

Metric	Description
IO Read Operations Per Second	The rate at which the app process is issuing read I/O operations.
IO Write Bytes Per Second	The rate at which the app process is writing bytes to I/O operations.
IO Write Operations Per Second	The rate at which the app process is issuing write I/O operations.
Memory working set	The current amount of memory used by the app, in MiB.
Private Bytes	Private Bytes is the current size, in bytes, of memory that the app process has allocated that can't be shared with other processes.
Requests	The total number of requests regardless of their resulting HTTP status code.
Requests In Application Queue	The number of requests in the application request queue.
Thread Count	The number of threads currently active in the app process.
Total App Domains	The current number of AppDomains loaded in this application.
Total App Domains Unloaded	The total number of AppDomains unloaded since the start of the application.

For an App Service plan, the available metrics are:

> **Note**
>
> App Service plan metrics are available only for plans in *Basic*, *Standard*, and *Premium* tiers.

Metric	Description
CPU Percentage	The average CPU used across all instances of the plan.

Metric	Description
Memory Percentage	The average memory used across all instances of the plan.
Data In	The average incoming bandwidth used across all instances of the plan.
Data Out	The average outgoing bandwidth used across all instances of the plan.
Disk Queue Length	The average number of both read and write requests that were queued on storage. A high disk queue length is an indication of an app that might be slowing down because of excessive disk I/O.
Http Queue Length	The average number of HTTP requests that had to sit on the queue before being fulfilled. A high or increasing HTTP Queue length is a symptom of a plan under heavy load.

CPU time vs CPU percentage

There are two metrics that reflect CPU usage:

CPU Time: Useful for apps hosted in Free or Shared plans, because one of their quotas is defined in CPU minutes used by the app.

CPU percentage: Useful for apps hosted in Basic, Standard, and Premium plans, because they can be scaled out. CPU percentage is a good indication of the overall usage across all instances.

Metrics granularity and retention policy

Metrics for an app and app service plan are logged and aggregated by the service 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 Receive alert notifications.

App Service apps hosted in Basic or higher App Service plans support autoscale. With autoscale, you can configure rules that monitor the App Service plan metrics. Rules can increase or decrease the instance count, which can provide additional resources as needed. Rules can also help you save money when the app is over-provisioned.

For more information about autoscale, see How to scale and Best practices for Azure Monitor autoscaling.

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**.

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.

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**.

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.

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**.

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.

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**.

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 • 03/09/2023

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.

The DNS record type you need to add with your domain provider depends on the domain you want to add to App Service.

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 `www` 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 and not **Free (F1)**. 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.

1. 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**.

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 **Basic** tier or higher. If you want to remain in **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.

Setting	Description

Setting	Description
Custom domain	The domain name to add the TLS/SSL binding for.
Private Certificate Thumbprint	The certificate to bind.
TLS/SSL Type	- **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.

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 `*.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. Select the respective **Copy** button to help you with the next step.

   The following screenshot shows the default selections for a `www.contoso.com` domain, which shows a CNAME record and a TXT record to add.

> ⚠ **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**.

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:

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 (e.g. contoso.com)

Create two records according to the following 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 root domain, App Service accesses `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>

3. Validate and complete

1. Back in the **Add custom domain** dialog in the Azure portal, select **Validate**.

2. If the **Domain validation** section shows green check marks next for both domain records, then you've configured them correctly. Select **Add**. If you see any errors or warnings, fix it in the DNS record settings on your domain provider's website.

> **Note**
>
> If you configured the TXT record but not the A or CNAME record, App Service treats it 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 may also see a red X with **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 **Secured**. If you selected **Add certificate later**, this red X will remain until you add a private certificate for the domain and configure the binding.

> **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. Test in a browser

Browse to the DNS names that you configured earlier.

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 DNS resolution again. On a Windows machine, you 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

Buy an App Service domain and configure an app with it

Article • 02/27/2023

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 and not **Free (F1)**. See Scale up an app to update the tier.
- Remove the spending limit on your subscription.

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**.

> **Note**
>
> You can also create an App Service domain independently from an app by going to the App Service Domains view and select **Add**, or navigating to **the create page directly**. But since it's independent from your app, you won't be able to assign hostnames like `www` to your app as if you launch it from your app's **Custom domains** page.

4. In the **Basics** tab, configure the settings using the following 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	Type the domain you want. For example, **contoso.com**. If the domain you want isn't available, you can select from a list of suggestions of available domains, or try a different domain.

> **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.

6. Select **Next: Hostname assignment** and verify the default hostnames to map to your app:

Hostname	Description
root(@)	The root or apex subdomain. If you buy the `contoso.com` domain, then it's the root domain. Select **No** if you don't want to map it to your app.
'www' subdomain	If you buy the `contoso.com` domain, the `www` subdomain would be `www.contoso.com`. Select **No** if you don't want to map it 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 them later by following the steps at **Map a hostname manually**.

7. Select **Next: Advanced** and configure the optional settings:

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 that 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 it disabled, you must renew it manually.
Privacy protection	Enabled by default. Privacy protection hides your domain registration contact information from the WHOIS database. Privacy protection is already included in the yearly domain registration fee. To opt out, select **Disable**.

8. Select **Next: Tags** and set the tags you want for your App Service domain. Tagging isn't required for using App Service domains, but is 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 its management page.

You're now ready to assign an App Service app to this custom domain.

> **Note**
>
> Depending on the subscription type, a sufficient payment history may be required prior to creating an App Service domain.

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**.

4. For **Domain provider**, select **App Service Domain**.

5. For **TLS/SSL certificate**, select **App Service Managed Certificate** if your app is in **Basic** tier or higher. If you want to remain in **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.

Setting	Description
Custom domain	The domain name to add the TLS/SSL binding for.
Private Certificate Thumbprint	The certificate to bind.
TLS/SSL Type	- **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.

7. In **App Service Domain**, select an App Service domain in your subscription.

> ⓘ **Note**

To map from 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:

Domain type	Description
Root domain	The root or apex subdomain. If you buy the `contoso.com` domain, then it's the root domain.
Subdomain	In the **Subdomain** textbox, specify a subdomain like `www` or `shoppingcart`.

9. Select **Add**.

10. You should see the custom domain added to the list. You may also see a red X with **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 **Secured**. If you selected **Add certificate later**, this red X will remain until you add a private certificate for the domain and configure the binding.

> **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 (like `shoppingcart.contoso.com`) in the browser.

Renew the domain

The App Service domain you bought is valid for one year from the time of purchase. You can configure to renew your domain automatically, which will charge your payment method when your domain renews the following year. You can also manually renew your domain name.

If you want to configure automatic renewal, or if you want to manually renew your domain, follow the steps here.

1. In the search bar, search for and select **App Service Domains**.

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.

> **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 like for an externally purchased domain. To manage custom DNS records:

1. In the search bar, search for and select **App Service Domains**.

2. Select the domain you want to configure.

3. From the **Overview** page, select **Manage DNS records**.

For information on how to edit DNS records, see How to manage DNS Zones in the Azure portal.

Cancel purchase (delete domain)

After you purchase the App Service Domain, you have five days to cancel your purchase for a full refund. After five days, you can delete the App Service Domain, but can't receive a refund.

1. In the search bar, search for and select **App Service Domains**.

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**.

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, do not 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 Visual Studio Enterprise Subscription, have a limit of 1 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 steps

Learn how to bind a custom TLS/SSL certificate to App Service.

Secure a custom DNS name with a TLS binding in Azure App Service

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.

In the left navigation of the app page, select **Scale up (App Service plan)**.

The app's current tier is highlighted by a blue border. Check to make sure that the app is 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.

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:

   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.

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, to 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.

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:

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)	*asuid*	Domain verification ID shown in the **Custom domains** management page.
www (sub)	*asuid.www*	Domain verification ID shown in the **Custom domains** management page.
* (wildcard)	*asuid*	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**.

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	- **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`.

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).

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.

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.

> **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.

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

Secure a custom DNS name with a TLS/SSL binding in Azure App Service

Article • 04/20/2023

This article shows you how to secure the custom domain in your App Service app or function app by creating a certificate binding. When you're finished, you can access your App Service app at the `https://` endpoint for your custom DNS name (for example, `https://www.contoso.com`).

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.

1. 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**

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 simplest. For more information, see Create a free managed certificate.
    - **Import App Service Certificate** - In **App Service Certificate**, choose an App Service certificate you've purchased for your selected domain. To purchase an App Service certificate, see Import an App Service certificate.
    - **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**, choose between **SNI SSL** and **IP based SSL**.

    - 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 based 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 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 **Secure**.

> **Note**
>
> A **Secure** state in the **Custom domains** means that it is secured with a certificate, but App Service doesn't check if the certificate is self-signed or expired, for example, which can also cause browsers to show an error or warning.

2. Remap records for IP based SSL

This step is needed only for IP based SSL. For an SNI SSL binding, skip to Test HTTPS for your custom domain.

There are two changes you need to make, potentially:

- By default, your app uses a shared public IP address. When you bind a certificate with IP SSL, App Service creates a new, dedicated IP address for your app. If you mapped an A record to your app, update your domain registry with this new, dedicated IP address.

  Your app's **Custom domain** page is updated with the new, dedicated IP address. Copy this IP address, then remap the A record to this new IP address.

- If you have an SNI SSL binding to `<app-name>.azurewebsites.net`, remap any CNAME mapping to point to `sni.<app-name>.azurewebsites.net` instead (add the `sni` prefix).

3. Test HTTPS

In various browsers, browse to `https://<your.custom.domain>` to verify that it serves up your app.

Your application code can inspect the protocol via the "x-appservice-proto" header. The header 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 may have left out intermediate certificates when you export 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 or not, inspect the `X-Forwarded-Proto` header.

Language specific configuration guides, such as the Linux Node.js configuration guide, shows you how to detect an HTTPS session in your application code.

Automate with scripts

Azure CLI

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 your web app's default domain name.

# Upgrade App Service plan to Basic tier (minimum required by custom SSL
certificates)
Set-AzAppServicePlan -Name $webappname -ResourceGroupName $webappname `
-Tier Basic

# Add a custom domain name to the web app.
Set-AzWebApp -Name $webappname -ResourceGroupName $webappname `
-HostNames @($fqdn,"$webappname.azurewebsites.net")

# Upload and bind the SSL certificate to the web app.
New-AzWebAppSSLBinding -WebAppName $webappname -ResourceGroupName
$webappname -Name $fqdn `
-CertificateFilePath $pfxPath -CertificatePassword $pfxPassword -SslState
SniEnabled
```

More resources

- Use a TLS/SSL certificate in your code in Azure App Service
- FAQ : App Service Certificates

Add and manage TLS/SSL certificates in Azure App Service

Article • 07/28/2023

You can add digital security certificates to use in your application code or to 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:

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 secure 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.
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.

> **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.
- 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 depends 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.

To secure a custom domain in a TLS binding, the certificate has more 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.

Create a free managed certificate

The free App Service managed certificate is a turn-key solution for securing 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: `0 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 (ASE).
- Only supports alphanumeric characters, dashes (-), and periods (.).

Apex domain

- Must have an A record pointing to your web app's IP address.
- Isn't supported on apps that aren't publicly accessible.
- Isn't supported with root domains that are integrated with Traffic Manager.
- Must meet all 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.

4. To 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 an App Service certificate

To import an App Service certificate, first buy and configure an App Service certificate, 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** list.

7. To 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 to the key vault.

> **Note**
>
> Currently, a Key Vault certificate supports only the Key Vault access policy, not RBAC model.

Resource provider	Service principal AppId	Key vault secret permissions	Key vault certificate permissions
Microsoft Azure App Service or Microsoft.Azure.WebSites	- `abfa0a7c-a6b6-4736-8310-5855508787cd`, which	Get	Get

Resource provider	Service principal AppId	Key vault secret permissions	Key vault certificate permissions
	is the same for all Azure subscriptions - For Azure Government cloud environment, use `6a02c803-dafd-4136-b4c3-5a6f318b4714`.		
Microsoft.Azure.CertificateRegistration		Get List Set Delete	Get List

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:

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**, 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.

> **Note**
>
> If you update your certificate in Key Vault with a new certificate, App Service automatically syncs your certificate within 24 hours.

7. To 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.

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 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 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 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 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:

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**, then **Add**.

   When the operation completes, the certificate appears in the **Bring your own certificates** list.

5. To 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.

Upload a public certificate

Public certificates are supported in the *.cer* format.

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:

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 uploaded certificate

When you replace an expiring certificate, the way you update the certificate binding with the new certificate might adversely affect 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 **...** actions button, and select **Update binding**.
3. Select the new certificate and 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 inside 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 **...** details button for the imported key vault certificate, and then select **Sync**.

Frequently asked questions

- How can I automate adding a bring-your-owncertificate to an app?
- Frequently asked questions for App Service certificates

How can I automate adding a bring-your-owncertificate 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

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

Create and manage an App Service certificate for your web app

Article • 07/28/2023

This article shows how to create an App Service certificate and manage it (such as renew, sync, and delete). 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

Start certificate purchase

1. Go to the App Service Certificate creation page, and start your purchase for an App Service certificate.

> **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`

2. To help you configure the certificate, use the following table. When you're done, select **Review + Create**, then select **Create**.

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 Host Name	Specify the root domain. The issued certificate secures *both* the root domain and the `www` subdomain. In the issued certificate, the **Common Name** field specifies the root domain, and the **Subject Alternative Name** field specifies the `www` domain. To secure any subdomain only, specify the fully qualified domain name for the subdomain, for example, `mysubdomain.contoso.com`.
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 and the cost is charged to your subscription.

3. When deployment is complete, select **Go to resource**.

Store 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, the storage of choice is Key Vault. After you finish the certificate purchase process, you must complete a few more steps before you start using this certificate.

1. On the App Service Certificates page, select the certificate. On the certificate menu, select **Certificate Configuration** > **Step 1: Store**.

2. On the **Key Vault Status** page, select **Select from Key Vault**.

3. 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.

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, in which objects remain recoverable (see Azure Key Vault soft-delete overview). Set a value between 7 and 90.
Purge protection	Prevents objects soft-deleted st objects to be manually purged. Enabling this option forces all deleted objects to remain in soft-deleted state for the entire duration of the retention period.

4. Select **Next** and select **Vault access policy**. Currently, App Service certificates support only Key Vault access policies, not the RBAC model.

5. Select **Review + create**, then select **Create**.

6. After the key vault is created, don't select **Go to resource** but 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 in the previous section, select **Step 2: Verify**.

2. Select **App Service Verification**. However, because you previously mapped the domain to your web app per the Prerequisites, 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:

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.

Method	Description
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, which applies only to **Standard** certificates per 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.

> **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 issuing, rekeying, or renewing 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 and nearer to 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 or rekey process. The new certificate order remains in "pending issuance" mode during the renew or rekey process until you complete the domain verification.

Unlike the free App Service managed certificate, domain re-verification for App Service certificates *isn't* automated. 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 well-known **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 select **Save**.

   If you turn on automatic renewal, certificates can start automatically renewing 32 days before expiration.

4. To manually renew the certificate instead, select **Manual Renew**. You can request to manually renew your certificate 60 days before expiration.

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 and App Service certificate

If you think your certificate's private key is compromised, you can rekey your certificate. This action rolls the certificate with a new certificate issued from the certificate authority.

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.

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.

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 this certificate becomes invalid.

1. On the App Service Certificates page, select the certificate.
2. From the left menu, select **Overview** > **Delete**.
3. When the confirmation box opens, enter the certificate name, and select **OK**.

Frequently asked questions

My App Service certificate doesn't have any value in Key Vault

Your App Service certificate is most likely still 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 most likely still 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 on the vault called: `AppServiceCertificateLock` to prevent accidental deletion of the key vault.

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

Use a TLS/SSL certificate in your code in Azure App Service

Article • 03/02/2023

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 above. If your app is in **Free** or **Shared** tier, you can include the certificate file in your app repository.

When you let App Service manage your 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 **TLS/SSL settings**, then select **Private Key Certificates (.pfx)** or **Public Key Certificates (.cer)**.

Find the certificate you want to use and copy the thumbprint.

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 above.

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, Java, or Ruby, see the documentation for the respective language or web platform.

Load certificate in Linux/Windows containers

The `WEBSITE_LOAD_CERTIFICATES` app settings 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:

Container platform	Public certificates	Private certificates
Windows container	`C:\appservice\certificates\public`	`C:\appservice\certificates\private`

Container platform	Public certificates	Private certificates
Linux container	`/var/ssl/certs`	`/var/ssl/private`

The certificate file names are the certificate thumbprints.

> **Note**
>
> App Service inject 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 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 the file paths provided above to Load the certificate directly from file.

The following C# code shows how to load a public certificate in a Linux app.

C#

```
using System;
using System.IO;
using System.Security.Cryptography.X509Certificates;

...
var bytes = File.ReadAllBytes("/var/ssl/certs/<thumbprint>.der");
var cert = new X509Certificate2(bytes);

// Use the loaded certificate
```

The following C# code shows how to load a private certificate in a Linux app.

C#

```
using System;
using System.IO;
using System.Security.Cryptography.X509Certificates;
...
var bytes = File.ReadAllBytes("/var/ssl/private/<thumbprint>.p12");
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, Java, or Ruby, 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 • 10/21/2022

You can restrict access to your Azure App Service app by enabling different types of authentication for it. One way to do it is to request a client certificate when the client request is over TLS/SSL and validate the certificate. This mechanism is called TLS mutual authentication or client certificate authentication. This article shows how to set up your app to use client certificate authentication.

> **Note**
>
> If you access your site over HTTP and not HTTPS, you will not receive any client certificate. So if your application requires client certificates, you should not allow requests to your application over HTTP.

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**.

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

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

To set up your app to require client certificates:

1. From the left navigation of your app's management page, select **Configuration** > **General Settings**.
2. Set **Client certificate mode** to **Require**. Click **Save** at the top of the page.

Azure CLI

To do the same with Azure CLI, run the following command in the Cloud Shell:

Azure CLI

```
az webapp update --set clientCertEnabled=true --name <app-name> --
resource-group <group-name>
```

Exclude paths from requiring authentication

When you enable mutual auth for your application, all paths under the root of your app require a client certificate for access. To remove this requirement for certain paths, define exclusion paths as part of your application configuration.

1. From the left navigation of your app's management page, select **Configuration** > **General Settings**.
2. Next to **Certificate exclusion paths**, click the edit icon.
3. Click **New path**, specify a path, or a list of paths separated by `,` or `;`, and click **OK**.
4. Click **Save** at the top of the page.

In the following screenshot, any path for your app that starts with `/public` does not request a client certificate. Path matching is case-insensitive.

Access client certificate

In App Service, TLS termination of the request happens at the frontend load balancer. When forwarding the request to your app code with client certificates enabled, App Service injects an `X-ARR-ClientCert` request header with the client certificate. App Service does not do anything with this client certificate other than forwarding it to your app. Your app code is responsible for validating the client certificate.

For ASP.NET, the client certificate is available through the **HttpRequest.ClientCertificate** property.

For other application stacks (Node.js, PHP, etc.), the client cert is available in your app through a base64 encoded value in the `X-ARR-ClientCert` request header.

ASP.NET 5+, ASP.NET Core 3.1 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 forwared 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.ToShortDateString() + " " +
certificate.NotBefore.ToShortTimeString();
                        certExpiryDate =
certificate.NotAfter.ToShortDateString() + " " +
certificate.NotAfter.ToShortTimeString();
                    }
                    catch (Exception ex)
                    {
                        errorString = ex.ToString();
                    }
                    finally
                    {
                        isValidCert = IsValidClientCertificate();
                        if (!isValidCert) Response.StatusCode = 403;
                        else Response.StatusCode = 200;
                    }
                }
                else
                {
                    certHeader = "";
                }
            }

            //
            // This is a SAMPLE verification routine. Depending on your
application logic and security requirements,
            // you should modify this method
            //
            private bool IsValidClientCertificate()
            {
                // In this example we will only accept the certificate as a
valid certificate if all the conditions below are met:
                // 1. The certificate is not expired and is active for the
current time on server.
                // 2. The subject name of the certificate has the common
name nildevecc
                // 3. The issuer name of the certificate has the common name
nildevecc and organization name Microsoft Corp
                // 4. The thumbprint of the certificate is
30757A2E831977D8BD9C8496E4C99AB26CB9622B
                //
```

```
                // This example does NOT test that this certificate is
chained to a Trusted Root Authority (or revoked) on the server
                // and it allows for self signed certificates
                //

                if (certificate == null ||
!String.IsNullOrEmpty(errorString)) return false;

                // 1. Check time validity of certificate
                if (DateTime.Compare(DateTime.Now, certificate.NotBefore) <
0 || DateTime.Compare(DateTime.Now, certificate.NotAfter) > 0) return false;

                // 2. Check subject name of certificate
                bool foundSubject = false;
                string[] certSubjectData = certificate.Subject.Split(new
char[] { ',' }, StringSplitOptions.RemoveEmptyEntries);
                foreach (string s in certSubjectData)
                {
                    if (String.Compare(s.Trim(), "CN=nildevecc") == 0)
                    {
                        foundSubject = true;
                        break;
                    }
                }
                if (!foundSubject) return false;

                // 3. Check issuer name of certificate
                bool foundIssuerCN = false, foundIssuerO = false;
                string[] certIssuerData = certificate.Issuer.Split(new
char[] { ',' }, StringSplitOptions.RemoveEmptyEntries);
                foreach (string s in certIssuerData)
                {
                    if (String.Compare(s.Trim(), "CN=nildevecc") == 0)
                    {
                        foundIssuerCN = true;
                        if (foundIssuerO) break;
                    }

                    if (String.Compare(s.Trim(), "O=Microsoft Corp") == 0)
                    {
                        foundIssuerO = true;
                        if (foundIssuerCN) break;
                    }
                }

                if (!foundIssuerCN || !foundIssuerO) return false;

                // 4. Check thumprint of certificate
                if (String.Compare(certificate.Thumbprint.Trim().ToUpper(),
"30757A2E831977D8BD9C8496E4C99AB26CB9622B") != 0) return false;

                return true;
            }
        }
    }
```

Node.js sample

The following Node.js sample code gets the `X-ARR-ClientCert` header and uses node-forge to convert the base64-encoded PEM string into a certificate object and validate it:

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)).getB
ytes()).digest().toHex();
            if (fingerPrint.toLowerCase() !==
'abcdef1234567890abcdef1234567890abcdef12') throw new Error('UNAUTHORIZED');

            // Validate time validity
            const currentDate = new Date();
            if (currentDate < incomingCert.validity.notBefore || currentDate
> incomingCert.validity.notAfter) throw new Error('UNAUTHORIZED');

            // Validate issuer
            if (incomingCert.issuer.hash.toLowerCase() !==
'abcdef1234567890abcdef1234567890abcdef12') throw new Error('UNAUTHORIZED');

            // Validate subject
            if (incomingCert.subject.hash.toLowerCase() !==
'abcdef1234567890abcdef1234567890abcdef12') throw new Error('UNAUTHORIZED');

            next();
        } catch (e) {
            if (e instanceof Error && e.message === 'UNAUTHORIZED') {
                res.status(401).send();
            } else {
```

```
                next(e);
            }
        }
    }
}
```

Java sample

The following Java class encodes the certificate from `X-ARR-ClientCert` to an `X509Certificate` instance. `certificateIsValid()` validates that the certificate's thumbprint matches the one given in the constructor and that certificate has not expired.

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 has not expired.
     * @return True if the certificate's thumbprint matches and has not
expired. False otherwise.
     */
    public boolean certificateIsValid() throws NoSuchAlgorithmException,
CertificateEncodingException {
        return certificateHasNotExpired() && thumbprintIsValid();
    }

    /**
     * Check certificate's timestamp.
     * @return Returns true if the certificate has not expired. Returns
false if it has expired.
     */
    private boolean certificateHasNotExpired() {
        Date currentTime = new java.util.Date();
        try {
            this.getCertificate().checkValidity(currentTime);
        } catch (CertificateExpiredException |
CertificateNotYetValidException e) {
            return false;
        }
        return true;
    }

    /**
     * Check the certificate's thumbprint matches the given one.
     * @return Returns true if the thumbprints match. False otherwise.
     */
    private boolean thumbprintIsValid() throws NoSuchAlgorithmException,
CertificateEncodingException {
        MessageDigest md = MessageDigest.getInstance("SHA-1");
        byte[] 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;
    }
}
```

Securely connect to Azure services and databases from Azure App Service

Article • 02/28/2022

Your app service may need to connect to other Azure services such as a database, storage, or another app. This overview recommends the more secure method for connecting.

Connection method	When to use
Direct connection from App Service managed identity	Dependent service supports managed identity * Best for enterprise-level security. * Connection to dependent service is secured with managed identity. * Large team or automated connection string and secret management. * Don't manage credentials manually. * Credentials aren't accessible to you. * An Azure Active Directory Identity is required to access. Services include Microsoft Graph or Azure management SDKs.
Connect using Key Vault secrets from App Service managed identity	Dependent service doesn't support managed identity. * Best for enterprise-level security. * Connection includes non-Azure services such as GitHub, Twitter, Facebook, Google * Large team or automated connection string and secret management * Don't manage credentials manually. * Credentials aren't accessible to you. * Manage connection information with environment variables.
Connect with app settings	* Best for small team or individual owner of Azure resources. * Stage 1 of multi-stage migration to Azure. * Temporary or proof-of-concept applications. * Manually manage connection information with environment variables.

Connect to Azure services with managed identity

Use managed identity to authenticate from one Azure resource, such as Azure app service, to another Azure resource whenever possible. This level of authentication lets Azure manage the authentication process, after the required setup is complete. Once the connection is set up, you won't need to manage the connection.

Benefits of managed identity:

- Automated credentials management
- Many Azure services are included
- No additional cost
- No code changes

Learn which services are supported with managed identity and what operations you can perform.

Example managed identity scenario

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 managed identity.
- C: Securely **connect from** App Service **to** Microsoft Graph.

Connect to Key Vault with managed identity

When managed identity isn't supported for your app's dependent services, use Key Vault to store your secrets, and connect your app to Key Vault with a managed identity.

Secrets include:

Secret	Example
Certificates	SSL certificates
Keys and access tokens	Cognitive service API Key GitHub personal access token Twitter consumer keys and authentication tokens
Connection strings	Database connection strings such as SQL server or MongoDB

Benefits of managed identity integrated with Key Vault include:

- Connectivity to Key Vault is secured by managed identities

- 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.
- No code change is required if your application code already accesses connection secrets with app settings.
- Monitoring and auditing of who accessed secrets.
- Rotation of connection information in Key Vault requires no changes in App Service.

Connect with app settings

The App Service provides App settings to store connection strings, API keys, and other environment variables. While App Service does provide encryption for app settings, for enterprise-level security, consider other services to manage these types of secrets that provide additional benefits.

App settings best used when:

- Security of connection information is manual and limited to a few people
- Web app is temporary, proof-of-concept, or in first migration stage to Azure

App Service managed identity to another Azure service best when:

- You don't need to manage Azure credentials. Credentials aren't even accessible to you.
- You can use managed identities to authenticate to any resource that supports Azure Active Directory authentication including your own applications.
- Managed identities can be used without any additional cost.

Key Vault integration from App Service with managed identity best used when:

- Connectivity to Key Vault is secured by managed identities.
- 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.
- No code change is required if your application code already accesses connection secrets with app settings.
- Monitoring and auditing of who accessed secrets.

Next steps

- Learn how to use App Service managed identity with:

- SQL server
- Azure storage
- Microsoft Graph

Tutorial: Deploy an ASP.NET Core and Azure SQL Database app to Azure App Service

Article • 05/24/2023

In this tutorial, you'll 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 7.0 app, the process is the same for other versions of ASP.NET Core and ASP.NET Framework.

This tutorial requires:

- 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.

Sample application

To explore the sample application used in this tutorial, download it from the repository https://github.com/Azure-Samples/msdocs-app-service-sqldb-dotnetcore or clone it using the following Git command:

terminal

```
git clone https://github.com/Azure-Samples/msdocs-app-service-sqldb-dotnetcore.git
cd msdocs-app-service-sqldb-dotnetcore
```

1. 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 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 .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 7 (STS)**.
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. **SQLAzure** is selected by default as the database engine. 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.

8. Select **Review + create**.
9. After validation completes, select **Create**.

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 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.
- **Private DNS zones** → Enable DNS resolution of the database server and the Redis cache in the virtual network.

2. Verify connection strings

The creation wizard generated connection strings for the SQL database and the Redis cache already. In this step, find the generated connection strings for later.

Step 1. In the App Service page, in the left menu, select Configuration.

Step 2.

1. Scroll to the bottom of the page and find **AZURE_SQL_CONNECTIONSTRING** in the **Connection strings** section. This string was generated from the new SQL database by the creation wizard. To set up your application, this name is all you need.
2. Also, find **AZURE_REDIS_CONNECTIONSTRING** in the **Application settings** section. This string was 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 to use `AZURE_SQL_CONNECTIONSTRING` and `AZURE_REDIS_CONNECTIONSTRING`.

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 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-app-service-sqldb-dotnetcore.
3. Select **Fork**.
4. Select **Create fork**.

Step 2. In the App Service page, in the left menu, select **Deployment Center**.

Step 3. 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 **main**.
6. In the top menu, select **Save**. App Service commits a workflow file into the chosen GitHub repository, in the `.github/workflows` directory.

Step 4. Back the GitHub page of the forked sample, open Visual Studio Code in the browser by pressing the `.` key.

Step 5. In Visual Studio Code in the browser:

1. Open *DotNetCoreSqlDb/appsettings.json* in the explorer.
2. Change the connection string name `MyDbConnection` to `AZURE_SQL_CONNECTIONSTRING`, which matches the connection string created in App Service earlier.

Step 6.

1. Open *DotNetCoreSqlDb/Program.cs* in the explorer.
2. In the `options.UseSqlServer` method, change the connection string name `MyDbConnection` to `AZURE_SQL_CONNECTIONSTRING`. This is where the connection string is used by the sample application.

3. Remove the `builder.Services.AddDistributedMemoryCache();` method and replace it with the following code. It changes your code from using an in-memory cache to the Redis cache in Azure, and it does so by using `AZURE_REDIS_CONNECTIONSTRING` from earlier.

```
C#

builder.Services.AddStackExchangeRedisCache(options =>
{
options.Configuration =
builder.Configuration["AZURE_REDIS_CONNECTIONSTRING"];
options.InstanceName = "SampleInstance";
});
```

Step 7.

1. Open *.github/workflows/main_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`.
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 -p DotNetCoreSqlDb/DotNetCoreSqlDb.csproj -o ${{env.DOTNET_ROOT}}/myapp/migrate`. 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.

Step 8.

1. Select the **Source Control** extension.
2. In the textbox, type a commit message like `Configure DB & Redis & add migration bundle`.
3. Select **Commit and Push**.

Step 9. 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.

Step 10. 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 a few minutes.

4. Generate database schema

With the SQL Database protected by the virtual network, the easiest way to run Run dotnet database migrations is in an SSH session with the App Service container.

Step 1. Back in the App Service page, in the left menu, select **SSH**.

Step 2. In the SSH terminal:

1. Run `cd /home/site/wwwroot`. Here are all your deployed files.
2. Run the migration bundle that's generated by the GitHub workflow with `./migrate`. If it succeeds, App Service is connecting successfully to the SQL Database. Only changes to files in `/home` can persist beyond app restarts. Changes outside of `/home` aren't persisted.

```
A P P   S E R V I C E   O N   L I N U X

Documentation: http://aka.ms/webapp-linux
Dotnet quickstart: https://aka.ms/dotnet-qs
ASP .NETCore Version: 7.0.5
Note: Any data outside '/home' is not persisted
root@223eb839fdcf:~/site/wwwroot# cd ~/site/wwwroot
root@223eb839fdcf:~/site/wwwroot# ./migrate
info: Microsoft.EntityFrameworkCore.Database.Command[20101]
      Executed DbCommand (67ms) [Parameters=[], CommandType='Text', CommandTimeout='
30']
      SELECT 1
info: Microsoft.EntityFrameworkCore.Database.Command[20101]
      Executed DbCommand (58ms) [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 1
info: Microsoft.EntityFrameworkCore.Database.Command[20101]
      Executed DbCommand (22ms) [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 (10ms) [Parameters=[], CommandType='Text', CommandTimeout='
30']
      SELECT 1
info: Microsoft.EntityFrameworkCore.Database.Command[20101]
      Executed DbCommand (9ms) [Parameters=[], CommandType='Text', CommandTimeout='3
0']
      SELECT OBJECT_ID(N'[__EFMigrationsHistory]');
info: Microsoft.EntityFrameworkCore.Database.Command[20101]
      Executed DbCommand (10ms) [Parameters=[], CommandType='Text', CommandTimeout='
30']
      SELECT [MigrationId], [ProductVersion]
      FROM [__EFMigrationsHistory]
      ORDER BY [MigrationId];
info: Microsoft.EntityFrameworkCore.Migrations[20402]
      Applying migration '20230509140701_InitialCreate'.
Applying migration '20230509140701_InitialCreate'.
info: Microsoft.EntityFrameworkCore.Database.Command[20101]
      Executed DbCommand (5ms) [Parameters=[], CommandType='Text', CommandTimeout='3
0']
      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 (5ms) [Parameters=[], CommandType='Text', CommandTimeout='3
0']
      INSERT INTO [__EFMigrationsHistory] ([MigrationId], [ProductVersion])
      VALUES (N'20230509140701_InitialCreate', N'7.0.5');
Done.
root@223eb839fdcf:~/site/wwwroot#
```

Menu | ssh://root@169.254.130.4:2222 | SSH CONNECTION ESTABLISHED

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`.

Step 2. Add a few tasks to the list. Congratulations, you're running a secure data-driven ASP.NET Core app in Azure App Service.

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.

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.

Step 1. In the App Service page:

1. From the left menu, select **App Service logs**.
2. Under **Application logging**, select **File System**.

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.

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.

Step 2. In the resource group page, select **Delete resource group**.

Step 3.

1. Enter the resource group name to confirm your deletion.
2. Select **Delete**.

:

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 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 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.

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:

Tutorial: Connect to SQL Database from App Service without secrets using a managed identity

Configure ASP.NET Core app

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.

> **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.

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 `Todoes` table and select **View Data**.

Update app with Code First Migrations

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

For more information about using Entity Framework Code First Migrations, see Getting Started with Entity Framework 6 Code First using MVC 5.

Update your data model

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

```
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 **MyDatabaseContext** dropdown, select the database connection for your Azure SQL Database.

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

Publish your changes

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

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

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

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

Stream application logs

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

Open *Controllers\TodosController.cs*.

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

Enable log streaming

1. 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.

Console

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

Stop log streaming

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

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.

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**.

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.

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.

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:Dsz40HWwbCqnq0oxMsjq7fItmKIeBfCBGORfspaI1Kw=* as the value. It's a Laravel encryption variable.

1. In the menu bar at the top, select **Save**.
2. When prompted, select **Continue**.

> **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**.

Step 2. In the GitHub page, open Visual Studio Code in the browser by pressing the `.` key.

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.

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.

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**.

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.

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**.

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.

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**.

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.

```
A P P   S E R V I C E   O N   L I N U X

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.

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.

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`.

Step 2. Add a few tasks to the list. Congratulations, you're running a secure data-driven PHP app in Azure App Service.

> **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**.

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.

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.

Step 2. In the resource group page, select **Delete resource group**.

Step 3.

1. Enter the resource group name to confirm your deletion.
2. Select **Delete**.

:

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

Deploy a Node.js + MongoDB web app to Azure

Article • 02/01/2023

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.

This article assumes you're already familiar with Node.js development and have Node and MongoDB installed locally. You'll also need an Azure account with an active subscription. If you don't have an Azure account, you can create one for free.

Sample application

To follow along with this tutorial, clone or download the sample application from the repository https://github.com/Azure-Samples/msdocs-nodejs-mongodb-azure-sample-app.

Bash

```
git clone https://github.com/Azure-Samples/msdocs-nodejs-mongodb-azure-sample-app.git
```

If you want to run the application locally, do the following:

- Install the package dependencies by running `npm install`.
- Copy the `.env.sample` file to `.env` and populate the DATABASE_URL value with your MongoDB URL (for example *mongodb://localhost:27017/*).
- Start the application using `npm start`.
- To view the app, browse to `http://localhost:3000`.

1. 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 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 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. This name must be unique across Azure.
4. *Runtime stack* → **Node 16 LTS**.
5. *Hosting plan* → **Basic**. When you're ready, you can scale up to a production pricing tier later.
6. **Azure Cosmos DB for MongoDB** is selected by default as the database engine. 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.
7. Select **Review + create**.
8. After validation completes, select **Create**.

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.

2. Set up database connectivity

The creation wizard generated the MongoDB URI for you already, but your app needs a `DATABASE_URL` variable and a `DATABASE_NAME` variable. In this step, you create app settings with the format that your app needs.

Step 1. In the App Service page, in the left menu, select Configuration.

Step 2. In the **Application settings** tab of the **Configuration** page, create a `DATABASE_NAME` setting:

1. Select **New application setting**.
2. In the **Name** field, enter *DATABASE_NAME*.
3. In the **Value** field, enter the automatically generated database name from the creation wizard, which looks like *msdocs-expressjs-mongodb-XYZ-database*.
4. Select **OK**.

Step 3.

1. Scroll to the bottom of the page and select the connection string **MONGODB_URI**. It was generated by the creation wizard.
2. In the **Value** field, select the **Copy** button and paste the value in a text file for the next step. It's in the MongoDB connection string URI format.
3. Select **Cancel**.

Step 4.

1. Using the same steps in **Step 2**, create an app setting named *DATABASE_URL* and set the value to the one you copied from the `MONGODB_URI` connection string (i.e. `mongodb://...`).
2. In the menu bar at the top, select **Save**.
3. When prompted, select **Continue**.

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 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-nodejs-mongodb-azure-sample-app.
3. Select **Fork**.
4. Select **Create fork**.

Step 2. In the GitHub page, open Visual Studio Code in the browser by pressing the `.` key.

Step 3. In Visual Studio Code in the browser, open *config/connection.js* in the explorer. In the `getConnectionInfo` function, see that the app settings you created earlier for the MongoDB connection are used (`DATABASE_URL` and `DATABASE_NAME`).

Step 4. Back in the App Service page, in the left menu, 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-nodejs-mongodb-azure-sample-app**.
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.

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**.

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 15 minutes.

4. 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`.

Step 2. Add a few tasks to the list. Congratulations, you're running a secure data-driven Node.js app in Azure App Service.

5. 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: 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**.

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.

6. 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`.

Step 2. In the Kudu page, select **Deployments**.

If you have deployed code to App Service using Git or zip deploy, you'll see a history of deployments of your web app.

[{"id":"c6bf4a4f-15ce-41fc-a05b-81bcfcdc86c9","status":4,"status_text":"","author_email":"N/A","author":"N/A","deployer":"GITHUB_ZIP_DEPLOY","message":"{\"type\":\"deployment\",\"sha\":\"8164afd05dc4ab3ccc8412d80a21f5cea63b00e7\",\"repoName\":\"lcephas/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-81bcfcdc86c9","log_url":"https://msdocs-expressjs-mongodb-234.scm.azurewebsites.net/api/deployments/c6bf4a4f-15ce-41fc-a05b-81bcfcdc86c9/log","site_name":"msdocs-expressjs-mongodb-234","build_summary":{"errors":[],"warnings":[]}}]

Step 3. Go back to the Kudu homepage and select **Site wwwroot**.

You can see the deployed folder structure and click 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

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.

Step 2. In the resource group page, select **Delete resource group**.

Step 3.

1. Enter the resource group name to confirm your deletion.
2. Select **Delete**.

:

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?

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 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.

Next steps

JavaScript on Azure developer center

Configure Node.js app in App Service

Secure with custom domain and certificate

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

Article • 02/01/2023

Azure App Service provides a highly scalable, self-patching web hosting service. This tutorial shows how to create a Ruby app and connect it to a PostgreSQL database. When you're finished, you'll have a Ruby on Rails app running on App Service on Linux.

In this tutorial, you learn how to:

- ✓ Create a PostgreSQL database in Azure
- ✓ Connect a Ruby on Rails app to PostgreSQL
- ✓ Deploy the app to Azure
- ✓ Update the data model and redeploy the app
- ✓ Stream diagnostic logs from Azure
- ✓ Manage the app in the Azure portal

If you don't have an Azure subscription, create an Azure free account before you begin.

Prerequisites

To complete this tutorial:

- Install Git
- Install Ruby 2.6
- Install Ruby on Rails 5.1

- Install and run PostgreSQL
- 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.

Prepare local Postgres

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

Connect to local Postgres server

1. Open the terminal window and run `psql` to connect to your local Postgres server.

   Bash

   ```
   sudo -u postgres psql
   ```

   If your connection is successful, your Postgres database is running. If not, make sure that your local Postgres database is started by following the steps at Downloads - PostgreSQL Core Distribution.

2. Type `\q` to exit the Postgres client.

3. Create a Postgres user that can create databases by running the following command, using your signed-in Linux username.

```bash
sudo -u postgres createuser -d <signed-in-user>
```

Create a Ruby on Rails app locally

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

Clone the sample

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

2. Clone the sample repository and change to the repository root.

```bash
git clone https://github.com/Azure-Samples/rubyrails-tasks.git
cd rubyrails-tasks
```

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`, this tutorial also shows you how to deploy a repository from `main`. For more information, see **Change deployment branch**.

4. Install the required packages.

```bash
bundle install --path vendor/bundle
```

Run the sample locally

1. Run the Rails migrations to create the tables the application needs. To see which tables are created in the migrations, look in the *db/migrate* directory in the Git repository.

```bash
rake db:create
rake db:migrate
```

2. Run the application.

```bash
rails server
```

3. Navigate to `http://localhost:3000` in a browser. Add a few tasks in the page.

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

Create Postgres in Azure

In this step, you create a Postgres database in Azure Database for PostgreSQL. Later, you configure the Ruby on Rails application to connect to this database.

Create a resource group

A resource group is a logical container into which Azure resources, 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 on Linux in **Basic** tier, run the az appservice list-locations --sku B1 --linux-workers-enabled 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 Postgres database in Azure

1. Install the `db-up` extension with the following command:

   Azure CLI

   ```
   az extension add --name db-up
   ```

2. Create the Postgres database in Azure with the az postgres up command, as shown in the following example. Replace *<postgresql-name>* with a *unique* name (the server endpoint is *https://<postgresql-name>.postgres.database.azure.com*). For *<admin-username>* and *<admin-password>*, specify credentials to create an administrator user for this Postgres server.

   Azure CLI

   ```
   az postgres up --resource-group myResourceGroup --location westeurope -
   -server-name <postgresql-name> --database-name sampledb --admin-user
   <admin-username> --admin-password <admin-password> --ssl-enforcement
   Enabled
   ```

   This command may take a while because it's doing the following:

   - Creates a resource group called `myResourceGroup`, if it doesn't exist. Every Azure resource needs to be in one of these. `--resource-group` is optional.
   - Creates a Postgres server with the administrative user.
   - Creates a `sampledb` database.
   - Allows access from your local IP address.
   - Allows access from Azure services.

- Create a database user with access to the `sampledb` database.

You can do all the steps separately with other `az postgres` commands and `psql`, but `az postgres up` does all of them in one step for you.

When the command finishes, find the output lines that being with `Ran Database Query:`. They show the database user that's created for you, with the username `root` and password `Sampledb1`. You'll use them later to connect your app to the database.

> **Tip**
>
> `--location <location-name>`, can be set to any one of the **Azure regions**. You can get the regions available to your subscription with the **az account list-locations** command. For production apps, put your database and your app in the same location.

Connect app to Azure Postgres

In this step, you connect the Ruby on Rails application to the Postgres database you created in Azure Database for PostgreSQL.

Configure the database connection

In the repository, open *config/database.yml*. At the bottom of the file, replace the production variables with the following code.

```txt
production:
  <<: *default
  host: <%= ENV['DB_HOST'] %>
  database: <%= ENV['DB_DATABASE'] %>
  username: <%= ENV['DB_USERNAME'] %>
  password: <%= ENV['DB_PASSWORD'] %>
```

Save the changes.

Test the application locally

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

```bash
export DB_HOST=<postgres-server-name>.postgres.database.azure.com
export DB_DATABASE=sampledb
export DB_USERNAME=root@<postgres-server-name>
export DB_PASSWORD=Sampledb1
```

2. Run Rails database migrations with the production values you just configured to create the tables in your Postgres database in Azure Database for PostgreSQL.

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

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

```bash
rake assets:precompile
```

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

```bash
rails secret
```

5. Save the secret key to the respective variables used by the Rails production environment. For convenience, you use the same key for both variables.

```bash
export RAILS_MASTER_KEY=<output-of-rails-secret>
export SECRET_KEY_BASE=<output-of-rails-secret>
```

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

```bash
export RAILS_SERVE_STATIC_FILES=true
```

7. Run the sample application in the production environment.

```bash
rails server -e production
```

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

9. Add a few tasks in the page.

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

Commit your changes

1. Run the following Git commands to commit your changes:

```bash
git add .
git commit -m "database.yml updates"
```

Your app is ready to be deployed.

Deploy to Azure

In this step, you deploy the Postgres-connected Rails application to Azure App Service.

Configure a deployment user

FTP and local Git can deploy to an Azure web app by using a *deployment user*. Once you configure your deployment user, you can use it for all your Azure deployments. Your

account-level deployment username and password are different from your Azure subscription credentials.

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

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

Azure CLI

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

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

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

Create an App Service plan

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

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

Azure CLI

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

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

```
{
  "freeOfferExpirationTime": null,
  "geoRegion": "West Europe",
  "hostingEnvironmentProfile": null,
  "id": "/subscriptions/0000-
0000/resourceGroups/myResourceGroup/providers/Microsoft.Web/serverfarms/myAp
```

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

Create a web app

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

In the Cloud Shell, you can use the az webapp create command. In the following example, replace `<app-name>` with a globally unique app name (valid characters are `a-z`, `0-9`, and `-`). The runtime is set to `RUBY|2.6.2`. To see all supported runtimes, run az webapp list-runtimes --os linux.

Azure CLI

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

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

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

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

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

Configure database settings

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

The following Cloud Shell command configures the app settings `DB_HOST`, `DB_DATABASE`, `DB_USERNAME`, and `DB_PASSWORD`. Replace the placeholders <*appname*> and <*postgres-server-name*>.

Azure CLI

```
az webapp config appsettings set --name <app-name> --resource-group
myResourceGroup --settings DB_HOST="<postgres-server-
name>.postgres.database.azure.com" DB_DATABASE="sampledb"
DB_USERNAME="root@<postgres-server-name>" DB_PASSWORD="Sampledb1"
```

Configure Rails environment variables

1. In the local terminal, generate a new secret for the Rails production environment in Azure.

   Bash

   ```
   rails secret
   ```

2. In the following Cloud Shell command, replace the two <*output-of-rails-secret*> placeholders with the new secret key you generated in the local terminal.

   Azure CLI

   ```
   az webapp config appsettings set --name <app-name> --resource-group
   myResourceGroup --settings RAILS_MASTER_KEY="<output-of-rails-secret>"
   SECRET_KEY_BASE="<output-of-rails-secret>"
   RAILS_SERVE_STATIC_FILES="true" ASSETS_PRECOMPILE="true"
   ```

`ASSETS_PRECOMPILE="true"` tells the default Ruby container to precompile assets at each Git deployment. For more information, see Precompile assets and Serve static assets.

Push to Azure from Git

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

   Bash

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

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

   Bash

   ```
   git push azure main
   ```

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

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

Browse to the Azure app

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

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

Update model locally and redeploy

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

For the tasks scenario, you modify the application so that you can mark a task as complete.

Add a column

1. In the terminal, navigate to the root of the Git repository.
2. Generate a new migration that adds a boolean column called `Done` to the `Tasks` table:

   ```bash
   rails generate migration AddDoneToTasks Done:boolean
   ```

   This command generates a new migration file in the *db/migrate* directory.

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

```bash
rake db:migrate
```

Update application logic

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

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

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

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

Update the views

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

2. Find the line `<%=f.error_span(:Description) %>` and insert the following code directly below it:

```erb
<%= f.label :Done, :class => 'control-label col-lg-2' %>
<div class="col-lg-10">
  <%= f.check_box :Done, :class => 'form-control' %>
</div>
```

3. Open the *app/views/tasks/show.html.erb* file, which is the single-record View page.

   Find the line `<dd><%= @task.Description %></dd>` and insert the following code directly below it:

```erb
```

```
  <dt><strong><%= model_class.human_attribute_name(:Done) %>:</strong>
</dt>
  <dd><%= check_box "task", "Done", {:checked => @task.Done, :disabled
=> true}%></dd>
```

Open the *app/views/tasks/index.html.erb* file, which is the Index page for all records.

Find the line `<th><%= model_class.human_attribute_name(:Description) %></th>` and insert the following code directly below it:

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

4. In the same file, find the line `<td><%= task.Description %></td>` and insert the following code directly below it:

```erb
<td><%= check_box "task", "Done", {:checked => task.Done, :disabled =>
true} %></td>
```

Test the changes locally

1. In the local terminal, run the Rails server.

```Bash
rails server
```

2. To see the task status change, navigate to `http://localhost:3000` and add or edit items.

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

Publish changes to Azure

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

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

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

```bash
git add .
git commit -m "added complete checkbox"
git push azure main
```

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

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

Stream diagnostic logs

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`.

Manage the Azure app

1. Go to the Azure portal to manage the app you created.

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

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

The left menu provides pages for configuring your app.

Clean up resources

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

Azure CLI

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

This command may take a minute to run.

Next steps

In this tutorial, you learned how to:

- ✓ Create a Postgres database in Azure
- ✓ Connect a Ruby on Rails app to Postgres
- ✓ Deploy the app to Azure
- ✓ Update the data model and redeploy the app
- ✓ Stream diagnostic logs from Azure
- ✓ Manage the app in the Azure portal

Advance to the next tutorial to learn how to secure your app with a custom domain and certificate.

Secure with custom domain and certificate

Or, check out other resources:

Configure Ruby app

Deploy a Python (Django or Flask) web app with PostgreSQL in Azure

Article • 08/21/2023

In this tutorial, you'll deploy a data-driven Python web app (**Django** or **Flask**) to **Azure App Service** with the **Azure Database for PostgreSQL** relational database service. Azure App Service supports Python in a Linux server environment.

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 Flask development or Python with Django development

Sample application

Sample Python applications using the Flask and Django framework are provided to help you follow along with this tutorial. To deploy them without running them locally, skip this part.

To run the application locally, make sure you have Python 3.7 or higher and PostgreSQL installed locally. Then, download or clone the app:

Flask

Bash

```
git clone https://github.com/Azure-Samples/msdocs-flask-postgresql-sample-app.git
```

Go to the application folder:

Flask

Bash

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

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 SECRET_KEY value for your app by running the following command at a terminal prompt: `python -c 'import secrets; print(secrets.token_hex())'`.

Set the returned value as the value of `SECRET_KEY` in the .env file.

```
SECRET_KEY=<secret-key>
```

Create a virtual environment for the app:

Windows

Cmd

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

Install the dependencies:

Bash

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

Run the sample application with the following commands:

Flask

Bash

```bash
# Run database migration
flask db upgrade
# Run the app at http://127.0.0.1:5000
flask run
```

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'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 **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 3.10**.
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**.

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.

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 Configuration.

Step 2: In the **Application settings** tab of the **Configuration** page, verify that `AZURE_POSTGRESQL_CONNECTIONSTRING` is present. That will be injected into the runtime environment as an environment variable.

Step 3: In a terminal or command prompt, run the following Python script to generate a unique secret: `python -c 'import secrets; print(secrets.token_hex())'`. Copy the

output value to use in the next step.

Step 4: In the **Application settings** tab of the **Configuration** page, select **New application setting**. Name the setting `SECRET_KEY`. Paste the value from the previous value. Select **OK**.

Step 5: Select **Save**.

Having issues? Check the Troubleshooting guide.

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 will kick off the build and deploy action.

Flask

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.
3. Select **Fork**.
4. Select **Create fork**.

Step 2: In the GitHub page, open Visual Studio Code in the browser by pressing the `.` key.

Step 3: In Visual Studio Code in the browser, open *azureproject/production.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.

Step 4: Back in the App Service page, in the left menu, 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-flask-postgresql-sample-app**.
5. In **Branch**, select **main**.
6. Keep the default option selected to **Add a workflow**.
7. In the top menu, select **Save**. App Service commits a workflow file into the chosen GitHub repository, in the `.github/workflows` directory.

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**.

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.

Having issues? Check the Troubleshooting guide.

4. Generate database schema

Flask

With the PostgreSQL database protected by the virtual network, the easiest way to run Flask database migrations is in an SSH session with the App Service container.

Step 1: Back in the App Service page, in the left menu, select **SSH**.

1. Select **Go**.

Step 2: In the SSH terminal, run `flask db upgrade`. 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.

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`.

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.

6. Stream diagnostic logs

Azure App Service captures all messages output to the console to help you diagnose issues with your application. The sample app includes `print()` statements to demonstrate this capability as shown below.

Flask

Python

```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 **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.

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.

Step 2: In the resource group page, select **Delete resource group**.

Step 3:

1. Enter the resource group name to confirm your deletion.
2. Select **Delete**.

:

Troubleshooting

Listed below are issues you may 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 may mean that the environment variable is missing (you may 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 is the Django sample configured to run on Azure App Service?

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
```

How is the Django sample configured to run on Azure App Service?

> **Note**
>
> If you are following along with this tutorial with your own app, look at the *requirements.txt* file description in each project's *README.md* file (**Flask**, **Django**) to see what packages you'll need.

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

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

For more information, see Production settings for Django apps.

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

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

Article • 02/01/2023

This tutorial walks you through the process of building, configuring, deploying, and scaling Java web apps on Azure. When you are finished, you will have a Spring Boot application storing data in Azure Cosmos DB running on Azure App Service on Linux.

In this tutorial, you learn how to:

- ✓ Create an Azure Cosmos DB database.
- ✓ Connect a sample app to the database and test it locally
- ✓ Deploy the sample app to Azure
- ✓ Stream diagnostic logs from App Service
- ✓ Add additional instances to scale out the sample app

If you don't have an Azure subscription, create an Azure free account before you begin.

Prerequisites

- Azure CLI, installed on your own computer.
- Git
- Java JDK
- Maven

Clone the sample TODO app and prepare the repo

This tutorial uses a sample TODO list app with a web UI that calls a Spring REST API backed by Spring Data for Azure Cosmos DB. The code for the app is available on GitHub. To learn more about writing Java apps using Spring and Azure Cosmos DB, see the Spring Boot Starter with the Azure Cosmos DB for NoSQL tutorial and the Spring Data for Azure Cosmos DB quick start.

Run the following commands in your terminal to clone the sample repo and set up the sample app environment.

Bash

```
git clone --recurse-submodules https://github.com/Azure-Samples/e2e-java-experience-in-app-service-linux-part-2.git
cd e2e-java-experience-in-app-service-linux-part-2
yes | cp -rf .prep/* .
```

Create an Azure Cosmos DB

Follow these steps to create an Azure Cosmos DB database in your subscription. The TODO list app will connect to this database and store its data when running, persisting

the application state no matter where you run the application.

1. Login to your Azure CLI, and optionally set your subscription if you have more than one connected to your login credentials.

   Azure CLI

   ```
   az login
   az account set -s <your-subscription-id>
   ```

2. Create an Azure Resource Group, noting the resource group name.

   Azure CLI

   ```
   az group create -n <your-azure-group-name> \
       -l <your-resource-group-region>
   ```

3. Create Azure Cosmos DB with the `GlobalDocumentDB` kind. The name of the Azure Cosmos DB instance must use only lower case letters. Note down the `documentEndpoint` field in the response from the command.

   Azure CLI

   ```
   az cosmosdb create --kind GlobalDocumentDB \
       -g <your-azure-group-name> \
       -n <your-azure-COSMOS-DB-name-in-lower-case-letters>
   ```

4. Get your Azure Cosmos DB key to connect to the app. Keep the `primaryMasterKey`, `documentEndpoint` nearby as you'll need them in the next step.

   Azure CLI

   ```
   az cosmosdb keys list -g <your-azure-group-name> -n <your-azure-
   COSMOSDB-name>
   ```

Configure the TODO app properties

Open a terminal on your computer. Copy the sample script file in the cloned repo so you can customize it for the Azure Cosmos DB database you just created.

Bash

```
cd initial/spring-todo-app
cp set-env-variables-template.sh .scripts/set-env-variables.sh
```

Edit `.scripts/set-env-variables.sh` in your favorite editor and supply Azure Azure Cosmos DB connection info. For the App Service Linux configuration, use the same region as before (`your-resource-group-region`) and resource group (`your-azure-group-name`) used when creating the Azure Cosmos DB database. Choose a WEBAPP_NAME that is unique since it cannot duplicate any web app name in any Azure deployment.

Bash

```bash
export COSMOSDB_URI=<put-your-COSMOS-DB-documentEndpoint-URI-here>
export COSMOSDB_KEY=<put-your-COSMOS-DB-primaryMasterKey-here>
export COSMOSDB_DBNAME=<put-your-COSMOS-DB-name-here>

# App Service Linux Configuration
export RESOURCEGROUP_NAME=<put-your-resource-group-name-here>
export WEBAPP_NAME=<put-your-Webapp-name-here>
export REGION=<put-your-REGION-here>
```

Then run the script:

Bash

```bash
source .scripts/set-env-variables.sh
```

These environment variables are used in `application.properties` in the TODO list app. The fields in the properties file set up a default repository configuration for Spring Data:

properties

```properties
azure.cosmosdb.uri=${COSMOSDB_URI}
azure.cosmosdb.key=${COSMOSDB_KEY}
azure.cosmosdb.database=${COSMOSDB_DBNAME}
```

Java

```java
@Repository
public interface TodoItemRepository extends DocumentDbRepository<TodoItem,
String> {
}
```

Then the sample app uses the `@Document` annotation imported from `com.microsoft.azure.spring.data.cosmosdb.core.mapping.Document` to set up an entity type to be stored and managed by Azure Cosmos DB:

Java

```java
@Document
public class TodoItem {
    private String id;
    private String description;
    private String owner;
    private boolean finished;
```

Run the sample app

Use Maven to run the sample.

Bash

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

The output should look like the following.

Output

```output
bash-3.2$ mvn package spring-boot:run
[INFO] Scanning for projects...
[INFO]
[INFO] ------------------------------------------------------------------------
---
[INFO] Building spring-todo-app 2.0-SNAPSHOT
[INFO] ------------------------------------------------------------------------
---
[INFO]


[INFO] SimpleUrlHandlerMapping - Mapped URL path [/webjars/**] onto handler
of type [class
org.springframework.web.servlet.resource.ResourceHttpRequestHandler]
[INFO] SimpleUrlHandlerMapping - Mapped URL path [/**] onto handler of type
[class org.springframework.web.servlet.resource.ResourceHttpRequestHandler]
[INFO] WelcomePageHandlerMapping - Adding welcome page: class path resource
[static/index.html]
2018-10-28 15:04:32.101  INFO 7673 --- [           main]
c.m.azure.documentdb.DocumentClient      : Initializing DocumentClient with
serviceEndpoint [https://sample-cosmos-db-westus.documents.azure.com:443/],
ConnectionPolicy [ConnectionPolicy [requestTimeout=60,
mediaRequestTimeout=300, connectionMode=Gateway, mediaReadMode=Buffered,
maxPoolSize=800, idleConnectionTimeout=60, userAgentSuffix=;spring-
data/2.0.6;098063be661ab767976bd5a2ec350e978faba99348207e8627375e8033277cb2,
retryOptions=com.microsoft.azure.documentdb.RetryOptions@6b9fb84d,
enableEndpointDiscovery=true, preferredLocations=null]], ConsistencyLevel
[null]
[INFO] AnnotationMBeanExporter - Registering beans for JMX exposure on
```

```
startup
[INFO] TomcatWebServer - Tomcat started on port(s): 8080 (http) with context
path ''
[INFO] TodoApplication - Started TodoApplication in 45.573 seconds (JVM
running for 76.534)
```

You can access Spring TODO App locally using this link once the app is started: `http://localhost:8080/`.

If you see exceptions instead of the "Started TodoApplication" message, check that the `bash` script in the previous step exported the environment variables properly and that the values are correct for the Azure Cosmos DB database you created.

Configure Azure deployment

Open the `pom.xml` file in the `initial/spring-boot-todo` directory and add the following Azure Web App Plugin for Maven configuration.

XML

```xml
<plugins>

    <!--**************************************************-->
    <!-- Deploy to Java SE in App Service Linux           -->
    <!--**************************************************-->

    <plugin>
        <groupId>com.microsoft.azure</groupId>
        <artifactId>azure-webapp-maven-plugin</artifactId>
        <version>2.5.0</version>
        <configuration>
            <schemaVersion>v2</schemaVersion>

            <!-- Web App information -->
            <resourceGroup>${RESOURCEGROUP_NAME}</resourceGroup>
            <appName>${WEBAPP_NAME}</appName>
            <region>${REGION}</region>
            <pricingTier>P1v2</pricingTier>
            <!-- Java Runtime Stack for Web App on Linux-->
            <runtime>
                 <os>linux</os>
                 <javaVersion>Java 8</javaVersion>
                 <webContainer>Java SE</webContainer>
             </runtime>
             <deployment>
                 <resources>
                 <resource>
                     <directory>${project.basedir}/target</directory>
                     <includes>
                     <include>*.jar</include>
                     </includes>
                 </resource>
                 </resources>
             </deployment>

            <appSettings>
                <property>
                    <name>COSMOSDB_URI</name>
                    <value>${COSMOSDB_URI}</value>
                </property>
                <property>
                    <name>COSMOSDB_KEY</name>
                    <value>${COSMOSDB_KEY}</value>
                </property>
                <property>
                    <name>COSMOSDB_DBNAME</name>
```

```
                    <value>${COSMOSDB_DBNAME}</value>
                </property>
                <property>
                    <name>JAVA_OPTS</name>
                    <value>-Dserver.port=80</value>
                </property>
            </appSettings>

        </configuration>
    </plugin>
    ...
</plugins>
```

Deploy to App Service on Linux

Use the `mvn azure-webapp:deploy` Maven goal to deploy the TODO app to Azure App Service on Linux.

Bash

```
# Deploy
bash-3.2$ mvn azure-webapp:deploy
[INFO] Scanning for projects...
[INFO]
[INFO] ------------------------------------------------------------------------
---
[INFO] Building spring-todo-app 2.0-SNAPSHOT
[INFO] ------------------------------------------------------------------------
---
[INFO]
[INFO] --- azure-webapp-maven-plugin:2.5.0:deploy (default-cli) @ spring-
todo-app ---
Auth Type: AZURE_CLI
Default subscription: xxxxxxxxx
Username: xxxxxxxxx
[INFO] Subscription: xxxxxxxxx
[INFO] Creating App Service Plan 'ServicePlanb6ba8178-5bbb-49e7'...
[INFO] Successfully created App Service Plan.
[INFO] Creating web App spring-todo-app...
[INFO] Successfully created Web App spring-todo-app.
[INFO] Trying to deploy artifact to spring-todo-app...
[INFO] Successfully deployed the artifact to https://spring-todo-
app.azurewebsites.net
[INFO] ------------------------------------------------------------------------
---
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
---
[INFO] Total time: 02:19 min
[INFO] Finished at: 2019-11-06T15:32:03-07:00
```

```
[INFO] Final Memory: 50M/574M
[INFO] ------------------------------------------------------------------------
---
```

The output contains the URL to your deployed application (in this example, `https://spring-todo-app.azurewebsites.net`). You can copy this URL into your web browser or run the following command in your Terminal window to load your app.

Bash

```
explorer https://spring-todo-app.azurewebsites.net
```

You should see the app running with the remote URL in the address bar:

Stream diagnostic logs

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

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`.

Scale out the TODO App

Scale out the application by adding another worker:

Azure CLI

```
az appservice plan update --number-of-workers 2 \
   --name ${WEBAPP_PLAN_NAME} \
   --resource-group <your-azure-group-name>
```

Clean up resources

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

Azure CLI

```
az group delete --name <your-azure-group-name> --yes
```

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 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

Tutorial: Build a Quarkus web app with Azure App Service on Linux and PostgreSQL

Article • 03/09/2023

This tutorial walks you through the process of building, configuring, deploying, and scaling Java web apps on Azure. When you are finished, you will have a Quarkus application storing data in PostgreSQL database running on Azure App Service on Linux.

In this tutorial, you learn how to:

- ✔ Create a App Service on Azure
- ✔ Create a PostgreSQL database on Azure
- ✔ Deploy the sample app to Azure App Service
- ✔ Connect a sample app to the database
- ✔ Stream diagnostic logs from App Service
- ✔ Add additional instances to scale out the sample app

If you don't have an Azure subscription, create an Azure free account before you begin.

Prerequisites

- Azure CLI, installed on your own computer.
- Git
- Java JDK
- Maven

Clone the sample app and prepare the repo

This tutorial uses a sample Fruits list app with a web UI that calls a Quarkus REST API backed by Azure Database for PostgreSQL. The code for the app is available on GitHub. To learn more about writing Java apps using Quarkus and PostgreSQL, see the Quarkus Hibernate ORM with Panache Guide and the Quarkus Datasource Guide.

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/quarkusio/quarkus-quickstarts
cd quarkus-quickstarts/hibernate-orm-panache-quickstart
```

Create an App Service on Azure

1. Sign in to your Azure CLI, and optionally set your subscription if you have more than one connected to your sign-in credentials.

   Azure CLI

   ```
   az login
   az account set -s <your-subscription-id>
   ```

2. Create an Azure Resource Group, noting the resource group name (referred to with `$RESOURCE_GROUP` later on)

Azure CLI

```
az group create \
    --name <a-resource-group-name> \
    --location <a-resource-group-region>
```

3. Create an App Service Plan. The App Service Plan is the compute container, it determines your cores, memory, price, and scale.

Azure CLI

```
az appservice plan create \
    --name "quarkus-tutorial-app-service-plan" \
    --resource-group $RESOURCE_GROUP \
    --sku B2 \
    --is-linux
```

4. Create an app service within the App Service Plan.

Azure CLI

```
    WEBAPP_NAME=<a unique name>
    az webapp create \
    --name $WEBAPP_NAME \
    --resource-group $RESOURCE_GROUP \
    --runtime "JAVA|11-java11" \
    --plan "quarkus-tutorial-app-service-plan"
```

> ⓘ **Important**
>
> The `WEBAPP_NAME` must be **unique across all Azure**. A good pattern is to use a combination of your company name or initials of your name along with a good webapp name, for example `johndoe-quarkus-app`.

Create an Azure PostgreSQL Database

Follow these steps to create an Azure PostgreSQL database in your subscription. The Quarkus Fruits app will connect to this database and store its data when running, persisting the application state no matter where you run the application.

1. Create the database service.

Azure CLI

```
DB_SERVER_NAME='msdocs-quarkus-postgres-webapp-db'
ADMIN_USERNAME='demoadmin'
ADMIN_PASSWORD='<admin-password>'

az postgres server create \
    --resource-group $RESOURCE_GROUP \
    --name $DB_SERVER_NAME \
    --location $LOCATION \
    --admin-user $ADMIN_USERNAME \
    --admin-password $ADMIN_PASSWORD \
    --sku-name GP_Gen5_2
```

The following parameters are used in the above Azure CLI command:

- *resource-group* → Use the same resource group name in which you created the web app, for example `msdocs-quarkus-postgres-webapp-rg`.
- *name* → The PostgreSQL database server name. This name must be **unique across all Azure** (the server endpoint becomes `https://<name>.postgres.database.azure.com`). Allowed characters are `A`-`Z`, `0`-`9`, and `-`. A good pattern is to use a combination of your company name and server identifier. (`msdocs-quarkus-postgres-webapp-db`)
- *location* → Use the same location used for the web app.
- *admin-user* → Username for the administrator account. It can't be `azure_superuser`, `admin`, `administrator`, `root`, `guest`, or `public`. For example, `demoadmin` is okay.
- *admin-password* Password of the administrator user. It must contain 8 to 128 characters from three of the following categories: English uppercase letters, English lowercase letters, numbers, and non-alphanumeric characters.

> **ⓘ Important**
>
> 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 Java apps.

- *public-access* → `None` which sets the server in public access mode with no firewall rules. Rules will be created in a later step.
- *sku-name* → The name of the pricing tier and compute configuration, for example `GP_Gen5_2`. For more information, see Azure Database for PostgreSQL pricing.

2. Configure the firewall rules on your server by using the az postgres server firewall-rule create command to give your local environment access to connect to the server.

Azure CLI

```
az postgres server firewall-rule create \
--resource-group $RESOURCE_GROUP_NAME \
--server-name $DB_SERVER_NAME \
--name AllowMyIP \
--start-ip-address <your IP> \
--end-ip-address <your IP>
```

Also, once your application runs on App Service, you'll need to give it access as well. run the following command to allow access to the database from services within Azure:

Azure CLI

```
az postgres server firewall-rule create \
  --resource-group $RESOURCE_GROUP_NAME \
  --server-name $DB_SERVER_NAME \
  --name AllowAllWindowsAzureIps \
  --start-ip-address 0.0.0.0 \
  --end-ip-address 0.0.0.0
```

3. Create a database named `fruits` within the Postgres service with this command:

Azure CLI

```
az postgres db create \
    --resource-group $RESOURCE_GROUP \
    --server-name $DB_SERVER_NAME \
    --name fruits
```

Configure the Quarkus app properties

Quarkus configuration is located in the `src/main/resources/application.properties` file. Open this file in your editor, and observe several default properties. The properties prefixed with `%prod` are only used when the application is built and deployed, for example when deployed to Azure App Service. When the application runs locally, `%prod` properties are ignored. Similarly, `%dev` properties are used in Quarkus' Live Coding / Dev mode, and `%test` properties are used during continuous testing.

Delete the existing content in `application.properties` and replace with the following to configure our database for dev, test, and production modes:

properties

```properties
quarkus.package.type=uber-jar
%dev.quarkus.datasource.db-kind=h2
%dev.quarkus.datasource.jdbc.url=jdbc:h2:mem:fruits

%test.quarkus.datasource.db-kind=h2
%test.quarkus.datasource.jdbc.url=jdbc:h2:mem:fruits

%prod.quarkus.datasource.db-kind=postgresql
%prod.quarkus.datasource.jdbc.url=jdbc:postgresql://${DBHOST}.postgres.database.azure.com:5432/${DBNAME}?user=${DBUSER}@${DBHOST}&password=${DBPASS}
%prod.quarkus.hibernate-orm.sql-load-script=import.sql

quarkus.hibernate-orm.database.generation=drop-and-create
```

> **ⓘ Important**
>
> Be sure to keep the dollar signs and braces intact when copying and pasting the above for the variables `${DBHOST}`, `${DBNAME}`, `${DBUSER}`, and `${DBPASS}`. We'll set the actual values later in our environment so that we don't expose them hard-coded in the properties file, and so that we can change them without having to re-deploy the app.

Run the sample app locally

Use Maven to run the sample.

Bash

```bash
mvn quarkus:dev
```

> **ⓘ Important**
>
> Be sure you have the H2 JDBC driver installed. You can add it using the following Maven command: `./mvnw quarkus:add-extension -Dextensions="jdbc-h2"`.

This will build the app, run its unit tests, and then start the application in developer live coding. You should see:

Output

```
__  ____  __  _____   ___  __ ____  ______
 --/ __ \/ / / / _ | / _ \/ //_/ / / / __/
 -/ /_/ / /_/ / __ |/ , _/ ,< / /_/ /\ \
--\___\_\____/_/ |_/_/|_/_/|_|\____/___/
INFO  [io.quarkus] (Quarkus Main Thread) hibernate-orm-panache-quickstart
1.0.0-SNAPSHOT on JVM (powered by Quarkus x.x.x.Final) started in x.xxxs.
Listening on: http://localhost:8080

INFO  [io.quarkus] (Quarkus Main Thread) Profile dev activated. Live Coding
activated.
INFO  [io.quarkus] (Quarkus Main Thread) Installed features: [agroal, cdi,
hibernate-orm, hibernate-orm-panache, jdbc-h2, jdbc-postgresql, narayana-
jta, resteasy-reactive, resteasy-reactive-jackson, smallrye-context-
propagation, vertx]
```

You can access Quarkus app locally by typing the `w` character into the console, or using this link once the app is started: `http://localhost:8080/`.

If you see exceptions in the output, double-check that the configuration values for `%dev` are correct.

> **Tip**
>
> You can enable continuous testing by typing `r` into the terminal. This will continuously run tests as you develop the application. You can also use Quarkus' *Live Coding* to see changes to your Java or `pom.xml` immediately. Simlply edit code and reload the browser.

When you're done testing locally, shut down the application with `CTRL-C` or type `q` in the terminal.

Configure App Service for Database

Our Quarkus app is expecting various environment variables to configure the database. Add these to the App Service environment with the following command:

Azure CLI

```
az webapp config appsettings set \
    -g $RESOURCE_GROUP \
    -n $WEBAPP_NAME \
    --settings \
        'DBHOST=$DB_SERVER_NAME' \
        'DBNAME=fruits' \
        'DBUSER=$ADMIN_USERNAME' \
        'DBPASS=$ADMIN_PASSWORD' \
        'PORT=8080' \
        'WEBSITES_PORT=8080'
```

> **Note**
>
> The use of single quotes (`'`) to surround the settings is required if your password has special characters.

Be sure to replace the values for `$RESOURCE_GROUP`, `$WEBAPP_NAME`, `$DB_SERVER_NAME`, `$ADMIN_USERNAME`, and `$ADMIN_PASSWORD` with the relevant values from previous steps.

Deploy to App Service on Linux

Build the production JAR file using the following command:

Azure CLI

```
mvn clean package
```

The final result will be a JAR file in the `target/` subfolder.

To deploy applications to Azure App Service, developers can use the Maven Plugin for App Service, VSCode Extension, or the Azure CLI to deploy apps. Use the following command to deploy our app to the App Service:

Azure CLI

```
az webapp deploy \
    --resource-group $RESOURCE_GROUP \
```

```
    --name $WEBAPP_NAME \
    --src-path target/*.jar --type jar
```

You can then access the application using the following command:

Azure CLI

```
az webapp browse \
    --resource-group $RESOURCE_GROUP \
    --name $WEBAPP_NAME
```

Tip

You can also manually open the location in your browser at `http://<webapp-name>.azurewebsites.net`. It may take a minute or so to upload the app and restart the App Service.

You should see the app running with the remote URL in the address bar:

If you see errors, use the following section to access the log file from the running app:

Stream diagnostic logs

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

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`.

Scale out the app

Scale out the application by adding another worker:

Azure CLI

```
az appservice plan update --number-of-workers 2 \
   --name quarkus-tutorial-app-service-plan \
   --resource-group $RESOURCE_GROUP
```

Clean up resources

If you don't need these resources for another tutorial (see Next steps), you can delete them by running the following command in the Cloud Shell or on your local terminal:

Azure CLI

```
az group delete --name $RESOURCE_GROUP --yes
```

Next steps

Azure for Java Developers Quarkus, Getting Started with Quarkus, and App Service Linux.

Learn more about running Java apps on App Service on Linux in the developer guide.

Java in App Service Linux dev guide

Learn how to secure your app with a custom domain and certificate.

Secure with custom domain and certificate

Configure an App Service app

Article • 04/21/2023

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

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
- Ruby
- Custom containers

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

> **Note**
>
> App settings can also be resolved from **Key Vault** using **Key Vault references**.

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** > **Application settings**.

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 **New application setting**. To edit a setting, select the **Edit** button on the right side.

4. In the dialog, you can stick the setting to the current slot.

   App setting names can't contain periods (`.`). If an app setting contains a period, the period is replaced with an underscore in the container.

> **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).

5. When finished, select **Update**. Don't forget to select **Save** back in the **Configuration** page.

Edit app settings in bulk

Azure portal

Select the **Advanced edit** button. Edit the settings in the text area. When finished, select **Update**. Don't forget to select **Save** back in the **Configuration** 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

In the Azure portal, search for and select **App Services**, and then select your app. In the app's left menu, select **Configuration** > **Application settings**.

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_`

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

- ASP.NET Core
- Node.js
- PHP
- Python
- Java
- Ruby
- Custom containers

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

> **Note**
>
> Connection strings can also be resolved from **Key Vault** using **Key Vault references**.

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** > **Application settings**.

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 **New connection string**. To edit a connection string, select the **Edit** button on the right side.

4. In the dialog, you can stick the connection string to the current slot.

5. When finished, select **Update**. Don't forget to select **Save** back in the **Configuration** page.

Edit connection strings in bulk

Azure portal

Select the **Advanced edit** button. Edit the connection strings in the text area. When finished, select **Update**. Don't forget to select **Save** back in the **Configuration** 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
- Ruby

Configure general settings

Azure portal

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**.

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.

- **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.
  - **ARR affinity**: In a multi-instance deployment, ensure that the client is routed to the same instance for the life of the session. You can set this option to **Off** for stateless applications.
  - **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.

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**.

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**.

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

   - **Extension**. The file extension you want to handle, such as **.php* or *handler.fcgi*.
   - **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**.

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

Use Key Vault references as app settings in Azure App Service and Azure Functions

Article • 07/31/2023

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 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:

   Azure CLI

   Azure CLI

   ```
   az webapp config set --subscription <sub> -g <group-name> -n <app-name> --generic-configurations '{"vnetRouteAllEnabled": true}'
   ```

2. Make sure that the vault's configuration allows the network or subnet that your app uses to access it.

> **Note**
>
> Windows container currently does not support key vault references over VNet Integration.

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

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:

Reference string	Description
SecretUri=*secretUri*	The **SecretUri** should be the full data-plane URI of a secret in the vault, optionally including a version, e.g., `https://myvault.vault.azure.net/secrets/mysecret/` or `https://myvault.vault.azure.net/secrets/mysecret/ec96f02080254f109c51a1f14cdb1931`

Reference string	Description
VaultName=*vaultName*;SecretName=*secretName*;SecretVersion=*secretVersion*	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**.

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.
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 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 $csResourceName --query id --output tsv)
csKeyKVUri=$(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 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 • 03/03/2022

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.
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 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 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 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|14-lts"
az webapp config appsettings set --resource-group $groupName --name $appName
```

```
--settings SCM_DO_BUILD_DURING_DEPLOYMENT=true
az webapp deployment source config-zip --resource-group $groupName --name
$appName --src ./default.zip
```

The preceding commands:

- Create a linux app service plan
- Create a web app for Node.js 14 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 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 $csResourceName --query id --output tsv)
csKeyKVUri=$(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 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 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.
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 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 $csResourceName --query id --output tsv)
   csKeyKVUri=$(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 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: 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** .

3. In the trigger information box, select **Use sample payload to generate schema**.

4. Copy the following sample JSON into the textbox and select **Done**.

JSON

```
{
    "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

How to use managed identities for App Service and Azure Functions

Article • 06/27/2023

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 Azure Active Directory (Azure AD) allows your app to easily access other Azure AD-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 Azure AD, see Managed identities for Azure resources.

Your application can be granted two types of identities:

- A **system-assigned identity** is tied to your application and is deleted if your app is deleted. An app can only have one system-assigned identity.
- A **user-assigned identity** is a standalone Azure resource that can be assigned to your app. An app can have multiple user-assigned identities.

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 Azure Active Directory 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>`.

Add a system-assigned identity

Azure portal

1. In the left navigation of your app's page, scroll down to the **Settings** group.
2. Select **Identity**.
3. Within the **System assigned** tab, switch **Status** to **On**. Click **Save**.

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.

Azure portal

First, you'll need to create a user-assigned identity resource.

1. Create a user-assigned managed identity resource according to these instructions.
2. 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 Azure Active Directory tokens, see Azure services that support Azure AD 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 Azure Active Directory, 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

```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
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": "5E29463D-71DA-4FE0-8E69-999B57DB23B0"
}
```

This response is the same as the response for the Azure AD 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 Azure Active Directory. System-assigned identities are also automatically removed from Azure Active Directory

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:

Parameter name	In	Description
resource	Query	The Azure AD resource URI of the resource for which a token should be obtained. This could be one of the Azure services that support Azure AD authentication or any other resource URI.
api-version	Query	The version of the token API to be used. Use `2019-08-01`.

Parameter name	In	Description
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 `principal_id`, `mi_res_id`, or `object_id`. If all ID parameters (`client_id`, `principal_id`, `object_id`, and `mi_res_id`) are omitted, the system-assigned identity is used.
principal_id	Query	(Optional) The principal ID of the user-assigned identity to be used. `object_id` is an alias that may be used instead. Cannot be used on a request that includes client_id, mi_res_id, or object_id. If all ID parameters (`client_id`, `principal_id`, `object_id`, and `mi_res_id`) 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 `principal_id`, `client_id`, or `object_id`. If all ID parameters (`client_id`, `principal_id`, `object_id`, and `mi_res_id`) 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

Tutorial: Connect to Azure databases from App Service without secrets using a managed identity

Article • 03/29/2023

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 Azure Active Directory 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 an Azure AD user as an administrator for your Azure database.
- ✓ Connect to your database as the Azure AD 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 Azure AD 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. Grant database access to Azure AD user

First, enable Azure Active Directory authentication to the Azure database by assigning an Azure AD user as the administrator of the server. For the scenario in the tutorial, you'll use this user to connect to your Azure database from the local development environment. Later, you set up the managed identity for your App Service app to connect from within Azure.

> **Note**
>
> This user is different from the Microsoft account you used to sign up for your Azure subscription. It must be a user that you created, imported, synced, or invited into

Azure AD. For more information on allowed Azure AD users, see **Azure AD features and limitations in SQL Database**.

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

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

Azure CLI

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

Azure SQL Database

3. Add this Azure AD user as an Active Directory administrator using az sql server ad-admin create command in the Cloud Shell. In the following command, replace *<group-name>* and *<server-name>* with your own parameters.

Azure CLI

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

For more information on adding an Active Directory administrator, see Provision an Azure Active Directory administrator for your server

2. Configure managed identity for app

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

1. Enable a managed identity for your App Service app with the az webapp identity assign command in the Cloud Shell. In the following command, replace *<app-name>*.

System-assigned identity

Azure CLI

```
az webapp identity assign --resource-group <group-name> --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>*.

2. The identity needs to be granted permissions to access the database. In the Cloud Shell, sign in to your database with the following command. Replace *<server-name>* with your server name, *<database-name>* with the database name your app uses, and *<aad-user-name>* and *<aad-password>* with your Azure AD user's credentials from 1. Grant database access to Azure AD user.

Azure SQL Database

Azure CLI

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

3. Run the following database commands to grant the permissions your app needs. For example,

System-assigned identity

SQL

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

For a deployment slot, use *<app-name>/slots/<slot-name>* instead of *<app-name>*.

3. Modify your code

In this section, connectivity to the Azure database in your code follows the `DefaultAzureCredential` pattern for all language stacks. `DefaultAzureCredential` is flexible enough to adapt to both the development environment and the Azure environment. When running locally, it can retrieve the logged-in Azure user from the environment of your choice (Visual Studio, Visual Studio Code, Azure CLI, or Azure PowerShell). When running in Azure, it retrieves the managed identity. So it's possible to have connectivity to database both at development time and in production. The pattern is as follows:

1. Instantiate a `DefaultAzureCredential` from the Azure Identity client library. If you're using a user-assigned identity, specify the client ID of the identity.
2. Get an access token for the resource URI respective to the database type.
   - For Azure SQL Database: `https://database.windows.net/.default`
   - For Azure Database for MySQL: `https://ossrdbms-aad.database.windows.net/.default`
   - For Azure Database for PostgreSQL: `https://ossrdbms-aad.database.windows.net/.default`
3. Add the token to your connection string.
4. Open the connection.

For Azure Database for MySQL and Azure Database for PostgreSQL, the database username that you created in 2. Configure managed identity for app is also required in the connection string.

.NET Framework

1. In Visual Studio, open the Package Manager Console and add the NuGet packages you need:

Azure SQL Database

PowerShell

```
Install-Package Azure.Identity
Install-Package System.Data.SqlClient
```

2. Connect to the Azure database by adding an access token. If you're using a user-assigned identity, make sure you uncomment the applicable lines.

Azure SQL Database

C#

```
// Uncomment one of the two lines depending on the identity
type
//var credential = new Azure.Identity.DefaultAzureCredential();
// system-assigned identity
//var credential = new
Azure.Identity.DefaultAzureCredential(new
DefaultAzureCredentialOptions { ManagedIdentityClientId =
'<client-id-of-user-assigned-identity>' }); // user-assigned
identity

// Get token for Azure SQL Database
var token = credential.GetToken(new
Azure.Core.TokenRequestContext(new[] {
"https://database.windows.net/.default" }));

// Add the token to the SQL connection
var connection = new
System.Data.SqlClient.SqlConnection("Server=tcp:<server-
name>.database.windows.net;Database=<database-
name>;TrustServerCertificate=True");
connection.AccessToken = token.Token;

// Open the SQL connection
connection.Open();
```

For a more detailed tutorial, see Tutorial: Connect to SQL Database from .NET App Service without secrets using a managed identity.

4. Set up your dev environment

This sample code uses `DefaultAzureCredential` to get a useable token for your Azure database from Azure Active Directory 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 Azure AD 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 Azure AD user. In this

step, you configure your environment of choice by signing in with your Azure AD user.

Visual Studio Windows

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

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

For more information about setting up your dev environment for Azure Active Directory 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 Azure AD authentication.

5. Test and publish

1. Run your code in your dev environment. Your code uses the signed-in Azure AD user) in your environment to connect to the back-end database. The user can access the database because it's configured as an Azure AD 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 an Azure AD group?
- I get the error mysql: unknown option '--enable-cleartext-plugin'.
- I get the error SSL connection is required. Please specify SSL options and retry.

Does managed identity support SQL Server?

Azure Active Directory and managed identities aren't supported for on-premises SQL Server.

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 an Azure AD group?

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

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 an Azure AD group, see documentation for the respective database type.

I get the error `mysql: unknown option '--enable-cleartext-plugin'`.

If you're using a MariaDB client, the `--enable-cleartext-plugin` option isn't required.

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

Next steps

What you learned:

- ✓ Configure an Azure AD user as an administrator for your Azure database.
- ✓ Connect to your database as the Azure AD 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 Azure AD 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

Tutorial: Connect to SQL Database from .NET App Service without secrets using a managed identity

Article • 04/01/2023

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 Azure AD authentication with SQL Database

✓ Connect to SQL Database from Visual Studio using Azure AD authentication

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

If you don't have an Azure subscription, create 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.

  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. Grant database access to Azure AD user

First, enable Azure Active Directory authentication to SQL Database by assigning an Azure AD 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 Azure AD. For more information on allowed Azure AD users, see Azure AD features and limitations in SQL Database.

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

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

   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 Azure AD, run `az ad user list --query '[].userPrincipalName'`.

3. Add this Azure AD user as an Active Directory admin using az sql server ad-admin create command in the Cloud Shell. In the following command, replace *<server-name>* with the 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 an Azure Active Directory administrator for your server

2. Set up your dev environment

Visual Studio Windows

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

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

For more information about setting up your dev environment for Azure Active Directory 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 Azure AD 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 (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 add the NuGet package Azure.Identity and update Entity Framework:

```
PowerShell

Install-Package Azure.Identity
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;
var credential = new Azure.Identity.DefaultAzureCredential();
var token = credential.GetToken(new
Azure.Core.TokenRequestContext(new[] {
"https://database.windows.net/.default" }));
conn.AccessToken = token.Token;
```

   This code uses Azure.Identity.DefaultAzureCredential to get a useable token for SQL Database from Azure Active Directory and then adds it to the database connection. While you can customize `DefaultAzureCredential`, by default it's already versatile. When it runs in App Service, it uses app's system-assigned managed identity. When it runs locally, it can get a token using the logged-in identity of Visual Studio, Visual Studio Code, Azure CLI, and Azure PowerShell.

3. In *Web.config*, find the connection string called `MyDbConnection` and replace its `connectionString` value with `"server=tcp:<server-name>.database.windows.net;database=<db-name>;"`. Replace <*server-name*> and <*db-name*> with your server name and database name. This connection string is used by the default constructor in *Models/MyDbContext.cs*.

   That's every thing you need to connect to SQL Database. When you debug in Visual Studio, your code uses the Azure AD 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.

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

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 the 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 **Azure AD group**, then grant SQL Database access to the Azure AD group instead of the identity. For example, the following commands add the managed identity from the previous step to a new group called *myAzureSQLDBAccessGroup*:

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 Azure AD 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 Azure AD. 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 an Azure AD 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**
>
> Azure Active Directory 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**.

> **ⓘ 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.

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 Azure AD authentication with SQL Database
- ✓ Connect to SQL Database from Visual Studio using Azure AD 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 • 08/15/2023

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 Spring Boot web application to use Azure AD 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 Spring Boot 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 Azure Active Directory (Azure AD) admin account to perform administrative tasks.

   Flexible Server

   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.

Flexible Server

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 $APPSERVICE_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.

Flexible Server

To make this connection, run the az webapp connection create command.

Azure CLI

```
az webapp connection create postgres-flexible \
    --resource-group $RESOURCE_GROUP \
    --name $APPSERVICE_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 $APPSERVICE_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}"'&authenticationPluginClassNa
me=com.azure.identity.extensions.jdbc.postgresql.AzurePostgresqlAuthenticati
onPlugin"'
```

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

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.

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.

5. In **Overview**, select **Permissions**, and you'll see the added permissions for Microsoft Graph.

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.

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.

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.

5. In **Overview**, select **Permissions**, and you'll see the added permissions for Microsoft Graph.

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

```bash
npm install @azure/identity @microsoft/microsoft-graph-client
```

Configure authentication information

Create an object to hold the authentication settings:

JavaScript

```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.

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.

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.

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.

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.

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 • 09/15/2023

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 Azure Active Directory (Azure AD).

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 an Azure AD application. In this step, you give the web app permissions to access Microsoft Graph for the user. (Technically, you give the web app's Azure AD application the permissions to access the Microsoft Graph Azure AD 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 Azure AD 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 Azure Active Directory. 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.

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.

In the app registration overview, 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 • 08/18/2023

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 Azure Active Directory (Azure AD).

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 an Azure AD application. In this step, you give the web app permissions to access Microsoft Graph for the user. (Technically, you give the web app's Azure AD application the permissions to access the Microsoft Graph Azure AD 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 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 Azure Active Directory. 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.

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.

In the app registration overview, 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 • 04/21/2023

This tutorial shows you how to enable built-in authentication in an App Service app using the Azure Active Directory 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 Azure Active Directory 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 Azure Active Directory as the identity provider
- ✓ Access Azure SQL Database on behalf of the signed-in Azure AD user

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

If you don't have an Azure subscription, create 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:

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.	
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 Azure AD authentication

First, enable Azure Active Directory authentication to SQL Database by assigning an Azure AD 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 Azure AD. For more information on allowed Azure AD users, see Azure AD features and limitations in SQL Database.

1. If your Azure AD tenant doesn't have a user yet, create one by following the steps at Add or delete users using Azure Active Directory.
2. Find the object ID of the Azure AD user using the az ad user list and replace <*user-principal-name*>. The result is saved to a variable.

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 Azure AD, run `az ad user list --query []`.`userPrincipalName`.

3. Add this Azure AD user as an Active Directory admin using az sql server ad-admin create command in the Cloud Shell. In the following command, replace *<server-name>* with the 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> --server-name <server-name>
```

For more information on adding an Active Directory admin, see Provision Azure AD admin (SQL Database).

2. Enable user authentication for your app

You enable authentication with Azure Active Directory as the identity provider. For more information, see Configure Azure Active Directory 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 Azure AD identities.

5. Accept the default settings and select **Add**.

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 Azure AD 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**.

4. Configure App Service to return a usable access token

The app registration in Azure Active Directory 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`.

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> --type SQLAzure --settings MyDbConnection="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 Azure AD user.

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 Azure AD 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?.
- Azure AD 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 Azure AD users or groups in Azure SQL Database?.

How do I add other Azure AD 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 Azure AD identities in SQL Database documentation.

   The following Transact-SQL example adds an Azure AD 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 Azure Active Directory as the identity provider
- ✓ Access Azure SQL Database on behalf of the signed-in Azure AD 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

Tutorial: Authenticate and authorize users end-to-end in Azure App Service

Article • 03/08/2023

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 Azure Active Directory as the identity provider
- ✓ Access a remote app on behalf of the signed-in user
- ✓ Secure service-to-service calls with token authentication
- ✓ Use access tokens from server code
- ✓ Use access tokens from client (browser) code

> **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 --runtime "NODE:16-lts"
```

3. Configure app setting

The frontend application needs to the 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.

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 Azure Active Directory 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 Azure Active Directory 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 Azure AD identities.

5. Accept the default settings and select **Add**.

Home > dotnet-core-back-end-auth >

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 Azure AD application to a notepad. You need this value later.

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 backend app's management page.

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 Azure AD identities.

5. Accept the default settings and select **Add**.

Home > dotnet-core-back-end-auth >

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 Azure AD 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**.

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

```
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

```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.

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`.

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 Azure Active Directory as the identity provider
- ✓ Access a remote app on behalf of the signed-in user
- ✓ Secure service-to-service calls with token authentication
- ✓ Use access tokens from server code
- ✓ Use access tokens from client (browser) code

Advance to the next tutorial to learn how to 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 Azure AD 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**.

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.

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 Azure Active Directory 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

```javascript
import { getGraphProfile } from './with-graph/graph';
```

3. In the same file, uncomment the `graphProfile` variable:

JavaScript

```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

```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 with 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-off 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 • 05/23/2023

This article shows you how to scale your app in Azure App Service. There are two workflows for scaling, scale up and scale out, and this article explains the scale up workflow.

- Scale up: Get more CPU, memory, disk space, and extra features like dedicated virtual machines (VMs), custom domains and certificates, staging slots, autoscaling, and more. You scale up by changing the pricing tier of the App Service plan that your app belongs to.
- Scale out: Increase the number of VM instances that run your app. You can scale out to as many as 30 instances, depending on your pricing tier. App Service Environments in **Isolated** tier further increases your scale-out count to 100 instances. For more information about scaling out, see Scale instance count manually or automatically. There, you find out how to use autoscaling, which is to scale instance count automatically based on predefined rules and schedules.

> **Important**
>
> **App Service now 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 Microsoft Azure App Service subscription, see **Microsoft Azure Subscriptions**.

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 (instan
Popular options					
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
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.

2. In the **Summary** part of the **Resource group** page, select a resource that you want to scale. The following screenshot shows a SQL Database resource.

To scale up the related resource, see the documentation for the specific resource type. For example, to scale up a single SQL Database, see Scale single database resources in Azure SQL Database. To scale up an Azure Database for MySQL resource, see Scale MySQL resources.

Compare pricing tiers

For detailed information, such as VM sizes for each pricing tier, see App Service Pricing Details.

For a table of service limits, quotas, and constraints, and supported features in each tier, see App Service limits.

More resources

- Scale instance count manually or automatically
- Configure Premium V3 tier for App Service
- Tutorial: Run a load test to identify performance bottlenecks in a web app

Automatic scaling in Azure App Service

Article • 08/02/2023

> **Note**
>
> Automatic scaling is in preview. It's available for Premium V2 (P1V2, P2V2, P3V2) and Premium V3 (P1V3, P2V3, P3V3) pricing tiers, and supported for all app types: Windows, Linux, and Windows container.

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 can use Application Insights Live Metrics to check your current instance count, and performanceCounters to see the instance count history. You're charged per second for every instance, including prewarmed instances.

A comparison of scale out and scale in options available on App Service:

	Manual	Autoscale	Automatic scaling
Available pricing tiers	Basic and Up	Standard and Up	Premium V2 (P1V2, P2V2, P3V2) and Premium V3 (P1V3, P2V3, P3V3)
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 instances	No	No	Yes (default 1)

	Manual	Autoscale	Automatic scaling
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.

> ⓘ **Important**
>
> **Always ON** needs to be disabled to use automatic scaling.

Azure portal

To enable automatic scaling, navigate to the web app's left menu and select **Scale out (App Service Plan)**. Select **Automatic (preview)**, update the **Maximum burst** value, and select the **Save** button.

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.

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.

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.

Does automatic scaling support Azure Function apps?

No, you can only have Azure App Service web apps in the App Service Plan where you wish to enable automatic scaling. If you have existing Azure Functions apps in the same App Service Plan, or if you create new Azure Functions apps, then automatic scaling is disabled. For Functions, it's recommended to use the Azure Functions Premium plan instead.

More resources

- Get started with autoscale in Azure
- Configure PremiumV3 tier for App Service
- Scale up server capacity
- High-density hosting

Get started with autoscale in Azure

Article • 04/10/2023

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?postJsllMsg=true

To discover the resources that you can autoscale, follow these steps.

1. Open the Azure portal.
2. Using the search bar at the top of the page, search for and select *Azure Monitor*
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 page shows the instance count and the autoscale status for each resource. Autoscale statuses are:

- **Not configured**: You haven't enabled autoscale yet for this resource.
- **Enabled**: You've enabled autoscale for this resource.
- **Disabled**: You've disabled autoscale for this resource.

You can also reach the scaling page by selecting **Scaling** from the **Settings** menu for each resource.

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.

6. The default rule scales your resource by one instance if the CPU percentage is greater than 70 percent. Keep the default values and select **Add**.

7. You've now 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**.

You now have a scale setting that scales out and scales in based on CPU usage, but you're still limited to a maximum of one instance.

12. Under **Instance limits** set **Maximum** to *3*

13. Select **Save**.

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 **Sunday**

6. 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.

7. Select **Save**

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. Enter the **Start date** and **End date** for when the scale condition should be applied.

7. Select **Save**

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 will scale to 3 instances when this scale condition becomes active.

Additional settings

View the history of your resource's scale events

Whenever your resource is scaled up or down, an event is logged in the activity log. You can view the history of the scale events in the **Run history** tab.

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.

You can make changes in JSON directly, if necessary. These changes will be reflected after you save them.

Cool-down period effects

Autoscale uses a cool-down period with is the amount of time to wait after a scale operation before scaling again. For example, if the cooldown is 10 minutes, Autoscale won't attempt to scale again until 10 minutes after the previous scale action. The cooldown period allows the metrics to stabilize and avoids scaling more than once for the same condition. 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 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 will be 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

Tutorial: Add Azure CDN to an Azure App Service web app

Article • 02/28/2023

This tutorial shows how to add Azure Content Delivery Network (CDN) 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:

What you learn:

- ✓ Create a CDN endpoint.
- ✓ Refresh cached assets.
- ✓ Use query strings to control cached versions.

Prerequisites

To complete this tutorial:

- Install Git
- Install the Azure CLI

If you don't have an Azure subscription, create 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 navigate to the Azure portal.

Dynamic site acceleration optimization

If you want to optimize your CDN endpoint for dynamic site acceleration (DSA), you should use the CDN portal to create your profile and endpoint. With DSA optimization, the performance of web pages with dynamic content is measurably improved. For instructions about how to optimize a CDN endpoint for DSA from the CDN portal, see CDN endpoint configuration to accelerate delivery of dynamic files. Otherwise, if you don't want to optimize your new endpoint, you can use the web app portal to create it by following the steps in the next section. For **Azure CDN from Verizon** profiles, you can't change the optimization of a CDN endpoint after it has been created.

Create a CDN profile and endpoint

In the left navigation, select **App Services**, and then select the app that you created in the static HTML quickstart.

In the **App Service** page, in the **Settings** section, select **Networking** > **Azure CDN**.

In the **Azure Content Delivery Network** page, provide the **New endpoint** settings as specified in the table.

Setting	Suggested value	Description
CDN profile	myCDNProfile	A CDN profile is a collection of CDN endpoints with the same pricing tier.
Pricing tier	Standard Akamai	The pricing tier specifies the provider and available features. This tutorial uses *Standard Akamai*.
CDN endpoint name	Any name that is unique in the azureedge.net domain	You access your cached resources at the domain *<endpointname>*.azureedge.net.

Select **Create** to create a CDN profile.

Azure creates the profile and endpoint. The new endpoint appears in the **Endpoints** list, and when it's provisioned, the status is **Running**.

Test the CDN endpoint

Because it takes time for the registration to propagate, the endpoint isn't immediately available for use:

- For **Azure CDN Standard from Microsoft** profiles, propagation usually completes in 10 minutes.
- For **Azure CDN Standard from Akamai** profiles, propagation usually completes within one minute.
- For **Azure CDN Standard from Verizon** and **Azure CDN Premium from Verizon** profiles, propagation usually completes within 90 minutes.

The sample app has an *index.html* file and *css*, *img*, and *js* folders that contain other static assets. The content paths for all of these files are the same at the CDN endpoint. For example, both of the following URLs access the *bootstrap.css* file in the *css* folder:

```
http://<appname>.azurewebsites.net/css/bootstrap.css
```

```
http://<endpointname>.azureedge.net/css/bootstrap.css
```

Navigate a browser to the following URL:

```
http://<endpointname>.azureedge.net/index.html
```

You see the same page that you ran earlier in an Azure web app. Azure CDN has retrieved the origin web app's assets and is serving them from the CDN endpoint

To ensure that this page is cached in the CDN, refresh the page. Two requests for the same asset are sometimes required for the CDN to cache the requested content.

For more information about creating Azure CDN profiles and endpoints, see Getting started with Azure CDN.

Purge the CDN

The CDN periodically refreshes its resources from the origin web app based on the time-to-live (TTL) configuration. The default TTL is seven days.

At times you might need to refresh the CDN before the TTL expiration; for example, when you deploy updated content to the web app. To trigger a refresh, manually purge the CDN resources.

In this section of the tutorial, you deploy a change to the web app and purge the CDN to trigger the CDN to refresh its cache.

Deploy a change to the web app

Open the *index.html* file and add - *V2* to the H1 heading, as shown in the following example:

```
<h1>Azure App Service - Sample Static HTML Site - V2</h1>
```

Commit your change and deploy it to the web app.

```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 CDN endpoint URL for the home page, you don't see the changes because the cached version in the CDN hasn't expired yet.

```
http://<endpointname>.azureedge.net/index.html
```

Purge the CDN in the portal

To trigger the CDN to update its cached version, purge the CDN.

In the portal left navigation, select **Resource groups**, and then select the resource group that you created for your web app (myResourceGroup).

In the list of resources, select your CDN endpoint.

At the top of the **Endpoint** page, select **Purge**.

Enter the content paths you want to purge. You can pass a complete file path to purge an individual file, or a path segment to purge and refresh all content in a folder. Because you changed *index.html*, ensure that is in one of the paths.

At the bottom of the page, select **Purge**.

Verify that the CDN is updated

Wait until the purge request finishes processing, which is typically a couple of minutes. To see the current status, select the bell icon at the top of the page.

When you browse to the CDN endpoint URL for *index.html*, you see the *V2* that you added to the title on the home page, which indicates that the CDN cache has been refreshed.

```
http://<endpointname>.azureedge.net/index.html
```

For more information, see Purge an Azure CDN endpoint.

Use query strings to version content

Azure CDN offers the following caching behavior options:

- Ignore query strings
- Bypass caching for query strings
- Cache every unique URL

The first option is the default, which means there'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** drop-down list.

Select **Save**.

Verify that unique URLs are cached separately

In a browser, navigate to the home page at the CDN endpoint, and include a query string:

```
http://<endpointname>.azureedge.net/index.html?q=1
```

Azure CDN returns the current web app content, which includes *V2* in the heading.

To ensure that this page is cached in the CDN, refresh the page.

Open *index.html*, change *V2* to *V3*, then deploy the change.

```bash
git commit -am "version 3"
git push azure main
```

In a browser, go to the CDN endpoint URL with a new query string, such as `q=2`. Azure CDN gets the current *index.html* file and displays *V3*. However, if you navigate to the CDN endpoint with the `q=1` query string, you see *V2*.

```
http://<endpointname>.azureedge.net/index.html?q=2
```

```
http://<endpointname>.azureedge.net/index.html?q=1
```

This output shows that each query string is treated differently:

- q=1 was used before, so cached contents are returned (V2).
- q=2 is new, so the latest web app contents are retrieved and returned (V3).

For more information, see Control Azure CDN caching behavior with query strings.

Clean up resources

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

Azure CLI

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

This command may take a minute to run.

Next steps

What you learned:

- ✓ Create a CDN endpoint.
- ✓ Refresh cached assets.
- ✓ Use query strings to control cached versions.

Learn how to optimize CDN performance in the following articles:

Tutorial: Add a custom domain to your Azure CDN endpoint

High-density hosting on Azure App Service using per-app scaling

Article • 06/29/2023

> **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**.

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",
```

```
                "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 precendnce 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`

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

Tutorial: Add app authentication to your web app running on Azure App Service

Article • 06/28/2023

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 Azure Active Directory (Azure AD) 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 are 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:
  - Azure AD
  - 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 Azure AD as the identity provider. For more information, see Configure Azure AD 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.

3. On your app's left menu, select **Authentication**, and then click **Add identity provider**.

4. In the **Add an identity provider** page, for example select **Microsoft** as the **Identity provider** to sign in Microsoft and Azure AD identities.

5. Select a **Tenant type**, for example **Workforce** for work and school accounts or Microsoft accounts.

6. For **App registration** > **App registration type**, select **Create new app registration** to create a new app registration in Azure AD.

7. Add a **Name** for the app registration, a public facing display name.

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 **App Service authentication settings** section, leave **Authentication** set to **Require authentication** and **Unauthenticated requests** set to **HTTP 302 Found redirect: recommended for websites**.

10. At the bottom of the **Add an identity provider** page, click **Add** to enable authentication for your web app.

Home > Resource groups > GetStartedWebApp > GetStartedWebApp | Authentication >

Add an identity provider ...

Basics Permissions

Identity provider * Microsoft

A tenant contains a directory for managing users and applications. Choose a tenant type based on the kind of users that will be accessing this application. Learn more

Tenant type *
- (●) **Workforce:** A tenant for managing using work and school (Azure AD) accounts or Microsoft accounts (such as Outlook.com and Live.com).
- ( ) **Customer (Preview):** A tenant for managing customer identities through social and local accounts.

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 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 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

Token store [✓]

Add < Previous Next: 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 Azure AD tenant. The app registration has the same display name as your web app.

1. To check the settings, select **Azure Active Directory** from the portal menu, and select **App registrations**.

2. Select the app registration that was created.

3. In the overview, verify that **Supported account types** is set to **My organization only**.

4. To verify that access to your app is limited to users in your organization, got 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`.

5. You should be directed to a secured sign-in page, verifying that unauthenticated users aren't allowed access to the site.

6. 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 Azure Active Directory. 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.

This command might take several minutes to run.

Delete the app registration

From the portal menu, select **Azure Active Directory** > **App registrations**. Then select the application you created.

In the app registration overview, 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 • 03/28/2023

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, Azure AD, Facebook, Google, Twitter.

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:

Provider	Sign-in endpoint	How-To guidance
Microsoft identity platform	`/.auth/login/aad`	App Service Microsoft Identity Platform login
Facebook	`/.auth/login/facebook`	App Service Facebook login
Google	`/.auth/login/google`	App Service Google login
Twitter	`/.auth/login/twitter`	App Service Twitter login
GitHub	`/.auth/login/github`	App Service GitHub login
Sign in with Apple	`/.auth/login/apple`	App Service Sign in With Apple login (Preview)
Any OpenID Connect provider	`/.auth/login/<providerName>`	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. 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.

Step	Without provider SDK	With provider SDK

Step	Without provider SDK	With provider SDK
1. Sign user in	Redirects client to `/.auth/login/<provider>`.	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 `/.auth/login/<provider>/callback`.	Client code posts token from provider to `/.auth/login/<provider>` 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 `X-ZUMO-AUTH` 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.

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. This rejection can be a redirect 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. 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` or `HTTP 403 Forbidden` for all requests.

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 Azure AD tenant can request a token for your application. You can **configure the application in Azure AD** 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 the Microsoft identity platform, see **Microsoft identity platform authorization basics**.

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 Azure AD 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)

Authentication scenarios and recommendations

Article • 08/10/2023

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 solutions, 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 Azure AD, Facebook, Google, Twitter.
- **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 Azure AD, Facebook, Google, Twitter.

Scenario recommendations

The following table lists each authentication solution and some important factors for when you would use it.

Authentication method	When to use
Built-in App Service authentication	* You want less code to own and manage. * Your app's language and SDKs don't provide user sign-in or

Authentication method	When to use
	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)	* 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	* 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.

Scenario	App Service built-in auth	Microsoft Authentication Library	Microsoft.Identity.Web
Need a fast and simple way to limit access to users in your organization?	✅	❌	❌
Unable to modify the application code (app migration scenario)?	✅	❌	❌
Your app's language and libraries support user sign-in/authorization?	❌	✅	✅
Even if you can use a code solution, would you rather *not* use	✅	❌	❌

Scenario	App Service built-in auth	Microsoft Authentication Library	Microsoft.Identity.Web
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

Identity and access management (IAM) fundamental concepts

Article • 06/12/2023

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 front line 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*.

Multi-factor 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

Here's a quick overview of authentication and authorization:

Authentication	Authorization
Can be thought of as a gatekeeper, allowing access only to those who provide valid credentials.	Can be thought of as a guard, ensuring that only those 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 ID is an example of a cloud-based identity provider. Other examples include Twitter, 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 Multi-factor authentication (MFA).

Configure your App Service or Azure Functions app to use Azure AD login

Article • 06/21/2023

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 (Azure AD) as the authentication provider.

The App Service Authentication feature can automatically create an app registration with the Microsoft identity platform. You can also use a registration that you or a directory admin creates separately.

- Create a new app registration automatically
- Use an existing registration created separately

> **Note**
>
> The option to create a new registration automatically is not available for government clouds or when using **Azure Active Directory for customers (Preview)**. Instead, **define a registration separately**.

Option 1: Create a new app registration automatically

Use this option unless you need to create an app registration separately. It makes enabling authentication simple and requires just a few clicks. You can customize the app registration in Azure AD once it's created.

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 **Microsoft** in the identity provider dropdown. The option to create a new registration is selected by default. You can change the name of the registration or the supported account types.

   A client secret will be created and stored 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.

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: 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.

6. Click **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 Azure AD login for a web app that accesses Azure Storage and Microsoft Graph, see this tutorial.

Option 2: Use an existing registration created separately

You can configure App Service authentication to use an existing app registration. 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 Azure AD tenant.
- You want to use an app registration from a different Azure AD tenant than the one your app is in.
- The option to create a new registration is not available for government clouds.

Step 1: Create an app registration in Azure AD for your App Service app

During creation of the app registration, collect the following information which you will need later when you configure the authentication in the App Service app:

- Client ID
- Tenant ID
- Client secret (optional, but recommended)
- Application ID URI

The instructions for creating an app registration depend on if you are using a workforce tenant or a customer tenant (Preview). Use the tabs below to select the right set of instructions for your scenario.

To register the app, perform the following steps:

1. Sign in to the Azure portal, search for and select **App Services**, and then select your app. Note your app's **URL**. You'll use it to configure your Azure Active Directory app registration.

2. Navigate to your tenant in the portal:

   Workforce tenant

   From the portal menu, select **Azure Active Directory**. If the tenant you are using is different from the one you use to configure the App Service application, you will need to change directories first.

3. On the "Overview" screen, make note of the **Tenant ID**, as well as the **Primary domain**.

4. From the left navigation, select **App registrations** > **New registration**.

5. In the **Register an application** page, enter a **Name** for your app registration.

6. In **Supported account types**, select the account type that can access this application.

7. 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`.

8. Select **Register**.

9. After the app registration is created, copy the **Application (client) ID** and the **Directory (tenant) ID** for later.

10. Under **Implicit grant and hybrid flows**, enable **ID tokens** to allow OpenID Connect user sign-ins from App Service. Select **Save**.

11. (Optional) From the left navigation, select **Branding & properties**. In **Home page URL**, enter the URL of your App Service app and select **Save**.

12. From the left navigation, select **Expose an API** > **Set** > **Save**. This value uniquely identifies the application when it is used as a resource, allowing tokens to be requested that grant access. It is 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 multi-tenant app, you must provide a custom URI. To learn more about accepted formats for App ID URIs, see the app registrations best practices reference.

13. 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**.

14. (Optional) 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.

15. (Optional) To add multiple **Reply URLs**, select **Authentication**.

16. Finish setting up your app registration:

    Workforce tenant

    No additional steps are required for a workforce tenant.

Step 2: Enable Azure Active Directory in your App Service app

1. Sign in to the Azure portal and navigate to your app.
2. From the left navigation, select **Authentication** > **Add identity provider** > **Microsoft**.
3. Select the **Tenant type** of the app registration you created.
4. Configure the app to use the registration you created, using the instructions for the appropriate tenant type:

Workforce tenant

For **App registration type**, choose one of the following:

- **Pick an existing app registration in this directory**: Choose an app registration from the current tenant and automatically gather the necessary app information. The system will attempt to create a new client secret against the app registration and automatically configure your app to use it. A default issuer URL is set based on the supported account types configured in the app registration. If you intend to change this default, consult the table below.
- **Provide the details of an existing app registration**: Specify details for an app registration from another tenant or if your account does not have permission in the current tenant to query the registrations. For this option, you must manually fill in the configuration values according to the table below.

The **authentication endpoint** for a workforce tenant should be a value specific to the cloud environment. For example, a workforce tenant in global Azure would use "https://login.microsoftonline.com" as its authentication endpoint. Make note of the authentication endpoint value, as it is needed to construct the right **Issuer URL**.

When filling in the configuration details directly, use the values you collected during the app registration creation process:

Field	Description
Application (client) ID	Use the **Application (client) ID** of the app registration.

Field	Description
Client Secret	Use the client secret you generated in the app registration. With a client secret, hybrid flow is used and the App Service will return access and refresh tokens. When the client secret is not set, implicit flow is used and only an ID token is returned. These tokens are sent by the provider and stored in the App Service authentication token store.
Issuer URL	Use `<authentication-endpoint>/<tenant-id>/v2.0`, and replace *<authentication-endpoint>* with the **authentication endpoint** you determined in the previous step for your tenant type and cloud environment, also replacing *<tenant-id>* with the **Directory (tenant) ID** in which the app registration was created. For applications that use Azure AD v1, omit `/v2.0` in the URL. This value is used to redirect users to the correct Azure AD tenant, as well as to download the appropriate metadata to determine the appropriate token signing keys and token issuer claim value for example. Any configuration other than a tenant-specific endpoint will be treated as multi-tenant. In multi-tenant configurations, no validation of the issuer or tenant ID is performed by the system, and these checks should be fully handled in your app's authorization logic.
Allowed Token Audiences	This field is optional. The configured **Application (client) ID** is *always* implicitly considered to be an allowed audience. If your application represents an API that will be called by other clients, you should also add the **Application ID URI** that you configured on the app registration. There is a limit of 500 characters total across the list of allowed audiences.

The client secret will be stored 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.

5. 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.

6. Click **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.

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 additional 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 does not 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 leverage 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 Azure AD 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 Azure Active Directory identity provider configuration has a `validation` section that can include a `defaultAuthorizationPolicy` object as in the following structure:

JSON

```
{
    "validation": {
        "defaultAuthorizationPolicy": {
            "allowedApplications": [],
            "allowedPrincipals": {
                "identities": []
            }
        }
    }
}
```

Property	Description
`defaultAuthorizationPolicy`	A grouping of requirements that must be met in order to access the app. Access is granted based on a logical `AND` over each of its configured properties. When `allowedApplications` and `allowedPrincipals` are both configured, the incoming request must satisfy both requirements in order to be accepted.
`allowedApplications`	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 `appid` or `azp` claim of the incoming token, which must be an access token. See the Microsoft Identity Platform claims reference.
`allowedPrincipals`	A grouping of checks that determine if the principal represented by the incoming request may access the app. Satisfaction of `allowedPrincipals` is based on a logical `OR` over its configured properties.

Property	Description
`identities` (under `allowedPrincipals`)	An allowlist of string **object IDs** representing users or applications that have access. When this property is configured as a nonempty array, the `allowedPrincipals` requirement can be satisfied if the user or application represented by the request is specified in the list. This policy evaluates the `oid` 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 (e.g., "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 Azure AD 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 is not 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 **Azure Active Directory**.

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 Create an app registration in Azure AD for your App Service app.

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 **Azure Active Directory**.
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 Azure AD 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 additional 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 click **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 is not 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 will keep your tenant and applications more secure:

- Configure each App Service app with its own app registration in Azure AD.
- 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 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 AAD 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. See Enable Azure Active Directory in your App Service app for general expectations around this value.

2. Remove requests for Azure AD Graph permissions from your login 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 login parameter `scope=openid profile email https://graph.microsoft.com/.default`.

With these changes, when App Service Authentication attempts to log 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 AAD.

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 • 01/25/2023

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

Configure your App Service or Azure Functions app to use Twitter login

Article • 03/31/2021

This article shows how to configure Azure App Service or Azure Functions to use Twitter as an authentication provider.

To complete the procedure in this article, you need a Twitter account that has a verified email address and phone number. To create a new Twitter account, go to twitter.com.

Register your application with Twitter

1. Sign in to the Azure portal and go to your application. Copy your **URL**. You'll use it to configure your Twitter app.

2. Go to the Twitter Developers website, sign in with your Twitter account credentials, and select **Create 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/twitter/callback`. For example, `https://contoso.azurewebsites.net/.auth/login/twitter/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 Twitter 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 Twitter 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 login using an OpenID Connect provider

Article • 05/23/2023

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 do not 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 will need to collect a **client ID** and **client secret** for your application.

> **Important**

The client secret is an important security credential. Do not share this secret with anyone or distribute it within a client application.

Additionally, you will 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 are unable to use a configuration metadata document, you will 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. Click **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.)

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.

5. Review the app registration information and select **Register**.
6. Again, on the **Identifiers** tab, select the **(+)** button.

7. On the **Register a New Identifier** page, choose **Services IDs** and select **Continue**.

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**.

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**.

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.eyJzdWIiOiJjb20ueW91cmNvbXBhbnkuYXBwMSIsIm5iZiI6MTU2MDIwMzIwNywiZXhwIjoxNTYwMjg5NjA3LCJpc3MiOiJBQkMxMjNERUZHIiwiYXVkIjoiaHR0cHM6Ly9hcHBsZWlkLmFwcGxlLmNvbSJ9.ABSXELWuTbgqfrIUz7bLi6nXvkXAz5O8vt0jB2dSHTQTib1x1DSP4__4UrlKI-pdzNg1sgeocolPNTmDKazO8-BHAZCsdeeTNlgFEzBytIpMKFfVEQbEtGRkam5IeclUK7S9oOva4EK4jV4VmgDrr-LGWWO3TaAxAvy3_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": "RedirectToLoginPage"
    },
    "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 • 01/25/2023

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 the Microsoft Identity Platform</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</a>
```

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:

Provider value	Required in request body	Comments
`aad`	`{"access_token":"<access_token>"}`	The `id_token`, `refresh_token`, and `expires_in` properties are optional.
`microsoftaccount`	`{"access_token":"<access_token>"}` or `{"authentication_token":"<token>"}`	`authentication_token` is preferred over `access_token`. The `expires_in` property is optional. When requesting the token from Live services, always request the `wl.basic` scope.
`google`	`{"id_token":"<id_token>"}`	The `authorization_code` property is optional. Providing an `authorization_code` value will add an access token and a refresh token to the token store. When specified, `authorization_code` can also optionally be accompanied by a `redirect_uri` property.
`facebook`	`{"access_token":"<user_access_token>"}`	Use a valid user access token from Facebook.

Provider value	Required in request body	Comments
`twitter`	`{"access_token":"<access_token>", "access_token_secret":"<access_token_secret>"}`	

If the provider token is validated successfully, the API returns with an `authenticationToken` in the response body, which is your session token.

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 Azure Active Directory and Google, performs a server-side sign-out on the identity provider.

Here's a simple sign-out link in a webpage:

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"
```

Limit the domain of sign-in accounts

Both Microsoft Account and Azure Active Directory lets you sign in from multiple domains. For example, Microsoft Account allows *outlook.com*, *live.com*, and *hotmail.com* accounts. Azure AD allows any number of custom domains for the sign-in accounts. However, you may want to accelerate your users straight to your own branded Azure AD sign-in page (such as `contoso.com`). To suggest the domain name of the sign-in accounts, follow these steps.

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
   <?xml version="1.0" encoding="utf-8"?>
   <configuration>
      <system.web>
         <authorization>
           <allow users="user1@contoso.com,user2@contoso.com"/>
           <deny users="*"/>
         </authorization>
      </system.web>
   </configuration>
   ```

Identity provider level

The identity provider may provide certain turn-key authorization. For example:

- For Azure App Service, you can manage enterprise-level access directly in Azure AD. For instructions, see How to remove a user's access to an application.
- For Google, Google API projects that belong to an organization can be configured to allow access only to users in your organization (see Google's **Setting up OAuth 2.0** support page).

Application level

If either of the other levels don't provide the authorization you need, or if your platform or identity provider isn't supported, you must write custom code to authorize users based on the user claims.

More resources

- Tutorial: Authenticate and authorize users end-to-end
- Environment variables and app settings for authentication

Work with user identities in Azure App Service authentication

Article • 09/04/2023

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:

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, e.g. Email Address, 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, the Microsoft Identity Provider also sets `X-MS-TOKEN-AAD-ACCESS-TOKEN` and `X-MS-TOKEN-AAD-ID-TOKEN` as appropriate.

> **Note**
>
> Different language frameworks may 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 may 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 may 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": ""
}
```

Property	Type	Description
`auth_typ`	string	The name of the identity provider used by App Service Authentication.
`claims`	array of objects	An array of objects representing the available claims. Each object contains `typ` and `val` properties.
`typ`	string	The name of the claim. This may have been subject to default claims mapping and could be different from the corresponding claim contained in a token.
`val`	string	The value of the claim.
`name_typ`	string	The name claim type, which is typically a URI providing scheme information about the `name` claim if one is defined.
`role_typ`	string	The role claim type, which is typically a URI providing scheme information about the `role` claim if one is defined.

To process this header, your app will need to decode the payload and iterate through the `claims` array to find the claims of interest. It may be convenient to convert these 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(HttpRequest 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.Claims`
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

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 • 02/17/2023

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 does not support creation or editing of Microsoft Account as a distinct provider as was done in V1. Rather, it leverages the converged Microsoft identity platform to sign-in users with both Azure AD and personal Microsoft accounts. When switching to the V2 API, the V1 Azure Active Directory (Azure AD) 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 it is recommended that you 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, click **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 do not 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:

   - Azure AD: `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` does not, and if it does, switch that value to `null`.

4. Add a property to `authsettings.json` which points to the application setting name you created earlier for each provider:

   - Azure AD: `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 Azure AD:

JSON

```
{
    "id": "/subscriptions/00d563f8-5b89-4c6a-bcec-
c1b9f6d607e0/resourceGroups/myresourcegroup/providers/Microsoft.Web/sit
es/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 does not contain an Azure AD provider, you can switch the configuration over to the Azure AD 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 URIs" 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 Azure AD 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 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 Azure AD 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 Azure AD 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 Azure AD app registration. However, this would force a new consent prompt for Microsoft Account users, and those users' identity claims may

be different in structure, `sub` 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":
"RedirectToLoginPage|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"
                ]
            },
        },
        "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 you're 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-lts" 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 Http Https --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	Http Https	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:
```

```yaml
  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.

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.

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 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.

6. Configure continuous deployment using GitHub Actions

1. Navigate to the Node.js backend sample app. 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. 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

```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
   ```

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.

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                                  02:05:58 PM
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                              02:06:16 PM
<!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";
```

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-
pc --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

Security recommendations for App Service

Article • 06/15/2023

This article contains security recommendations for Azure App Service. Implementing these recommendations will help you fulfill your security obligations as described in our shared responsibility model and will improve the overall security for your Web App solutions. For more information on what Microsoft does to fulfill service provider responsibilities, read Azure infrastructure security.

General

Recommendation	Comments
Stay up to date	Use the latest versions of supported platforms, programming languages, protocols, and frameworks.

Identity and access management

Recommendation	Comments
Disable anonymous access	Unless you need to support anonymous requests, disable anonymous access. For more information on Azure App Service authentication options, see Authentication and authorization in Azure App Service.
Require authentication	Whenever possible, use the App Service authentication module instead of writing code to handle authentication and authorization. See Authentication and authorization in Azure App Service.
Protect back-end resources with authenticated access	You can either use the user's identity or use an application identity to authenticate to a back-end resource. When you choose to use an application identity use a managed identity.
Require client certificate authentication	Client certificate authentication improves security by only allowing connections from clients that can authenticate using certificates that you provide.

Data protection

Recommendation	Comments

Recommendation	Comments
Redirect HTTP to HTTPs	By default, clients can connect to web apps by using both HTTP or HTTPS. We recommend redirecting HTTP to HTTPs because HTTPS uses the SSL/TLS protocol to provide a secure connection, which is both encrypted and authenticated.
Encrypt communication to Azure resources	When your app connects to Azure resources, such as SQL Database or Azure Storage, the connection stays in Azure. Since the connection goes through the shared networking in Azure, you should always encrypt all communication.
Require the latest TLS version possible	Since 2018 new Azure App Service apps use TLS 1.2. Newer versions of TLS include security improvements over older protocol versions.
Use FTPS	App Service supports both FTP and FTPS for deploying your files. Use FTPS instead of FTP when possible. When one or both of these protocols are not in use, you should disable them.
Secure application data	Don't store application secrets, such as database credentials, API tokens, or private keys in your code or configuration files. The commonly accepted approach is to access them as environment variables using the standard pattern in your language of choice. In Azure App Service, you can define environment variables through app settings and connection strings. App settings and connection strings are stored encrypted in Azure. The app settings are decrypted only before being injected into your app's process memory when the app starts. The encryption keys are rotated regularly. Alternatively, you can integrate your Azure App Service app with Azure Key Vault for advanced secrets management. By accessing the Key Vault with a managed identity, your App Service app can securely access the secrets you need.
Secure application code	Follow the steps to ensure the application code is secured.
Static Content	When authoring a web application serving static content, ensure that only the intended files/folders are processed. A configuration/code which serves out all files may not be sure by default. Follow application runtime/framework's best practices to secure the static content.
Hidden Folders	Ensure hidden folders like .git, bin, obj, objd, etc., doesn't get accidentally included as part of deployment artifact. Take adequate steps to ensure deployment scripts only deploy required files and nothing more.

Recommendation	Comments
In-place deployments	Understand nuances of in place deployment in local Git deployment. In-place deployment results in the creation and storage of the .git folder in the content root of the web application. Local Git deployment can activate in-place deployments automatically in some scenarios, even if in-place deployment isn't explicitly configured (for example, if the web app contains previously-deployed content when the local Git repository is initialized). Follow application runtime/framework's best practices to secure the content.

Networking

Recommendation	Comments
Use static IP restrictions	Azure App Service on Windows lets you define a list of IP addresses that are allowed to access your app. The allowed list can include individual IP addresses or a range of IP addresses defined by a subnet mask. For more information, see Azure App Service Static IP Restrictions.
Use the isolated pricing tier	Except for the isolated pricing tier, all tiers run your apps on the shared network infrastructure in Azure App Service. The isolated tier gives you complete network isolation by running your apps inside a dedicated App Service environment. An App Service environment runs in your own instance of Azure Virtual Network.
Use secure connections when accessing on-premises resources	You can use Hybrid connections, Virtual Network integration, or App Service environment's to connect to on-premises resources.
Limit exposure to inbound network traffic	Network security groups allow you to restrict network access and control the number of exposed endpoints. For more information, see How To Control Inbound Traffic to an App Service Environment.
Protect against DDoS attacks	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.

Monitoring

Recommendation	Comments

Recommendation	Comments
Use Microsoft Defender for Cloud's Microsoft Defender for App Service	Microsoft Defender for App Service is natively integrated with Azure App Service. Defender for Cloud 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. Microsoft Defender for Cloud also provides threat protection and can detect a multitude of threats covering almost the complete list of MITRE ATT&CK tactics from pre-attack to command and control. For a full list of the Azure App Service alerts, see Microsoft Defender for App Service alerts.

Next steps

Check with your application provider to see if there are additional security requirements. For more information on developing secure applications, see Secure Development Documentation.

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 loose 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 `https://<appname>.scm.azurewebsites.net/Env.cshtml` (under System info)
.NET version	At `https://<appname>.scm.azurewebsites.net/DebugConsole`, run the following command in the command prompt: `reg query "HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\NET Framework Setup\NDP\v4\Full"`
.NET Core version	At `https://<appname>.scm.azurewebsites.net/DebugConsole`, run the following command in the command prompt: `dotnet --version`
PHP version	At `https://<appname>.scm.azurewebsites.net/DebugConsole`, run the following command in the command prompt: `php --version`
Default Node.js version	In the Cloud Shell, run the following command: `az webapp config appsettings list --resource-group <groupname> --name <appname> --query "[?name=='WEBSITE_NODE_DEFAULT_VERSION']"`
Python version	At `https://<appname>.scm.azurewebsites.net/DebugConsole`, run the following command in the command prompt: `python --version`

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 • 04/03/2023

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 you haven't configured custom DNS, 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've configured the servers. If none of the servers respond within the timeout and you have 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']"
```

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 a DNS server to end of the configured DNS servers. You use the setting to configure a fallback server to custom DNS servers from the virtual network.

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 may influence performance.

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	Cache timeout defined in seconds. Setting cache to zero means you've disabled caching.
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 enable DNS caching on Web App for Containers and Linux-based apps, you must add the app setting `WEBSITE_ENABLE_DNS_CACHE`. This setting defaults

to 30 seconds.

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 managed 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 • 09/19/2023

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.

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	2.0.1

Domain	Control ID	Control title	Policy (Azure portal)	Policy version (GitHub)
Guidelines for Cryptography - Transport Layer Security	1139	Using Transport Layer Security - 1139	Function apps should use the latest TLS version	2.0.1
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

Azure Security Benchmark v1

The Azure Security Benchmark provides recommendations on how you can secure your cloud solutions on Azure. To see how this service completely maps to the Azure 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 - Azure Security Benchmark.

Domain	Control ID	Control title	Policy (Azure portal)	Policy version (GitHub)
Network Security	1.1	Protect resources using Network Security Groups or Azure Firewall on your Virtual Network	App Service apps should use a virtual network service endpoint	2.0.1
Network Security	1.3	Protect critical web applications	App Service apps should have 'Client Certificates	3.0.0

Domain	Control ID	Control title	Policy (Azure portal)	Policy version (GitHub)
			(Incoming client certificates)' enabled	
Network Security	1.3	Protect critical web applications	App Service apps should have remote debugging turned off	2.0.0
Network Security	1.3	Protect critical web applications	App Service apps should not have CORS configured to allow every resource to access your apps	2.0.0
Network Security	1.3	Protect critical web applications	Function apps should have remote debugging turned off	2.0.0
Network Security	1.3	Protect critical web applications	Function apps should not have CORS configured to allow every resource to access your apps	2.0.0
Logging and Monitoring	2.3	Enable audit logging for Azure resources	App Service apps should have resource logs enabled	2.0.1
Data Protection	4.4	Encrypt all sensitive information in transit	App Service apps should only be accessible over HTTPS	4.0.0
Data Protection	4.4	Encrypt all sensitive information in transit	App Service apps should require FTPS only	3.0.0
Data Protection	4.4	Encrypt all sensitive information in transit	App Service apps should use the latest TLS version	2.0.1
Data Protection	4.4	Encrypt all sensitive information in transit	Function apps should only be accessible over HTTPS	5.0.0
Data Protection	4.4	Encrypt all sensitive information in transit	Function apps should require FTPS only	3.0.0
Data Protection	4.4	Encrypt all sensitive information in transit	Function apps should use the latest TLS version	2.0.1
Secure Configuration	7.12	Manage identities securely and automatically	App Service apps should use managed identity	3.0.0
Secure Configuration	7.12	Manage identities securely and automatically	Function apps should use managed identity	3.0.0

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.

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 resource to access your apps	2.0.0
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.

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
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	2.0.1
9 AppService	9.3	Ensure web app is using the latest version of TLS encryption	Function apps should use the latest TLS version	2.0.1
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	3.0.0
9 AppService	9.4	Ensure the web app has 'Client Certificates (Incoming client certificates)' set to 'On'	Function apps should have 'Client Certificates (Incoming client certificates)' enabled	3.0.0
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.

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	2.0.1
9 AppService	9.3	Ensure web app is using the latest version of TLS encryption	Function apps should use the latest TLS version	2.0.1
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	3.0.0
9 AppService	9.4	Ensure the web app has 'Client Certificates (Incoming client certificates)' set to 'On'	Function apps should have 'Client Certificates (Incoming client certificates)' enabled	3.0.0
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

Domain	Control ID	Control title	Policy (Azure portal)	Policy version (GitHub)
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 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 .

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
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

Domain	Control ID	Control title	Policy (Azure portal)	Policy version (GitHub)
9 AppService	9.3	Ensure Web App is using the latest version of TLS encryption	App Service apps should use the latest TLS version	2.0.1
9 AppService	9.3	Ensure Web App is using the latest version of TLS encryption	Function apps should use the latest TLS version	2.0.1
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	3.0.0
9 AppService	9.4	Ensure the web app has 'Client Certificates (Incoming client certificates)' set to 'On'	Function apps should have 'Client Certificates (Incoming client certificates)' enabled	3.0.0
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 .

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	2.0.1
9	9.3	Ensure Web App is using the latest version of TLS encryption	Function apps should use the latest TLS version	2.0.1
9	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	3.0.0
9	9.4	Ensure the web app has 'Client Certificates (Incoming client certificates)' set to 'On'	Function apps should have 'Client Certificates (Incoming client certificates)' enabled	3.0.0
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
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 Service apps that use PHP should use a specified 'PHP	3.2.0

Domain	Control ID	Control title	Policy (Azure portal)	Policy version (GitHub)
		App	version'	
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).

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	App Service apps should not have CORS configured to	2.0.0

Domain	Control ID	Control title	Policy (Azure portal)	Policy version (GitHub)
		devices (including other information systems).	allow every resource to access your apps	
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 functions that authorized users are permitted to execute.	Function apps should only be accessible over HTTPS	5.0.0
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	2.0.0

Domain	Control ID	Control title	Policy (Azure portal)	Policy version (GitHub)
			debugging turned off	
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
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	2.0.1

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.	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	2.0.1
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	2.0.1
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 external boundaries and key internal boundaries of organizational systems.	Function apps should use the latest TLS version	2.0.1
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

Domain	Control ID	Control title	Policy (Azure portal)	Policy version (GitHub)
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	2.0.1
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	2.0.1
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	2.0.1
System and Communications Protection	SC.3.190	Protect the authenticity of communications sessions.	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.3.190	Protect the authenticity of communications sessions.	Function apps should use the latest TLS version	2.0.1
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	2.0.1
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	2.0.1

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.

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

Domain	Control ID	Control title	Policy (Azure portal)	Policy version (GitHub)
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 (Incoming client certificates)' enabled	3.0.0
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 'Client Certificates (Incoming client certificates)' enabled	3.0.0
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	2.0.0

Domain	Control ID	Control title	Policy (Azure portal)	Policy version (GitHub)
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	2.0.1
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	2.0.1
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	2.0.1

Domain	Control ID	Control title	Policy (Azure portal)	Policy version (GitHub)
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	2.0.1
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

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.

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	3.0.0

Domain	Control ID	Control title	Policy (Azure portal)	Policy version (GitHub)
			managed identity	
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-12	Audit Generation	App Service apps should have resource logs enabled	2.0.1
Configuration Management	CM-6	Configuration Settings	App Service apps should have 'Client Certificates (Incoming client certificates)' enabled	3.0.0
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 'Client Certificates (Incoming client certificates)' enabled	3.0.0
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

Domain	Control ID	Control title	Policy (Azure portal)	Policy version (GitHub)
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	2.0.1
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	2.0.1
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	2.0.1

Domain	Control ID	Control title	Policy (Azure portal)	Policy version (GitHub)
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	2.0.1
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.

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	1403.05i1Organizational.67 - 05.i	Access granted to external parties is limited to the minimum	App Service apps should only	4.0.0

Domain	Control ID	Control title	Policy (Azure portal)	Policy version (GitHub)
External Parties		necessary and granted only for the duration required.	be accessible over HTTPS	
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	3.0.0
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	2.0.1
08 Network Protection	0809.01n2Organizational.1234-01.n	0809.01n2Organizational.1234-01.n 01.04 Network Access Control	Function apps should only be	5.0.0

Domain	Control ID	Control title	Policy (Azure portal)	Policy version (GitHub)
			accessible over HTTPS	
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	2.0.1
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	2.0.1
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	2.0.1
08 Network Protection	0811.01n2Organizational.6-01.n	0811.01n2Organizational.6-01.n 01.04 Network Access Control	App Service apps should only be accessible over HTTPS	4.0.0
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	2.0.1

Domain	Control ID	Control title	Policy (Azure portal)	Policy version (GitHub)
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	2.0.1
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	2.0.1
08 Network Protection	0812.01n2Organizational.8-01.n	0812.01n2Organizational.8-01.n 01.04 Network Access Control	Function apps should only be accessible over HTTPS	5.0.0
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	2.0.1
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	2.0.1

Domain	Control ID	Control title	Policy (Azure portal)	Policy version (GitHub)
			should use the latest TLS version	
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	2.0.1
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
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	2.0.0

Domain	Control ID	Control title	Policy (Azure portal)	Policy version (GitHub)
			to allow every resource to access your apps	
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 debugging turned off	2.0.0
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	3.0.0
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	4.0.0

Domain	Control ID	Control title	Policy (Azure portal)	Policy version (GitHub)
			accessible over HTTPS	
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	2.0.1
09 Transmission Protection	0949.09y2Organizational.5-09.y	0949.09y2Organizational.5-09.y 09.09 Electronic Commerce Services	Function apps should only be accessible over HTTPS	5.0.0
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	2.0.1
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

Domain	Control ID	Control title	Policy (Azure portal)	Policy version (GitHub)
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 Awareness	1325.09s1Organizational.3-09.s	1325.09s1Organizational.3-09.s 09.08 Exchange of Information	Function apps should have remote debugging turned off	2.0.0

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.

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.

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.

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	2.0.1
Network Security	NS-8	Detect and disable insecure services and protocols	Function apps should use the latest TLS version	2.0.1
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	2.0.1
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	2.0.1
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	App Service apps should have 'Client Certificates (Incoming client certificates)' enabled	3.0.0
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 'Client Certificates (Incoming client certificates)' enabled	3.0.0
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

New Zealand ISM Restricted

To review how the available Azure Policy built-ins for all Azure services map to this compliance standard, see Azure Policy Regulatory Compliance - New Zealand ISM Restricted. For more information about this compliance standard, see New Zealand ISM Restricted.

Domain	Control ID	Control title	Policy (Azure portal)	Policy version (GitHub)
Software security	SS-2	14.1.8 Developing hardened SOEs	App Service apps should have remote debugging turned off	2.0.0
Software security	SS-2	14.1.8 Developing hardened SOEs	Function apps should have remote debugging turned off	2.0.0
Software security	SS-9	14.5.8 Web applications	App Service apps should not have CORS configured to allow every resource to access your apps	2.0.0
Software security	SS-9	14.5.8 Web applications	App Service apps should only be accessible over HTTPS	4.0.0
Software security	SS-9	14.5.8 Web applications	Function apps should not have CORS configured to allow every resource to access your apps	2.0.0
Software security	SS-9	14.5.8 Web applications	Function apps should only be accessible over HTTPS	5.0.0
Access Control and Passwords	AC-2	16.1.32 System User Identitfication	App Service apps should use managed identity	3.0.0
Access Control and Passwords	AC-2	16.1.32 System User Identitfication	Function apps should use managed identity	3.0.0
Access Control and Passwords	AC-17	16.6.9 Events to be logged	App Service apps should have resource logs enabled	2.0.1
Cryptography	CR-7	17.4.16 Using TLS	App Service apps should require FTPS only	3.0.0
Cryptography	CR-7	17.4.16 Using TLS	App Service apps should use the latest TLS version	2.0.1
Cryptography	CR-7	17.4.16 Using TLS	Function apps should require FTPS only	3.0.0
Cryptography	CR-7	17.4.16 Using TLS	Function apps should use the latest TLS version	2.0.1

NIST SP 800-171 R2

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.

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
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

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.	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
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	2.0.1
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.	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	Function apps should require FTPS only	3.0.0

Domain	Control ID	Control title	Policy (Azure portal)	Policy version (GitHub)
		otherwise protected by alternative physical safeguards.		
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	2.0.1
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
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.	App Service apps should have 'Client Certificates (Incoming client certificates)' enabled	3.0.0
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	App Service apps should not have	2.0.0

Domain	Control ID	Control title	Policy (Azure portal)	Policy version (GitHub)
		organizational systems (including hardware, software, firmware, and documentation) throughout the respective system development life cycles.	CORS configured to allow every resource to access your apps	
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 have 'Client Certificates (Incoming client certificates)' enabled	3.0.0
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.	Function 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.	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.	App Service apps should have 'Client Certificates (Incoming client certificates)' enabled	3.0.0
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

Domain	Control ID	Control title	Policy (Azure portal)	Policy version (GitHub)
Configuration Management	3.4.2	Establish and enforce security configuration settings for information technology products employed in organizational systems.	Function apps should have 'Client Certificates (Incoming client certificates)' enabled	3.0.0
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 products employed in organizational systems.	Function apps should not have CORS configured to allow every resource to access your apps	2.0.0
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	3.0.0

Domain	Control ID	Control title	Policy (Azure portal)	Policy version (GitHub)
			identity	
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.

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

Domain	Control ID	Control title	Policy (Azure portal)	Policy version (GitHub)
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 (Incoming client certificates)' enabled	3.0.0
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 'Client Certificates (Incoming client certificates)' enabled	3.0.0
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

Domain	Control ID	Control title	Policy (Azure portal)	Policy version (GitHub)
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	2.0.1
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	2.0.1
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	2.0.1
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	SC-8 (1)	Cryptographic Or Alternate Physical	Function apps should use the latest TLS version	2.0.1

Domain	Control ID	Control title	Policy (Azure portal)	Policy version (GitHub)
Protection		Protection		
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	App Service apps should use latest 'HTTP Version'	4.0.0
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	App Service apps should use latest 'HTTP Version'	4.0.0
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	Flaw Remediation	Function 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	Flaw Remediation	Function 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	App Service apps should use latest 'HTTP Version'	4.0.0
System and Information Integrity	SI-2 (6)	Removal of Previous Versions of Software /	App Service apps should use latest 'HTTP Version'	4.0.0

Domain	Control ID	Control title	Policy (Azure portal)	Policy version (GitHub)
		Firmware		
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	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
System and Information Integrity	SI-2 (6)	Removal of Previous Versions of Software / Firmware	Function 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
System And Information Integrity	SI-2 (6)	Removal of Previous Versions of Software / Firmware	Function 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.

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

Domain	Control ID	Control title	Policy (Azure portal)	Policy version (GitHub)
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)	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	App Service apps should have 'Client Certificates (Incoming client certificates)' enabled	3.0.0
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	2.0.0

Domain	Control ID	Control title	Policy (Azure portal)	Policy version (GitHub)
			allow every resource to access your apps	
Configuration Management	CM-6	Configuration Settings	Function apps should have 'Client Certificates (Incoming client certificates)' enabled	3.0.0
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	2.0.1
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

Domain	Control ID	Control title	Policy (Azure portal)	Policy version (GitHub)
System and Communications Protection	SC-8	Transmission Confidentiality and Integrity	Function apps should use the latest TLS version	2.0.1
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	2.0.1
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	2.0.1
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 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

NZ ISM Restricted v3.5

To review how the available Azure Policy built-ins for all Azure services map to this compliance standard, see Azure Policy Regulatory Compliance - NZ ISM Restricted v3.5. For more information about this compliance standard, see NZ ISM Restricted v3.5 .

Domain	Control ID	Control title	Policy (Azure portal)	Policy version (GitHub)
Access Control and Passwords	AC-18	16.6.9 Events to be logged	App Service apps should have resource logs enabled	2.0.1
Access Control and Passwords	AC-2	16.1.32 System User Identitfication	App Service apps should use managed identity	3.0.0
Access Control and Passwords	AC-2	16.1.32 System User Identitfication	Function apps should use managed identity	3.0.0
Cryptography	CR-8	17.4.16 Using TLS	App Service apps should use the latest TLS version	2.0.1
Cryptography	CR-8	17.4.16 Using TLS	Function apps should use the latest TLS version	2.0.1
Software security	SS-2	14.1.8 Developing hardened SOEs	App Service apps should have remote debugging turned off	2.0.0
Software security	SS-2	14.1.8 Developing hardened SOEs	Function apps should have remote debugging turned off	2.0.0
Software security	SS-9	14.5.8 Web applications	App Service apps should have 'Client Certificates (Incoming client certificates)' enabled	3.0.0
Software security	SS-9	14.5.8 Web applications	App Service apps should have authentication enabled	2.0.1
Software security	SS-9	14.5.8 Web applications	App Service apps should not have CORS configured to allow every resource to access your apps	2.0.0
Software security	SS-9	14.5.8 Web applications	App Service apps should only be accessible over HTTPS	4.0.0
Software security	SS-9	14.5.8 Web applications	App Service apps should require FTPS only	3.0.0
Software security	SS-9	14.5.8 Web applications	App Service apps should use latest 'HTTP Version'	4.0.0
Software security	SS-9	14.5.8 Web applications	Function apps should have 'Client Certificates (Incoming client	3.0.0

Domain	Control ID	Control title	Policy (Azure portal)	Policy version (GitHub)
			certificates)' enabled	
Software security	SS-9	14.5.8 Web applications	Function apps should have authentication enabled	3.0.0
Software security	SS-9	14.5.8 Web applications	Function apps should not have CORS configured to allow every resource to access your apps	2.0.0
Software security	SS-9	14.5.8 Web applications	Function apps should only be accessible over HTTPS	5.0.0
Software security	SS-9	14.5.8 Web applications	Function apps should require FTPS only	3.0.0
Software security	SS-9	14.5.8 Web applications	Function apps should use latest 'HTTP Version'	4.0.0

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.

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
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.

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.

Domain	Control ID	Control title	Policy (Azure portal)	Policy version (GitHub)
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 'Client Certificates (Incoming client certificates)' enabled	3.0.0
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	App Service apps should only be accessible over HTTPS	4.0.0

Domain	Control ID	Control title	Policy (Azure portal)	Policy version (GitHub)
		(PKI)-3.1		
Information and Cyber Security	3.1.h	Public Key Infrastructure (PKI)-3.1	App Service apps should use the latest TLS version	2.0.1
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	2.0.1
Information and Cyber Security	3.8	Digital Signatures-3.8	App Service apps should have 'Client Certificates (Incoming client certificates)' enabled	3.0.0
Information and Cyber Security	3.8	Digital Signatures-3.8	Function apps should have 'Client Certificates (Incoming client certificates)' enabled	3.0.0

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).

Domain	Control ID	Control title	Policy (Azure portal)	Policy version (GitHub)
Advanced Real-Timethreat Defenceand Management		Advanced Real-Timethreat Defenceand Management-13.1	App Service apps should have 'Client Certificates (Incoming client certificates)' enabled	3.0.0
Network Management And Security		Network Device Configuration Management-4.3	App Service apps should have 'Client Certificates	3.0.0

Domain	Control ID	Control title	Policy (Azure portal)	Policy version (GitHub)
			(Incoming client certificates)' enabled	
Network Management And Security		Network Device Configuration Management-4.3	App Service apps should have remote debugging turned off	2.0.0
Advanced Real-Timethreat Defenceand Management		Advanced Real-Timethreat Defenceand Management-13.1	App Service apps should have remote debugging turned off	2.0.0
Audit Log Settings		Audit Log Settings-17.1	App Service apps should have resource logs enabled	2.0.1
Application Security Life Cycle (Aslc)		Application Security Life Cycle (Aslc)-6.4	App Service apps should have resource logs enabled	2.0.1
Application Security Life Cycle (Aslc)		Application Security Life Cycle (Aslc)-6.4	App Service apps should have resource logs enabled	2.0.1
Advanced Real-Timethreat Defenceand Management		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-10.2	App Service apps should only be accessible over HTTPS	4.0.0
Advanced Real-Timethreat Defenceand Management		Advanced Real-Timethreat Defenceand Management-13.4	App Service apps should only be accessible over HTTPS	4.0.0
Secure Mail And Messaging Systems		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	App Service apps should require FTPS only	3.0.0
Advanced Real-Timethreat Defenceand Management		Advanced Real-Timethreat Defenceand Management-13.4	App Service apps should require FTPS only	3.0.0
Secure Mail And Messaging Systems		Secure Mail And Messaging Systems-	App Service apps should require FTPS only	3.0.0

Domain	Control ID	Control title	Policy (Azure portal)	Policy version (GitHub)
Application Security Life Cycle (Aslc)		Application Security Life Cycle (Aslc)-6.4	App Service apps should use managed identity	3.0.0
User Access Control / Management		User Access Control / Management-8.4	App Service apps should use managed identity	3.0.0
Application Security Life Cycle (Aslc)		Application Security Life Cycle (Aslc)-6.4	App Service apps should use managed identity	3.0.0
Advanced Real-Timethreat Defenceand Management		Advanced Real-Timethreat Defenceand Management-13.1	App Service apps should use the latest TLS version	2.0.1
Advanced Real-Timethreat Defenceand Management		Advanced Real-Timethreat Defenceand Management-13.4	App Service apps should use the latest TLS version	2.0.1
Secure Mail And Messaging Systems		Secure Mail And Messaging Systems-10.1	App Service apps should use the latest TLS version	2.0.1
Secure Mail And Messaging Systems		Secure Mail And Messaging Systems-10.2	App Service apps should use the latest TLS version	2.0.1
Advanced Real-Timethreat Defenceand Management		Advanced Real-Timethreat Defenceand Management-13.1	Function apps should have 'Client Certificates (Incoming client certificates)' enabled	3.0.0
Advanced Real-Timethreat Defenceand Management		Advanced Real-Timethreat Defenceand Management-13.1	Function apps should have remote debugging turned off	2.0.0
Network Management And Security		Network Device Configuration Management-4.3	Function apps should have remote debugging turned off	2.0.0
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
Advanced Real-Timethreat Defenceand Management		Advanced Real-Timethreat Defenceand Management-13.4	Function apps should only be accessible over HTTPS	5.0.0

Domain	Control ID	Control title	Policy (Azure portal)	Policy version (GitHub)
Secure Mail And Messaging Systems		Secure Mail And Messaging Systems-10.2	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 only be accessible over HTTPS	5.0.0
Secure Mail And Messaging Systems		Secure Mail And Messaging Systems-10.2	Function apps should require FTPS only	3.0.0
Advanced Real-Timethreat Defenceand Management		Advanced Real-Timethreat Defenceand Management-13.4	Function apps should require FTPS only	3.0.0
Secure Mail And Messaging Systems		Secure Mail And Messaging Systems-10.1	Function apps should require FTPS only	3.0.0
Application Security Life Cycle (Aslc)		Application Security Life Cycle (Aslc)-6.4	Function apps should use managed identity	3.0.0
User Access Control / Management		User Access Control / Management-8.4	Function apps should use managed identity	3.0.0
Application Security Life Cycle (Aslc)		Application Security Life Cycle (Aslc)-6.4	Function apps should use managed identity	3.0.0
Advanced Real-Timethreat Defenceand Management		Advanced Real-Timethreat Defenceand Management-13.1	Function apps should use the latest TLS version	2.0.1
Secure Mail And Messaging Systems		Secure Mail And Messaging Systems-10.2	Function apps should use the latest TLS version	2.0.1
Advanced Real-Timethreat Defenceand Management		Advanced Real-Timethreat Defenceand Management-13.4	Function apps should use the latest TLS version	2.0.1
Secure Mail And Messaging Systems		Secure Mail And Messaging Systems-10.1	Function apps should use the latest TLS version	2.0.1

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.

Domain	Control ID	Control title	Policy (Azure portal)	Policy version (GitHub)
Cryptography	10.20	Cryptography - 10.20	App Service apps should have 'Client Certificates (Incoming client certificates)' enabled	3.0.0
Cryptography	10.20	Cryptography - 10.20	Function apps should have 'Client Certificates (Incoming client certificates)' enabled	3.0.0
Access Control	10.54	Access Control - 10.54	App Service apps should have authentication enabled	2.0.1
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	2.0.1
Security of Digital Services	10.68	Security of Digital Services - 10.68	Function apps should use the latest TLS version	2.0.1
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

Domain	Control ID	Control title	Policy (Azure portal)	Policy version (GitHub)
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
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

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.

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

Domain	Control ID	Control title	Policy (Azure portal)	Policy version (GitHub)
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 (Incoming client certificates)' enabled	3.0.0
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	2.0.1
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	2.0.1
Reduce Attack Surface and Vulnerabilities	2.4A	Back-office Data Flow Security	App Service apps should have 'Client Certificates (Incoming client certificates)' enabled	3.0.0
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

Domain	Control ID	Control title	Policy (Azure portal)	Policy version (GitHub)
Reduce Attack Surface and Vulnerabilities	2.6	Operator Session Confidentiality and Integrity	App Service apps should use the latest TLS version	2.0.1
Reduce Attack Surface and Vulnerabilities	2.6	Operator Session Confidentiality and Integrity	Function apps should use the latest TLS version	2.0.1
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

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.

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
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

Release notes

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

- Learn more about Azure Policy Regulatory Compliance.
- See the built-ins on the Azure Policy GitHub repo.

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 property called `possibleOutboundAddresses` that will list them.

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 • 04/05/2023

Azure App Service is a multi-tenant service, except for App Service Environments. Apps that are not in an App Service environment (not in the Isolated tier) share network infrastructure with other apps. As a result, the inbound and outbound IP addresses of an app can be different, and can even change in certain situations.

App Service Environments use dedicated network infrastructures, so apps running in an App Service environment get static, dedicated IP addresses both for inbound and outbound connections.

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 provisions a static IP address to make it happen.

When outbound IPs change

Regardless of the number of scaled-out instances, each app has a set number of outbound IP addresses at any given time. Any outbound connection from the App Service app, such as to a back-end database, uses one of the outbound IP addresses as the origin IP address. 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** blade in the Azure portal. See Find outbound IPs.

Note that 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 will increase. 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 will also increase. This has no immediate effect since the outbound IP addresses for running applications do not 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 will be used and customers will 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, click **Properties** in your app's left-hand navigation. They are listed in the **Outbound IP Addresses** field.

You can find the same information by running the following command in the Cloud Shell.

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, click **Properties** in your app's left-hand navigation. They are listed in the **Additional Outbound IP Addresses** field.

You can find the same information by running the following command in the Cloud Shell.

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 regional VNet integration together with a virtual network NAT gateway to direct traffic through a static public IP address. Regional VNet integration is available on **Basic**, **Standard**, **Premium**, **PremiumV2** and **PremiumV3** App Service plans. To learn more about this setup, see NAT gateway integration.

Next steps

Learn how to restrict inbound traffic by source IP addresses.

Static IP restrictions

Azure App Service access restrictions

Article • 08/01/2023

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's sent 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. If you've never configured the setting, the default behavior is to enable access unless a private endpoint exists after which it's implicitly disabled. You have the ability to explicitly configure this behavior to either enabled or disabled even if private endpoints exist.

In the Azure Resource Manager API, app access is called `publicNetworkAccess`. For 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 web application; 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 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 has never been configured, the unmatched rule behavior is to allow all access unless one or more rules exists after which it's implicitly changed 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 may want to 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 CIDR values.
- **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 for third party proxies be used 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 for third party proxies be used 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

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.

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 may want to 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 • 03/09/2023

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 that's 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 is enabled by service endpoints. With service endpoints, you can restrict access to a multi-tenant 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:

1. Sign in to the Azure portal.
2. Select the app that you want to add access restrictions to.
3. On the left pane, select **Networking**.
4. On the **Networking** pane, under **Access Restrictions**, select **Configure Access Restrictions**.

5. On the **Access Restrictions** page, review the list of access restriction rules that are defined for your app.

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, the app is accessible from anywhere.

Permissions

You must have at least the following Role-based access control permissions on the subnet or at a higher level to configure access restrictions through Azure portal, CLI or when setting the site config properties directly:

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 are 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 it will also automatically be 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** pane, select **Add rule**. After you add a rule, it becomes effective immediately.

Rules are enforced in priority order, starting from the lowest number in the **Priority** column. 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**.

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. After typing in the rule specific input select **Save** to save the changes.

> **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**.

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'll be 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 will be 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 currently 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**.

All 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.

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.

> ⓘ **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.

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 additional 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 might use this if you've more than 512 IP ranges or 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.

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**.

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 that's 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 will be hidden, and it will use the rules from the main site. If you clear the check box, your SCM site settings will appear again.

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'll need to further filter the incoming requests based on the unique http header that Azure Front Door sends.

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

Integrate your app with an Azure virtual network

Article • 07/21/2023

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's already integrated 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's placed 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.

For Windows App Service plans, 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, a Windows 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. Linux App Service plans support only one virtual network integration per plan.

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 size or in/out in number of instances, the required address space is doubled for a short period of time. This is because the scale operation adds the same number of new instances and then deletes the existing instances. 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.

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.

> **Note**
>
> 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 x 10 = 50 IP addresses per App Service plan
>
> Since you have 1 App Service plan, 1 x 50 = 50 IP addresses.

When you want your apps in your plan to reach a virtual network that's already connected to by apps in another plan, select a different subnet than the one being used by the pre-existing virtual network integration.

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:

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's automatically registered 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 it's sent through the virtual network integration.

Application routing

Application routing applies to traffic that is sent from your app after it has been started. 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 **Route All** setting. If **Route All** 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 **Route All** 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 **Route All** is enabled, the source address for your outbound public 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 in 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.

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**
>
> - Backup/restore to private storage accounts is currently not supported.
> - 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.

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's applied 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 Azure Active Directory. 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 can't be used by Isolated plan apps that are 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.
- 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.
- The integration subnet can be used by only one App Service plan.
- 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 Windows App Service plan. You can't have more than one virtual network integration per Linux 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/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.

1. Go to **Networking** > **Virtual network integration** in your app portal.

2. Uncheck the **Outbound internet traffic** setting.

3. Select **Apply** to confirm.

Configure with the Azure CLI

You can also configure **Outbound internet traffic** by using the Azure CLI.

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 • 01/20/2023

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.

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 • 03/03/2023

> **Important**
>
> Private endpoint is available for Windows and Linux apps, containerized or not, hosted on these App Service plans : **Basic**, **Standard**, **PremiumV2**, **PremiumV3**, **IsolatedV2**, **Functions Premium** (sometimes referred to as the Elastic Premium plan).

You can use private endpoint for your App Service apps to allow 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, eliminating exposure from the public Internet.

Using 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 won'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 plug up to 100 private endpoints per slot. You can't share a private endpoint between slots. The sub-resource 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 cannot use the same subnet as private endpoint, this is a limitation of the virtual network integration feature.

From a security perspective:

- Private endpoint and public access can co-exist on an app. For more information, see overview of access restrictions
- When you enable private endpoints to your app, ensure that public network access is disabled to ensure isolation.
- 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 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 mywebappname.azurewebsites.net.

By default, without private endpoint, the public name of your web app is a canonical name to the cluster. For example, the name resolution is:

Name	Type	Value
mywebapp.azurewebsites.net	CNAME	clustername.azurewebsites.windows.net
clustername.azurewebsites.windows.net	CNAME	cloudservicename.cloudapp.net
cloudservicename.cloudapp.net	A	40.122.110.154

When you deploy a private endpoint, we update the DNS entry to point to the canonical name mywebapp.privatelink.azurewebsites.net. For example, the name resolution is:

Name	Type	Value	Remark

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	40.122.110.154	<--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:

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 endpoint address

Name	Type	Value	Remark
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 mywebappname.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, you must 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 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, the second is for the SCM of your app.

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), you have to 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, 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 you also 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 private endpoint, to run or execute the function in Azure portal, you must have direct network access or you receive an HTTP 403 error. In other words, your browser must be able to reach the private endpoint to execute 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. The recommendation is to deploy the code to a slot and remote debug it there.
- FTP access is provided through the inbound public IP address. Private endpoint doesn't support FTP access to the app.
- IP-Based SSL isn't supported with private endpoints.
- Apps that you configure with private endpoints cannot use service endpoint-based access restriction rules.

We're improving Azure Private Link feature and private endpoint regularly, check this article for up-to-date information about limitations.

Next steps

- 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
- End-to-end example, how to connect a frontend app to a secured backend app with virtual network integration and private endpoint with ARM template, see this quickstart
- End-to-end example, how to connect a frontend app to a secured backend app with virtual network integration and private endpoint with terraform, see this sample

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**.

Home > Resource groups > rg-eus-appservice-networking-demo > Create a resource > Marketplace >

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

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.

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:

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`.

> **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.

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.

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.

More resources

Monitor Azure Firewall logs and metrics.

Application Gateway integration

Article • 04/09/2023

There are three variations of App Service that require slightly different configuration of the integration with Azure Application Gateway. The variations include regular App Service - also known as multi-tenant, Internal Load Balancer (ILB) App Service Environment and External App Service Environment. This article will walk through how to configure it with App Service (multi-tenant) using service endpoint to secure traffic. The article will also discuss considerations around using private endpoint and integrating with ILB, and External App Service Environment. Finally the article has considerations on scm/kudu site.

Integration with App Service (multi-tenant)

App Service (multi-tenant) has a public internet facing endpoint. Using service endpoints you can allow traffic only from a specific subnet within an Azure Virtual Network and block everything else. In the following scenario, we'll use this functionality to ensure that an App Service instance can only receive traffic from a specific Application Gateway instance.

There are two parts to this configuration besides creating the App Service and the Application Gateway. The first part is enabling service endpoints in the subnet of the Virtual Network where the Application Gateway is deployed. Service endpoints will ensure all network traffic leaving the subnet towards the App Service will be tagged with the specific subnet ID. The second part is to set an access restriction of the specific web app to ensure that only traffic tagged with this specific subnet ID is allowed. You can configure it using different tools depending on preference.

Using Azure portal

With Azure portal, you follow four steps to provision and configure the setup. If you have existing resources, you can skip the first steps.

1. Create an App Service using one of the Quickstarts in the App Service documentation, for example .NET Core Quickstart
2. Create an Application Gateway using the portal Quickstart, but skip the Add backend targets section.
3. Configure App Service as a backend in Application Gateway, but skip the Restrict access section.
4. Finally create the access restriction using service endpoints.

You can now access the App Service through Application Gateway, but if you try to access the App Service directly, you should receive a 403 HTTP error indicating that the web site is stopped.

Using Azure Resource Manager template

The Resource Manager deployment template will provision a complete scenario. The scenario consists of an App Service instance locked down with service endpoints and access restriction to only receive traffic from Application Gateway. The template includes many Smart Defaults and unique postfixes added to the resource names for it to be simple. To override them, you'll have to clone the repo or download the template and edit it.

To apply the template you can use the Deploy to Azure button found in the description of the template, or you can use appropriate PowerShell/CLI.

Using Azure CLI

The Azure CLI sample will provision an App Service locked down with service endpoints and access restriction to only receive traffic from Application Gateway. If you only need to isolate traffic to an existing App Service from an existing Application Gateway, the following command is sufficient.

Azure CLI

```
az webapp config access-restriction add --resource-group myRG --name
myWebApp --rule-name AppGwSubnet --priority 200 --subnet mySubNetName --
vnet-name myVnetName
```

In the default configuration, the command will ensure both setup of the service endpoint configuration in the subnet and the access restriction in the App Service.

Considerations when using private endpoint

As an alternative to service endpoint, you can use private endpoint to secure traffic between Application Gateway and App Service (multi-tenant). You will need to ensure that Application Gateway can DNS resolve the private IP of the App Service apps or alternatively that you use the private IP in the backend pool and override the host name in the http settings.

Application Gateway will cache the DNS lookup results, so if you use FQDNs and rely on DNS lookup to get the private IP address, then you may need to restart the Application Gateway if the DNS update or link to Azure private DNS zone was done after configuring the backend pool. To restart the Application Gateway, you must start and stop the instance. You can do this with Azure CLI:

Azure CLI

```
az network application-gateway stop --resource-group myRG --name myAppGw
az network application-gateway start --resource-group myRG --name myAppGw
```

Considerations for ILB ASE

ILB App Service Environment isn't exposed to the internet and traffic between the instance and an Application Gateway is therefore already isolated to the Virtual Network. The following how-to guide configures an ILB App Service Environment and integrates it with an Application Gateway using Azure portal.

If you want to ensure that only traffic from the Application Gateway subnet is reaching the App Service Environment, you can configure a Network security group (NSG) which affect all web apps in the App Service Environment. For the NSG, you are able to specify the subnet IP range and optionally the ports (80/443). Make sure you don't override the required NSG rules for App Service Environment to function correctly.

To isolate traffic to an individual web app you'll need to use ip-based access restrictions as service endpoints will not work for ASE. The IP address should be the private IP of the Application Gateway instance.

Considerations for External ASE

External App Service Environment has a public facing load balancer like multi-tenant App Service. Service endpoints don't work for App Service Environment, and that's why you'll have to use ip-based access restrictions using the public IP of the Application Gateway instance. To create an External App Service Environment using the Azure portal, you can follow this Quickstart

Considerations for kudu/scm site

The scm site, also known as kudu, is an admin site, which exists for every web app. It isn't possible to reverse proxy the scm site and you most likely also want to lock it down to individual IP addresses or a specific subnet.

If you want to use the same access restrictions as the main site, you can inherit the settings using the following command.

Azure CLI

```
az webapp config access-restriction set --resource-group myRG --name
myWebApp --use-same-restrictions-for-scm-site
```

If you want to set individual access restrictions for the scm site, you can add access restrictions using the --scm-site flag like shown below.

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
```

Next steps

For more information on the App Service Environment, see App Service Environment documentation.

To further secure your web app, information about Web Application Firewall on Application Gateway can be found in the Azure Web Application Firewall documentation.

Azure NAT Gateway integration

Article • 04/21/2023

Azure NAT gateway is a fully managed, highly resilient service, which can be associated with one or more subnets and ensures that all outbound Internet-facing traffic will be routed through the gateway. With App Service, there are two important scenarios that you can use NAT gateway for.

The NAT gateway gives you a static predictable public IP for outbound Internet-facing traffic. It also significantly increases the available SNAT ports in scenarios where you have a high number of concurrent connections to the same public address/port combination.

For more information and pricing. Go to the Azure NAT Gateway overview.

> **Note**
>
> - Using a NAT gateway with App Service is dependent on virtual network integration, and therefore a supported App Service plan pricing tier is required.
> - When using a NAT gateway together with App Service, all traffic to Azure Storage must be using private endpoint or service endpoint.
> - A NAT gateway cannot be used together with App Service Environment v1 or v2.

Configuring NAT gateway integration

To configure NAT gateway integration with App Service, you need to complete the following steps:

- Configure regional virtual network integration with your app as described in Integrate your app with an Azure virtual network
- Ensure Route All is enabled for your virtual network integration so the Internet bound traffic will be affected by routes in your virtual network.
- Provision a NAT gateway with a public IP and associate it with the virtual network integration subnet.

Set up Azure NAT Gateway through the portal:

1. Go to the **Networking** UI in the App Service portal and select virtual network integration in the Outbound Traffic section. Ensure that your app is integrated with a subnet and **Route All** has been enabled.

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 pick the region where your app is located.

5. In the **Outbound IP** tab, create a new or select an existing public IP.

6. In the **Subnet** tab, select the subnet used for virtual network integration.

7. Fill in tags if needed and **Create** the NAT gateway. After the NAT gateway is provisioned, click on the **Go to resource group** and select the new NAT gateway. You can see the public IP that your app will use for outbound Internet-facing traffic in the Outbound IP blade.

If you prefer using CLI to configure your environment, these are the important commands. As a prerequisite, you should create an app with virtual network integration configured.

Ensure **Route All** is configured for your virtual network integration:

Azure CLI

```
az webapp config set --resource-group [myResourceGroup] --name [myWebApp] --
vnet-route-all-enabled
```

Create Public IP and 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
```

Associate the NAT gateway with the virtual network integration subnet:

Azure CLI

```
az network vnet subnet update --resource-group [myResourceGroup] --vnet-name
[myVnet] --name [myIntegrationSubnet] --nat-gateway myNATgateway
```

Scaling a NAT gateway

The same NAT gateway can be used across multiple subnets in the same virtual network allowing a NAT gateway to be used 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) allowing up to 1M available ports. Learn more in the Scaling section of Azure NAT Gateway.

Next steps

For more information on Azure NAT Gateway, see Azure NAT Gateway documentation.

For more information on virtual network integration, see Virtual network integration documentation.

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 • 03/03/2023

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 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 as well as to 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 will be redirected through the Hybrid Connection.

> **Note**

This means that you should try to always use a DNS name for your Hybrid Connection. Some client software does not do a DNS lookup if the endpoint uses an IP address instead.

App Service Hybrid Connection benefits

There are a number of benefits to the Hybrid Connections capability, including:

- Apps can access on-premises systems and services securely.
- The feature 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.
- It's supported in GA for Windows apps and Linux apps. It isn't supported for Windows custom containers.

Things you cannot do with Hybrid Connections

Things you cannot do with Hybrid Connections include:

- Mount a drive.
- Use UDP.
- Access TCP-based services that use dynamic ports, such as FTP Passive Mode or Extended Passive Mode.
- Support LDAP, because it can require UDP.
- Support Active Directory, because you cannot domain join an App Service worker.

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.

To add a new Hybrid Connection, select **[+] Add hybrid connection**. You'll see a list of the Hybrid Connections that you already created. To add one or more of them to your app, select the ones you want, and then select **Add selected Hybrid Connection**.

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.

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, 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.

Pricing plan	Number of Hybrid Connections usable in the plan
Basic	5 per plan
Standard	25 per plan
Premium (v1-v3)	220 per app
Isolated (v1-v2)	220 per app

The App Service plan UI shows you how many Hybrid Connections are being used and by what apps.

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 additional 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**.

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 that.

4. Choose a subscription.

5. Select the Hybrid Connections that you want the HCM to relay.

6. Select **Save**.

You can now see the Hybrid Connections you added. You can also select the configured Hybrid Connection to see details.

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 will show up in the HCM UI. Applying the upgrade will apply the changes and restart the HCM.

Adding a Hybrid Connection to your app programmatically

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 will use. 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 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 additional security measures in place to prevent unauthorized access), you must lock down access to the Azure Service Bus Relay.

Anyone with `Reader` access to the Relay will be able to *see* the Hybrid Connection when attempting to add it to their Web App in the Azure portal, but they will not be able to *add* it as they lack the permissions to retrieve the connection string 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 will allow 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 steps below:

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 -P 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 cannot coexist with Biztalk Hybrid Connection Manager or Service Bus for Windows Server. Hence when installing HCM, any versions of these packages should be removed first.

If your status says **Connected** but your app cannot 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 may not happen. If the client running in your web app doesn't do a DNS lookup, then the Hybrid Connection will not 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 cannot 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 • 06/01/2023

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.

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

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.

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="<primaryConnectionString>"
```

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 publically 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.

> ⓘ **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

Azure Policy built-in definitions for Azure App Service

Article • 09/19/2023

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

Name (Azure portal)	Description	Effect(s)	Version (GitHub)
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 https://aka.ms/appservice-vnet-configuration-routing.	Audit, Deny, Disabled	1.0.0
App Service app slots	By default, if one uses regional Azure Virtual Network (VNET) integration, the app only routes	Audit, Deny, Disabled	1.0.0

Name (Azure portal)	Description	Effect(s)	Version (GitHub)
should enable outbound non-RFC 1918 traffic to Azure Virtual Network	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.		
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.	Audit, 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 Azure Active Directory identities for authentication. Learn more at: https://aka.ms/app-service-disable-basic-auth .	AuditIfNotExists, Disabled	1.0.2
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 Azure Active Directory 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 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	Audit enabling of resource logs on the app. This enables you to recreate activity trails for	AuditIfNotExists, Disabled	1.0.0

Name (Azure portal)	Description	Effect(s)	Version (GitHub)
resource logs enabled	investigation purposes if a security incident occurs or your network is compromised.		
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
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	Periodically, newer versions are released for TLS either due to security flaws, include additional	AuditIfNotExists, Disabled	1.0.0

Name (Azure portal)	Description	Effect(s)	Version (GitHub)
should use the latest TLS version	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.		
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 Linux apps. This policy requires you to specify a Python version that meets your requirements.	AuditIfNotExists, Disabled	1.0.0
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.	Audit, Disabled, Deny	1.1.0

Name (Azure portal)	Description	Effect(s)	Version (GitHub)
	Learn more at: https://aka.ms/app-service-private-endpoint.		
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 '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
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 local authentication methods disabled for	Disabling local authentication methods for FTP deployments improves security by ensuring that App Services exclusively require Azure Active Directory identities for authentication. Learn more at: https://aka.ms/app-service-disable-basic-auth.	AuditIfNotExists, Disabled	1.0.2

Name (Azure portal)	Description	Effect(s)	Version (GitHub)
FTP deployments			
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 Azure Active Directory identities for authentication. Learn more at: https://aka.ms/app-service-disable-basic-auth .	AuditIfNotExists, Disabled	1.0.2
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	Audit, Deny, Disabled	4.1.0

Name (Azure portal)	Description	Effect(s)	Version (GitHub)
	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 .		
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
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	AuditIfNotExists, Disabled	2.0.1

Name (Azure portal)	Description	Effect(s)	Version (GitHub)
	take advantage of security fixes, if any, and/or new functionalities of the latest version.		
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 Linux apps. This policy requires you to specify a PHP version that meets your requirements.	AuditIfNotExists, Disabled	3.2.0
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	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

Name (Azure portal)	Description	Effect(s)	Version (GitHub)
TLS Cipher suites			
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 Azure Active Directory identities for authentication. Learn more at: https://aka.ms/app-service-disable-basic-auth .	DeployIfNotExists, Disabled	1.0.2
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 Azure Active Directory identities for authentication. Learn more at: https://aka.ms/app-service-disable-basic-auth .	DeployIfNotExists, Disabled	1.0.2
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	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)
slots to only be accessible over HTTPS	network layer eavesdropping attacks.		
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	1.1.0
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 Azure Active Directory identities for authentication. Learn more at: https://aka.ms/app-service-disable-basic-auth.	DeployIfNotExists, Disabled	1.0.2
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 Azure Active Directory identities for authentication. Learn more at: https://aka.ms/app-service-disable-basic-auth.	DeployIfNotExists, Disabled	1.0.2
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 network layer eavesdropping attacks.	Modify, Disabled	2.0.0
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.	DeployIfNotExists, Disabled	1.0.0

Name (Azure portal)	Description	Effect(s)	Version (GitHub)
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.	DeployIfNotExists, Disabled	1.0.1
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.	DeployIfNotExists, 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.	DeployIfNotExists, Disabled	1.1.0
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 requires inbound ports to be opened on Function apps. Remote debugging should be turned off.	DeployIfNotExists, Disabled	1.0.0

Name (Azure portal)	Description	Effect(s)	Version (GitHub)
remote debugging			
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.	DeployIfNotExists, Disabled	1.0.1
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
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 with valid certificates will be able to reach the app.	Audit, 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

Name (Azure portal)	Description	Effect(s)	Version (GitHub)
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	1.0.0
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
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

Name (Azure portal)	Description	Effect(s)	Version (GitHub)
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 '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 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 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	The content directory of a Function app should be located on an Azure file share. The storage	Audit, Disabled	3.0.0

Name (Azure portal)	Description	Effect(s)	Version (GitHub)
Azure file share for its content directory	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 .		
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	2.0.1
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

Next steps

- See the built-ins on the Azure Policy GitHub repo.
- Review the Azure Policy definition structure.
- Review Understanding policy effects.

Azure App Service plan overview

Article • 05/26/2023

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. There are the following categories of pricing tiers:

- **Shared compute**: **Free** and **Shared**, the two base tiers, runs an app on the same Azure VM as other App Service apps, including apps of other customers. These tiers allocate CPU quotas to each app that runs on the shared resources, and the resources cannot scale out. 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 are available to you for scale-out.
- **Isolated**: The **Isolated** and **IsolatedV2** tiers run dedicated Azure VMs on dedicated Azure Virtual Networks. It provides network isolation on top of compute isolation to your apps. It provides the maximum scale-out capabilities.

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 are available. To

find out which features are supported in each pricing tier, see App Service plan details.

Premium V3 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 **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** tier or above gives 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 is as simple as changing the pricing tier of the plan. You can choose a lower pricing tier at first and scale up later when you need more App Service features.

For example, you can start testing your web app in a **Free** App Service plan and pay nothing. When you 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 the reverse. When you feel you no longer need the capabilities or features of a higher tier, you can scale down to a lower tier, which saves you money.

For information on scaling up the App Service plan, see Scale up an app in Azure.

If your app is in the same App Service plan with other apps, you may want to improve the app's performance by isolating the compute resources. You can do it by moving the app into a separate App Service plan. For more information, see Move an app to another App Service plan.

Should I put an app in a new plan or an existing plan?

Since you pay for the computing resources your App Service plan allocates (see How much does my App Service plan cost?), you can potentially save money by putting

multiple apps into one App Service plan. You can continue to add apps to an existing plan as long as the plan has enough resources to handle the load. However, keep in mind that apps in the same App Service plan all share the same compute resources. To determine whether the new app has the necessary resources, you need to understand the capacity of the existing App Service plan, and the expected load for the new app. Overloading an App Service plan can potentially cause downtime for your new and existing apps.

Isolate your app into a new App Service plan when:

- The app is resource-intensive. The number may actually be lower depending on how resource intensive the hosted applications are, however as a general guidance, you may refer to the table below:

App Service Plan SKU	Max 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	Max density bounded by vCPU usage
P3mv3, P4mv3, P5mv3	Max density bounded by vCPU usage

- You want to scale the app independently from the other apps in the existing plan.
- The app needs resource in a different geographical region.

> **Note**
>
> An active slot is also classified as an active app as it too is competing for resources on the same App Service Plan.

This way you can allocate a new set of resources for your app and gain greater control of your apps.

Next steps

Manage an App Service plan

Manage an App Service plan in Azure

Article • 08/02/2023

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, browse to https://ms.portal.azure.com/#create/Microsoft.AppServicePlanCreate .

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, then select the **Operating System** and **Region**. Region determines 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 creating an 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 this error 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, geographical region, and of the same OS type*. Any change in type such as Windows to Linux or any type that is different from the originating type is not 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**.

3. In the **App Service plan** dropdown, select an existing plan to move the app to. The dropdown shows only plans that are in the same resource group and geographical region as the current App Service plan. If no such plan exists, it lets you create a plan by default. You can also create a new plan manually by selecting **Create new**.

4. If you create a plan, you can select the pricing tier of the new plan. In **Pricing Tier**, select the existing tier to change it.

> **Important**
>
> If you're moving an app from a higher-tiered plan to a lower-tiered plan, such as from **D1** to **F1**, the app may lose certain capabilities in the target plan. For

example, if your app uses 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 cannot change an App Service plan's region. If you want to run your app in a different region, one alternative is app cloning. Cloning makes a copy of your app in a new or existing App Service plan in any region.

You can find **Clone App** in the **Development Tools** section of the menu.

> **Important**
>
> Cloning has some limitations. You can read about them in **Azure App Service App cloning**.

Scale an App Service plan

To scale up an App Service plan's pricing tier, see Scale up an app in Azure.

To scale out an app's instance count, see Scale instance count manually or automatically.

Delete an App Service plan

To avoid unexpected charges, when you delete the last app in an App Service plan, App Service also deletes the plan by default. If you choose to keep the plan instead, you should change the plan to **Free** tier so you're not charged.

> **Important**
>
> App Service plans that have no apps associated with them still incur charges because they continue to reserve the configured VM instances.

Next steps

Scale up an app in Azure

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 (instan
Popular options					
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
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)**.

Select one of the Premium V3 plans and select **Select**.

Name	ACU/vCPU	vCPU	Memory (GB)	Remote Storage (GB)	Scale (instan
Popular options					
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
Dev/Test (For less demanding workloads)					

If your operation finishes successfully, your app's overview page shows that it's now in a Premium V3 tier.

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 • 09/20/2023

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 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 are supported in **Basic**, **Standard**, **Premium**, and **Isolated** tiers. For **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 target app of the same OS platform as the source app.

Back up & restore vs. disaster recovery

Platform	Back up & restore guidance	Disaster recovery guidance
App Service Web Apps (Free & Shared Pricing Tier)	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 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 region as explained in the recover from region-wide failure article. It's recommended to

Platform	Back up & restore guidance	Disaster recovery guidance
		implement commonly used disaster recovery techniques to prevent downtime and data loss during a regional disaster.
App Service Web Apps (Basic\Standard\Premium Pricing Tier)	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 by restoring to a new app or slot. Refer to Back up and restore your app in Azure App Service for more info.	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. 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.
App Service Environment (V2 & V3)	In Azure App Service Environment, You can make on-demand custom backups or utilize automatic backups. Automatic backups can be restored to a target app within the same ASE, not in another ASE. Custom backups can be restored to a target app in another ASE (such as from a V2 ASE to a V3 ASE). Backups can be restored to target app of the same OS platform as the source app. Refer to Back up and restore your app in Azure App Service for more details.	We encourage you to implement disaster recovery guidance for web apps deployed to App Service Environment using commonly used disaster recovery techniques.
Azure Functions (Dedicated)	In Azure Functions, You can make on-demand custom backups or utilize automatic	Current guidance regarding how to bring Azure Functions app (dedicated) resources back online

Platform	Back up & restore guidance	Disaster recovery guidance
	backups. You can restore a backup by overwriting an existing app by restoring to a new app or slot. Refer to Back up and restore your app in Azure App Service for more info.	in a different Azure region during a regional disaster is available at Recover from region-wide failure - Azure App Service. 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. Instead, implement Azure Functions geo-disaster recovery. Additionally you may also refer to commonly used disaster recovery techniques for Azure Functions dedicated.
Azure Functions Consumption & Premium	Azure functions deployed to consumption and premium plans don't provide access to custom and automatic backups. Function app's content is in an Azure storage account. Use Azure Storage redundancy options to ensure your storage account meets its availability and durability targets during an outage. If you created your functions by using the editor in the Azure portal, you can also download your existing function app project as a .zip file.	We strongly encourage you to implement Azure Functions geo-disaster recovery and reliability.

Automatic vs. custom backups

There are two types of backups in App Service. Automatic backups made 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.

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 3 hourly backup retained. - Days 15-30: every 6 hourly backup retained.	0-30 days or indefinite.
Downloadable	No.	Yes, as Azure Storage blobs.
Partial backups	Not supported.	Supported.
Back up over VNet	Not supported.	Supported.

Restore a backup

> **Note**
>
> App Service stops the target app or target slot while restoring a backup. To minimize downtime for the production app, restore the backup to a **deployment slot** first, then **swap** into production.

Azure portal

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.

2. Select the automatic backup or custom backup to restore by clicking its **Restore** link.

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. Click **Restore**.

Create a custom backup

1. In your app management page in the Azure portal, in the left menu, select **Backups**.

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 with **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 backed up always without any configuration. If you make settings for in-app MySQL databases manually, such as adding connection strings, the backups may not work correctly.

4. Click **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. In the **Configure custom backups** page, select **Set schedule**.
2. Configure the backup schedule as desired and select **Configure**.

Back up and restore a linked database

Custom backups can include linked databases (except when the backup is configured over an 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 (preview)

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 has granted 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 you have fulfilled the aforementioned requirements.

Once the configuration is saved, any manual, 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 will fail.

Configure partial backups

Partial backups are supported for custom backups (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, you want to filter out the highlighted folder and files.

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. So the content of the file should be:

```
\site\wwwroot\Images\brand.png
\site\wwwroot\Images\2014
\site\wwwroot\Images\2013
```

Upload `_backup.filter` file to the `D:\home\site\wwwroot\` directory of your site using ftp or any other method. If you wish, you can create the file directly using Kudu `DebugConsole` and insert the content there.

Run backups the same way you would normally do it, 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 have made one or more backups for your app, the backups are visible on the **Containers** page of your storage account, and your app. In the storage account, each

backup consists of a `.zip` file that contains the backup data and an `.xml` file that contains a manifest of the `.zip` file contents. You can unzip and browse these files if you want to access your backups without actually performing an app restore.

The database backup for the app is stored in the root of the .zip file. For SQL Database, this is a BACPAC file (no file extension) and can be imported. To create a database in Azure SQL Database based on the BACPAC export, see Import a BACPAC file to create a database in Azure SQL Database.

> ⚠ **Warning**
>
> Altering any of the files in your **websitebackups** container can cause the backup to become invalid and therefore non-restorable.

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.

Error	Fix
Storage access failed.	Delete backup schedule and reconfigure it. Or, reconfigure the backup storage.
The website + database size exceeds the {0} GB limit for backups. Your content size is {1} GB.	Exclude some files from the backup, or remove the database portion of the backup and use externally offered backups instead.
Error occurred while connecting to the database {0} on server {1}: Authentication to host '{1}' for user '<username>' using method 'mysql_native_password' failed with message: Unknown database '<db-name>'	Update database connection string.
Cannot resolve {0}. {1} (CannotResolveStorageAccount)	Delete the backup schedule and reconfigure it.
Login failed for user '{0}'.	Update the database connection string.

Error	Fix
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).	Check that the connection string is valid. Allow the app's outbound IPs in the database server settings.
Cannot open server "<name>" requested by the login. The login failed.	Check that the connection string is valid.
Missing mandatory parameters for valid Shared Access Signature.	Delete the backup schedule and reconfigure it.
SSL connection is required. Please specify SSL options and retry when trying to connect.	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, using the 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 the 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** or **Standard** or **Premium** tiers. Function apps in the **Consumption** or **Elastic Premium** pricing tiers aren't supported for automatic backups.

What's included in an automatic backup?

The following table shows which content is backed up in an automatic backup:

Settings	Restored?
Windows apps: All app content under `%HOME%` directory **Linux apps**: All app content under `/home` 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 configuration is restored when you choose to restore app configuration:

Settings	Restored?
Native log settings, including the Azure Storage account and container settings	Yes
Application Insights configuration	Yes
Health check	Yes

Settings	Restored?
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 an 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:

- Backups of TLS enabled Azure Database for MySQL isn't supported. If a backup is configured, you'll encounter backup failures.
- Backups of TLS enabled Azure Database for PostgreSQL isn't supported. If a backup is configured, you'll encounter 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 may 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 (preview).

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. In 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 like 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 simple and stored in the same datacenter as the App Service and should not be relied upon as your disaster recovery plan.

How do I stop the automatic backup?

You cannot stop automatic backup. The automatic backup is stored on the platform and has no effect on the underlying app instance or its storage.

Next Steps

Azure Blob Storage documentation

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 • 04/06/2023

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.
> - App Service apps running in an App Service Environment don't support snapshots. Therefore, undelete functionality and clone functionality aren't supported for App Service apps running in an App Service Environment.

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:

- **DeletedSiteId**: Unique identifier for the app, used for scenarios where multiple apps with the same name have been deleted
- **SubscriptionID**: Subscription containing the deleted resource
- **Location**: Location of the original app
- **ResourceGroupName**: Name of the original resource group
- **Name**: Name of the original app.
- **Slot**: the name of the slot.
- **Deletion Time**: When was the app deleted

Restore deleted app

> ⓘ **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.

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>
```

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/00000000-0000-0000-0000-
000000000000/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
- **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.

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.

Move an App Service resource to another region

Article • 02/01/2023

This article describes how to move App Service resources to a different Azure region. You might move your resources to another region for a number of reasons. For example, to take advantage of a new Azure region, to deploy features or services available in specific regions only, to meet internal policy and governance requirements, or in response to capacity planning requirements.

App Service resources are region-specific and can't be moved across regions. You must create a copy of your existing App Service resources in the target region, then move your content over to the new app. If your source app uses a custom domain, you can migrate it to the new app in the target region when you're finished.

To make copying your app easier, you can clone an individual App Service app into an App Service plan in another region, but it does have limitations, especially that it doesn't support Linux apps.

Prerequisites

- Make sure that the App Service app is in the Azure region from which you want to move.
- Make sure that the target region supports App Service and any related service, whose resources you want to move.

Prepare

Identify all the App Service resources that you're currently using. For example:

- App Service apps
- App Service plans
- Deployment slots
- Custom domains purchased in Azure
- 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.

Move

1. Create a back up of the source app.
2. Create an app in a new App Service plan, in the target region.
3. Restore the back up in the target app
4. If you use a custom domain, bind it preemptively to the target app with `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.

Clean up source resources

Delete the source app and App Service plan. An App Service plan in the non-free tier carries a charge, even if no app is running in it.

Next steps

Azure App Service App Cloning Using PowerShell

Move resources to a new resource group or subscription

Article • 04/24/2023

This article shows you how to move Azure resources to either another Azure subscription or another resource group under the same subscription. You can use the Azure portal, Azure PowerShell, Azure CLI, or the REST API to move resources.

Both the source group and the target group are locked during the move operation. Write and delete operations are blocked on the resource groups until the move completes. This lock means you can't add, update, or delete resources in the resource groups. It doesn't mean the resources are frozen. For example, if you move an Azure SQL logical server, its databases and other dependent resources to a new resource group or subscription, applications that use the databases experience no downtime. They can still read and write to the databases. The lock can last for a maximum of four hours, but most moves complete in much less time.

If your move requires setting up new dependent resources, you'll experience an interruption in those services until they've been reconfigured.

Moving a resource only moves it to a new resource group or subscription. It doesn't change the location of the resource.

> **Note**
>
> This article was partially created with the help of artificial intelligence. Before publishing, an author reviewed and revised the content as needed. See **Our principles for using AI-generated content in Microsoft Learn**.

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'll need to change that value. For example, if you have a custom dashboard in the portal that references a resource ID, you'll 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

There are some important steps to do before moving a resource. By verifying these conditions, you can avoid errors.

1. The source and destination subscriptions must be active. If you have trouble enabling an account that has been disabled, create an Azure support request. Select **Subscription Management** for the issue type.

2. The source and destination subscriptions must exist within the same Azure Active Directory tenant. To check that both subscriptions have the same tenant ID, use Azure PowerShell or Azure CLI.

   For Azure PowerShell, use:

   Azure PowerShell

   ```
   (Get-AzSubscription -SubscriptionName <your-source-subscription>).TenantId
   (Get-AzSubscription -SubscriptionName <your-destination-subscription>).TenantId
   ```

   For Azure CLI, use:

   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 aren't the same, use the following methods to reconcile the tenant IDs:

   - Transfer ownership of an Azure subscription to another account
   - How to associate or add an Azure subscription to Azure Active Directory

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.

4. The resources you want to move must support the move operation. For a list of which resources support move, see Move operation support for resources.

5. Some services have specific limitations or requirements when moving resources. If you're moving any of the following services, check that guidance before moving.

    - If you're using Azure Stack Hub, you can't move resources between groups.
    - App Services move guidance
    - Azure DevOps Services move guidance
    - Classic deployment model move guidance - Classic Compute, Classic Storage, Classic Virtual Networks, and Cloud Services
    - Cloud Services (extended support) move guidance
    - Networking move guidance
    - Recovery Services move guidance
    - Virtual Machines move guidance
    - To move an Azure subscription to a new management group, see Move subscriptions.

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.

    For PowerShell, use the following commands to get the registration status:

    Azure PowerShell

    ```
    Set-AzContext -Subscription <destination-subscription-name-or-id>
    Get-AzResourceProvider -ListAvailable | Select-Object
    ProviderNamespace, RegistrationState
    ```

    To register a resource provider, use:

    Azure PowerShell

    ```
    Register-AzResourceProvider -ProviderNamespace Microsoft.Batch
    ```

    For Azure CLI, use the following commands 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, use:

Azure CLI

```
az provider register --namespace Microsoft.Batch
```

7. Before moving the resources, check the subscription quotas for the subscription you're moving the resources to. If moving the resources means the subscription will exceed its limits, you need to review whether you can request an increase in the quota. For a list of limits and how to request an increase, see Azure subscription and service limits, quotas, and constraints.

8. The account moving the resources must have at least the following permissions:

   - **Microsoft.Resources/subscriptions/resourceGroups/moveResources/action** on the source resource group.
   - **Microsoft.Resources/subscriptions/resourceGroups/write** on the destination resource group.

9. If you move a resource that has an Azure role assigned directly to the resource (or a child resource), the role assignment isn't moved and becomes orphaned. After the move, you must re-create the role assignment. Eventually, the orphaned role assignment is automatically removed, but we recommend removing the role assignment before the move.

   For information about how to manage role assignments, see List Azure role assignments and Assign Azure roles.

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 VM with managed disks would require the VM and the managed disks to be moved together, along with other dependent resources.

    If you're moving a resource to a new subscription, check to see whether the resource has any dependent resources, and whether they're located in the same resource group. If the resources aren't in the same resource group, check to see whether the resources can be combined into the same resource group. If so, bring all these resources into the same resource group by using a move operation across resource groups.

    For more information, see Scenario for move across subscriptions.

Scenario for move across subscriptions

Moving resources from one subscription to another is a three-step process:

For illustration purposes, we have only one dependent resource.

- Step 1: If dependent resources are distributed across different resource groups, first move them into one resource group.
- Step 2: Move the resource and dependent resources together from the source subscription to the target subscription.
- Step 3: Optionally, redistribute the dependent resources to different resource groups within the target subscription.

Use the portal

To move resources, select the resource group that contains those resources.

Select the resources you want to move. To move all of the resources, select the checkbox at the top of list. Or, select resources individually.

Select the **Move** button.

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).

Select whether you're moving the resources to a new resource group or a new subscription.

The source resource group is automatically set. Specify the destination resource group. If you're moving to a new subscription, also specify the subscription. Select **Next**.

The portal validates that the resources can be moved. Wait for validation to complete.

When validation completes successfully, select **Next**.

Acknowledge that you need to update tools and scripts for these resources. To start moving the resources, select **Move**.

When the move has completed, you're notified of the result.

Use Azure PowerShell

Validate

To test your move scenario without actually moving the resources, use the Invoke-AzResourceAction command. Use this command only when you need to predetermine the results.

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;targetResourceGroup =
$destinationResourceGroup.ResourceId }
```

If validation passes, you see no output.

If validation fails, you see an error message describing why the resources can't be moved.

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 Azure CLI

Validate

To test your move scenario without actually moving the resources, use the az resource invoke-action command. Use this command only when you need to predetermine the results. To run this operation, you need the:

- Resource ID of the source resource group
- Resource ID of the target resource group
- Resource ID of each resource to move

In the request body, use `\"` to escape double quotes.

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 describing why the resources can't be moved.

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 example shows how to move several resources to a new resource group. It works when using Azure CLI in a **Bash** terminal.

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
```

The next example shows how to run the same commands in a **PowerShell** console.

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
```

To move to a new subscription, provide the `--destination-subscription-id` parameter.

Use Python

Validate

To test your move scenario without actually moving the resources, use the ResourceManagementClient.resources.begin_validate_move_resources method. Use this method only when you need to predetermine the results.

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))
```

If validation passes, you see no output.

If validation fails, you see an error message describing why the resources can't be moved.

Move

To move existing resources to another resource group or subscription, use the ResourceManagementClient.resources.begin_move_resources method. 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 the resources. Use this operation to check if the move will succeed. Validation is automatically called when you send a move request. Use this operation only when you need to predetermine the results. To run this operation, you need the:

- Name of the source resource group
- Resource ID of the target resource group
- Resource ID of each resource to move
- The access token for your account

Send the following request:

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
...
```

The 202 status code indicates the validation request was accepted, but it hasn't yet determined if the move operation will succeed. The `location` value contains a URL that you use to check the status of the long-running operation.

To check the status, send the following request:

```HTTP
GET <location-url>
Authorization: Bearer <access-token>
```

While the operation is still running, you continue to receive the 202 status code. Wait the number of seconds indicated in the `retry-after` value before trying again. If the move operation validates successfully, you receive the 204 status code. If the move validation fails, you receive an error message, such as:

```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}
```

In the request body, you specify the target resource group and the resources to move.

JSON

```
{
 "resources": ["<resource-id-1>", "<resource-id-2>"],
 "targetResourceGroup": "/subscriptions/<subscription-id>/resourceGroups/<target-group>"
}
```

Frequently asked questions

Question: My resource move operation, which usually takes a few minutes, has been running for almost an hour. Is there something wrong?

Moving a resource is a complex operation that has different phases. It can involve more than just the resource provider of the resource you're trying to move. Because of the dependencies between resource providers, Azure Resource Manager allows 4 hours for the operation to complete. This time period gives resource providers a chance to recover from transient issues. If your move request is within the four-hour period, the operation keeps trying to complete and may still succeed. The source and destination resource groups are locked during this time to avoid consistency issues.

Question: Why is my resource group locked for four hours during resource move?

A move request is allowed a maximum of four hours to complete. To prevent modifications on the resources being moved, both the source and destination resource groups are locked during the resource move.

There are two phases in a move request. In the first phase, the resource is moved. In the second phase, notifications are sent to other resource providers that are dependent on the resource being moved. A resource group can be locked for the entire four hours when a resource provider fails either phase. During the allowed time, Resource Manager retries the failed step.

If a resource can't be moved within four hours, Resource Manager unlocks both resource groups. Resources that were successfully moved are in the destination resource group. Resources that failed to move are left the source resource group.

Question: What are the implications of the source and destination resource groups being locked during the resource move?

The lock prevents you from deleting either resource group, creating a new resource in either resource group, or deleting any of the resources involved in the move.

The following image shows an error message from the Azure portal when a user tries to delete a resource group that is part of an ongoing move.

Question: What does the error code "MissingMoveDependentResources" mean?

When you move a resource, its dependent resources must either exist in the destination resource group or subscription, or be included in the move request. You get the MissingMoveDependentResources error code when a dependent resource doesn't meet this requirement. The error message has details about the dependent resource that needs to be included in the move request.

For example, moving a virtual machine could require moving seven resource types with three different resource providers. Those resource providers and types are:

- Microsoft.Compute
  - virtualMachines
  - disks
- Microsoft.Network
  - networkInterfaces
  - publicIPAddresses
  - networkSecurityGroups
  - virtualNetworks
- Microsoft.Storage
  - storageAccounts

Another common example involves moving a virtual network. You may have to move several other resources associated with that virtual network. The move request could require moving public IP addresses, route tables, virtual network gateways, network security groups, and others. In general, a virtual network gateway must always be in the same resource group as its virtual network, they can't be moved separately.

Question: What does the error code "RequestDisallowedByPolicy" mean?

Resource Manager validates your move request before attempting the move. This validation includes checking policies defined on 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 the creation of a key vault in the target resource group, validation fails and the move is blocked. The returned error code is **RequestDisallowedByPolicy**.

For more information about policies, see What is Azure Policy?.

Question: Why can't I move some resources in Azure?

Currently, not all resources in Azure support move. For a list of resources that support move, see Move operation support for resources.

Question: How many resources can I move in a single operation?

When possible, break large moves into separate move operations. Resource Manager immediately returns an error when there are more than 800 resources in a single operation. However, moving less than 800 resources may also fail by timing out.

Question: What is the meaning of the error that a resource isn't in succeeded state?

When you get an error message that indicates a resource can't be moved because it isn't in a succeeded state, it may actually be a dependent resource that 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 dependent resources for the virtual network are checked during the move. The check includes those resources directly and indirectly dependent on the virtual network. If any of those resources are in a failed state, the move is blocked. For example, if a virtual machine that uses the virtual network has failed, the move is blocked. The move is blocked even when the virtual machine isn't one of the resources being moved and isn't in one of the resource groups for the move.

When you receive this error, you have two options. Either move your resources to a resource group that doesn't have a virtual network, or contact support.

Next steps

For a list of which resources support move, see Move operation support for resources.

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.

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.

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 • 09/14/2023

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've never used Azure App Service, follow the custom container quickstart and tutorial first. There's also a multi-container app quickstart and tutorial.

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 login 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 ACR using managed identity. The steps will 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, which you use shortly.

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 will 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 this is configured on the web app and then set an additional `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 are all set, and the web app will now use 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 will need to be connected to a virtual network using the virtual network integration feature. This 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 may 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 have been 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 additional instances. There are also rare cases where the app instances may 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 may 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` will contribute to the storage space quota included with your App Service Plan.

When persistent storage is disabled, then writes to the `/home` directory are not 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 will always persist 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 will not affect the application logging behavior.

It is recommended to write data to `/home` or a mounted Azure storage path. Data written outside these paths will not be persistent during restarts and will be saved to platform-managed host disk space separate from the App Service Plans file storage quota.

By default, persistent storage is *enabled* on Linux custom containers. To disable it, set the `WEBSITES_ENABLE_APP_SERVICE_STORAGE` app setting value to `false` 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=false
```

In PowerShell:

Azure PowerShell

```
Set-AzWebApp -ResourceGroupName <group-name> -Name <app-name> -AppSettings
@{"WEBSITES_ENABLE_APP_SERVICE_STORAGE"=false}
```

> **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 will always be 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. Please replace the last command according to the project language/stack:

Debian

Bash

```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. The same will 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

```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 is 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 is not 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.

For further troubleshooting additional information is available at the Azure App Service OSS 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

- 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 are not supported:

- Authentication / Authorization
- Managed Identities
- CORS
- Virtual network integration is not 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 cannot 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: Multi-container WordPress app

Or, see additional resources:

- Environment variables and app settings reference
- Load certificate in Windows/Linux containers

Continuous deployment with custom containers in Azure App Service

Article • 11/18/2022

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

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.

For other private registries, your 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 Azure Active Directory.

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> \
                            --sdk-auth
```

> **Important**
>
> For security, grant the minimum required access to the service principal. The scope in the previous example is limited to the specific App Service app and not the entire resource group.

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

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

Article • 05/11/2023

> **Note**
>
> Multi-container is in preview.

Web App for Containers provides a flexible way to use Docker images. This quickstart shows how to deploy a multi-container app (preview) to Web App for Containers in the Cloud Shell using a Docker Compose configuration.

If you don't have an Azure subscription, create an Azure free account before you begin.

Prerequisites

- 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.

This article requires version 2.0.32 or later of the Azure CLI. If using Azure Cloud Shell, the latest version is already installed.

Download the sample

For this quickstart, you use the compose file from Docker. The configuration file can be found at Azure Samples.

yml

```yml
version: '3.3'

services:
   db:
     image: mysql:5.7
     volumes:
       - db_data:/var/lib/mysql
     restart: always
     environment:
       MYSQL_ROOT_PASSWORD: somewordpress
       MYSQL_DATABASE: wordpress
       MYSQL_USER: wordpress
       MYSQL_PASSWORD: wordpress

   wordpress:
     depends_on:
```

```
        - db
      image: wordpress:latest
      ports:
        - "8000:80"
      restart: always
      environment:
        WORDPRESS_DB_HOST: db:3306
        WORDPRESS_DB_USER: wordpress
        WORDPRESS_DB_PASSWORD: wordpress
volumes:
    db_data:
```

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

Bash

```bash
mkdir quickstart

cd $HOME/quickstart
```

Next, run the following command to clone the sample app repository to your quickstart directory. Then change to the `multicontainerwordpress` directory.

Bash

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

cd multicontainerwordpress
```

Create a resource group

A resource group is a logical container into which Azure resources, 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 *South Central US* location. To see all supported locations for App Service on Linux in **Standard** tier, run the az appservice list-locations --sku S1 --linux-workers-enabled command.

Azure CLI

```bash
az group create --name myResourceGroup --location "South Central US"
```

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

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

Create an Azure App Service plan

In the Cloud Shell, create an App Service plan in the resource group with the az appservice plan create command.

The following example creates an App Service plan named `myAppServicePlan` in the **Standard** pricing tier (`--sku S1`) and in a Linux container (`--is-linux`).

Azure CLI

```
az appservice plan create --name myAppServicePlan --resource-group
myResourceGroup --sku S1 --is-linux
```

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

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

Create a Docker Compose app

> **Note**
>
> Docker Compose on Azure App Services currently has a limit of 4,000 characters when converted to Base64 at this time.

In your Cloud Shell terminal, create a multi-container web app in the `myAppServicePlan` App Service plan with the az webapp create command. Don't forget to replace *<app_name>* with a unique app name (valid characters are `a-z`, `0-9`, and `-`).

Azure CLI

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

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

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

Browse to the app

Browse to the deployed app at (`http://<app_name>.azurewebsites.net`). The app may take a few minutes to load. If you receive an error, allow a few more minutes then refresh the browser.

Congratulations, you've created a multi-container app in Web App for Containers.

Clean up deployment

After the sample script has been run, the following command can be used to remove the resource group and all resources associated with it.

Azure CLI

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

Next steps

Tutorial: Multi-container WordPress app

Configure a custom container

Secure with custom domain and certificate

Deploy a custom container to App Service using GitHub Actions

Article • 11/29/2022

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:

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 do not 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
```

```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

```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 Login 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 is 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:

Parameter	Explanation
app-name	(Required) Name of the App Service app
publish-profile	(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
slot-name	(Optional) Enter an existing Slot other than the Production slot
package	(Optional) Applies to Web App only: Path to package or folder. *.zip, *.war, *.jar or a folder to deploy

Parameter	Explanation
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

```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 login
- Azure WebApp
- Docker login/logout
- Events that trigger workflows
- K8s deploy
- Starter Workflows

Migrate custom software to Azure App Service using a custom container

Article • 02/01/2023

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.

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 image into App Service using a managed identity
- ✓ Access diagnostic logs
- ✓ Enable CI/CD from Azure Container Registry to App Service
- ✓ Connect to the container using SSH

Completing this tutorial incurs a small charge in your Azure account for the container registry and can incur more costs for hosting 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, select **Clone**, and then select **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

The file in the sample named *Dockerfile* that 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 create an SSH server for secure 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 (see Data transfer)**. 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 followed by 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 CR-LF 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 the web app and container are functioning correctly.

I. Create a user-assigned managed identity

App Service can either use 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 with the az acr create command and 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 a set of administrative credentials.

2. Retrieve the administrative credentials by running the az 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 step.

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 is free for the first month. You can control the tier with 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, which 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:

- The sample container is listening on port 8000 for web requests, and you configure the app to send requests to port 8000.
- Tell your app to use the managed identity to pull images from your container registry.
- Configure continuous deployment from the container registry (or, 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.

For more information on this environment variable, see the readme in the sample's GitHub repository.

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 the message after the webhook ping: `Starting container for site`, 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`, replacing `<app-name>` with the name of your web app.

On first access, 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, type `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 the 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 secure communication between a container and a client. To enable SSH connection to your container, your custom image must be configured 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 SSH. SCM/KUDU will use your Azure Portal credentials. Changing this value will result in an error when using 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 SSH connection to 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 it 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 image into App Service using a managed identity
- ✓ Access diagnostic logs
- ✓ Enable CI/CD from Azure Container Registry to App Service
- ✓ Connect to the container using SSH

In the next tutorial, you learn how to secure your app with a custom domain and certificate.

Secure with custom domain and certificate

Or, check out other resources:

Configure custom container

Tutorial: Multi-container WordPress app

Tutorial: Use GitHub Actions to deploy to an App Service custom container and connect to a database

Article • 01/09/2023

This tutorial walks you through setting up a GitHub Actions workflow to deploy a containerized ASP.NET Core application with an Azure SQL Database backend. When you're finished, you have an ASP.NET app running in Azure and connected to SQL Database. You'll first create Azure resources with an ARM template GitHub Actions workflow.

In this tutorial, you learn how to:

- ✓ Use a GitHub Actions workflow to add resources to Azure with a Azure Resource Manager template (ARM template)
- ✓ Use a GitHub Actions workflow to build a container with the latest web app changes

If you don't have an Azure subscription, create an Azure free account before you begin.

Prerequisites

To complete this tutorial, you'll need:

- 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.
  - A GitHub repository to store your Resource Manager templates and your workflow files. To create one, see Creating a new repository.

Download the sample

Fork the sample project in the Azure Samples repo.

```
https://github.com/Azure-Samples/dotnetcore-containerized-sqldb-ghactions/
```

Create the resource group

Open the Azure Cloud Shell at https://shell.azure.com. You can alternately use the Azure CLI if you've installed it locally. (For more information on Cloud Shell, see the Cloud Shell Overview.)

Azure CLI

```
az group create --name {resource-group-name} --location {resource-group-location}
```

Generate deployment credentials

OpenID Connect is an authentication method that uses short-lived tokens. Setting up OpenID Connect with GitHub Actions is more complex process that offers hardened security.

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 will output JSON with an `appId` that is your `client-id`. Save the value to use as the `AZURE_CLIENT_ID` GitHub secret later.

   You'll 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` and will be used 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 will be tied to your default subscription. Replace `$subscriptionId` with your subscription ID, `$resourceGroupName` with your resource group name, and `$assigneeObjectId` with the generated `assignee-object-id`. Learn how to manage Azure subscriptions with the Azure CLI.

   Azure CLI

   ```
   az role assignment create --role contributor --scope
   /subscriptions/$subscriptionId/resourceGroups/$resourceGroupName --
   subscription $subscriptionId --assignee-object-id  $assigneeObjectId --
   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 **objectId (generated while creating app)** for your Active Directory application.
   - Set a value for `CREDENTIAL-NAME` to reference later.
   - Set the `subject`. The value of this is defined by GitHub depending on your workflow:
     - 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 rest --method POST --uri
   'https://graph.microsoft.com/beta/applications/<APPLICATION-OBJECT-
   ID>/federatedIdentityCredentials' --body '{"name":"<CREDENTIAL-
   NAME>","issuer":"https://token.actions.githubusercontent.com","subject"
   :"repo:organization/repository:ref:refs/heads/main","description":"Test
   ing","audiences":["api://AzureADTokenExchange"]}'
   ```

To learn how to create a Create an active directory application, service principal, and federated credentials in Azure portal, see Connect GitHub and Azure.

Configure the GitHub secret for authentication

You need to provide your application's **Client ID**, **Tenant ID**, and **Subscription ID** to the 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. In GitHub, go to your repository.
2. Go to **Settings** in the navigation menu.
3. Select **Security** > **Secrets and variables** > **Actions**.

4. Select **New repository secret**.
5. 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:

GitHub Secret	Active Directory Application
AZURE_CLIENT_ID	Application (client) ID
AZURE_TENANT_ID	Directory (tenant) ID
AZURE_SUBSCRIPTION_ID	Subscription ID

6. Save each secret by selecting **Add secret**.

Add a SQL Server secret

Create a new secret in your repository for `SQL_SERVER_ADMIN_PASSWORD`. This secret can be any password that meets the Azure standards for password security. You won't be able to access this password again so save it separately.

Create Azure resources

The create Azure resources workflow runs an ARM template to deploy resources to Azure. The workflow:

- Checks out source code with the Checkout action.
- Logs into Azure with the Azure Login action and gathers environment and Azure resource information.
- Deploys resources with the Azure Resource Manager Deploy action.

To run the create Azure resources workflow:

1. Open the `azuredeploy.yaml` file in `.github/workflows` within your repository.

2. Update the value of `AZURE_RESOURCE_GROUP` to your resource group name.

3. Update the values of `WEB_APP_NAME` and `SQL_SERVER_NAME` to your web app name and sql server name.

4. Go to **Actions** and select **Run workflow**.

5. Verify that your action ran successfully by checking for a green checkmark on the **Actions** page.

Add container registry and SQL secrets

1. In the Azure portal, open your newly created Azure Container Registry in your resource group.

2. Go to **Access keys** and copy the username and password values.

3. Create new GitHub secrets for `ACR_USERNAME` and `ACR_PASSWORD` password in your repository.

4. In the Azure portal, open your Azure SQL database. Open **Connection strings** and copy the value.

5. Create a new secret for `SQL_CONNECTION_STRING`. Replace `{your_password}` with your `SQL_SERVER_ADMIN_PASSWORD`.

Build, push, and deploy your image

The build, push, and deploy workflow builds a container with the latest app changes, pushes the container to Azure Container Registry and, updates the web application staging slot to point to the latest container pushed. The workflow containers a build and deploy job:

- The build job checks out source code with the Checkout action. The job then uses the Docker login action and a custom script to authenticate with Azure Container Registry, build a container image, and deploy it to Azure Container Registry.
- The deployment job logs into Azure with the Azure Login action and gathers environment and Azure resource information. The job then updates Web App Settings with the Azure App Service Settings action and deploys to an App Service staging slot with the Azure Web Deploy action. Last, the job runs a custom script to update the SQL database and swaps staging slot to production.

To run the build, push, and deploy workflow:

1. Open your `build-deploy.yaml` file in `.github/workflows` within your repository.

2. Verify that the environment variables for `AZURE_RESOURCE_GROUP` and `WEB_APP_NAME` match the ones in `azuredeploy.yaml`.

3. Update the `ACR_LOGIN_SERVER` value for your Azure Container Registry login server.

Next steps

Learn about Azure and GitHub integration

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

Article • 03/29/2023

> **Note**
>
> Multi-container is in preview.

Web App for Containers provides a flexible way to use Docker images. In this tutorial, you'll learn how to create a multi-container app using WordPress and MySQL. You'll complete this tutorial in Cloud Shell, but you can also run these commands locally with the Azure CLI command-line tool (2.0.32 or later).

In this tutorial, you learn how to:

- ✓ Convert a Docker Compose configuration to work with Web App for Containers
- ✓ Deploy a multi-container app to Azure
- ✓ Add application settings
- ✓ Use persistent storage for your containers
- ✓ Connect to Azure Database for MySQL
- ✓ Troubleshoot errors

If you don't have an Azure subscription, create an Azure free account before you begin.

Prerequisites

To complete this tutorial, you need experience with Docker Compose.

Download the sample

For this tutorial, you use the compose file from Docker, but you'll modify it to include Azure Database for MySQL, persistent storage, and Redis. The configuration file can be found at Azure Samples. For supported configuration options, see Docker Compose options.

```yml
```

```yaml
version: '3.3'

services:
   db:
     image: mysql:5.7
     volumes:
       - db_data:/var/lib/mysql
     restart: always
     environment:
       MYSQL_ROOT_PASSWORD: somewordpress
       MYSQL_DATABASE: wordpress
       MYSQL_USER: wordpress
       MYSQL_PASSWORD: wordpress

   wordpress:
     depends_on:
       - db
     image: wordpress:latest
     ports:
       - "8000:80"
     restart: always
     environment:
       WORDPRESS_DB_HOST: db:3306
       WORDPRESS_DB_USER: wordpress
       WORDPRESS_DB_PASSWORD: wordpress
volumes:
    db_data:
```

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

Bash

```bash
mkdir tutorial

cd tutorial
```

Next, run the following command to clone the sample app repository to your tutorial directory. Then change to the `multicontainerwordpress` directory.

Bash

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

cd multicontainerwordpress
```

Create a resource group

A resource group is a logical container into which Azure resources, 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 Cloud Shell, create a resource group with the az group create command. The following example creates a resource group named *myResourceGroup* in the *South Central US* location. To see all supported locations for App Service on Linux in **Standard** tier, run the az appservice list-locations --sku S1 --linux-workers-enabled command.

Azure CLI

```
az group create --name myResourceGroup --location "South Central US"
```

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

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

Create an Azure App Service plan

In Cloud Shell, create an App Service plan in the resource group with the az appservice plan create command.

The following example creates an App Service plan named `myAppServicePlan` in the **Standard** pricing tier (`--sku S1`) and in a Linux container (`--is-linux`).

Azure CLI

```
az appservice plan create --name myAppServicePlan --resource-group
myResourceGroup --sku S1 --is-linux
```

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

```
{
  "adminSiteName": null,
  "appServicePlanName": "myAppServicePlan",
  "geoRegion": "South Central US",
  "hostingEnvironmentProfile": null,
  "id": "/subscriptions/0000-
0000/resourceGroups/myResourceGroup/providers/Microsoft.Web/serverfarms/myAp
pServicePlan",
  "kind": "linux",
  "location": "South Central US",
  "maximumNumberOfWorkers": 1,
  "name": "myAppServicePlan",
```

```
  < JSON data removed for brevity. >
  "targetWorkerSizeId": 0,
  "type": "Microsoft.Web/serverfarms",
  "workerTierName": null
}
```

Docker Compose with WordPress and MySQL containers

Create a Docker Compose app

In your Cloud Shell, create a multi-container web app in the `myAppServicePlan` App Service plan with the az webapp create command. Don't forget to replace *<app-name>* with a unique app name.

Azure CLI

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

When the web app has been created, Cloud Shell shows output similar to the following example:

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

Browse to the app

Browse to the deployed app at (`http://<app-name>.azurewebsites.net`). The app may take a few minutes to load. If you receive an error, allow a few more minutes then refresh the browser. If you're having trouble and would like to troubleshoot, review container logs.

Congratulations, you've created a multi-container app in Web App for Containers. Next you'll configure your app to use Azure Database for MySQL. Don't install WordPress at this time.

Connect to production database

It's not recommended to use database containers in a production environment. The local containers aren't scalable. Instead, you'll use Azure Database for MySQL which can be scaled.

Create an Azure Database for MySQL server

Create an Azure Database for MySQL server with the az mysql server create command.

In the following command, substitute your MySQL server name where you see the *<mysql-server-name>* placeholder (valid characters are `a-z`, `0-9`, and `-`). This name is

part of the MySQL server's hostname (`<mysql-server-name>.database.windows.net`), it needs to be globally unique.

Azure CLI

```
az mysql server create --resource-group myResourceGroup --name <mysql-server-name>  --location "South Central US" --admin-user adminuser --admin-password My5up3rStr0ngPaSw0rd! --sku-name B_Gen5_1 --version 5.7
```

Creating the server may take a few minutes to complete. When the MySQL server is created, Cloud Shell shows information similar to the following example:

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

Configure server firewall

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

Azure CLI

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

Tip

You can be even more restrictive in your firewall rule by **using only the outbound IP addresses your app uses**.

Create the WordPress database

Azure CLI

```
az mysql db create --resource-group myResourceGroup --server-name <mysql-server-name> --name wordpress
```

When the database has been created, Cloud Shell shows information similar to the following example:

```
{
  "additionalProperties": {},
  "charset": "latin1",
  "collation": "latin1_swedish_ci",
  "id": "/subscriptions/12db1644-4b12-4cab-ba54-8ba2f2822c1f/resourceGroups/myResourceGroup/providers/Microsoft.DBforMySQL/servers/<mysql-server-name>/databases/wordpress",
  "name": "wordpress",
  "resourceGroup": "myResourceGroup",
  "type": "Microsoft.DBforMySQL/servers/databases"
}
```

Configure database variables in WordPress

To connect the WordPress app to this new MySQL server, you'll configure a few WordPress-specific environment variables, including the SSL CA path defined by `MYSQL_SSL_CA`. The Baltimore CyberTrust Root from DigiCert is provided in the custom image below.

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

Azure CLI

```
az webapp config appsettings set --resource-group myResourceGroup --name <app-name> --settings WORDPRESS_DB_HOST="<mysql-server-name>.mysql.database.azure.com" WORDPRESS_DB_USER="adminuser" WORDPRESS_DB_PASSWORD="My5up3rStr0ngPaSw0rd!" WORDPRESS_DB_NAME="wordpress" MYSQL_SSL_CA="BaltimoreCyberTrustroot.crt.pem"
```

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

```
[
  {
    "name": "WORDPRESS_DB_HOST",
    "slotSetting": false,
```

```
    "value": "<mysql-server-name>.mysql.database.azure.com"
  },
  {
    "name": "WORDPRESS_DB_USER",
    "slotSetting": false,
    "value": "adminuser"
  },
  {
    "name": "WORDPRESS_DB_NAME",
    "slotSetting": false,
    "value": "wordpress"
  },
  {
    "name": "WORDPRESS_DB_PASSWORD",
    "slotSetting": false,
    "value": "My5up3rStr0ngPaSw0rd!"
  },
  {
    "name": "MYSQL_SSL_CA",
    "slotSetting": false,
    "value": "BaltimoreCyberTrustroot.crt.pem"
  }
]
```

For more information on environment variables, see Configure environment variables.

Use a custom image for MySQL TLS/SSL and other configurations

By default, TLS/SSL is used by Azure Database for MySQL. WordPress requires additional configuration to use TLS/SSL with MySQL. The WordPress 'official image' doesn't provide the additional configuration, but a custom image has been prepared for your convenience. In practice, you would add desired changes to your own image.

The custom image is based on the 'official image' of WordPress from Docker Hub. The following changes have been made in this custom image for Azure Database for MySQL:

- Adds Baltimore Cyber Trust Root Certificate file for SSL to MySQL.
- Uses App Setting for MySQL SSL Certificate Authority certificate in WordPress wp-config.php.
- Adds WordPress define for MYSQL_CLIENT_FLAGS needed for MySQL SSL.

The following changes have been made for Redis (to be used in a later section):

- Adds PHP extension for Redis v4.0.2.
- Adds unzip needed for file extraction.
- Adds Redis Object Cache 1.3.8 WordPress plugin.

- Uses App Setting for Redis host name in WordPress wp-config.php.

To use the custom image, you'll update your docker-compose-wordpress.yml file. In Cloud Shell, open a text editor and change the `image: wordpress` to use `image: mcr.microsoft.com/azuredocs/multicontainerwordpress`. You no longer need the database container. Remove the `db`, `environment`, `depends_on`, and `volumes` section from the configuration file. Your file should look like the following code:

YAML

```yaml
version: '3.3'

services:
   wordpress:
     image: mcr.microsoft.com/azuredocs/multicontainerwordpress
     ports:
       - "8000:80"
     restart: always
```

Update app with new configuration

In Cloud Shell, reconfigure your multi-container web app with the az webapp config container set command. Don't forget to replace <*app-name*> with the name of the web app you created earlier.

Azure CLI

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

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


```
[
  {
    "name": "DOCKER_CUSTOM_IMAGE_NAME",
    "value":
"COMPOSE|dmVyc2lvbjogJzMuMycKCnNlcnZpY2VzOgogICB3b3JkcHJlc3M6CiAgICAgaW1hZ2U
6IG1zYW5nYXB1L3dvcmRwcmVzcwogICAgIHBvcnRzOgogICAgICAgLSAiODAwMDo4MCIKICAgICB
yZXN0YXJ0OiBhbHdheXM="
  }
]
```

Browse to the app

Browse to the deployed app at (`http://<app-name>.azurewebsites.net`). The app is now using Azure Database for MySQL.

Add persistent storage

Your multi-container is now running in Web App for Containers. However, if you install WordPress now and restart your app later, you'll find that your WordPress installation is gone. This happens because your Docker Compose configuration currently points to a storage location inside your container. The files installed into your container don't persist beyond app restart. In this section, you'll add persistent storage to your WordPress container.

Configure environment variables

To use persistent storage, you'll enable this setting within App Service. To make this change, use the az webapp config appsettings set command in Cloud Shell. App settings are case-sensitive and space-separated.

Azure CLI

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

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

```
[
  < JSON data removed for brevity. >
  {
    "name": "WORDPRESS_DB_NAME",
    "slotSetting": false,
    "value": "wordpress"
  },
  {
    "name": "WEBSITES_ENABLE_APP_SERVICE_STORAGE",
    "slotSetting": false,
    "value": "TRUE"
  }
]
```

Modify configuration file

In the Cloud Shell, open the file `docker-compose-wordpress.yml` in a text editor.

The `volumes` option maps the file system to a directory within the container. `${WEBAPP_STORAGE_HOME}` is an environment variable in App Service that is mapped to persistent storage for your app. You'll use this environment variable in the volumes option so that the WordPress files are installed into persistent storage instead of the container. Make the following modifications to the file:

In the `wordpress` section, add a `volumes` option so it looks like the following code:

YAML

```
version: '3.3'

services:
   wordpress:
     image: mcr.microsoft.com/azuredocs/multicontainerwordpress
     volumes:
```

```
      - ${WEBAPP_STORAGE_HOME}/site/wwwroot:/var/www/html
    ports:
      - "8000:80"
    restart: always
```

Update app with new configuration

In Cloud Shell, reconfigure your multi-container web app with the az webapp config container set command. Don't forget to replace <*app-name*> with a unique app name.

Azure CLI

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

After your command runs, it shows output similar to the following example:


```
[
  {
    "name": "WEBSITES_ENABLE_APP_SERVICE_STORAGE",
    "slotSetting": false,
    "value": "TRUE"
  },
  {
    "name": "DOCKER_CUSTOM_IMAGE_NAME",
    "value":
"COMPOSE|dmVyc2lvbjogJzMuMycKCnNlcnZpY2VzOgogICBteXNxbDoKICAgICBpbWFnZTogbXl
zcWw6NS43CiAgICAgdm9sdW1lczoKICAgICAgIC0gZGJfZGF0YTovdmFyL2xpYi9teXNxbAogICA
gIHJlc3RhcnQ6IGFsd2F5cwogICAgIGVudmlyb25tZW50OgogICAgICAgTVlTUUxfUk9PVF9QQVN
TV09SRDogZXhhbXBsZXBhc3MKCiAgIHdvcmRwcmVzczoKICAgICBkZXBlbmRzX29uOgogICAgICA
gLSBteXNxbAogICAgIGltYWdlOiB3b3JkcHJlc3M6bGF0ZXN0CiAgICAgcG9ydHM6CiAgICAgICA
tICI4MDAwOjgwIgogICAgIHJlc3RhcnQ6IGFsd2F5cwogICAgIGVudmlyb25tZW50OgogICAgICA
gV09SRFBSRVNTX0RCX1BBU1NXT1JEOiBleGFtcGxlcGFzcwp2b2x1bWVzOgogICAgZGJfZGF0YTo
="
  }
]
```


Browse to the app

Browse to the deployed app at (`http://<app-name>.azurewebsites.net`).

The WordPress container is now using Azure Database for MySQL and persistent storage.

Add Redis container

The WordPress 'official image' does not include the dependencies for Redis. These dependencies and additional configuration needed to use Redis with WordPress have been prepared for you in this custom image. In practice, you would add desired changes to your own image.

The custom image is based on the 'official image' of WordPress from Docker Hub. The following changes have been made in this custom image for Redis:

- Adds PHP extension for Redis v4.0.2.
- Adds unzip needed for file extraction.
- Adds Redis Object Cache 1.3.8 WordPress plugin.
- Uses App Setting for Redis host name in WordPress wp-config.php.

Add the redis container to the bottom of the configuration file so it looks like the following example:

YAML

```yaml
version: '3.3'

services:
   wordpress:
     image: mcr.microsoft.com/azuredocs/multicontainerwordpress
     ports:
       - "8000:80"
     restart: always

   redis:
     image: mcr.microsoft.com/oss/bitnami/redis:6.0.8
     environment:
      - ALLOW_EMPTY_PASSWORD=yes
     restart: always
```

Configure environment variables

To use Redis, you'll enable this setting, `WP_REDIS_HOST`, within App Service. This is a *required setting* for WordPress to communicate with the Redis host. To make this change, use the az webapp config appsettings set command in Cloud Shell. App settings are case-sensitive and space-separated.

Azure CLI

```cpp
az webapp config appsettings set --resource-group myResourceGroup --name
<app-name> --settings WP_REDIS_HOST="redis"
```

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

```
[
  < JSON data removed for brevity. >
  {
    "name": "WORDPRESS_DB_USER",
    "slotSetting": false,
    "value": "adminuser"
  },
  {
    "name": "WP_REDIS_HOST",
    "slotSetting": false,
    "value": "redis"
  }
]
```

Update app with new configuration

In Cloud Shell, reconfigure your multi-container web app with the az webapp config container set command. Don't forget to replace <*app-name*> with a unique app name.

Azure CLI

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

After your command runs, it shows output similar to the following example:

```
[
  {
    "name": "DOCKER_CUSTOM_IMAGE_NAME",
    "value":
"COMPOSE|dmVyc2lvbjogJzMuMycKCnNlcnZpY2VzOgogICBteXNxbDoKICAgICBpbWFnZTogbXl
zcWw6NS43CiAgICAgdm9sdW1lczoKICAgICAgIC0gZGJfZGF0YTovdmFyL2xpYi9teXNxbAogICA
gIHJlc3RhcnQ6IGFsd2F5cwogICAgIGVudmlyb25tZW50OgogICAgICAgTVlTUUxfUk9PVF9QQVN
TV09SRDogZXhhbXBsZXBhc3MKCiAgIHdvcmRwcmVzczoKICAgICBkZXBlbmRzX29uOgogICAgICA
gLSBteXNxbAogICAgIGltYWdlOiB3b3JkcHJlc3M6bGF0ZXN0CiAgICAgcG9ydHM6CiAgICAgICA
tICI4MDAwOjgwIgogICAgIHJlc3RhcnQ6IGFsd2F5cwogICAgIGVudmlyb25tZW50OgogICAgICA
gV09SRFBSRVNTX0RCX1BBU1NXT1JEOiBleGFtcGxlcGFzcwp2b2x1bWVzOgogICAgZGJfZGF0YTo
="
  }
]
```

Browse to the app

Browse to the deployed app at (`http://<app-name>.azurewebsites.net`).

Complete the steps and install WordPress.

Connect WordPress to Redis

Sign in to WordPress admin. In the left navigation, select **Plugins**, and then select **Installed Plugins**.

Show all plugins here

In the plugins page, find **Redis Object Cache** and click **Activate**.

Click on **Settings**.

Click the **Enable Object Cache** button.

WordPress connects to the Redis server. The connection **status** appears on the same page.

Congratulations, you've connected WordPress to Redis. The production-ready app is now using **Azure Database for MySQL, persistent storage, and Redis**. You can now scale out your App Service Plan to multiple instances.

Find Docker Container logs

If you run into issues using multiple containers, you can access the container logs by browsing to: `https://<app-name>.scm.azurewebsites.net/api/logs/docker`.

You'll see output similar to the following example:

```
[
   {
      "machineName":"RD00XYZYZE567A",
      "lastUpdated":"2018-05-10T04:11:45Z",
      "size":25125,
      "href":"https://<app-
name>.scm.azurewebsites.net/api/vfs/LogFiles/2018_05_10_RD00XYZYZE567A_docke
r.log",
      "path":"/home/LogFiles/2018_05_10_RD00XYZYZE567A_docker.log"
   }
]
```

You see a log for each container and an additional log for the parent process. Copy the respective `href` value into the browser to view the log.

Clean up deployment

After the sample script has been run, the following command can be used to remove the resource group and all resources associated with it.

Azure CLI

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

Next steps

In this tutorial, you learned how to:

- ✓ Convert a Docker Compose configuration to work with Web App for Containers
- ✓ Deploy a multi-container app to Azure
- ✓ Add application settings
- ✓ Use persistent storage for your containers
- ✓ Connect to Azure Database for MySQL
- ✓ Troubleshoot errors

Advance to the next tutorial to learn how to secure your app with a custom domain and certificate.

Secure with custom domain and certificate

Or, check out other resources:

- Configure custom container
- Environment variables and app settings reference

Reliability in Azure App Service

Article • 09/26/2023

This article describes reliability support in Azure App Service, and covers both intra-regional resiliency with availability zones and cross-region disaster recovery and business continuity. For a more detailed overview of reliability principles in Azure, see Azure reliability.

Azure App Service is an HTTP-based service for hosting web applications, REST APIs, and mobile back ends; and adds the power of Microsoft Azure to your application, such as:

- Security
- Load balancing
- Autoscaling
- Automated management

To explore how Azure App Service can bolster the reliability and resiliency of your application workload, see Why use App Service?

Reliability recommendations

This section contains recommendations for achieving resiliency and availability. Each recommendation falls into one of two categories:

- **Health items** cover areas such as configuration items and the proper function of the major components that make up your Azure Workload, such as Azure Resource configuration settings, dependencies on other services, and so on.
- **Risk items** cover areas such as availability and recovery requirements, testing, monitoring, deployment, and other items that, if left unresolved, increase the chances of problems in the environment.

Reliability recommendations priority matrix

Each recommendation is marked in accordance with the following priority matrix:

Image	Priority	Description
	High	Immediate fix needed.
	Medium	Fix within 3-6 months.
	Low	Needs to be reviewed.

Image	Priority	Description

Reliability recommendations summary

Category	Priority	Recommendation
High Availability	!	ASP-1 - Deploy zone-redundant App Service plans
Resiliency	!	ASP-2 -Use an App Service plan that supports availability zones
	!	ASP-4 - Create separate App Service plans for production and test
Scalability	⚠	ASP-3 - Avoid frequently scaling up or down
	⚠	ASP-5 - Enable Autoscale/Automatic scaling to ensure adequate resources are available to service requests

High availability

ASP-1 - Deploy zone-redundant App Service plans

To enhance the resiliency and reliability of your business-critical workloads, it's recommended that you deploy your new App Service Plans with zone-redundancy. Follow the steps to redeploy to availability zone support, configure your pipelines to redeploy your WebApp on the new App Services Plan, and then use a Blue-Green deployment approach to failover to the new site.

By distributing your applications across multiple availability zones, you can ensure their continued operation even in the event of a datacenter-level failure. For more information on availability zone support in Azure App Service, see Availability zone support.

Azure Resource Graph

Kusto

```
// Azure Resource Graph Query
// The query filters the qualified App Service Plans that do not have
Zone Redundancy enabled.
resources
| where type =~ 'microsoft.web/serverfarms'
| extend zoneRedundant = tobool(properties.zoneRedundant)
| extend sku_tier = tostring(sku.tier)
| where sku_tier !in ("Free", "LinuxFree", "Basic", "WorkflowStandard",
"ElasticPremium", "Dynamic", "Shared", "", "Isolated", "Standard") and
```

```
zoneRedundant == false
| project recommendationid="asp-1", name, id, sku_tier, zoneRedundant
```

Resiliency

ASP-2 -Use an App Service plan that supports availability zones

Availability zone support is only available on certain App Service plans. To see which plan you need in order to use availability zones, see Availability zone prerequisites.

Azure Resource Graph

Kusto

```
// Azure Resource Graph Query
// The query filters the resources based on the type
"microsoft.web/serverfarms" i.e. App Service Plans that are not setup to
use standard or premium SKUs.
resources
| where type =~ 'microsoft.web/serverfarms'
| extend sku_tier = tostring(sku.tier)
| where sku_tier !contains 'premium' and sku_tier !contains 'standard'
| project recommendationid="asp-2", name, id, sku_tier
```

ASP-4 - Create separate App Service plans for production and test

To enhance the resiliency and reliability of your business-critical workloads, you should migrate your existing App Service plans and App Service Environments to availability zone support. By distributing your applications across multiple availability zones, you can ensure their continued operation even in the event of a datacenter-level failure. For more information on availability zone support in Azure App Service, see Availability zone support.

Scalability

ASP-3 - Avoid frequently scaling up or down

It's recommended that you avoid frequently scaling up or down your Azure App Service instances. Instead, choose an appropriate tier and instance size that can handle your

typical workload, and scale out the instances to accommodate changes in traffic volume. Scaling up or down can potentially trigger an application restart, which may result in service disruptions.

⚠ ASP-5 - Enable Autoscale/Automatic scaling to ensure that adequate resources are available to service requests

It's recommended that you enable autoscale/automatic scaling for your Azure App Service to ensure that sufficient resources are available to handle incoming requests. Autoscaling is rule based scaling, while automatic scaling performs automatic in and out scaling based on HTTP traffic. For more information see, automatic scaling in Azure App Service or get started with autoscale in Azure.

Azure Resource Graph

Kusto

```
// Azure Resource Graph Query
// Azure Resource Graph
// App Service Plans that are not setup to use standard or premium SKUs.
// TODO: Extend this query to show automatic scaling or App level
automatic scale
resources
| where type =~ "microsoft.insights/autoscalesettings"
| where properties.targetResourceUri contains
"microsoft.web/serverfarms"
| project AppServicePlan = split(properties.targetResourceUri, '/')[8],
ResourceGroup = split(properties.targetResourceUri, '/')
[4],id=properties.targetResourceUri
| project recommendationid="asp-5", ["name"]=AppServicePlan, ["id"]=id
```

Availability zone support

Azure availability zones are at least three physically separate groups of datacenters within each Azure region. Datacenters within each zone are equipped with independent power, cooling, and networking infrastructure. In the case of a local zone failure, availability zones are designed so that if the one zone is affected, regional services, capacity, and high availability are supported by the remaining two zones.

Failures can range from software and hardware failures to events such as earthquakes, floods, and fires. Tolerance to failures is achieved with redundancy and logical isolation of Azure services. For more detailed information on availability zones in Azure, see Regions and availability zones.

Azure availability zones-enabled services are designed to provide the right level of reliability and flexibility. They can be configured in two ways. They can be either zone redundant, with automatic replication across zones, or zonal, with instances pinned to a specific zone. You can also combine these approaches. For more information on zonal vs. zone-redundant architecture, see Recommendations for using availability zones and regions.

Azure App Service can be deployed across availability zones (AZ) to help you achieve resiliency and reliability for your business-critical workloads. This architecture is also known as zone redundancy.

When you configure to be zone redundant, the platform automatically spreads the instances of the Azure App Service plan across three zones in the selected region. This means that the minimum App Service Plan instance count will always be three. If you specify a capacity larger than three, and the number of instances is divisible by three, the instances are spread evenly. Otherwise, instance counts beyond 3*N are spread across the remaining one or two zones.

Availability zone support is a property of the App Service plan. App Service plans can be created on managed multi-tenant environment or dedicated environment using App Service Environment v3. To Learn more regarding App Service Environment v3, see App Service Environment v3 overview.

For App Services that aren't configured to be zone redundant, VM instances are not zone resilient and can experience downtime during an outage in any zone in that region.

For information on enterprise deployment architecture, see High availability enterprise deployment using App Service Environment.

Prerequisites

The current requirements/limitations for enabling availability zones are:

- Both Windows and Linux are supported.
- Availability zones are only supported on the newer App Service footprint. Even if you're using one of the supported regions, you'll receive an error if availability zones aren't supported for your resource group. To ensure your workloads land on a stamp that supports availability zones, you may need to create a new resource group, App Service plan, and App Service.

- Your App Services plan must be one of the following plans that support availability zones:
  - In a multi-tenant environment using App Service Premium v2 or Premium v3 plans.
  - In a dedicated environment using App Service Environment v3, which is used with Isolated v2 App Service plans.

- For dedicated environments, your App Service Environment must be v3.

> **Important**
>
> **App Service Environment v2 and v1 will be retired on 31 August 2024**. App Service Environment v3 is easier to use and runs on more powerful infrastructure. To learn more about App Service Environment v3, see **App Service Environment overview**. If you're currently using App Service Environment v2 or v1 and you want to upgrade to v3, please follow the **steps in this article** to migrate to the new version.

- Minimum instance count of three zones is enforced. The platform will enforce this minimum count behind the scenes if you specify an instance count fewer than three.

- Availability zones can only be specified when creating a **new** App Service plan. A pre-existing App Service plan can't be converted to use availability zones.

- The following regions support Azure App Services running on multi-tenant environments:
  - Australia East
  - Brazil South
  - Canada Central
  - Central India
  - Central US
  - East Asia
  - East US
  - East US 2
  - France Central
  - Germany West Central
  - Japan East
  - Korea Central
  - North Europe
  - Norway East

- Poland Central
- Qatar Central
- South Africa North
- South Central US
- Southeast Asia
- Sweden Central
- Switzerland North
- UAE North
- UK South
- West Europe
- West US 2
- West US 3
- Microsoft Azure operated by 21Vianet - China North 3
- Azure Government - US Gov Virginia

- To see which regions support availability zones for App Service Environment v3, see Regions.

Create a resource with availability zone enabled

To deploy a multi-tenant zone-redundant App Service

Azure CLI

To enable availability zones using the Azure CLI, include the `--zone-redundant` parameter when you create your App Service plan. You can also include the `--number-of-workers` parameter to specify capacity. If you don't specify a capacity, the platform defaults to three. Capacity should be set based on the workload requirement, but no less than three. A good rule of thumb to choose capacity is to ensure sufficient instances for the application such that losing one zone of instances leaves sufficient capacity to handle expected load.

Azure CLI

```
az appservice plan create --resource-group MyResourceGroup --name MyPlan
--sku P1v2 --zone-redundant --number-of-workers 6
```

Tip

To decide instance capacity, you can use the following calculation:

Since the platform spreads VMs across three zones and you need to account for at least the failure of one zone, multiply peak workload instance count by a factor of zones/(zones-1), or 3/2. For example, if your typical peak workload requires four instances, you should provision six instances: (2/3 * 6 instances) = 4 instances.

Deploy a zone-redundant App Service using a dedicated environment

To learn how to create an App Service Environment v3 on the Isolated v2 plan, see Create an App Service Environment.

Troubleshooting

Error message	Description	Recommendation
Zone redundancy is not available for resource group 'RG-NAME'. Please deploy app service plan 'ASP-NAME' to a new resource group.	Availability zones are only supported on the newer App Service footprint. Even if you're using one of the supported regions, you'll receive an error if availability zones aren't supported for your resource group.	To ensure your workloads land on a stamp that supports availability zones, create a new resource group, App Service plan, and App Service.

Fault tolerance

To prepare for availability zone failure, you should over-provision capacity of service to ensure that the solution can tolerate 1/3 loss of capacity and continue to function without degraded performance during zone-wide outages. Since the platform spreads VMs across three zones and you need to account for at least the failure of one zone, multiply peak workload instance count by a factor of zones/(zones-1), or 3/2. For example, if your typical peak workload requires four instances, you should provision six instances: (2/3 * 6 instances) = 4 instances.

Zone down experience

Traffic is routed to all of your available App Service instances. In the case when a zone goes down, the App Service platform will detect lost instances and automatically attempt to find new replacement instances and spread traffic as needed. If you have autoscale configured, and if it decides more instances are needed, autoscale will also

issue a request to App Service to add more instances. Note that autoscale behavior is independent of App Service platform behavior and that your autoscale instance count specification doesn't need to be a multiple of three.

> **Note**
>
> There's no guarantee that requests for additional instances in a zone-down scenario will succeed. The back filling of lost instances occurs on a best-effort basis. The recommended solution is to create and configure your App Service plans to account for losing a zone as described in the next section.

Applications that are deployed in an App Service plan that has availability zones enabled will continue to run and serve traffic even if other zones in the same region suffer an outage. However it's possible that non-runtime behaviors including App Service plan scaling, application creation, application configuration, and application publishing may still be impacted from an outage in other Availability Zones. Zone redundancy for App Service plans only ensures continued uptime for deployed applications.

When the App Service platform allocates instances to 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 will be "balanced" if each zone has either the same number of VMs, or +/- one VM in all of the other zones used by the App Service plan.

Availability zone migration

You cannot migrate existing App Service instances or environment resources from non-availability zone support to availability zone support. To get support for availability zones, you'll need to create your resources with availability zones enabled.

Pricing

There's no additional cost associated with enabling availability zones. Pricing for a zone redundant App Service is the same as a single zone App Service. You'll be charged based on your App Service plan SKU, the capacity you specify, and any instances you scale to based on your autoscale criteria. If you enable availability zones but specify a capacity less than three, the platform will enforce a minimum instance count of three and charge you for those three instances. For pricing information for App Service Environment v3, see Pricing.

Cross-region disaster recovery and business continuity

Disaster recovery (DR) is about recovering from high-impact events, such as natural disasters or failed deployments, that result in downtime and data loss. Regardless of the cause, the best remedy for a disaster is a well-defined and tested DR plan and an application design that actively supports DR.

When it comes to DR, Microsoft uses the shared responsibility model. The shared responsibility models means that Microsoft ensures that the baseline infrastructure and platform services are available. However, in some scenarios, usage requires that the customer to duplicate their deployments and storage in a multi-region capacity, if they opt to.

Most services that run on Azure platform as a service (PaaS) offerings provide features and guidance to support DR. For some scenarios, you can use service-specific features to support fast recovery.

This section covers some common strategies for web apps deployed to App Service.

When you create a web app in App Service and choose an Azure region during resource creation, it's a single-region app. When the region becomes unavailable during a disaster, your application also becomes unavailable. If you create an identical deployment in a secondary Azure region using a multi-region geography architecture, your application becomes less susceptible to a single-region disaster, which guarantees business continuity. Any data replication across the regions lets you recover your last application state.

For IT, business continuity plans are largely driven by Recovery Time Objective (RTO) and Recovery Point Objective (RPO). For more information on RTO and RPO, see Recovery objectives.

Normally, maintaining an SLA around RTO is impractical for regional disasters, and you would typically design your disaster recovery strategy around RPO alone (i.e. focus on recovering data and not on minimizing interruption). With Azure, however, it's not only practical but can even be straightforward to deploy App Service for automatic geo-failovers. This lets you disaster-proof your applications further by taking care of both RTO and RPO.

Depending on your desired RTO and RPO metrics, three disaster recovery architectures are commonly used for both App Service multitenant and App Service Environments. Each architecture is described in the following table:

Metric	Active-Active	Active-Passive	Passive/Cold
RTO	Real-time or seconds	Minutes	Hours
RPO	Real-time or seconds	Minutes	Hours
Cost	$$$	$$	$
Scenarios	Mission-critical apps	High-priority apps	Low-priority apps
Ability to serve multi-region user traffic	Yes	Yes/maybe	No
Code deployment	CI/CD pipelines preferred	CI/CD pipelines preferred	Backup and restore
Creation of new App Service resources during downtime	Not required	Not required	Required

> **Note**
>
> Your application most likely depends on other data services in Azure, such as Azure SQL Database and Azure Storage accounts. It's recommended that you develop disaster recovery strategies for each of these dependent Azure Services as well. For SQL Database, see **Active geo-replication for Azure SQL Database**. For Azure Storage, see **Azure Storage redundancy**.

Disaster recovery in multi-region geography

There are multiple ways to replicate your web apps content and configurations across Azure regions in an active-active or active-passive architecture, such as using App service backup and restore. However, backup and restore create point-in-time snapshots and eventually lead to web app versioning challenges across regions. See the following table below for a comparison between back and restore guidance vs. diaster recovery guidance:

Back up & restore vs. disaster recovery

Platform	Back up & restore guidance	Disaster recovery guidance
App Service Web Apps (Free & Shared Pricing Tier)	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 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 region as explained in the recover from region-wide failure article. It's recommended to implement commonly used disaster recovery techniques to prevent downtime and data loss during a regional disaster.
App Service Web Apps (Basic\Standard\Premium Pricing Tier)	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 by restoring to a new app or slot. Refer to Back up and restore your app in Azure App Service for more info.	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. 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.
App Service Environment (V2 & V3)	In Azure App Service Environment, You can make on-demand custom backups or utilize automatic backups. Automatic backups can be restored to a target app within the same ASE, not in another ASE. Custom backups can be restored to a target app in another ASE (such as from a V2 ASE to a V3 ASE). Backups can	We encourage you to implement disaster recovery guidance for web apps deployed to App Service Environment using commonly used disaster recovery techniques.

Platform	Back up & restore guidance	Disaster recovery guidance
	be restored to target app of the same OS platform as the source app. Refer to Back up and restore your app in Azure App Service for more details.	
Azure Functions (Dedicated)	In Azure Functions, You can make on-demand custom backups or utilize automatic backups. You can restore a backup by overwriting an existing app by restoring to a new app or slot. Refer to Back up and restore your app in Azure App Service for more info.	Current guidance regarding how to bring Azure Functions app (dedicated) resources back online in a different Azure region during a regional disaster is available at Recover from region-wide failure - Azure App Service. 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. Instead, implement Azure Functions geo-disaster recovery. Additionally you may also refer to commonly used disaster recovery techniques for Azure Functions dedicated.
Azure Functions Consumption & Premium	Azure functions deployed to consumption and premium plans don't provide access to custom and automatic backups. Function app's content is in an Azure storage account. Use Azure Storage redundancy options to ensure your storage account meets its availability and durability targets during an outage. If you created your functions by using the editor in the Azure portal, you can also download your existing function app project as a .zip file.	We strongly encourage you to implement Azure Functions geo-disaster recovery and reliability.

To avoid the limitations of backup and restore methods, configure your CI/CD pipelines to deploy code to both Azure regions. Consider using Azure Pipelines or GitHub Actions. For more information, see Continuous deployment to Azure App Service.

Outage detection, notification, and management

- It's recommended that you set up monitoring and alerts for your web apps to for timely notifications during a disaster. For more information, see Application Insights availability tests.
- To manage your application resources in Azure, use an infrastructure-as-Code (IaC) mechanism. In a complex deployment across multiple regions, to manage the regions independently and to keep the configuration synchronized across regions in a reliable manner requires a predictable, testable, and repeatable process. Consider an IaC tool such as Azure Resource Manager templates or Terraform.

Set up disaster recovery and outage detection

To prepare for disaster recovery in a multi-region geography, you can use either an active-active or active-passive architecture.

Active-Active architecture

In active-active disaster recovery architecture, identical web apps are deployed in two separate regions and Azure Front door is used to route traffic to both the active regions.

With this example architecture:

- Identical App Service apps are deployed in two separate regions, including pricing tier and instance count.
- Public traffic directly to the App Service apps is blocked.
- Azure Front Door is used to route traffic to both the active regions.
- During a disaster, one of the regions becomes offline, and Azure Front Door routes traffic exclusively to the region that remains online. The RTO during such a geo-failover is near-zero.
- Application files should be deployed to both web apps with a CI/CD solution. This ensures that the RPO is practically zero.
- If your application actively modifies the file system, the best way to minimize RPO is to only write to a mounted Azure Storage share instead of writing directly to the web app's */home* content share. Then, use the Azure Storage redundancy features (GZRS or GRS) for your mounted share, which has an RPO of about 15 minutes.

Steps to create an active-active architecture for your web app in App Service are summarized as follows:

1. Create two App Service plans in two different Azure regions. Configure the two App Service plans identically.

2. Create two instances of your web app, with one in each App Service plan.

3. Create an Azure Front Door profile with:

    - An endpoint.
    - Two origin groups, each with a priority of 1. The equal priority tells Azure Front Door to route traffic to both regions equally (thus active-active).
    - A route.

4. Limit network traffic to the web apps only from the Azure Front Door instance.

5. Setup and configure all other back-end Azure service, such as databases, storage accounts, and authentication providers.

6. Deploy code to both the web apps with continuous deployment.

Tutorial: Create a highly available multi-region app in Azure App Service shows you how to set up an *active-passive* architecture. The same steps with minimal changes (setting priority to “1” for both origin groups in Azure Front Door) give you an *active-active* architecture.

Active-passive architecture

In this disaster recovery approach, identical web apps are deployed in two separate regions and Azure Front door is used to route traffic to one region only (the *active* region).

With this example 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 to route traffic to the primary region.
- To save cost, the secondary App Service plan is configured to have fewer instances and/or be in a lower pricing tier. There are three possible approaches:
  - **Preferred** The secondary App Service plan has the same pricing tier as the primary, with the same number of instances or fewer. This approach ensures parity in both feature and VM sizing for the two App Service plans. The RTO during a geo-failover only depends on the time to scale out the instances.
  - **Less preferred** The secondary App Service plan has the same pricing tier type (such as PremiumV3) but smaller VM sizing, with lesser instances. For example, the primary region may be in P3V3 tier while the secondary region is in P1V3 tier. This approach still ensures feature parity for the two App Service plans, but the lack of size parity may require a manual scale-up when the secondary region becomes the active region. The RTO during a geo-failover depends on the time to both scale up and scale out the instances.

- **Least-preferred** The secondary App Service plan has a different pricing tier than the primary and lesser instances. For example, the primary region may be in P3V3 tier while the secondary region is in S1 tier. Make sure that the secondary App Service plan has all the features your application needs in order to run. Differences in features availability between the two may cause delays to your web app recovery. The RTO during a geo-failover depends on the time to both scale up and scale out the instances.

- Autoscale is configured on the secondary region in the event the active region becomes inactive. It's advisable to have similar autoscale rules in both active and passive regions.

- During a disaster, the primary region becomes inactive, and the secondary region starts receiving traffic and becomes the active region.

- Once the secondary region becomes active, the network load triggers preconfigured autoscale rules to scale out the secondary web app.

- You may need to scale up the pricing tier for the secondary region manually, if it doesn't already have the needed features to run as the active region. For example, autoscaling requires Standard tier or higher.

- When the primary region is active again, Azure Front Door automatically directs traffic back to it, and the architecture is back to active-passive as before.

- Application files should be deployed to both web apps with a CI/CD solution. This ensures that the RPO is practically zero.

- If your application actively modifies the file system, the best way to minimize RPO is to only write to a mounted Azure Storage share instead of writing directly to the web app's */home* content share. Then, use the Azure Storage redundancy features (GZRS or GRS) for your mounted share, which has an RPO of about 15 minutes.

Steps to create an active-passive architecture for your web app in App Service are summarized as follows:

1. Create two App Service plans in two different Azure regions. The secondary App Service plan may be provisioned using one of the approaches mentioned previously.
2. Configure autoscaling rules for the secondary App Service plan so that it scales to the same instance count as the primary when the primary region becomes inactive.
3. Create two instances of your web app, with one in each App Service plan.
4. Create an Azure Front Door profile with:

    - An endpoint.
    - An origin group with a priority of 1 for the primary region.
    - A second origin group with a priority of 2 for the secondary region. The difference in priority tells Azure Front Door to prefer the primary region when it's online (thus active-passive).
    - A route.
5. Limit network traffic to the web apps only from the Azure Front Door instance.
6. Setup and configure all other back-end Azure service, such as databases, storage accounts, and authentication providers.
7. Deploy code to both the web apps with continuous deployment.

Tutorial: Create a highly available multi-region app in Azure App Service shows you how to set up an *active-passive* architecture.

Passive-cold architecture

Use a passive/cold architecture to create and maintain regular backups of your web apps in an Azure Storage account that's located in another region.

With this example architecture:

- A single web app is deployed to a single region.
- The web app is regularly backed up to an Azure Storage account in the same region.
- The cross-region replication of your backups depends on the data redundancy configuration in the Azure storage account. You should set your Azure Storage account as GZRS if possible. GZRS offers both synchronous zone redundancy within a region and asynchronous in a secondary region. If GZRS isn't available, configure the account as GRS. Both GZRS and GRS have an RPO of about 15 minutes.
- To ensure that you can retrieve backups when the storage account's primary region becomes unavailable, **enable read only access to secondary region** (making the storage account **RA-GZRS** or **RA-GRS**, respectively). For more information on designing your applications to take advantage of geo-redundancy, see Use geo-redundancy to design highly available applications.
- During a disaster in the web app's region, you must manually deploy all required App Service dependent resources by using the backups from the Azure Storage account, most likely from the secondary region with read access. The RTO may be hours or days.

- To minimize RTO, it's highly recommended that you have a comprehensive playbook outlining all the steps required to restore your web app backup to another Azure Region.

Steps to create a passive-cold region for your web app in App Service are summarized as follows:

1. Create an Azure storage account in the same region as your web app. Choose Standard performance tier and select redundancy as Geo-redundant storage (GRS) or Geo-Zone-redundant storage (GZRS).

2. Enable RA-GRS or RA-GZRS (read access for the secondary region).

3. Configure custom backup for your web app. You may decide to set a schedule for your web app backups, such as hourly.

4. Verify that the web app backup files can be retrieved the secondary region of your storage account.

> **Tip**
>
> Aside from Azure Front Door, Azure provides other load balancing options, such as Azure Traffic Manager. For a comparison of the various options, see **Load-balancing options - Azure Architecture Center**.

Disaster recovery in single-region geography

If your web app's region doesn't have GZRS or GRS storage or if you are in an Azure region that isn't one of a regional pair, you'll need to utilize zone-redundant storage (ZRS) or locally redundant storage (LRS) to create a similar architecture. For example, you can manually create a secondary region for the storage account as follows:

Steps to create a passive-cold region without GRS and GZRS are summarized as follows:

1. Create an Azure storage account in the same region of your web app. Choose Standard performance tier and select redundancy as zone-redundant storage (ZRS).

2. Configure custom backup for your web app. You may decide to set a schedule for your web app backups, such as hourly.

3. Verify that the web app backup files can be retrieved the secondary region of your storage account.

4. Create a second Azure storage account in a different region. Choose Standard performance tier and select redundancy as locally redundant storage (LRS).

5. By using a tool like AzCopy, replicate your custom backup (Zip, XML and log files) from primary region to the secondary storage. For example:

   ```
   azcopy copy 'https://<source-storage-account-
   name>.blob.core.windows.net/<container-name>/<blob-path>'
   'https://<destination-storage-account-
   name>.blob.core.windows.net/<container-name>/<blob-path>'
   ```

   You can use Azure Automation with a PowerShell Workflow runbook to run your replication script on a schedule. Make sure that the replication schedule follows a similar schedule to the web app backups.

Next steps

- Tutorial: Create a highly available multi-region app in Azure App Service
- Reliability in Azure

Reliability in Azure App Service

Article • 09/26/2023

This article describes reliability support in Azure App Service, and covers both intra-regional resiliency with availability zones and cross-region disaster recovery and business continuity. For a more detailed overview of reliability principles in Azure, see Azure reliability.

Azure App Service is an HTTP-based service for hosting web applications, REST APIs, and mobile back ends; and adds the power of Microsoft Azure to your application, such as:

- Security
- Load balancing
- Autoscaling
- Automated management

To explore how Azure App Service can bolster the reliability and resiliency of your application workload, see Why use App Service?

Reliability recommendations

This section contains recommendations for achieving resiliency and availability. Each recommendation falls into one of two categories:

- **Health items** cover areas such as configuration items and the proper function of the major components that make up your Azure Workload, such as Azure Resource configuration settings, dependencies on other services, and so on.
- **Risk items** cover areas such as availability and recovery requirements, testing, monitoring, deployment, and other items that, if left unresolved, increase the chances of problems in the environment.

Reliability recommendations priority matrix

Each recommendation is marked in accordance with the following priority matrix:

Image	Priority	Description
	High	Immediate fix needed.
	Medium	Fix within 3-6 months.
	Low	Needs to be reviewed.

Image	Priority	Description

Reliability recommendations summary

Category	Priority	Recommendation
High Availability	!	ASP-1 - Deploy zone-redundant App Service plans
Resiliency	!	ASP-2 -Use an App Service plan that supports availability zones
	!	ASP-4 - Create separate App Service plans for production and test
Scalability	⚠	ASP-3 - Avoid frequently scaling up or down
	⚠	ASP-5 - Enable Autoscale/Automatic scaling to ensure adequate resources are available to service requests

High availability

ASP-1 - Deploy zone-redundant App Service plans

To enhance the resiliency and reliability of your business-critical workloads, it's recommended that you deploy your new App Service Plans with zone-redundancy. Follow the steps to redeploy to availability zone support, configure your pipelines to redeploy your WebApp on the new App Services Plan, and then use a Blue-Green deployment approach to failover to the new site.

By distributing your applications across multiple availability zones, you can ensure their continued operation even in the event of a datacenter-level failure. For more information on availability zone support in Azure App Service, see Availability zone support.

Azure Resource Graph

Kusto

```
// Azure Resource Graph Query
// The query filters the qualified App Service Plans that do not have
Zone Redundancy enabled.
resources
| where type =~ 'microsoft.web/serverfarms'
| extend zoneRedundant = tobool(properties.zoneRedundant)
| extend sku_tier = tostring(sku.tier)
| where sku_tier !in ("Free", "LinuxFree", "Basic", "WorkflowStandard",
"ElasticPremium", "Dynamic", "Shared", "", "Isolated", "Standard") and
```

```
zoneRedundant == false
| project recommendationid="asp-1", name, id, sku_tier, zoneRedundant
```

Resiliency

ASP-2 -Use an App Service plan that supports availability zones

Availability zone support is only available on certain App Service plans. To see which plan you need in order to use availability zones, see Availability zone prerequisites.

Azure Resource Graph

Kusto

```
// Azure Resource Graph Query
// The query filters the resources based on the type
"microsoft.web/serverfarms" i.e. App Service Plans that are not setup to
use standard or premium SKUs.
resources
| where type =~ 'microsoft.web/serverfarms'
| extend sku_tier = tostring(sku.tier)
| where sku_tier !contains 'premium' and sku_tier !contains 'standard'
| project recommendationid="asp-2", name, id, sku_tier
```

ASP-4 - Create separate App Service plans for production and test

To enhance the resiliency and reliability of your business-critical workloads, you should migrate your existing App Service plans and App Service Environments to availability zone support. By distributing your applications across multiple availability zones, you can ensure their continued operation even in the event of a datacenter-level failure. For more information on availability zone support in Azure App Service, see Availability zone support.

Scalability

ASP-3 - Avoid frequently scaling up or down

It's recommended that you avoid frequently scaling up or down your Azure App Service instances. Instead, choose an appropriate tier and instance size that can handle your

typical workload, and scale out the instances to accommodate changes in traffic volume. Scaling up or down can potentially trigger an application restart, which may result in service disruptions.

⚠ ASP-5 - Enable Autoscale/Automatic scaling to ensure that adequate resources are available to service requests

It's recommended that you enable autoscale/automatic scaling for your Azure App Service to ensure that sufficient resources are available to handle incoming requests. Autoscaling is rule based scaling, while automatic scaling performs automatic in and out scaling based on HTTP traffic. For more information see, automatic scaling in Azure App Service or get started with autoscale in Azure.

Azure Resource Graph

Kusto

```
// Azure Resource Graph Query
// Azure Resource Graph
// App Service Plans that are not setup to use standard or premium SKUs.
// TODO: Extend this query to show automatic scaling or App level
automatic scale
resources
| where type =~ "microsoft.insights/autoscalesettings"
| where properties.targetResourceUri contains
"microsoft.web/serverfarms"
| project AppServicePlan = split(properties.targetResourceUri, '/')[8],
ResourceGroup = split(properties.targetResourceUri, '/')
[4],id=properties.targetResourceUri
| project recommendationid="asp-5", ["name"]=AppServicePlan, ["id"]=id
```

Availability zone support

Azure availability zones are at least three physically separate groups of datacenters within each Azure region. Datacenters within each zone are equipped with independent power, cooling, and networking infrastructure. In the case of a local zone failure, availability zones are designed so that if the one zone is affected, regional services, capacity, and high availability are supported by the remaining two zones.

Failures can range from software and hardware failures to events such as earthquakes, floods, and fires. Tolerance to failures is achieved with redundancy and logical isolation of Azure services. For more detailed information on availability zones in Azure, see Regions and availability zones.

Azure availability zones-enabled services are designed to provide the right level of reliability and flexibility. They can be configured in two ways. They can be either zone redundant, with automatic replication across zones, or zonal, with instances pinned to a specific zone. You can also combine these approaches. For more information on zonal vs. zone-redundant architecture, see Recommendations for using availability zones and regions.

Azure App Service can be deployed across availability zones (AZ) to help you achieve resiliency and reliability for your business-critical workloads. This architecture is also known as zone redundancy.

When you configure to be zone redundant, the platform automatically spreads the instances of the Azure App Service plan across three zones in the selected region. This means that the minimum App Service Plan instance count will always be three. If you specify a capacity larger than three, and the number of instances is divisible by three, the instances are spread evenly. Otherwise, instance counts beyond 3*N are spread across the remaining one or two zones.

Availability zone support is a property of the App Service plan. App Service plans can be created on managed multi-tenant environment or dedicated environment using App Service Environment v3. To Learn more regarding App Service Environment v3, see App Service Environment v3 overview.

For App Services that aren't configured to be zone redundant, VM instances are not zone resilient and can experience downtime during an outage in any zone in that region.

For information on enterprise deployment architecture, see High availability enterprise deployment using App Service Environment.

Prerequisites

The current requirements/limitations for enabling availability zones are:

- Both Windows and Linux are supported.
- Availability zones are only supported on the newer App Service footprint. Even if you're using one of the supported regions, you'll receive an error if availability zones aren't supported for your resource group. To ensure your workloads land on a stamp that supports availability zones, you may need to create a new resource group, App Service plan, and App Service.

- Your App Services plan must be one of the following plans that support availability zones:
  - In a multi-tenant environment using App Service Premium v2 or Premium v3 plans.
  - In a dedicated environment using App Service Environment v3, which is used with Isolated v2 App Service plans.

- For dedicated environments, your App Service Environment must be v3.

> **Important**
>
> **App Service Environment v2 and v1 will be retired on 31 August 2024**. App Service Environment v3 is easier to use and runs on more powerful infrastructure. To learn more about App Service Environment v3, see **App Service Environment overview**. If you're currently using App Service Environment v2 or v1 and you want to upgrade to v3, please follow the **steps in this article** to migrate to the new version.

- Minimum instance count of three zones is enforced. The platform will enforce this minimum count behind the scenes if you specify an instance count fewer than three.

- Availability zones can only be specified when creating a **new** App Service plan. A pre-existing App Service plan can't be converted to use availability zones.

- The following regions support Azure App Services running on multi-tenant environments:
  - Australia East
  - Brazil South
  - Canada Central
  - Central India
  - Central US
  - East Asia
  - East US
  - East US 2
  - France Central
  - Germany West Central
  - Japan East
  - Korea Central
  - North Europe
  - Norway East

- Poland Central
- Qatar Central
- South Africa North
- South Central US
- Southeast Asia
- Sweden Central
- Switzerland North
- UAE North
- UK South
- West Europe
- West US 2
- West US 3
- Microsoft Azure operated by 21Vianet - China North 3
- Azure Government - US Gov Virginia

- To see which regions support availability zones for App Service Environment v3, see Regions.

Create a resource with availability zone enabled

To deploy a multi-tenant zone-redundant App Service

Azure CLI

To enable availability zones using the Azure CLI, include the `--zone-redundant` parameter when you create your App Service plan. You can also include the `--number-of-workers` parameter to specify capacity. If you don't specify a capacity, the platform defaults to three. Capacity should be set based on the workload requirement, but no less than three. A good rule of thumb to choose capacity is to ensure sufficient instances for the application such that losing one zone of instances leaves sufficient capacity to handle expected load.

Azure CLI

```
az appservice plan create --resource-group MyResourceGroup --name MyPlan
--sku P1v2 --zone-redundant --number-of-workers 6
```

Tip

To decide instance capacity, you can use the following calculation:

Since the platform spreads VMs across three zones and you need to account for at least the failure of one zone, multiply peak workload instance count by a factor of zones/(zones-1), or 3/2. For example, if your typical peak workload requires four instances, you should provision six instances: (2/3 * 6 instances) = 4 instances.

Deploy a zone-redundant App Service using a dedicated environment

To learn how to create an App Service Environment v3 on the Isolated v2 plan, see Create an App Service Environment.

Troubleshooting

Error message	Description	Recommendation
Zone redundancy is not available for resource group 'RG-NAME'. Please deploy app service plan 'ASP-NAME' to a new resource group.	Availability zones are only supported on the newer App Service footprint. Even if you're using one of the supported regions, you'll receive an error if availability zones aren't supported for your resource group.	To ensure your workloads land on a stamp that supports availability zones, create a new resource group, App Service plan, and App Service.

Fault tolerance

To prepare for availability zone failure, you should over-provision capacity of service to ensure that the solution can tolerate 1/3 loss of capacity and continue to function without degraded performance during zone-wide outages. Since the platform spreads VMs across three zones and you need to account for at least the failure of one zone, multiply peak workload instance count by a factor of zones/(zones-1), or 3/2. For example, if your typical peak workload requires four instances, you should provision six instances: (2/3 * 6 instances) = 4 instances.

Zone down experience

Traffic is routed to all of your available App Service instances. In the case when a zone goes down, the App Service platform will detect lost instances and automatically attempt to find new replacement instances and spread traffic as needed. If you have autoscale configured, and if it decides more instances are needed, autoscale will also

issue a request to App Service to add more instances. Note that autoscale behavior is independent of App Service platform behavior and that your autoscale instance count specification doesn't need to be a multiple of three.

> **Note**
>
> There's no guarantee that requests for additional instances in a zone-down scenario will succeed. The back filling of lost instances occurs on a best-effort basis. The recommended solution is to create and configure your App Service plans to account for losing a zone as described in the next section.

Applications that are deployed in an App Service plan that has availability zones enabled will continue to run and serve traffic even if other zones in the same region suffer an outage. However it's possible that non-runtime behaviors including App Service plan scaling, application creation, application configuration, and application publishing may still be impacted from an outage in other Availability Zones. Zone redundancy for App Service plans only ensures continued uptime for deployed applications.

When the App Service platform allocates instances to 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 will be "balanced" if each zone has either the same number of VMs, or +/- one VM in all of the other zones used by the App Service plan.

Availability zone migration

You cannot migrate existing App Service instances or environment resources from non-availability zone support to availability zone support. To get support for availability zones, you'll need to create your resources with availability zones enabled.

Pricing

There's no additional cost associated with enabling availability zones. Pricing for a zone redundant App Service is the same as a single zone App Service. You'll be charged based on your App Service plan SKU, the capacity you specify, and any instances you scale to based on your autoscale criteria. If you enable availability zones but specify a capacity less than three, the platform will enforce a minimum instance count of three and charge you for those three instances. For pricing information for App Service Environment v3, see Pricing.

Cross-region disaster recovery and business continuity

Disaster recovery (DR) is about recovering from high-impact events, such as natural disasters or failed deployments, that result in downtime and data loss. Regardless of the cause, the best remedy for a disaster is a well-defined and tested DR plan and an application design that actively supports DR.

When it comes to DR, Microsoft uses the shared responsibility model. The shared responsibility models means that Microsoft ensures that the baseline infrastructure and platform services are available. However, in some scenarios, usage requires that the customer to duplicate their deployments and storage in a multi-region capacity, if they opt to.

Most services that run on Azure platform as a service (PaaS) offerings provide features and guidance to support DR. For some scenarios, you can use service-specific features to support fast recovery.

This section covers some common strategies for web apps deployed to App Service.

When you create a web app in App Service and choose an Azure region during resource creation, it's a single-region app. When the region becomes unavailable during a disaster, your application also becomes unavailable. If you create an identical deployment in a secondary Azure region using a multi-region geography architecture, your application becomes less susceptible to a single-region disaster, which guarantees business continuity. Any data replication across the regions lets you recover your last application state.

For IT, business continuity plans are largely driven by Recovery Time Objective (RTO) and Recovery Point Objective (RPO). For more information on RTO and RPO, see Recovery objectives.

Normally, maintaining an SLA around RTO is impractical for regional disasters, and you would typically design your disaster recovery strategy around RPO alone (i.e. focus on recovering data and not on minimizing interruption). With Azure, however, it's not only practical but can even be straightforward to deploy App Service for automatic geo-failovers. This lets you disaster-proof your applications further by taking care of both RTO and RPO.

Depending on your desired RTO and RPO metrics, three disaster recovery architectures are commonly used for both App Service multitenant and App Service Environments. Each architecture is described in the following table:

Metric	Active-Active	Active-Passive	Passive/Cold
RTO	Real-time or seconds	Minutes	Hours
RPO	Real-time or seconds	Minutes	Hours
Cost	$$$	$$	$
Scenarios	Mission-critical apps	High-priority apps	Low-priority apps
Ability to serve multi-region user traffic	Yes	Yes/maybe	No
Code deployment	CI/CD pipelines preferred	CI/CD pipelines preferred	Backup and restore
Creation of new App Service resources during downtime	Not required	Not required	Required

> **Note**
>
> Your application most likely depends on other data services in Azure, such as Azure SQL Database and Azure Storage accounts. It's recommended that you develop disaster recovery strategies for each of these dependent Azure Services as well. For SQL Database, see **Active geo-replication for Azure SQL Database**. For Azure Storage, see **Azure Storage redundancy**.

Disaster recovery in multi-region geography

There are multiple ways to replicate your web apps content and configurations across Azure regions in an active-active or active-passive architecture, such as using App service backup and restore. However, backup and restore create point-in-time snapshots and eventually lead to web app versioning challenges across regions. See the following table below for a comparison between back and restore guidance vs. diaster recovery guidance:

Back up & restore vs. disaster recovery

Platform	Back up & restore guidance	Disaster recovery guidance
App Service Web Apps (Free & Shared Pricing Tier)	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 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 region as explained in the recover from region-wide failure article. It's recommended to implement commonly used disaster recovery techniques to prevent downtime and data loss during a regional disaster.
App Service Web Apps (Basic\Standard\Premium Pricing Tier)	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 by restoring to a new app or slot. Refer to Back up and restore your app in Azure App Service for more info.	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. 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.
App Service Environment (V2 & V3)	In Azure App Service Environment, You can make on-demand custom backups or utilize automatic backups. Automatic backups can be restored to a target app within the same ASE, not in another ASE. Custom backups can be restored to a target app in another ASE (such as from a V2 ASE to a V3 ASE). Backups can	We encourage you to implement disaster recovery guidance for web apps deployed to App Service Environment using commonly used disaster recovery techniques.

Platform	Back up & restore guidance	Disaster recovery guidance
	be restored to target app of the same OS platform as the source app. Refer to Back up and restore your app in Azure App Service for more details.	
Azure Functions (Dedicated)	In Azure Functions, You can make on-demand custom backups or utilize automatic backups. You can restore a backup by overwriting an existing app by restoring to a new app or slot. Refer to Back up and restore your app in Azure App Service for more info.	Current guidance regarding how to bring Azure Functions app (dedicated) resources back online in a different Azure region during a regional disaster is available at Recover from region-wide failure - Azure App Service. 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. Instead, implement Azure Functions geo-disaster recovery. Additionally you may also refer to commonly used disaster recovery techniques for Azure Functions dedicated.
Azure Functions Consumption & Premium	Azure functions deployed to consumption and premium plans don't provide access to custom and automatic backups. Function app's content is in an Azure storage account. Use Azure Storage redundancy options to ensure your storage account meets its availability and durability targets during an outage. If you created your functions by using the editor in the Azure portal, you can also download your existing function app project as a .zip file.	We strongly encourage you to implement Azure Functions geo-disaster recovery and reliability.

To avoid the limitations of backup and restore methods, configure your CI/CD pipelines to deploy code to both Azure regions. Consider using Azure Pipelines or GitHub Actions. For more information, see Continuous deployment to Azure App Service.

Outage detection, notification, and management

- It's recommended that you set up monitoring and alerts for your web apps to for timely notifications during a disaster. For more information, see Application Insights availability tests.
- To manage your application resources in Azure, use an infrastructure-as-Code (IaC) mechanism. In a complex deployment across multiple regions, to manage the regions independently and to keep the configuration synchronized across regions in a reliable manner requires a predictable, testable, and repeatable process. Consider an IaC tool such as Azure Resource Manager templates or Terraform.

Set up disaster recovery and outage detection

To prepare for disaster recovery in a multi-region geography, you can use either an active-active or active-passive architecture.

Active-Active architecture

In active-active disaster recovery architecture, identical web apps are deployed in two separate regions and Azure Front door is used to route traffic to both the active regions.

With this example architecture:

- Identical App Service apps are deployed in two separate regions, including pricing tier and instance count.
- Public traffic directly to the App Service apps is blocked.
- Azure Front Door is used to route traffic to both the active regions.
- During a disaster, one of the regions becomes offline, and Azure Front Door routes traffic exclusively to the region that remains online. The RTO during such a geo-failover is near-zero.
- Application files should be deployed to both web apps with a CI/CD solution. This ensures that the RPO is practically zero.
- If your application actively modifies the file system, the best way to minimize RPO is to only write to a mounted Azure Storage share instead of writing directly to the web app's */home* content share. Then, use the Azure Storage redundancy features (GZRS or GRS) for your mounted share, which has an RPO of about 15 minutes.

Steps to create an active-active architecture for your web app in App Service are summarized as follows:

1. Create two App Service plans in two different Azure regions. Configure the two App Service plans identically.

2. Create two instances of your web app, with one in each App Service plan.

3. Create an Azure Front Door profile with:

    - An endpoint.
    - Two origin groups, each with a priority of 1. The equal priority tells Azure Front Door to route traffic to both regions equally (thus active-active).
    - A route.

4. Limit network traffic to the web apps only from the Azure Front Door instance.

5. Setup and configure all other back-end Azure service, such as databases, storage accounts, and authentication providers.

6. Deploy code to both the web apps with continuous deployment.

Tutorial: Create a highly available multi-region app in Azure App Service shows you how to set up an *active-passive* architecture. The same steps with minimal changes (setting priority to “1” for both origin groups in Azure Front Door) give you an *active-active* architecture.

Active-passive architecture

In this disaster recovery approach, identical web apps are deployed in two separate regions and Azure Front door is used to route traffic to one region only (the *active* region).

With this example 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 to route traffic to the primary region.
- To save cost, the secondary App Service plan is configured to have fewer instances and/or be in a lower pricing tier. There are three possible approaches:
  - **Preferred** The secondary App Service plan has the same pricing tier as the primary, with the same number of instances or fewer. This approach ensures parity in both feature and VM sizing for the two App Service plans. The RTO during a geo-failover only depends on the time to scale out the instances.
  - **Less preferred** The secondary App Service plan has the same pricing tier type (such as PremiumV3) but smaller VM sizing, with lesser instances. For example, the primary region may be in P3V3 tier while the secondary region is in P1V3 tier. This approach still ensures feature parity for the two App Service plans, but the lack of size parity may require a manual scale-up when the secondary region becomes the active region. The RTO during a geo-failover depends on the time to both scale up and scale out the instances.

- **Least-preferred** The secondary App Service plan has a different pricing tier than the primary and lesser instances. For example, the primary region may be in P3V3 tier while the secondary region is in S1 tier. Make sure that the secondary App Service plan has all the features your application needs in order to run. Differences in features availability between the two may cause delays to your web app recovery. The RTO during a geo-failover depends on the time to both scale up and scale out the instances.

- Autoscale is configured on the secondary region in the event the active region becomes inactive. It's advisable to have similar autoscale rules in both active and passive regions.

- During a disaster, the primary region becomes inactive, and the secondary region starts receiving traffic and becomes the active region.

- Once the secondary region becomes active, the network load triggers preconfigured autoscale rules to scale out the secondary web app.

- You may need to scale up the pricing tier for the secondary region manually, if it doesn't already have the needed features to run as the active region. For example, autoscaling requires Standard tier or higher.

- When the primary region is active again, Azure Front Door automatically directs traffic back to it, and the architecture is back to active-passive as before.

- Application files should be deployed to both web apps with a CI/CD solution. This ensures that the RPO is practically zero.

- If your application actively modifies the file system, the best way to minimize RPO is to only write to a mounted Azure Storage share instead of writing directly to the web app's */home* content share. Then, use the Azure Storage redundancy features (GZRS or GRS) for your mounted share, which has an RPO of about 15 minutes.

Steps to create an active-passive architecture for your web app in App Service are summarized as follows:

1. Create two App Service plans in two different Azure regions. The secondary App Service plan may be provisioned using one of the approaches mentioned previously.
2. Configure autoscaling rules for the secondary App Service plan so that it scales to the same instance count as the primary when the primary region becomes inactive.
3. Create two instances of your web app, with one in each App Service plan.
4. Create an Azure Front Door profile with:

    - An endpoint.
    - An origin group with a priority of 1 for the primary region.
    - A second origin group with a priority of 2 for the secondary region. The difference in priority tells Azure Front Door to prefer the primary region when it's online (thus active-passive).
    - A route.
5. Limit network traffic to the web apps only from the Azure Front Door instance.
6. Setup and configure all other back-end Azure service, such as databases, storage accounts, and authentication providers.
7. Deploy code to both the web apps with continuous deployment.

Tutorial: Create a highly available multi-region app in Azure App Service shows you how to set up an *active-passive* architecture.

Passive-cold architecture

Use a passive/cold architecture to create and maintain regular backups of your web apps in an Azure Storage account that's located in another region.

With this example architecture:

- A single web app is deployed to a single region.
- The web app is regularly backed up to an Azure Storage account in the same region.
- The cross-region replication of your backups depends on the data redundancy configuration in the Azure storage account. You should set your Azure Storage account as GZRS if possible. GZRS offers both synchronous zone redundancy within a region and asynchronous in a secondary region. If GZRS isn't available, configure the account as GRS. Both GZRS and GRS have an RPO of about 15 minutes.
- To ensure that you can retrieve backups when the storage account's primary region becomes unavailable, **enable read only access to secondary region** (making the storage account **RA-GZRS** or **RA-GRS**, respectively). For more information on designing your applications to take advantage of geo-redundancy, see Use geo-redundancy to design highly available applications.
- During a disaster in the web app's region, you must manually deploy all required App Service dependent resources by using the backups from the Azure Storage account, most likely from the secondary region with read access. The RTO may be hours or days.

- To minimize RTO, it's highly recommended that you have a comprehensive playbook outlining all the steps required to restore your web app backup to another Azure Region.

Steps to create a passive-cold region for your web app in App Service are summarized as follows:

1. Create an Azure storage account in the same region as your web app. Choose Standard performance tier and select redundancy as Geo-redundant storage (GRS) or Geo-Zone-redundant storage (GZRS).

2. Enable RA-GRS or RA-GZRS (read access for the secondary region).

3. Configure custom backup for your web app. You may decide to set a schedule for your web app backups, such as hourly.

4. Verify that the web app backup files can be retrieved the secondary region of your storage account.

> **Tip**
>
> Aside from Azure Front Door, Azure provides other load balancing options, such as Azure Traffic Manager. For a comparison of the various options, see **Load-balancing options - Azure Architecture Center**.

Disaster recovery in single-region geography

If your web app's region doesn't have GZRS or GRS storage or if you are in an Azure region that isn't one of a regional pair, you'll need to utilize zone-redundant storage (ZRS) or locally redundant storage (LRS) to create a similar architecture. For example, you can manually create a secondary region for the storage account as follows:

Steps to create a passive-cold region without GRS and GZRS are summarized as follows:

1. Create an Azure storage account in the same region of your web app. Choose Standard performance tier and select redundancy as zone-redundant storage (ZRS).

2. Configure custom backup for your web app. You may decide to set a schedule for your web app backups, such as hourly.

3. Verify that the web app backup files can be retrieved the secondary region of your storage account.

4. Create a second Azure storage account in a different region. Choose Standard performance tier and select redundancy as locally redundant storage (LRS).

5. By using a tool like AzCopy, replicate your custom backup (Zip, XML and log files) from primary region to the secondary storage. For example:

   ```
   azcopy copy 'https://<source-storage-account-
   name>.blob.core.windows.net/<container-name>/<blob-path>'
   'https://<destination-storage-account-
   name>.blob.core.windows.net/<container-name>/<blob-path>'
   ```

   You can use Azure Automation with a PowerShell Workflow runbook to run your replication script on a schedule. Make sure that the replication schedule follows a similar schedule to the web app backups.

Next steps

- Tutorial: Create a highly available multi-region app in Azure App Service
- Reliability in Azure

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.

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	

4. Configure everything else in the target app to mirror the impacted app and verify your configuration.

5. 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**.

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.

Create an App Service app and deploy files with FTP using Azure CLI

Article • 01/13/2023

This sample script creates an app in App Service with its related resources, and then deploys a static HTML page using FTP. For FTP upload, the script uses cURL as an example. You can use whatever FTP tool to upload your files.

If you don't have an Azure subscription, create an Azure free account before you begin.

Prerequisites

- 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.

Sample script

Launch Azure Cloud Shell

The Azure Cloud Shell is a free interactive shell that you can use to run the steps in this article. It has common Azure tools preinstalled and configured to use with your account.

To open the Cloud Shell, just select **Try it** from the upper right corner of a code block. You can also launch Cloud Shell in a separate browser tab by going to https://shell.azure.com.

When Cloud Shell opens, verify that **Bash** is selected for your environment. Subsequent sessions will use Azure CLI in a Bash environment, Select **Copy** to copy the blocks of code, paste it into the Cloud Shell, and press **Enter** to run it.

Sign in to Azure

Cloud Shell is automatically authenticated under the initial account signed-in with. Use the following script to sign in using a different subscription, replacing `<Subscription ID>` with your Azure Subscription ID. If you don't have an Azure subscription, create an Azure free account before you begin.

Azure CLI

```
subscription="<subscriptionId>" # add subscription here

az account set -s $subscription # ...or use 'az login'
```

For more information, see set active subscription or log in interactively

Run the script

Azure CLI

```
# Create an App Service app and deploy files with FTP
# set -e # exit if error
# Variable block
let "randomIdentifier=$RANDOM*$RANDOM"
location="East US"
resourceGroup="msdocs-app-service-rg-$randomIdentifier"
tag="deploy-ftp.sh"
warurl="https://raw.githubusercontent.com/Azure-Samples/html-docs-hello-
world/master/index.html"
appServicePlan="msdocs-app-service-plan-$randomIdentifier"
webapp="msdocs-web-app-$randomIdentifier"

# Download sample static HTML page
curl $warurl --output index.html

# Create a resource group.
echo "Creating $resourceGroup in "$location"..."
az group create --name $resourceGroup --location "$location" --tag $tag

# Create an App Service plan in `FREE` tier.
```

```
echo "Creating $appServicePlan"
az appservice plan create --name $appServicePlan --resource-group
$resourceGroup --sku FREE

# Create a web app.
echo "Creating $webapp"
az webapp create --name $webapp --resource-group $resourceGroup --plan
$appServicePlan

# Get FTP publishing profile and query for publish URL and credentials
creds=($(az webapp deployment list-publishing-profiles --name $webapp --
resource-group $resourceGroup \
--query "[?contains(publishMethod, 'FTP')].[publishUrl,userName,userPWD]" --
output tsv))

# Use cURL to perform FTP upload. You can use any FTP tool to do this
instead.
curl -T index.html -u ${creds[1]}:${creds[2]} ${creds[0]}/

# Copy the result of the following command into a browser to see the static
HTML site.
site="http://$webapp.azurewebsites.net"
echo $site
curl "$site"
```

Clean up resources

Use the following command to remove the resource group and all resources associated with it using the az group delete command - unless you have an ongoing need for these resources. Some of these resources may take a while to create, as well as to delete.

Azure CLI

```
az group delete --name $resourceGroup
```

Sample reference

This script uses the following commands. Each command in the table links to command specific documentation.

Command	Notes
az group create	Creates a resource group in which all resources are stored.
az appservice plan create	Creates an App Service plan.

Command	Notes
az webapp create	Creates an App Service app.
az webapp deployment list-publishing-profiles	Get the details for available app deployment profiles.

Next steps

For more information on the Azure CLI, see Azure CLI documentation.

Additional App Service CLI script samples can be found in the Azure App Service documentation.

Create an App Service app with deployment from GitHub using Azure CLI

Article • 01/13/2023

This sample script creates an app in App Service with its related resources. It then deploys your app code from a public GitHub repository (without continuous deployment). For GitHub deployment with continuous deployment, see Create an app with continuous deployment from GitHub.

If you don't have an Azure subscription, create an Azure free account before you begin.

Prerequisites

- 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.

Sample script

Launch Azure Cloud Shell

The Azure Cloud Shell is a free interactive shell that you can use to run the steps in this article. It has common Azure tools preinstalled and configured to use with your account.

To open the Cloud Shell, just select **Try it** from the upper right corner of a code block. You can also launch Cloud Shell in a separate browser tab by going to https://shell.azure.com.

When Cloud Shell opens, verify that **Bash** is selected for your environment. Subsequent sessions will use Azure CLI in a Bash environment, Select **Copy** to copy the blocks of code, paste it into the Cloud Shell, and press **Enter** to run it.

Sign in to Azure

Cloud Shell is automatically authenticated under the initial account signed-in with. Use the following script to sign in using a different subscription, replacing `<Subscription ID>` with your Azure Subscription ID. If you don't have an Azure subscription, create an Azure free account before you begin.

Azure CLI

```
subscription="<subscriptionId>" # add subscription here

az account set -s $subscription # ...or use 'az login'
```

For more information, see set active subscription or log in interactively

Run the script

Azure CLI

```
# Create an App Service app with deployment from GitHub
# set -e # exit if error
# Variable block
let "randomIdentifier=$RANDOM*$RANDOM"
location="East US"
resourceGroup="msdocs-app-service-rg-$randomIdentifier"
tag="deploy-github.sh"
gitrepo=https://github.com/Azure-Samples/php-docs-hello-world # Replace the
following URL with your own public GitHub repo URL if you have one
appServicePlan="msdocs-app-service-plan-$randomIdentifier"
webapp="msdocs-web-app-$randomIdentifier"

# Create a resource group.
echo "Creating $resourceGroup in "$location"..."
az group create --name $resourceGroup --location "$location" --tag $tag
```

```
# Create an App Service plan in `FREE` tier.
echo "Creating $appServicePlan"
az appservice plan create --name $appServicePlan --resource-group
$resourceGroup --sku FREE

# Create a web app.
echo "Creating $webapp"
az webapp create --name $webapp --resource-group $resourceGroup --plan
$appServicePlan

# Deploy code from a public GitHub repository.
az webapp deployment source config --name $webapp --resource-group
$resourceGroup \
--repo-url $gitrepo --branch master --manual-integration

# Use curl to see the web app.
site="http://$webapp.azurewebsites.net"
echo $site
curl "$site" # Optionally, copy and paste the output of the previous command
into a browser to see the web app
```

Clean up resources

Use the following command to remove the resource group and all resources associated with it using the az group delete command - unless you have an ongoing need for these resources. Some of these resources may take a while to create, as well as to delete.

Azure CLI

```
az group delete --name $resourceGroup
```

Sample reference

This script uses the following commands. Each command in the table links to command specific documentation.

Command	Notes
az group create	Creates a resource group in which all resources are stored.
az appservice plan create	Creates an App Service plan.
az webapp create	Creates an App Service app.
az webapp deployment source config	Associates an App Service app with a Git or Mercurial repository.

Next steps

For more information on the Azure CLI, see Azure CLI documentation.

Additional App Service CLI script samples can be found in the Azure App Service documentation.

Create an App Service app with continuous deployment from GitHub using CLI

Article • 01/13/2023

This sample script creates an app in App Service with its related resources, and then sets up continuous deployment from a GitHub repository. For GitHub deployment without continuous deployment, see Create an app and deploy code from GitHub. For this sample, you need:

- A GitHub repository with application code, that you have administrative permissions for. To get automatic builds, structure your repository according to the Prepare your repository table.
- A Personal Access Token (PAT) for your GitHub account.

If you don't have an Azure subscription, create an Azure free account before you begin.

Prerequisites

- 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.

Sample script

Launch Azure Cloud Shell

The Azure Cloud Shell is a free interactive shell that you can use to run the steps in this article. It has common Azure tools preinstalled and configured to use with your account.

To open the Cloud Shell, just select **Try it** from the upper right corner of a code block. You can also launch Cloud Shell in a separate browser tab by going to https://shell.azure.com.

When Cloud Shell opens, verify that **Bash** is selected for your environment. Subsequent sessions will use Azure CLI in a Bash environment, Select **Copy** to copy the blocks of code, paste it into the Cloud Shell, and press **Enter** to run it.

Sign in to Azure

Cloud Shell is automatically authenticated under the initial account signed-in with. Use the following script to sign in using a different subscription, replacing `<Subscription ID>` with your Azure Subscription ID. If you don't have an Azure subscription, create an Azure free account before you begin.

Azure CLI

```
subscription="<subscriptionId>" # add subscription here

az account set -s $subscription # ...or use 'az login'
```

For more information, see set active subscription or log in interactively

To create the web app

Azure CLI

```
# Create an App Service app with deployment from GitHub
# set -e # exit if error
# Variable block
let "randomIdentifier=$RANDOM*$RANDOM"
location="East US"
resourceGroup="msdocs-app-service-rg-$randomIdentifier"
tag="deploy-github.sh"
gitrepo=https://github.com/Azure-Samples/php-docs-hello-world # Replace the
following URL with your own public GitHub repo URL if you have one
appServicePlan="msdocs-app-service-plan-$randomIdentifier"
```

```
webapp="msdocs-web-app-$randomIdentifier"

# Create a resource group.
echo "Creating $resourceGroup in "$location"..."
az group create --name $resourceGroup --location "$location" --tag $tag

# Create an App Service plan in `FREE` tier.
echo "Creating $appServicePlan"
az appservice plan create --name $appServicePlan --resource-group
$resourceGroup --sku FREE

# Create a web app.
echo "Creating $webapp"
az webapp create --name $webapp --resource-group $resourceGroup --plan
$appServicePlan

# Deploy code from a public GitHub repository.
az webapp deployment source config --name $webapp --resource-group
$resourceGroup \
--repo-url $gitrepo --branch master --manual-integration

# Use curl to see the web app.
site="http://$webapp.azurewebsites.net"
echo $site
curl "$site" # Optionally, copy and paste the output of the previous command
into a browser to see the web app
```

To configure continuous deployment from GitHub

1. Create the following variables containing your GitHub information.

   Azure CLI

   ```
   gitrepo=<replace-with-URL-of-your-own-GitHub-repo>
   token=<replace-with-a-GitHub-access-token>
   ```

2. Configure continuous deployment from GitHub.

   > **Tip**
   >
   > The `--git-token` parameter is required only once per Azure account (Azure remembers token).

   Azure CLI

   ```
   az webapp deployment source config --name $webapp --resource-group
   $resourceGroup --repo-url $gitrepo --branch master --git-token $token
   ```

Clean up resources

Use the following command to remove the resource group and all resources associated with it using the az group delete command - unless you have an ongoing need for these resources. Some of these resources may take a while to create, as well as to delete.

Azure CLI

```
az group delete --name $resourceGroup
```

Sample reference

This script uses the following commands. Each command in the table links to command specific documentation.

Command	Notes
az group create	Creates a resource group in which all resources are stored.
az appservice plan create	Creates an App Service plan.
az webapp create	Creates an App Service app.
az webapp deployment source config	Associates an App Service app with a Git or Mercurial repository.

Next steps

For more information on the Azure CLI, see Azure CLI documentation.

Additional App Service CLI script samples can be found in the Azure App Service documentation.

Create an App Service app and deploy code into a local Git repository using Azure CLI

Article • 01/13/2023

This sample script creates an app in App Service with its related resources, and then deploys your app code in a local Git repository.

If you don't have an Azure subscription, create an Azure free account before you begin.

Prerequisites

- 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.

Sample script

Launch Azure Cloud Shell

The Azure Cloud Shell is a free interactive shell that you can use to run the steps in this article. It has common Azure tools preinstalled and configured to use with your account.

To open the Cloud Shell, just select **Try it** from the upper right corner of a code block. You can also launch Cloud Shell in a separate browser tab by going to https://shell.azure.com.

When Cloud Shell opens, verify that **Bash** is selected for your environment. Subsequent sessions will use Azure CLI in a Bash environment, Select **Copy** to copy the blocks of code, paste it into the Cloud Shell, and press **Enter** to run it.

Sign in to Azure

Cloud Shell is automatically authenticated under the initial account signed-in with. Use the following script to sign in using a different subscription, replacing `<Subscription ID>` with your Azure Subscription ID. If you don't have an Azure subscription, create an Azure free account before you begin.

Azure CLI

```
subscription="<subscriptionId>" # add subscription here

az account set -s $subscription # ...or use 'az login'
```

For more information, see set active subscription or log in interactively

To create the web app

Azure CLI

```
# Create an App Service app with deployment from GitHub
# set -e # exit if error
# Variable block
let "randomIdentifier=$RANDOM*$RANDOM"
location="East US"
resourceGroup="msdocs-app-service-rg-$randomIdentifier"
tag="deploy-github.sh"
gitrepo=https://github.com/Azure-Samples/php-docs-hello-world # Replace the
following URL with your own public GitHub repo URL if you have one
appServicePlan="msdocs-app-service-plan-$randomIdentifier"
webapp="msdocs-web-app-$randomIdentifier"

# Create a resource group.
echo "Creating $resourceGroup in "$location"..."
az group create --name $resourceGroup --location "$location" --tag $tag

# Create an App Service plan in `FREE` tier.
echo "Creating $appServicePlan"
az appservice plan create --name $appServicePlan --resource-group
$resourceGroup --sku FREE
```

```
# Create a web app.
echo "Creating $webapp"
az webapp create --name $webapp --resource-group $resourceGroup --plan
$appServicePlan

# Deploy code from a public GitHub repository.
az webapp deployment source config --name $webapp --resource-group
$resourceGroup \
--repo-url $gitrepo --branch master --manual-integration

# Use curl to see the web app.
site="http://$webapp.azurewebsites.net"
echo $site
curl "$site" # Optionally, copy and paste the output of the previous command
into a browser to see the web app
```

To deploy to your local Git repository

1. Create the following variables containing your GitHub information.

   Azure CLI

   ```
   gitdirectory=<Replace with path to local Git repo>
   username=<Replace with desired deployment username>
   password=<Replace with desired deployment password>
   ```

2. Configure local Git and get deployment URL.

   Azure CLI

   ```
   url=$(az webapp deployment source config-local-git --name $webapp --
   resource-group $resourceGroup --query url --output tsv)
   ```

3. Add the Azure remote to your local Git repository and push your code. When prompted for password, use the value of $password that you specified.

   Bash

   ```
   cd $gitdirectory
   git remote add azure $url
   git push azure main
   ```

Clean up resources

Use the following command to remove the resource group and all resources associated with it using the az group delete command - unless you have an ongoing need for these resources. Some of these resources may take a while to create, as well as to delete.

Azure CLI

```
az group delete --name $resourceGroup
```

Sample reference

This script uses the following commands. Each command in the table links to command specific documentation.

Command	Notes
az group create	Creates a resource group in which all resources are stored.
az appservice plan create	Creates an App Service plan.
az webapp create	Creates an App Service app.
az webapp deployment user set	Sets the account-level deployment credentials for App Service.
az webapp deployment source config-local-git	Creates a source control configuration for a local Git repository.

Next steps

For more information on the Azure CLI, see Azure CLI documentation.

Additional App Service CLI script samples can be found in the Azure App Service documentation.

Create an App Service app and deploy code to a staging environment using Azure CLI

Article • 01/13/2023

This sample script creates an app in App Service with an additional deployment slot called "staging", and then deploys a sample app to the "staging" slot.

If you don't have an Azure subscription, create an Azure free account before you begin.

Prerequisites

- 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.

Sample script

Launch Azure Cloud Shell

The Azure Cloud Shell is a free interactive shell that you can use to run the steps in this article. It has common Azure tools preinstalled and configured to use with your account.

To open the Cloud Shell, just select **Try it** from the upper right corner of a code block. You can also launch Cloud Shell in a separate browser tab by going to https://shell.azure.com.

When Cloud Shell opens, verify that **Bash** is selected for your environment. Subsequent sessions will use Azure CLI in a Bash environment, Select **Copy** to copy the blocks of code, paste it into the Cloud Shell, and press **Enter** to run it.

Sign in to Azure

Cloud Shell is automatically authenticated under the initial account signed-in with. Use the following script to sign in using a different subscription, replacing `<Subscription ID>` with your Azure Subscription ID. If you don't have an Azure subscription, create an Azure free account before you begin.

Azure CLI

```
subscription="<subscriptionId>" # add subscription here

az account set -s $subscription # ...or use 'az login'
```

For more information, see set active subscription or log in interactively

Run the script

Azure CLI

```
# Create an App Service app and deploy code to a staging environment
# set -e # exit if error
# Variable block
let "randomIdentifier=$RANDOM*$RANDOM"
location="East US"
resourceGroup="msdocs-app-service-rg-$randomIdentifier"
tag="deploy-deployment-slot.sh"
appServicePlan="msdocs-app-service-plan-$randomIdentifier"
webapp="msdocs-web-app-$randomIdentifier"
gitrepo=https://github.com/Azure-Samples/php-docs-hello-world # Replace with
your public GitHub repo URL

# Create a resource group.
echo "Creating $resourceGroup in "$location"..."
az group create --name $resourceGroup --location "$location" --tag $tag

# Create an App Service plan in STANDARD tier (minimum required by
deployment slots).
echo "Creating $appServicePlan"
az appservice plan create --name $appServicePlan --resource-group
```

```
$resourceGroup --location "$location" \
--sku S1

# Create a web app
echo "Creating $webapp"
az webapp create --name $webapp --plan $appServicePlan --resource-group
$resourceGroup

# Create a deployment slot with the name "staging".
az webapp deployment slot create --name $webapp --resource-group
$resourceGroup --slot staging

# Deploy sample code to "staging" slot from GitHub.
az webapp deployment source config --name $webapp --resource-group
$resourceGroup --slot staging --repo-url $gitrepo --branch master --manual-
integration

# Copy the result of the following command into a browser to see the staging
slot.
site="http://$webapp-staging.azurewebsites.net"
echo $site
curl "$site"

# Swap the verified/warmed up staging slot into production.
az webapp deployment slot swap --name $webapp --resource-group
$resourceGroup \
--slot staging

# Copy the result of the following command into a browser to see the web app
in the production slot.
site="http://$webapp.azurewebsites.net"
echo $site
curl "$site"
```

Clean up resources

Use the following command to remove the resource group and all resources associated with it using the az group delete command - unless you have an ongoing need for these resources. Some of these resources may take a while to create, as well as to delete.

Azure CLI

```
az group delete --name $resourceGroup
```

Sample reference

This script uses the following commands. Each command in the table links to command specific documentation.

Command	Notes
az group create	Creates a resource group in which all resources are stored.
az appservice plan create	Creates an App Service plan.
az webapp create	Creates an App Service app.
az webapp deployment slot create	Create a deployment slot.
az webapp deployment source config	Associates an App Service app with a Git or Mercurial repository.
az webapp deployment slot swap	Swap a specified deployment slot into production.

Next steps

For more information on the Azure CLI, see Azure CLI documentation.

Additional App Service CLI script samples can be found in the Azure App Service documentation.

Create an ASP.NET Core app in a Docker container from Docker Hub using Azure CLI

Article • 01/13/2023

This sample script creates a resource group, a Linux App Service plan, and an app. It then deploys an ASP.NET Core application using a Docker Container.

If you don't have an Azure subscription, create an Azure free account before you begin.

Prerequisites

- 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.

Sample script

Launch Azure Cloud Shell

The Azure Cloud Shell is a free interactive shell that you can use to run the steps in this article. It has common Azure tools preinstalled and configured to use with your account.

To open the Cloud Shell, just select **Try it** from the upper right corner of a code block. You can also launch Cloud Shell in a separate browser tab by going to https://shell.azure.com.

When Cloud Shell opens, verify that **Bash** is selected for your environment. Subsequent sessions will use Azure CLI in a Bash environment, Select **Copy** to copy the blocks of code, paste it into the Cloud Shell, and press **Enter** to run it.

Sign in to Azure

Cloud Shell is automatically authenticated under the initial account signed-in with. Use the following script to sign in using a different subscription, replacing `<Subscription ID>` with your Azure Subscription ID. If you don't have an Azure subscription, create an Azure free account before you begin.

Azure CLI

```
subscription="<subscriptionId>" # add subscription here

az account set -s $subscription # ...or use 'az login'
```

For more information, see set active subscription or log in interactively

To create the web app

Azure CLI

```
# Create an ASP.NET Core app in a Docker container from Docker Hub
# set -e # exit if error
# Variable block
let "randomIdentifier=$RANDOM*$RANDOM"
location="East US"
resourceGroup="msdocs-app-service-rg-$randomIdentifier"
tag="deploy-linux-docker-app-only.sh"
appServicePlan="msdocs-app-service-plan-$randomIdentifier"
webapp="msdocs-web-app-$randomIdentifier"

# Create a resource group.
echo "Creating $resourceGroup in "$location"..."
az group create --name $resourceGroup --location "$location" --tag $tag

# Create an App Service plan in S1 tier
echo "Creating $appServicePlan"
az appservice plan create --name $appServicePlan --resource-group
$resourceGroup --sku S1 --is-linux

# Create a web app. To see list of available runtimes, run 'az webapp list-
```

```
runtimes --linux'
echo "Creating $webapp"
az webapp create --name $webapp --resource-group $resourceGroup --plan
$appServicePlan  --runtime "NODE|14-lts"

# Copy the result of the following command into a browser to see the static
HTML site.
site="http://$webapp.azurewebsites.net"
echo $site
curl "$site"
```

Configure Web App with a Custom Docker Container from Docker Hub

1. Create the following variable containing your GitHub information.

   Azure CLI

   ```
   dockerHubContainerPath="<replace-with-docker-container-path>" #format:
   <username>/<container-or-image>:<tag>
   ```

2. Configure the web app with a custom docker container from Docker Hub.

   Azure CLI

   ```
   az webapp config container set --docker-custom-image-name
   $dockerHubContainerPath --name $webApp --resource-group $resourceGroup
   ```

3. Copy the result of the following command into a browser to see the web app.

   Azure CLI

   ```
   site="http://$webapp.azurewebsites.net"
   echo $site
   curl "$site"
   ```

Clean up resources

Use the following command to remove the resource group and all resources associated with it using the az group delete command - unless you have an ongoing need for these resources. Some of these resources may take a while to create, as well as to delete.

Azure CLI

```
az group delete --name $resourceGroup
```

Sample reference

This script uses the following commands to create a resource group, App Service app, and all related resources. Each command in the table links to command specific documentation.

Command	Notes
az group create	Creates a resource group in which all resources are stored.
az appservice plan create	Creates an App Service plan.
az webapp create	Creates an App Service app.
az webapp config container set	Sets the Docker container for the App Service app.

Next steps

For more information on the Azure CLI, see Azure CLI documentation.

Additional App Service CLI script samples can be found in the Azure App Service documentation.

Create an App Service app and deploy Private Endpoint using Azure CLI

Article • 03/09/2023

This sample script creates an app in App Service with its related resources, and then deploys a Private Endpoint.

If you don't have an Azure subscription, create an Azure free account before you begin.

Prerequisites

- 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.
- This tutorial requires version 2.0.28 or later of the Azure CLI. If using Azure Cloud Shell, the latest version is already installed.

Create a resource group

Before you can create any resource, you have to create a resource group to host the Web App, the Virtual Network, and other network components. Create a resource group with az group create. This example creates a resource group named *myResourceGroup* in the *francecentral* location:

Azure CLI

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

Create an App Service Plan

You need to create an App Service Plan to host your Web App. Create an App Service Plan with az appservice plan create. This example creates App Service Plan named *myAppServicePlan* in the *francecentral* location with *P1V2* sku and only one worker:

Azure CLI

```
az appservice plan create \
--name myAppServicePlan \
--resource-group myResourceGroup \
--location francecentral \
--sku P1V2 \
--number-of-workers 1
```

Create a Web App

Now that you have an App Service Plan you can deploy a Web App. Create a Web App with az webapp create. This example creates a Web App named *mySiteName* in the Plan named *myAppServicePlan*

Azure CLI

```
az webapp create \
--name mySiteName \
--resource-group myResourceGroup \
--plan myAppServicePlan
```

Create a VNet

Create a Virtual Network with az network vnet create. This example creates a default Virtual Network named *myVNet* with one subnet named *mySubnet*:

Azure CLI

```
az network vnet create \
--name myVNet \
--resource-group myResourceGroup \
--location francecentral \
```

```
--address-prefixes 10.8.0.0/16 \
--subnet-name mySubnet \
--subnet-prefixes 10.8.100.0/24
```

Configure the Subnet

You need to update the subnet to disable private endpoint network policies. Update a subnet configuration named *mySubnet* with az network vnet subnet update:

Azure CLI

```
az network vnet subnet update \
--name mySubnet \
--resource-group myResourceGroup \
--vnet-name myVNet \
--disable-private-endpoint-network-policies true
```

Create the Private Endpoint

Create the Private Endpoint for your Web App with az network private-endpoint create. This example creates a Private Endpoint named *myPrivateEndpoint* in the VNet *myVNet* in the Subnet *mySubnet* with a connection named *myConnectionName* to the resource ID of my Web App /subscriptions/SubscriptionID/resourceGroups/myResourceGroup/providers/Microsoft.Web/sites/myWebApp, the group parameter is *sites* for Web App type.

Azure CLI

```
az network private-endpoint create \
--name myPrivateEndpoint \
--resource-group myResourceGroup \
--vnet-name myVNet \
--subnet mySubnet \
--connection-name myConnectionName \
--private-connection-resource-id
/subscriptions/SubscriptionID/resourceGroups/myResourceGroup/providers/Microsoft.Web/sites/myWebApp \
--group-id sites
```

Configure the private zone

At the end, you need to create a private DNS zone named *privatelink.azurewebsites.net* linked to the VNet to resolve DNS name of the Web App.

Azure CLI

```
az network private-dns zone create \
--name privatelink.azurewebsites.net \
--resource-group myResourceGroup

az network private-dns link vnet create \
--name myDNSLink \
--resource-group myResourceGroup \
--registration-enabled false \
--virtual-network myVNet \
--zone-name privatelink.azurewebsites.net

az network private-endpoint dns-zone-group create \
--name myZoneGroup \
--resource-group myResourceGroup \
--endpoint-name myPrivateEndpoint \
--private-dns-zone privatelink.azurewebsites.net \
--zone-name privatelink.azurewebsites.net
```

Clean up deployment

After the sample script has been run, the following command can be used to remove the resource group and all resources associated with it.

Azure CLI

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

Next steps

- For more information on the Azure CLI, see Azure CLI documentation.
- Additional App Service CLI script samples can be found in the Azure App Service documentation.

Map a custom domain to an App Service app using CLI

Article • 01/13/2023

This sample script creates an app in App Service with its related resources, and then maps `www.<yourdomain>` to it.

If you don't have an Azure subscription, create an Azure free account before you begin.

Prerequisites

- 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.

Sample script

Launch Azure Cloud Shell

The Azure Cloud Shell is a free interactive shell that you can use to run the steps in this article. It has common Azure tools preinstalled and configured to use with your account.

To open the Cloud Shell, just select **Try it** from the upper right corner of a code block. You can also launch Cloud Shell in a separate browser tab by going to

https://shell.azure.com .

When Cloud Shell opens, verify that **Bash** is selected for your environment. Subsequent sessions will use Azure CLI in a Bash environment, Select **Copy** to copy the blocks of code, paste it into the Cloud Shell, and press **Enter** to run it.

Sign in to Azure

Cloud Shell is automatically authenticated under the initial account signed-in with. Use the following script to sign in using a different subscription, replacing `<Subscription ID>` with your Azure Subscription ID. If you don't have an Azure subscription, create an Azure free account before you begin.

Azure CLI

```
subscription="<subscriptionId>" # add subscription here

az account set -s $subscription # ...or use 'az login'
```

For more information, see set active subscription or log in interactively

To create the web app

Azure CLI

```
# Map a custom domain to an App Service app
# set -e # exit if error
# Variable block
let "randomIdentifier=$RANDOM*$RANDOM"
location="East US"
resourceGroup="msdocs-app-service-rg-$randomIdentifier"
tag="configure-custom-domain-webapp-only.sh"
appServicePlan="msdocs-app-service-plan-$randomIdentifier"
webapp="msdocs-web-app-$randomIdentifier"

# Create a resource group.
echo "Creating $resourceGroup in "$location"..."
az group create --name $resourceGroup --location "$location" --tag $tag

# Create an App Service plan in SHARED tier (minimum required by custom
domains).
echo "Creating $appServicePlan"
az appservice plan create --name $appServicePlan --resource-group
$resourceGroup --sku SHARED

# Create a web app.
echo "Creating $webapp"
```

```
az webapp create --name $webapp --resource-group $resourceGroup --plan
$appServicePlan

# Copy the result of the following command into a browser to see the static
HTML site.
site="http://$webapp.azurewebsites.net"
echo $site
curl "$site"
```

Map your prepared custom domain name to the web app

1. Create the following variable containing your fully qualified domain name.

   Azure CLI

   ```
   fqdn=<Replace with www.{yourdomain}>
   ```

2. Configure a CNAME record that maps your fully qualified domain name to your web app's default domain name ($webappname.azurewebsites.net).

3. Map your domain name to the web app.

   Azure CLI

   ```
   az webapp config hostname add --webapp-name $webappname --resource-
   group myResourceGroup --hostname $fqdn

   echo "You can now browse to http://$fqdn"
   ```

Clean up resources

Use the following command to remove the resource group and all resources associated with it using the az group delete command - unless you have an ongoing need for these resources. Some of these resources may take a while to create, as well as to delete.

Azure CLI

```
az group delete --name $resourceGroup
```

Sample reference

This script uses the following commands. Each command in the table links to command specific documentation.

Command	Notes
az group create	Creates a resource group in which all resources are stored.
az appservice plan create	Creates an App Service plan.
az webapp create	Creates an App Service app.
az webapp config hostname add	Maps a custom domain to an App Service app.

Next steps

For more information on the Azure CLI, see Azure CLI documentation.

Additional App Service CLI script samples can be found in the Azure App Service documentation.

Bind a custom TLS/SSL certificate to an App Service app using CLI

Article • 01/13/2023

This sample script creates an app in App Service with its related resources, then binds the TLS/SSL certificate of a custom domain name to it. For this sample, you need:

- Access to your domain registrar's DNS configuration page.
- A valid .PFX file and its password for the TLS/SSL certificate you want to upload and bind.

If you don't have an Azure subscription, create an Azure free account before you begin.

Prerequisites

- 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.

Sample script

Launch Azure Cloud Shell

The Azure Cloud Shell is a free interactive shell that you can use to run the steps in this article. It has common Azure tools preinstalled and configured to use with your account.

To open the Cloud Shell, just select **Try it** from the upper right corner of a code block. You can also launch Cloud Shell in a separate browser tab by going to https://shell.azure.com.

When Cloud Shell opens, verify that **Bash** is selected for your environment. Subsequent sessions will use Azure CLI in a Bash environment, Select **Copy** to copy the blocks of code, paste it into the Cloud Shell, and press **Enter** to run it.

Sign in to Azure

Cloud Shell is automatically authenticated under the initial account signed-in with. Use the following script to sign in using a different subscription, replacing `<Subscription ID>` with your Azure Subscription ID. If you don't have an Azure subscription, create an Azure free account before you begin.

Azure CLI

```
subscription="<subscriptionId>" # add subscription here

az account set -s $subscription # ...or use 'az login'
```

For more information, see set active subscription or log in interactively

To create the web app

Azure CLI

```
# Bind a custom TLS/SSL certificate to an App Service app
# set -e # exit if error
# Variable block
let "randomIdentifier=$RANDOM*$RANDOM"
location="East US"
resourceGroup="msdocs-app-service-rg-$randomIdentifier"
tag="configure-ssl-certificate-webapp-only.sh"
appServicePlan="msdocs-app-service-plan-$randomIdentifier"
webapp="msdocs-web-app-$randomIdentifier"

# Create a resource group.
echo "Creating $resourceGroup in "$location"..."
az group create --name $resourceGroup --location "$location" --tag $tag

# Create an App Service plan in Basic tier (minimum required by custom
domains).
```

```
echo "Creating $appServicePlan"
az appservice plan create --name $appServicePlan --resource-group
$resourceGroup --sku B1

# Create a web app.
echo "Creating $webapp"
az webapp create --name $webapp --resource-group $resourceGroup --plan
$appServicePlan

# Copy the result of the following command into a browser to see the static
HTML site.
site="http://$webapp.azurewebsites.net"
echo $site
curl "$site"
```

Map your prepared custom domain name to the web app

1. Create the following variable containing your fully qualified domain name.

   Azure CLI

   ```
   fqdn=<Replace with www.{yourdomain}>
   ```

2. Configure a CNAME record that maps your fully qualified domain name to your web app's default domain name ($webappname.azurewebsites.net).

3. Map your domain name to the web app.

   Azure CLI

   ```
   az webapp config hostname add --webapp-name $webappname --resource-
   group myResourceGroup --hostname $fqdn

   echo "You can now browse to http://$fqdn"
   ```

Upload and bind the SSL certificate

1. Create the following variable containing your pfx path and password.

   Azure CLI

   ```
   pfxPath=<replace-with-path-to-your-.PFX-file>
   pfxPassword=<replace-with-your=.PFX-password>
   ```

2. Upload the SSL certificate and get the thumbprint.

Azure CLI

```
thumbprint=$(az webapp config ssl upload --certificate-file $pfxPath --
certificate-password $pfxPassword --name $webapp --resource-group
$resourceGroup --query thumbprint --output tsv)
```

3. Bind the uploaded SSL certificate to the web app.

Azure CLI

```
az webapp config ssl bind --certificate-thumbprint $thumbprint --ssl-
type SNI --name $webapp --resource-group $resourceGroup

echo "You can now browse to https://$fqdn"
```

Clean up resources

Use the following command to remove the resource group and all resources associated with it using the az group delete command - unless you have an ongoing need for these resources. Some of these resources may take a while to create, as well as to delete.

Azure CLI

```
az group delete --name $resourceGroup
```

Sample reference

This script uses the following commands. Each command in the table links to command specific documentation.

Command	Notes
az group create	Creates a resource group in which all resources are stored.
az appservice plan create	Creates an App Service plan.
az webapp create	Creates an App Service app.
az webapp config hostname add	Maps a custom domain to an App Service app.
az webapp config ssl upload	Uploads a TLS/SSL certificate to an App Service app.
az webapp config ssl bind	Binds an uploaded TLS/SSL certificate to an App Service app.

Next steps

For more information on the Azure CLI, see Azure CLI documentation.

Additional App Service CLI script samples can be found in the Azure App Service documentation.

Scale an App Service app manually using Azure CLI

Article • 01/13/2023

This sample script creates a resource group, an App Service plan, and an app. It then scales the App Service plan from a single instance to multiple instances.

If you don't have an Azure subscription, create an Azure free account before you begin.

Prerequisites

- 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.

Sample script

Launch Azure Cloud Shell

The Azure Cloud Shell is a free interactive shell that you can use to run the steps in this article. It has common Azure tools preinstalled and configured to use with your account.

To open the Cloud Shell, just select **Try it** from the upper right corner of a code block. You can also launch Cloud Shell in a separate browser tab by going to

https://shell.azure.com .

When Cloud Shell opens, verify that **Bash** is selected for your environment. Subsequent sessions will use Azure CLI in a Bash environment, Select **Copy** to copy the blocks of code, paste it into the Cloud Shell, and press **Enter** to run it.

Sign in to Azure

Cloud Shell is automatically authenticated under the initial account signed-in with. Use the following script to sign in using a different subscription, replacing `<Subscription ID>` with your Azure Subscription ID. If you don't have an Azure subscription, create an Azure free account before you begin.

Azure CLI

```
subscription="<subscriptionId>" # add subscription here

az account set -s $subscription # ...or use 'az login'
```

For more information, see set active subscription or log in interactively

Run the script

Azure CLI

```
# Scale an App Service app manually
# set -e # exit if error
# Variable block
let "randomIdentifier=$RANDOM*$RANDOM"
location="East US"
resourceGroup="msdocs-app-service-rg-$randomIdentifier"
tag="scale-manual.sh"
appServicePlan="msdocs-app-service-plan-$randomIdentifier"
webapp="msdocs-web-app-$randomIdentifier"

# Create a resource group.
echo "Creating $resourceGroup in "$location"..."
az group create --name $resourceGroup --location "$location" --tag $tag

# Create an App Service plan in Basic tier
echo "Creating $appServicePlan"
az appservice plan create --name $appServicePlan --resource-group
$resourceGroup --location "$location" --sku B1

# Create a web app.
echo "Creating $webapp"
az webapp create --name $webapp --resource-group $resourceGroup --plan
```

```
$appServicePlan

# Scale Web App to 2 Workers
az appservice plan update --number-of-workers 2 --name $appServicePlan --
resource-group $resourceGroup
```

Clean up resources

Use the following command to remove the resource group and all resources associated with it using the az group delete command - unless you have an ongoing need for these resources. Some of these resources may take a while to create, as well as to delete.

Azure CLI

```
az group delete --name $resourceGroup
```

Sample reference

This script uses the following commands to create a resource group, App Service app, and all related resources. Each command in the table links to command specific documentation.

Command	Notes
az group create	Creates a resource group in which all resources are stored.
az appservice plan create	Creates an App Service plan.
az webapp create	Creates an App Service app.
az appservice plan update	Updates properties of the App Service plan.

Next steps

For more information on the Azure CLI, see Azure CLI documentation.

Additional App Service CLI script samples can be found in the Azure App Service documentation.

Scale an App Service app worldwide with a high-availability architecture using Azure CLI

Article • 01/13/2023

This sample script creates a resource group, two App Service plans, two apps, a traffic manager profile, and two traffic manager endpoints. Once the exercise is complete, you have a high-available architecture, which provides global availability of your app based on the lowest network latency.

If you don't have an Azure subscription, create an Azure free account before you begin.

Prerequisites

- 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.

Sample script

Launch Azure Cloud Shell

The Azure Cloud Shell is a free interactive shell that you can use to run the steps in this article. It has common Azure tools preinstalled and configured to use with your account.

To open the Cloud Shell, just select **Try it** from the upper right corner of a code block. You can also launch Cloud Shell in a separate browser tab by going to https://shell.azure.com.

When Cloud Shell opens, verify that **Bash** is selected for your environment. Subsequent sessions will use Azure CLI in a Bash environment, Select **Copy** to copy the blocks of code, paste it into the Cloud Shell, and press **Enter** to run it.

Sign in to Azure

Cloud Shell is automatically authenticated under the initial account signed-in with. Use the following script to sign in using a different subscription, replacing `<Subscription ID>` with your Azure Subscription ID. If you don't have an Azure subscription, create an Azure free account before you begin.

Azure CLI

```
subscription="<subscriptionId>" # add subscription here

az account set -s $subscription # ...or use 'az login'
```

For more information, see set active subscription or log in interactively

Run the script

Azure CLI

```
# Scale an App Service app worldwide with a high-availability architecture
# set -e # exit if error
# Variable block
let "randomIdentifier=$RANDOM*$RANDOM"
location="East US"
resourceGroup="msdocs-app-service-rg-$randomIdentifier"
tag="scale-geographic.sh"
appServicePlan="msdocs-app-service-plan-$randomIdentifier"
trafficManagerDns="msdocs-dns-$randomIdentifier"
app1Name="msdocs-appServiceTM1-$randomIdentifier"
app2Name="msdocs-appServiceTM2-$randomIdentifier"
location1="West US"
location2="East US"

# Create a resource group.
echo "Creating $resourceGroup in "$location"..."
```

```
az group create --name $resourceGroup --location "$location" --tag $tag

# Create a Traffic Manager Profile
echo "Creating $trafficManagerDNS"
az network traffic-manager profile create --name $trafficManagerDns-tmp --
resource-group $resourceGroup --routing-method Performance --unique-dns-name
$trafficManagerDns

# Create App Service Plans in two Regions
az appservice plan create --name $app1Name-Plan --resource-group
$resourceGroup --location "$location1" --sku S1
az appservice plan create --name $app2Name-Plan --resource-group
$resourceGroup --location "$location2" --sku S1

# Add a Web App to each App Service Plan
site1=$(az webapp create --name $app1Name --plan $app1Name-Plan --resource-
group $resourceGroup --query id --output tsv)
site2=$(az webapp create --name $app2Name --plan $app2Name-Plan --resource-
group $resourceGroup --query id --output tsv)

# Assign each Web App as an Endpoint for high-availabilty
az network traffic-manager endpoint create -n $app1Name-"$location1" --
profile-name $trafficManagerDns-tmp -g $resourceGroup --type azureEndpoints
--target-resource-id $site1
az network traffic-manager endpoint create -n $app2Name-"$location2" --
profile-name $trafficManagerDns-tmp -g $resourceGroup --type azureEndpoints
--target-resource-id $site2
```

Clean up resources

Use the following command to remove the resource group and all resources associated with it using the az group delete command - unless you have an ongoing need for these resources. Some of these resources may take a while to create, as well as to delete.

Azure CLI

```
az group delete --name $resourceGroup
```

Sample reference

This script uses the following commands to create a resource group, App Service app, traffic manager profile, and all related resources. Each command in the table links to command specific documentation.

Command	Notes

Command	Notes
az group create	Creates a resource group in which all resources are stored.
az appservice plan create	Creates an App Service plan.
az webapp create	Creates an App Service app.
az network traffic-manager profile create	Creates an Azure Traffic Manager profile.
az network traffic-manager endpoint create	Adds an endpoint to an Azure Traffic Manager Profile.

Next steps

For more information on the Azure CLI, see Azure CLI documentation.

Additional App Service CLI script samples can be found in the Azure App Service documentation.

Integrate App Service with Application Gateway using CLI

Article • 01/13/2023

This sample script creates an Azure App Service web app, an Azure Virtual Network and an Application Gateway. It then restricts the traffic for the web app to only originate from the Application Gateway subnet.

If you don't have an Azure subscription, create an Azure free account before you begin.

Prerequisites

- 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.

Sample script

Launch Azure Cloud Shell

The Azure Cloud Shell is a free interactive shell that you can use to run the steps in this article. It has common Azure tools preinstalled and configured to use with your account.

To open the Cloud Shell, just select **Try it** from the upper right corner of a code block. You can also launch Cloud Shell in a separate browser tab by going to https://shell.azure.com.

When Cloud Shell opens, verify that **Bash** is selected for your environment. Subsequent sessions will use Azure CLI in a Bash environment, Select **Copy** to copy the blocks of code, paste it into the Cloud Shell, and press **Enter** to run it.

Sign in to Azure

Cloud Shell is automatically authenticated under the initial account signed-in with. Use the following script to sign in using a different subscription, replacing `<Subscription ID>` with your Azure Subscription ID. If you don't have an Azure subscription, create an Azure free account before you begin.

Azure CLI

```
subscription="<subscriptionId>" # add subscription here

az account set -s $subscription # ...or use 'az login'
```

For more information, see set active subscription or log in interactively

Run the script

Azure CLI

```
# Integrate App Service with Application Gateway
# set -e # exit if error
# Variable block
let "randomIdentifier=$RANDOM*$RANDOM"
location="East US"
resourceGroup="msdocs-app-service-rg-$randomIdentifier"
tag="integrate-with-app-gateway.sh"
vNet="msdocs-app-service-vnet-$randomIdentifier"
subnet="msdocs-app-service-subnet-$randomIdentifier"
appServicePlan="msdocs-app-service-plan-$randomIdentifier"
webapp="msdocs-web-app-$randomIdentifier"
appGateway="msdocs-app-gateway-$randomIdentifier"
publicIpAddress="msdocs-public-ip-$randomIdentifier"

# Create a resource group.
echo "Creating $resourceGroup in "$location"..."
az group create --name $resourceGroup --location "$location" --tag $tag

# Create network resources
az network vnet create --resource-group $resourceGroup --name $vNet --
```

```
location "$location" --address-prefix 10.0.0.0/16 --subnet-name $subnet --
subnet-prefix 10.0.1.0/24

az network public-ip create --resource-group $resourceGroup --location
"$location" --name $publicIpAddress --dns-name $webapp --sku Standard --zone
1

# Create an App Service plan in `S1` tier
echo "Creating $appServicePlan"
az appservice plan create --name $appServicePlan --resource-group
$resourceGroup --sku S1

# Create a web app.
echo "Creating $webapp"
az webapp create --name $webapp --resource-group $resourceGroup --plan
$appServicePlan

appFqdn=$(az webapp show --name $webapp --resource-group $resourceGroup --
query defaultHostName -o tsv)

# Create an Application Gateway
az network application-gateway create --resource-group $resourceGroup --name
$appGateway --location "$location" --vnet-name $vNet --subnet $subnet --min-
capacity 2 --sku Standard_v2 --http-settings-cookie-based-affinity Disabled
--frontend-port 80 --http-settings-port 80 --http-settings-protocol Http --
public-ip-address $publicIpAddress --servers $appFqdn --priority 1

az network application-gateway http-settings update --resource-group
$resourceGroup --gateway-name $appGateway --name
appGatewayBackendHttpSettings --host-name-from-backend-pool

# Apply Access Restriction to Web App
az webapp config access-restriction add --resource-group $resourceGroup --
name $webapp --priority 200 --rule-name gateway-access --subnet $subnet --
vnet-name $vNet

# Get the App Gateway Fqdn
az network public-ip show --resource-group $resourceGroup --name
$publicIpAddress --query {AppGatewayFqdn:dnsSettings.fqdn} --output table
```

Clean up resources

Use the following command to remove the resource group and all resources associated with it using the az group delete command - unless you have an ongoing need for these resources. Some of these resources may take a while to create, as well as to delete.

Azure CLI

```
az group delete --name $resourceGroup
```

Sample reference

This script uses the following commands to create a resource group, an App Service app, an Azure Cosmos DB instance, and all related resources. Each command in the table links to command specific documentation.

Command	Notes
az group create	Creates a resource group in which all resources are stored.
az network vnet create	Creates a virtual network.
az network public-ip create	Creates a public IP address.
az network public-ip show	Show details of a public IP address.
az appservice plan create	Creates an App Service plan.
az webapp create	Creates an App Service web app.
az webapp show	Show details of an App Service web app.
az webapp config access-restriction add	Adds an access restriction to the App Service web app.
az network application-gateway create	Creates an Application Gateway.
az network application-gateway http-settings update	Updates Application Gateway HTTP settings.

Next steps

For more information on the Azure CLI, see Azure CLI documentation.

Additional App Service CLI script samples can be found in the Azure App Service documentation.

Connect an App Service app to SQL Database using CLI

Article • 01/13/2023

This sample script creates a database in Azure SQL Database and an App Service app. It then links the database to the app using app settings.

If you don't have an Azure subscription, create an Azure free account before you begin.

Prerequisites

- 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.

Sample script

Launch Azure Cloud Shell

The Azure Cloud Shell is a free interactive shell that you can use to run the steps in this article. It has common Azure tools preinstalled and configured to use with your account.

To open the Cloud Shell, just select **Try it** from the upper right corner of a code block. You can also launch Cloud Shell in a separate browser tab by going to

https://shell.azure.com .

When Cloud Shell opens, verify that **Bash** is selected for your environment. Subsequent sessions will use Azure CLI in a Bash environment, Select **Copy** to copy the blocks of code, paste it into the Cloud Shell, and press **Enter** to run it.

Sign in to Azure

Cloud Shell is automatically authenticated under the initial account signed-in with. Use the following script to sign in using a different subscription, replacing `<Subscription ID>` with your Azure Subscription ID. If you don't have an Azure subscription, create an Azure free account before you begin.

Azure CLI

```
subscription="<subscriptionId>" # add subscription here

az account set -s $subscription # ...or use 'az login'
```

For more information, see set active subscription or log in interactively

Run the script

Azure CLI

```
# Connect an App Service app to SQL Database
# set -e # exit if error
# Variable block
let "randomIdentifier=$RANDOM*$RANDOM"
location="East US"
resourceGroup="msdocs-app-service-rg-$randomIdentifier"
tag="connect-to-sql.sh"
appServicePlan="msdocs-app-service-plan-$randomIdentifier"
webapp="msdocs-web-app-$randomIdentifier"
server="msdocs-azuresql-$randomIdentifier"
database="msdocsazuresqldb$randomIdentifier"
login="azureuser"
password="Pa$$w0rD-$randomIdentifier"
startIp="0.0.0.0"
endIp="0.0.0.0"

# Create a resource group.
echo "Creating $resourceGroup in "$location"..."
az group create --name $resourceGroup --location "$location" --tag $tag

# Create an App Service Plan
echo "Creating $appServicePlan"
```

```
az appservice plan create --name $appServicePlan --resource-group
$resourceGroup \
--location "$location"

# Create a Web App
echo "Creating $webapp"
az webapp create --name $webapp --plan $appServicePlan --resource-group
$resourceGroup

# Create a SQL Database server
echo "Creating $server"
az sql server create --name $server --resource-group $resourceGroup --
location "$location" --admin-user $login --admin-password $password

# Configure firewall for Azure access
echo "Creating firewall rule with starting ip of $startIp" and ending ip of
$endIp
az sql server firewall-rule create \
--server $server \
--resource-group $resourceGroup \
--name AllowYourIp \
--start-ip-address $startIp --end-ip-address $endIp

# Create a database called 'MySampleDatabase' on server
echo "Creating $database"
az sql db create --server $server \
 --resource-group $resourceGroup --name $database \
--service-objective S0

# Get connection string for the database
connstring=$(az sql db show-connection-string --name $database --server
$server \
--client ado.net --output tsv)

# Add credentials to connection string
connstring=${connstring//<username>/$login}
connstring=${connstring//<password>/$password}

# Assign the connection string to an app setting in the web app
az webapp config appsettings set --name $webapp \
--resource-group $resourceGroup \
--settings "SQLSRV_CONNSTR=$connstring"
```

Clean up resources

Use the following command to remove the resource group and all resources associated with it using the az group delete command - unless you have an ongoing need for these resources. Some of these resources may take a while to create, as well as to delete.

Azure CLI

```
az group delete --name $resourceGroup
```

Sample reference

This script uses the following commands to create a resource group, App Service app, SQL Database, and all related resources. Each command in the table links to command specific documentation.

Command	Notes
az group create	Creates a resource group in which all resources are stored.
az appservice plan create	Creates an App Service plan.
az webapp create	Creates an App Service app.
az sql server create	Creates a server.
az sql db create	Creates a new database.
az sql db show-connection-string	Generates a connection string to a database.
az webapp config appsettings set	Creates or updates an app setting for an App Service app. App settings are exposed as environment variables for your app.

Next steps

For more information on the Azure CLI, see Azure CLI documentation.

Additional App Service CLI script samples can be found in the Azure App Service documentation.

Connect an App Service app to a storage account using CLI

Article • 01/13/2023

This sample script creates an Azure storage account and an App Service app. It then links the storage account to the app using app settings.

If you don't have an Azure subscription, create an Azure free account before you begin.

Prerequisites

- 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.

Sample script

Launch Azure Cloud Shell

The Azure Cloud Shell is a free interactive shell that you can use to run the steps in this article. It has common Azure tools preinstalled and configured to use with your account.

To open the Cloud Shell, just select **Try it** from the upper right corner of a code block. You can also launch Cloud Shell in a separate browser tab by going to

https://shell.azure.com .

When Cloud Shell opens, verify that **Bash** is selected for your environment. Subsequent sessions will use Azure CLI in a Bash environment, Select **Copy** to copy the blocks of code, paste it into the Cloud Shell, and press **Enter** to run it.

Sign in to Azure

Cloud Shell is automatically authenticated under the initial account signed-in with. Use the following script to sign in using a different subscription, replacing `<Subscription ID>` with your Azure Subscription ID. If you don't have an Azure subscription, create an Azure free account before you begin.

Azure CLI

```
subscription="<subscriptionId>" # add subscription here

az account set -s $subscription # ...or use 'az login'
```

For more information, see set active subscription or log in interactively

Run the script

Azure CLI

```
# Connect an App Service app to a storage account
# set -e # exit if error
# Variable block
let "randomIdentifier=$RANDOM*$RANDOM"
location="East US"
resourceGroup="msdocs-app-service-rg-$randomIdentifier"
tag="connect-to-storage.sh"
appServicePlan="msdocs-app-service-plan-$randomIdentifier"
webapp="msdocs-web-app-$randomIdentifier"
storage="webappstore$randomIdentifier"

# Create a resource group.
echo "Creating $resourceGroup in "$location"..."
az group create --name $resourceGroup --location "$location" --tag $tag

# Create an App Service Plan
echo "Creating $appServicePlan"
az appservice plan create --name $appServicePlan --resource-group
$resourceGroup \
--location "$location"

# Create a Web App
```

```
echo "Creating $webapp"
az webapp create --name $webapp --plan $appServicePlan --resource-group
$resourceGroup

# Create a storage account
echo "Creating $storage"
az storage account create --name $storage --resource-group $resourceGroup \
--location "$location" --sku Standard_LRS

# Retrieve the storage account connection string
connstr=$(az storage account show-connection-string --name $storage --
resource-group $resourceGroup \
--query connectionString --output tsv)

# Assign the connection string to an App Setting in the Web App
az webapp config appsettings set --name $webapp --resource-group
$resourceGroup \
--settings "STORAGE_CONNSTR=$connstr"
```

Clean up resources

Use the following command to remove the resource group and all resources associated with it using the az group delete command - unless you have an ongoing need for these resources. Some of these resources may take a while to create, as well as to delete.

Azure CLI

```
az group delete --name $resourceGroup
```

Sample reference

This script uses the following commands to create a resource group, App Service app, storage account, and all related resources. Each command in the table links to command specific documentation.

Command	Notes
az group create	Creates a resource group in which all resources are stored.
az appservice plan create	Creates an App Service plan.
az webapp create	Creates an App Service app.
az storage account create	Creates a storage account.

Command	Notes
az storage account show-connection-string	Get the connection string for a storage account.
az webapp config appsettings set	Creates or updates an app setting for an App Service app. App settings are exposed as environment variables for your app.

Next steps

For more information on the Azure CLI, see Azure CLI documentation.

Additional App Service CLI script samples can be found in the Azure App Service documentation.

Connect an App Service app to an Azure Cache for Redis using CLI

Article • 01/13/2023

This sample script creates an Azure Cache for Redis and an App Service app. It then links the Azure Cache for Redis to the app using app settings.

If you don't have an Azure subscription, create an Azure free account before you begin.

Prerequisites

- 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.

Sample script

Launch Azure Cloud Shell

The Azure Cloud Shell is a free interactive shell that you can use to run the steps in this article. It has common Azure tools preinstalled and configured to use with your account.

To open the Cloud Shell, just select **Try it** from the upper right corner of a code block. You can also launch Cloud Shell in a separate browser tab by going to

https://shell.azure.com .

When Cloud Shell opens, verify that **Bash** is selected for your environment. Subsequent sessions will use Azure CLI in a Bash environment, Select **Copy** to copy the blocks of code, paste it into the Cloud Shell, and press **Enter** to run it.

Sign in to Azure

Cloud Shell is automatically authenticated under the initial account signed-in with. Use the following script to sign in using a different subscription, replacing `<Subscription ID>` with your Azure Subscription ID. If you don't have an Azure subscription, create an Azure free account before you begin.

Azure CLI

```
subscription="<subscriptionId>" # add subscription here

az account set -s $subscription # ...or use 'az login'
```

For more information, see set active subscription or log in interactively

Run the script

Azure CLI

```
# Connect an App Service app to an Azure Cache for Redis
# set -e # exit if error
# Variable block
let "randomIdentifier=$RANDOM*$RANDOM"
location="East US"
resourceGroup="msdocs-app-service-rg-$randomIdentifier"
tag="connect-to-redis.sh"
appServicePlan="msdocs-app-service-plan-$randomIdentifier"
webapp="msdocs-web-app-$randomIdentifier"

# Create a resource group.
echo "Creating $resourceGroup in "$location"..."
az group create --name $resourceGroup --location "$location" --tag $tag

# Create an App Service Plan
echo "Creating $appServicePlan"
az appservice plan create --name $appServicePlan --resource-group
$resourceGroup \
--location "$location"

# Create a Web App
echo "Creating $webapp"
```

```
az webapp create --name $webapp --plan $appServicePlan --resource-group
$resourceGroup

# Create a Redis Cache
redis=($(az redis create --name $webapp --resource-group $resourceGroup --
location "$location" --vm-size C0 --sku Basic --query [hostName,sslPort] --
output tsv))

# Get access key
key=$(az redis list-keys --name $webapp --resource-group $resourceGroup --
query primaryKey --output tsv)

# Assign the connection string to an App Setting in the Web App
az webapp config appsettings set --name $webapp --resource-group
$resourceGroup --settings "REDIS_URL=${redis[0]}" "REDIS_PORT=${redis[1]}"
"REDIS_KEY=$key"
```

Clean up resources

Use the following command to remove the resource group and all resources associated with it using the az group delete command - unless you have an ongoing need for these resources. Some of these resources may take a while to create, as well as to delete.

Azure CLI

```
az group delete --name $resourceGroup
```

Sample reference

This script uses the following commands to create a resource group, App Service app, Azure Cache for Redis, and all related resources. Each command in the table links to command specific documentation.

Command	Notes
az group create	Creates a resource group in which all resources are stored.
az appservice plan create	Creates an App Service plan.
az webapp create	Creates an App Service app.
az redis create	Create new Azure Cache for Redis instance.
az redis list-keys	Lists the access keys for the Azure Cache for Redis instance.

Command	Notes
az webapp config appsettings set	Creates or updates an app setting for an App Service app. App settings are exposed as environment variables for your app.

Next steps

For more information on the Azure CLI, see Azure CLI documentation.

Additional App Service CLI script samples can be found in the Azure App Service documentation.

Connect an App Service app to Azure Cosmos DB via the Azure CLI

Article • 01/13/2023

This sample script creates an Azure Cosmos DB account using Azure Cosmos DB for MongoDB and an App Service app. It then links a MongoDB connection string to the web app using app settings.

If you don't have an Azure subscription, create an Azure free account before you begin.

Prerequisites

- 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.

Sample script

Launch Azure Cloud Shell

The Azure Cloud Shell is a free interactive shell that you can use to run the steps in this article. It has common Azure tools preinstalled and configured to use with your account.

To open the Cloud Shell, just select **Try it** from the upper right corner of a code block. You can also launch Cloud Shell in a separate browser tab by going to https://shell.azure.com.

When Cloud Shell opens, verify that **Bash** is selected for your environment. Subsequent sessions will use Azure CLI in a Bash environment, Select **Copy** to copy the blocks of code, paste it into the Cloud Shell, and press **Enter** to run it.

Sign in to Azure

Cloud Shell is automatically authenticated under the initial account signed-in with. Use the following script to sign in using a different subscription, replacing `<Subscription ID>` with your Azure Subscription ID. If you don't have an Azure subscription, create an Azure free account before you begin.

Azure CLI

```
subscription="<subscriptionId>" # add subscription here

az account set -s $subscription # ...or use 'az login'
```

For more information, see set active subscription or log in interactively

Run the script

Azure CLI

```
# Connect an App Service app to Cosmos DB
# set -e # exit if error
# Variable block
let "randomIdentifier=$RANDOM*$RANDOM"
location="East US"
resourceGroup="msdocs-app-service-rg-$randomIdentifier"
tag="connect-to-documentdb.sh"
appServicePlan="msdocs-app-service-plan-$randomIdentifier"
webapp="msdocs-web-app-$randomIdentifier"

# Create a resource group.
echo "Creating $resourceGroup in "$location"..."
az group create --name $resourceGroup --location "$location" --tag $tag

# Create an App Service Plan
echo "Creating $appServicePlan"
az appservice plan create --name $appServicePlan --resource-group
$resourceGroup \
--location "$location"
```

```
# Create a Web App
az webapp create --name $webapp --plan $appServicePlan --resource-group
$resourceGroup

# Create a Cosmos DB with MongoDB API
az cosmosdb create --name $webapp --resource-group $resourceGroup --kind
MongoDB

# Get the MongoDB URL
connectionString=$(az cosmosdb keys list  --name $webapp --resource-group
$resourceGroup --type connection-strings --query
connectionStrings[0].connectionString --output tsv)

# Assign the connection string to an App Setting in the Web App
az webapp config appsettings set --name $webapp --resource-group
$resourceGroup \
--settings "MONGODB_URL=$connectionString"
```

Clean up resources

Use the following command to remove the resource group and all resources associated with it using the az group delete command - unless you have an ongoing need for these resources. Some of these resources may take a while to create, as well as to delete.

Azure CLI

```
az group delete --name $resourceGroup
```

Sample reference

This script uses the following commands to create a resource group, App Service app, Azure Cosmos DB, and all related resources. Each command in the table links to command specific documentation.

Command	Notes
az group create	Creates a resource group in which all resources are stored.
az appservice plan create	Creates an App Service plan.
az webapp create	Creates an App Service app.

Command	Notes
az cosmosdb create	Creates an Azure Cosmos DB account.
az cosmosdb list-connection-strings	Lists connection strings for the specified Azure Cosmos DB account.
az webapp config appsettings set	Creates or updates an app setting for an App Service app. App settings are exposed as environment variables for your app (see Environment variables and app settings reference).

Next steps

For more information on the Azure CLI, see Azure CLI documentation.

Additional App Service CLI script samples can be found in the Azure App Service documentation.

Backup and restore a web app from a backup using CLI

Article • 01/13/2023

This sample script creates a web app in App Service with its related resources. It then creates a one-time backup for it, and also a scheduled backup for it. Finally, it restores the web app from backup.

If you don't have an Azure subscription, create an Azure free account before you begin.

Prerequisites

- 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.

Sample script

Launch Azure Cloud Shell

The Azure Cloud Shell is a free interactive shell that you can use to run the steps in this article. It has common Azure tools preinstalled and configured to use with your account.

To open the Cloud Shell, just select **Try it** from the upper right corner of a code block. You can also launch Cloud Shell in a separate browser tab by going to https://shell.azure.com.

When Cloud Shell opens, verify that **Bash** is selected for your environment. Subsequent sessions will use Azure CLI in a Bash environment, Select **Copy** to copy the blocks of code, paste it into the Cloud Shell, and press **Enter** to run it.

Sign in to Azure

Cloud Shell is automatically authenticated under the initial account signed-in with. Use the following script to sign in using a different subscription, replacing `<Subscription ID>` with your Azure Subscription ID. If you don't have an Azure subscription, create an Azure free account before you begin.

Azure CLI

```
subscription="<subscriptionId>" # add subscription here

az account set -s $subscription # ...or use 'az login'
```

For more information, see set active subscription or log in interactively

Run the script

Azure CLI

```
# Backup and restore a web app from a backup
# set -e # exit if error
# Variable block
let "randomIdentifier=$RANDOM*$RANDOM"
location="East US"
resourceGroup="msdocs-app-service-rg-$randomIdentifier"
tag="backup-restore.sh"
appServicePlan="msdocs-app-service-plan-$randomIdentifier"
webapp="msdocs-web-app-$randomIdentifier"
storage="webappstorage$randomIdentifier"
container="appbackup$randomIdentifier"
backup="backup$randomIdentifier"
expirydate=$(date -I -d "$(date) + 1 month")

# Create a resource group.
echo "Creating $resourceGroup in "$location"..."
az group create --name $resourceGroup --location "$location" --tag $tag

# Create a Storage Account
echo "Creating $storage"
```

```
az storage account create --name $storage --resource-group $resourceGroup --
location "$location" \
--sku Standard_LRS

# Create a storage container
echo "Creating $container on $storage..."
key=$(az storage account keys list --account-name $storage --resource-group
$resourceGroup -o json --query [0].value | tr -d '"')

az storage container create --name $container --account-key $key --account-
name $storage

# Generate a SAS token for the storage container, valid for one month.
# NOTE: You can use the same SAS token to make backups in App Service until
--expiry
sastoken=$(az storage container generate-sas --account-name $storage --name
$container --account-key $key \
--expiry $expirydate --permissions rwdl --output tsv)

# Construct the SAS URL for the container
sasurl=https://$storage.blob.core.windows.net/$container?$sastoken

# Create an App Service plan in Standard tier. Standard tier allows one
backup per day.
echo "Creating $appServicePlan"
az appservice plan create --name $appServicePlan --resource-group
$resourceGroup --location "$location" \
--sku S1

# Create a web app
echo "Creating $webapp"
az webapp create --name $webapp --plan $appServicePlan --resource-group
$resourceGroup

# Create a one-time backup
echo "Creating $backup"
az webapp config backup create --resource-group $resourceGroup --webapp-name
$webapp \
--backup-name $backup --container-url $sasurl

# List statuses of all backups that are complete or currently executing.
az webapp config backup list --resource-group $resourceGroup --webapp-name
$webapp

# Schedule a backup every day and retain for 10 days
az webapp config backup update --resource-group $resourceGroup --webapp-name
$webapp \
--container-url $sasurl --frequency 1d --retain-one true --retention 10

# Show the current scheduled backup configuration
az webapp config backup show --resource-group $resourceGroup --webapp-name
$webapp

# List statuses of all backups that are complete or currently executing
az webapp config backup list --resource-group $resourceGroup --webapp-name
```

```
$webapp

# (OPTIONAL) Change the backup schedule to every 2 days
az webapp config backup update --resource-group $resourceGroup --webapp-name
$webapp \
--container-url $sasurl --frequency 2d --retain-one true --retention 10

# Restore the app by overwriting it with the backup data

az webapp config backup restore --resource-group $resourceGroup --webapp-
name $webapp \
--backup-name $backup --container-url $sasurl --overwrite

# fails - https://github.com/Azure/azure-cli/issues/19492
```

Clean up resources

Use the following command to remove the resource group and all resources associated with it using the az group delete command - unless you have an ongoing need for these resources. Some of these resources may take a while to create, as well as to delete.

Azure CLI

```
az group delete --name $resourceGroup
```

Sample reference

This script uses the following commands. Each command in the table links to command specific documentation.

Command	Notes
az webapp config backup list	Gets a list of backups for a web app.
az webapp config backup restore	Restores a web app from a backup.

Next steps

For more information on the Azure CLI, see Azure CLI documentation.

Additional App Service CLI script samples can be found in the Azure App Service documentation.

Monitor an App Service app with web server logs using Azure CLI

Article • 01/13/2023

This sample script creates a resource group, App Service plan, and app, and configures the app to enable web server logs. It then downloads the log files for review.

If you don't have an Azure subscription, create an Azure free account before you begin.

Prerequisites

- 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.

Sample script

Launch Azure Cloud Shell

The Azure Cloud Shell is a free interactive shell that you can use to run the steps in this article. It has common Azure tools preinstalled and configured to use with your account.

To open the Cloud Shell, just select **Try it** from the upper right corner of a code block. You can also launch Cloud Shell in a separate browser tab by going to

https://shell.azure.com .

When Cloud Shell opens, verify that **Bash** is selected for your environment. Subsequent sessions will use Azure CLI in a Bash environment, Select **Copy** to copy the blocks of code, paste it into the Cloud Shell, and press **Enter** to run it.

Sign in to Azure

Cloud Shell is automatically authenticated under the initial account signed-in with. Use the following script to sign in using a different subscription, replacing `<Subscription ID>` with your Azure Subscription ID. If you don't have an Azure subscription, create an Azure free account before you begin.

Azure CLI

```
subscription="<subscriptionId>" # add subscription here

az account set -s $subscription # ...or use 'az login'
```

For more information, see set active subscription or log in interactively

Run the script

Azure CLI

```
# Monitor an App Service app with web server logs
# set -e # exit if error
# Variable block
let "randomIdentifier=$RANDOM*$RANDOM"
location="East US"
resourceGroup="msdocs-app-service-rg-$randomIdentifier"
tag="monitor-with-logs.sh"
appServicePlan="msdocs-app-service-plan-$randomIdentifier"
webapp="msdocs-web-app-$randomIdentifier"

# Create a resource group.
echo "Creating $resourceGroup in "$location"..."
az group create --name $resourceGroup --location "$location" --tag $tag

# Create an App Service Plan
echo "Creating $appServicePlan"
az appservice plan create --name $appServicePlan --resource-group
$resourceGroup

# Create a Web App and save the URL
echo "Creating $webapp"
url=$(az webapp create --name $webapp --resource-group $resourceGroup --plan
```

```
$appServicePlan --query defaultHostName | sed -e 's/^"//' -e 's/"$//')

# Enable all logging options for the Web App
az webapp log config --name $webapp --resource-group $resourceGroup --
application-logging azureblobstorage --detailed-error-messages true --
failed-request-tracing true --web-server-logging filesystem

# Create a Web Server Log
curl -s -L $url/404

# Download the log files for review
az webapp log download --name $webapp --resource-group $resourceGroup
```

Clean up resources

Use the following command to remove the resource group and all resources associated with it using the az group delete command - unless you have an ongoing need for these resources. Some of these resources may take a while to create, as well as to delete.

Azure CLI

```
az group delete --name $resourceGroup
```

Sample reference

This script uses the following commands to create a resource group, App Service app, and all related resources. Each command in the table links to command specific documentation.

Command	Notes
az group create	Creates a resource group in which all resources are stored.
az appservice plan create	Creates an App Service plan.
az webapp create	Creates an App Service app.
az webapp log config	Configures which logs an App Service app persists.
az webapp log download	Downloads the logs of an App Service app to your local machine.

Next steps

For more information on the Azure CLI, see Azure CLI documentation.

Additional App Service CLI script samples can be found in the Azure App Service documentation.

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 • 04/09/2023

The following table includes links to Azure Resource Manager templates for Azure App Service. For recommendations about avoiding common errors when you're creating app templates, see Guidance on deploying apps with Azure Resource Manager templates.

To learn about the JSON syntax and properties for App Services resources, see Microsoft.Web resource types.

Deploying an app	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 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 v2	Creates an App Service environment v2 in your virtual network.
Create an App Service environment v2 with an ILB address	Creates an App Service environment v2 in your virtual network with a private internal load balancer address.

Deploying an app	Description
Configure the default SSL certificate for an ILB App Service environment or an ILB App Service environment v2	Configures the default TLS/SSL certificate for an ILB App Service environment or an ILB App Service environment v2.

Terraform samples for Azure App Service

Article • 04/09/2023

The following table includes links to terraform scripts.

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.

Bicep files for App Service

Article • 04/09/2023

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.

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.

Deploying an app	Description
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 v2	Creates an App Service environment v2 in your virtual network.

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

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
az appservice ase update	Update app service environment.	Core	GA

Name	Description	Type	Status
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
az appservice plan	Manage app service plans.	Core and Extension	GA

Name	Description	Type	Status
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, I1v2, I2,
I2v2, I3, I3v2, I4v2, I5v2, I6v2, P0V3, P1MV3, P1V2, P1V3, P2MV3, P2V2,
P2V3, P3MV3, P3V2, P3V3, P4MV3, P5MV3, S1, S2, S3, SHARED, WS1, WS2, WS3}
                                [--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 Small), I2v2 (Isolated V2 Medium), I3v2 (Isolated V2 Large), I4v2 (Isolated V2 I4v2), I5v2 (Isolated V2 I5v2), I6v2 (Isolated V2 I6v2), 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, I1v2, I2, I2v2, I3, I3v2, I4v2, I5v2, I6v2, P0V3, P1MV3, P1V2, P1V3, P2MV3, P2V2, P2V3, P3MV3, P3V2, P3V3, P4MV3, P5MV3, S1, S2, S3, SHARED, WS1, WS2, WS3

Optional Parameters

`--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.

`--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

Add-AzWebAppAccessRestrictionRule	Adds an Access Restiction 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 • 04/29/2022

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.

REST Operation Groups

Operation Group	Description
App Service Certificate Orders	Provides operations for managing SSL certificates bought directly from Microsoft Azure.
App Service Environments	Provides operations for managing App Service Environments.
App Service Plans	Provides operations for managing App Service plans.
Certificates	Provides operations for managing SSL certificates that come from third-party providers.
Domains	Provides operations for managing custom domains.
Recommendations	Provides operations for retrieving current recommendations for App Service apps.
Top Level Domains	Provides operations for accessing information on top-level domains.
Web Apps	Provides operations for managing App Service apps, including web apps, mobile app back ends, and API apps.

Azure subscription and service limits, quotas, and constraints

Article • 08/24/2023

This document lists some of the most common Microsoft Azure limits, which are also sometimes called quotas.

To learn more about Azure pricing, see Azure pricing overview. There, you can estimate your costs by using the pricing calculator. You also can go to the pricing details page for a particular service, for example, Windows VMs. For tips to help manage your costs, see Prevent unexpected costs with Azure billing and cost management.

Managing limits

> **Note**
>
> Some services have adjustable limits.
>
> When 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* often are used informally to describe the current, adjustable limit (soft limit) and the maximum limit (hard limit). If a limit isn't adjustable, there won't be a soft limit, only a hard limit.

Free Trial subscriptions aren't eligible for limit or quota increases. If you have a Free Trial subscription, you can upgrade to a Pay-As-You-Go subscription. For more information, see Upgrade your Azure Free Trial subscription to a Pay-As-You-Go subscription and the Free Trial subscription FAQ.

Some limits are managed at a regional level.

Let's use vCPU quotas as an example. To request a quota increase with support for vCPUs, you must decide how many vCPUs you want to use in which regions. You then request an increase in vCPU quotas for the amounts and regions that you want. If you need to use 30 vCPUs in West Europe to run your application there, you specifically request 30 vCPUs in West Europe. Your vCPU quota isn't increased in any other region--only West Europe has the 30-vCPU quota.

As a result, decide what your quotas must be for your workload in any one region. Then request that amount in each region into which you want to deploy. For help in how to determine your current quotas for specific regions, see Resolve errors for resource quotas.

General limits

For limits on resource names, see Naming rules and restrictions for Azure resources.

For information about Resource Manager API read and write limits, see Throttling Resource Manager requests.

Management group limits

The following limits apply to management groups.

Resource	Limit
Management groups per Azure AD tenant	10,000
Subscriptions per management group	Unlimited.
Levels of management group hierarchy	Root level plus 6 levels[1]

Resource	Limit
Direct parent management group per management group	One
Management group level deployments per location	800[2]
Locations of Management group level deployments	10

[1]The 6 levels don't include the subscription level.

[2]If you reach the limit of 800 deployments, delete deployments from the history that are no longer needed. To delete management group level deployments, use Remove-AzManagementGroupDeployment or az deployment mg delete.

Subscription limits

The following limits apply when you use Azure Resource Manager and Azure resource groups.

Resource	Limit
Azure subscriptions associated with an Azure Active Directory tenant	Unlimited
Coadministrators per subscription	Unlimited
Resource groups per subscription	980
Azure Resource Manager API request size	4,194,304 bytes
Tags per subscription[1]	50
Unique tag calculations per subscription[2]	80,000
Subscription-level deployments per location	800[3]
Locations of Subscription-level deployments	10

[1]You 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.

[2]Resource 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.

[3]Deployments are automatically deleted from the history as you near the limit. For more information, see Automatic deletions from deployment history.

Resource group limits

Resource	Limit
Resources per resource group	Resources aren't limited by resource group. Instead, they're limited by resource type in a resource group. See next row.
Resources per resource group, per resource type	800 - Some resource types can exceed the 800 limit. See Resources not limited to 800 instances per resource group.
Deployments per resource group in the deployment history	800[1]
Resources per deployment	800
Management locks per unique scope	20
Number of tags per resource or resource group	50
Tag key length	512
Tag value length	256

[1]Deployments are automatically deleted from the history as you near the limit. Deleting an entry from the deployment history doesn't affect the deployed resources. For more information, see Automatic deletions from deployment history.

Template limits

Value	Limit
Parameters	256
Variables	256
Resources (including copy count)	800
Outputs	64
Template expression	24,576 chars
Resources in exported templates	200
Template size	4 MB
Parameter file size	4 MB

You can exceed some template limits by using a nested template. For more information, see Use linked templates when you deploy Azure resources. To reduce the number of parameters, variables, or outputs, you can combine several values into an object. For more information, see Objects as parameters.

You may get an error with a template or parameter file of less than 4 MB, if the total size of the request is too large. For more information about how to simplify your template to avoid a large request, see Resolve errors for job size exceeded.

Azure Active Directory limits

Here are the usage constraints and other service limits for the Azure AD service.

Category	Limit
Tenants	• A single user can belong to a maximum of 500 Azure AD tenants as a member or a guest. • A single user can create a maximum of 200 directories. • Limit of 300 license-based subscriptions (such as Microsoft 365 subscriptions) per tenant
Domains	• 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	• By default, a maximum of 50,000 Azure AD resources can be created in a single tenant by users of the Azure Active Directory Free edition. If you have at least one verified domain, the default Azure AD service quota for your organization is extended to 300,000 Azure AD resources. The Azure AD service quota for organizations created by self-service sign-up remains 50,000 Azure AD 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 Azure AD pricing page. To go beyond the default quota, you must contact Microsoft Support. • A non-admin user can create no more than 250 Azure AD resources. Both active resources and deleted resources that are available to restore count toward this quota. Only deleted Azure AD resources that were deleted fewer than 30 days ago are available to restore. Deleted Azure AD resources that are no longer available to restore count toward this quota at a value of one-quarter for 30 days. If you have developers who are likely to repeatedly exceed this quota in the course of their regular duties, you can create and assign a custom role with permission to create a limitless number of app registrations. • Resource limitations apply to all directory objects in a given Azure AD tenant, including users, groups, applications, and service principals.
Schema extensions	• String-type extensions can have a maximum of 256 characters. • Binary-type extensions are limited to 256 bytes. • Only 100 extension values, across *all* types and *all* applications, can be written to any single Azure AD resource. • Only User, Group, TenantDetail, Device, Application, and ServicePrincipal entities can be extended with string-type or binary-type single-valued attributes.

Category	Limit
Applications	• 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 which is assigned. • A group can have credentials configured for a maximum of 48 apps using password-based single sign-on. • See additional limits in Validation differences by supported account types.
Application manifest	A maximum of 1,200 entries can be added to the application manifest. See additional limits in Validation differences by supported account types.
Groups	• A non-admin user can create a maximum of 250 groups in an Azure AD organization. Any Azure AD admin who can manage groups in the organization can also create an unlimited number of groups (up to the Azure AD 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. • An Azure AD organization can have a maximum of 5,000 dynamic groups and dynamic administrative units combined. • A maximum of 500 role-assignable groups can be created in a single Azure AD organization (tenant). • A maximum of 100 users can be owners of a single group. • Any number of Azure AD resources can be members of a single group. • A user can be a member of any number of groups. 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 Azure Active Directory by using Azure AD Connect is limited to 50,000 members. If you need to sync a group membership that's over this limit, you must onboard the Azure AD 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: • One group can be added as a member of another group, and you can achieve group nesting. • Group membership claims. When an app is configured to receive group membership claims in the token, nested groups in which the signed-in user is a member are included. • Conditional access (when a conditional access policy has a group scope). • Restricting access to self-serve password reset. • Restricting which users can do Azure AD Join and device registration. The following scenarios are *not* supported with nested groups: • 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	• A maximum of 500 transactions* per second per Application Proxy application. • A maximum of 750 transactions per second for the Azure AD organization. *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).
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 additional data is truncated.
Administrative units	• An Azure AD resource can be a member of no more than 30 administrative units. • A maximum of 100 restricted management administrative units in a tenant. • An Azure AD organization can have a maximum of 5,000 dynamic groups and dynamic administrative units combined.
Azure AD roles and permissions	• A maximum of 100 Azure AD custom roles can be created in an Azure AD organization. • A maximum of 150 Azure AD custom role assignments for a single principal at any scope. • A maximum of 100 Azure AD built-in role assignments for a single principal at non-tenant scope (such as an administrative unit or Azure AD object). There is no limit to Azure AD built-in role assignments at tenant scope. For more information, see Assign Azure AD roles at different scopes. • A group can't be added as a group owner.

Category	Limit
	• A user's ability to read other users' tenant information can be restricted only by the Azure AD organization-wide switch to disable all non-admin users' access to all tenant information (not recommended). For more information, see To restrict the default permissions for member users. • It 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 Azure AD organization (tenant).
Terms of use	You can add no more than 40 terms to a single Azure AD organization (tenant).
Multi-tenant organizations	• 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 multi-tenant organization if the limit is exceeded. This limit is applied at the time a pending tenant joins a multi-tenant organization. • A maximum of 100,000 internal users per active tenant. This limit is applied at the time a pending tenant joins a multi-tenant organization.

API Center (preview) limits

Resource	Limit[1]
Maximum number of APIs	1,000
Maximum number of versions per API	100
Maximum number of environments	1,000
Maximum number of custom metadata properties	100
Maximum number of child properties in custom metadata property of type "object"	100

[1] To increase an API Center limit, contact support.

API Management limits

Resource	Limit
Maximum number of scale units	12 per region[1]
Cache size	5 GiB per unit[2]
Concurrent back-end connections[3] per HTTP authority	2,048 per unit[4]
Maximum cached response size	2 MiB
Maximum policy document size	256 KiB[5]
Maximum custom gateway domains per service instance[6]	20
Maximum number of CA certificates per service instance[7]	10
Maximum number of service instances per subscription[8]	20
Maximum number of subscriptions per service instance[8]	500
Maximum number of client certificates per service instance[8]	50
Maximum number of APIs per service instance[8]	50
Maximum number of API management operations per service instance[8]	1,000
Maximum total request duration[8]	30 seconds
Maximum request payload size[8]	1 GiB
Maximum buffered payload size[8]	2 MiB
Maximum request/response payload size in diagnostic logs	8,192 bytes

Resource	Limit
Maximum request URL size[9]	16,384 bytes
Maximum length of URL path segment[10]	1,024 characters
Maximum size of API schema used by validation policy[10]	4 MB
Maximum number of schemas[10]	100
Maximum size of request or response body in validate-content policy[10]	100 KB
Maximum number of self-hosted gateways[11]	25
Maximum number of active WebSocket connections per unit	5,000[12]
Maximum number of tags supported by an API Management resource	15
Maximum number of authorization providers per service instance	1,000
Maximum number of authorizations per authorization provider	10,000
Maximum number of access policies per authorization	100
Maximum number of authorization requests per minute per authorization	250
Maximum number of workspaces per service instance[10]	100

[1] Scaling limits depend on the pricing tier. For details on the pricing tiers and their scaling limits, see API Management pricing.
[2] Per unit cache size depends on the pricing tier. To see the pricing tiers and their scaling limits, see API Management pricing.
[3] Connections are pooled and reused unless explicitly closed by the back end.
[4] This limit is per unit of the Basic, Standard, and Premium tiers. The Developer tier is limited to 1,024. This limit doesn't apply to the Consumption tier.
[5] This limit applies to the Basic, Standard, and Premium tiers. In the Consumption tier, policy document size is limited to 16 KiB.
[6] Multiple custom domains are supported in the Developer and Premium tiers only.
[7] CA certificates are not supported in the Consumption tier.
[8] This limit applies to the Consumption tier only. There are no specific limits in other tiers but depends on service infrastructure, policy configuration, number of concurrent requests, and other factors.
[9] Applies to the Consumption tier only. Includes an up to 2048-bytes long query string.
[10] To increase this limit, contact support.
[11] Self-hosted gateways are supported in the Developer and Premium tiers only. The limit applies to the number of self-hosted gateway resources. To raise this limit contact support. Note, that the number of nodes (or replicas) associated with a self-hosted gateway resource is unlimited in the Premium tier and capped at a single node in the Developer tier.
[12] This limit does not apply to Developer tier. In the Developer tier, the limit is 2,500.

App Service limits

Resource	Free	Shared	Basic	Standard	Premium (v1-v3)	Isolated
Web, mobile, or API apps per Azure App Service plan[1]	10	100	Unlimited[2]	Unlimited[2]	Unlimited[2]	Unlimited[2]
App Service plan	10 per region 1 free Linux App Service plan 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]

Resource	Free	Shared	Basic	Standard	Premium (v1-v3)	Isolated
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
Hybrid connections			5 per plan	25 per plan	220 per app	220 per app
Virtual Network Integration			X	X	X	X
Private Endpoints			X	X	100 per app	
Integrated load balancer		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	Scheduled backups every 2 hours, a maximum of 12 backups per day	Scheduled backups every hour, a maximum of 50 backups per day	Scheduled backups every hour, a maximum of 50 backups per day

Resource	Free	Shared	Basic	Standard	Premium (v1-v3)	Isolated
			(manual + scheduled	(manual + scheduled)	(manual + scheduled)	(manual + scheduled)
Autoscale				X	X	X
WebJobs[10]	X	X	X	X	X	X
Endpoint monitoring			X	X	X	X
Staging slots per app				5	20	20
Testing in Production				X	X	X
Diagnostic Logs	X	X	X	X	X	X
Kudu	X	X	X	X	X	X
Authentication and Authorization	X	X	X	X	X	X
App Service Managed Certificates [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).

[7]If 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/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).

Automation limits

Process automation

Resource	Limit	Notes
Maximum number of new jobs that can be submitted every 30 seconds per Azure Automation account (nonscheduled jobs)	100	When this limit is reached, the subsequent requests to create a job fail. The client receives an error response.
Maximum number of concurrent running jobs at the same instance of time per Automation account (nonscheduled jobs)	200	When this limit is reached, the subsequent requests to create a job fail. The client receives an error response.
Maximum storage size of job metadata for a 30-day rolling period	10 GB (approximately 4 million jobs)	When this limit is reached, the subsequent requests to create a job fail.
Maximum job stream limit	1 MiB	A single stream cannot be larger than 1 MiB.
Maximum 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	
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 Automation accounts in a subscription	No limit	
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	

[1]A sandbox is a shared environment that can be used by multiple jobs. Jobs that use the same sandbox are bound by the resource limitations of the sandbox.

Change Tracking and Inventory

The following table shows the tracked item limits per machine for change tracking.

Resource	Limit	Notes
File	500	
File size	5 MB	
Registry	250	
Windows software	250	Doesn't include software updates.
Linux packages	1,250	
Services	250	
Daemon	250	

Update Management

The following table shows the limits for Update Management.

Resource	Limit	Notes
Number of machines per update deployment	1000	
Number of dynamic groups per update deployment	500	

Azure App Configuration

Resource	Limit	Comment
Configuration stores for Free tier	1 store per subscription	
Configuration stores for Standard 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 will be 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
Storage for Free tier	10 MB	
Storage for Standard tier	1 GB	
Keys and Values	10 KB	For a single key-value item, including all metadata

Azure Cache for Redis limits

Resource	Limit
Cache size	1.2 TB
Databases	64
Maximum connected clients	40,000
Azure Cache for Redis replicas, for high availability	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

Resource	Limit
Web or worker roles per deployment[1]	25
Instance input endpoints per deployment	25
Input endpoints per deployment	25
Internal endpoints per deployment	25
Hosted service certificates per deployment	199

[1]Each Azure Cloud Service with web or worker roles can have two deployments, one for production and one for staging. This limit refers to the number of distinct roles, that is, configuration. This limit doesn't refer to the number of instances per role, that is, scaling.

Azure Cognitive Search limits

Pricing tiers determine the capacity and limits of your search service. Tiers include:

- **Free** multi-tenant service, shared with other Azure subscribers, is intended for evaluation and small development projects.
- **Basic** provides dedicated computing resources for production workloads at a smaller scale, with up to three replicas for highly available query workloads.
- **Standard**, which includes S1, S2, S3, and S3 High Density, is for larger production workloads. Multiple levels exist within the Standard tier so that you can choose a resource configuration that best matches your workload profile.

Limits per subscription

You can create multiple *billable* search services (Basic and higher), limited only by the number of services allowed at each tier. 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. For more information about tiers, see Choose an SKU or tier for Azure Cognitive Search.

Maximum service limits can be raised upon request. If you need more services within the same subscription, file a support request.

Resource	Free[1]	Basic	S1	S2	S3	S3 HD	L1	L2
Maximum services	1	16	16	8	6	6	6	6
Maximum scale in search units (SU)[2]	N/A	3 SU	36 SU	36 SU	36 SU	36 SU	36 SU	36 SU

[1] You can have one free search service per Azure subscription. The free tier is based on infrastructure that's shared with other customers. Because the hardware isn't dedicated, scale-up isn't supported, and storage is limited to 50 MB.

[2] Search units are billing units, allocated as either a *replica* or a *partition*. You need both resources for storage, indexing, and query operations. To learn more about SU computations, see Scale resource levels for query and index workloads.

Limits per search service

A search service is constrained by disk space or by a hard limit on the maximum number of indexes or indexers, whichever comes first. The following table documents storage limits. For maximum object limits, see Limits by resource.

Resource	Free	Basic[1]	S1	S2	S3	S3 HD	L1	L2
Service level agreement (SLA)[2]	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes

Resource	Free	Basic[1]	S1	S2	S3	S3 HD	L1	L2
Storage per partition	50 MB	2 GB	25 GB	100 GB	200 GB	200 GB	1 TB	2 TB
Partitions per service	N/A	1	12	12	12	3	12	12
Partition size	N/A	2 GB	25 GB	100 GB	200 GB	200 GB	1 TB	2 TB
Replicas	N/A	3	12	12	12	12	12	12

[1] Basic has one fixed partition. Additional search units can be used to add replicas for larger query volumes.

[2] Service level agreements are in effect for billable services on dedicated resources. Free services and preview features have no SLA. For billable services, SLAs take effect when you provision sufficient redundancy for your service. Two or more replicas are required for query (read) SLAs. Three or more replicas are required for query and indexing (read-write) SLAs. The number of partitions isn't an SLA consideration.

To learn more about limits on a more granular level, such as document size, queries per second, keys, requests, and responses, see Service limits in Azure Cognitive Search.

Azure 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 Communications Gateway limits

Some of the following default limits and quotas can be increased. To request a change, create a change request stating the limit you want 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 the Azure Communications Gateway as defined in Plan and manage costs for Azure Communications Gateway.

Azure Communications Gateway also has limits on the SIP signaling.

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

Resource	Limit
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 will strip them, because this behavior isn't permitted by RFC 3261.

- Request URI
- To header
- From header

Azure Container Apps limits

For Azure Container Apps limits, see Quotas in Azure Container Apps.

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.

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

* 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

For Azure Cosmos DB limits, see Limits in Azure Cosmos DB.

Azure Data Explorer limits

The following table describes the maximum limits for Azure Data Explorer clusters.

Resource	Limit
Clusters per region per subscription	20
Instances per cluster	1000
Number of databases in a cluster	10,000
Number of follower clusters (data share consumers) per leader cluster (data share producer)	100

The following table describes the limits on management operations performed on Azure Data Explorer clusters.

Scope	Operation	Limit
Cluster	read (for example, get a cluster)	500 per 5 minutes
Cluster	write (for example, create a database)	1000 per hour

Azure Database for MySQL

For Azure Database for MySQL limits, see Limitations in Azure Database for MySQL.

Azure Database for PostgreSQL

For Azure Database for PostgreSQL limits, see Limitations in Azure Database for PostgreSQL.

Azure Deployment Environments limits

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

To learn more about the limits for Azure Files and File Sync, see Azure Files scalability and performance targets.

Azure Functions limits

Resource	Consumption plan	Premium plan	Dedicated plan	ASE	Kubernetes
Default timeout duration (min)	5	30	30[1]	30	30
Max timeout duration (min)	10	unbounded[7]	unbounded[2]	unbounded	unbounded
Max outbound connections (per instance)	600 active (1200 total)	unbounded	unbounded	unbounded	unbounded
Max request size (MB)[3]	100	100	100	100	Depends on cluster
Max query string length[3]	4096	4096	4096	4096	Depends on cluster
Max request URL length[3]	8192	8192	8192	8192	Depends on cluster
ACU per instance	100	210-840	100-840	210-250[8]	AKS pricing
Max memory (GB per instance)	1.5	3.5-14	1.75-14	3.5 - 14	Any node is supported
Max instance count	200/100	100/20	varies by SKU[9]	100[9]	Depends on

Resource	Consumption plan	Premium plan	Dedicated plan	ASE	Kubernetes
(Windows/Linux)					cluster
Function apps per plan[11]	100	100	unbounded[4]	unbounded	unbounded
App Service plans	100 per region	100 per resource group	100 per resource group	-	-
Deployment slots per app[10]	2	3	1-20[9]	20	n/a
Storage[5]	5 GB	250 GB	50-1000 GB	1 TB	n/a
Custom domains per app	500[6]	500	500	500	n/a
Custom domain SSL support	unbounded SNI SSL connection included	unbounded SNI SSL and 1 IP SSL connections included	unbounded SNI SSL and 1 IP SSL connections included	unbounded SNI SSL and 1 IP SSL connections included	n/a

[1] By default, the timeout for the Functions 1.x runtime in an App Service plan is unbounded.
[2] Requires the App Service plan be set to Always On. Pay at standard rates.
[3] These limits are set in the host.
[4] The actual number of function apps that you can host depends on the activity of the apps, the size of the machine instances, and the corresponding resource utilization.
[5] The storage limit is the total content size in temporary storage across all apps in the same App Service plan. Consumption plan uses Azure Files for temporary storage.
[6] When your function app is hosted in a Consumption plan, only the CNAME option is supported. For function apps in a Premium plan or an App Service plan, you can map a custom domain using either a CNAME or an A record.
[7] Guaranteed for up to 60 minutes.
[8] Workers are roles that host customer apps. Workers are available in three fixed sizes: One vCPU/3.5 GB RAM; Two vCPU/7 GB RAM; Four vCPU/14 GB RAM.
[9] See App Service limits for details.
[10] Including the production slot.
[11] There's currently a limit of 5000 function apps in a given subscription.

For more information, see Functions Hosting plans comparison.

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.

FHIR service in Azure Health Data Services has a limit of 4 TB for structured storage.

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.

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

Resource	Limit
Maximum clusters per subscription	5000 Note: spread clusters across different regions to account for Azure API throttling limits
Maximum nodes per cluster with Virtual Machine Scale Sets and Standard Load Balancer SKU	5000 across all node-pools (default limit: 1000) Note: Running more than a 1000 nodes per cluster requires increasing the default node limit quota. Contact support for assistance.
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	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	Maximum: 250 Default: 30
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

Kubernetes Control Plane tier	Limit
Standard tier	Automatically scales Kubernetes API server based on load. Larger control plane component limits and API server/etc 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

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

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
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

For Azure Load Testing limits, see Service limits in Azure Load Testing.

Azure Machine Learning limits

The latest values for Azure Machine Learning Compute quotas can be found in the Azure Machine Learning quota page

Azure Maps limits

The following table shows the usage limit for the Azure Maps S0 pricing tier. Usage limit depends on the pricing tier.

Resource	S0 pricing tier limit
Maximum request rate per subscription	50 requests per second

The following table shows the cumulative data size limit for Azure Maps accounts in an Azure subscription. The Azure Maps Data service is available only at the S1 pricing tier.

Resource	Limit
Maximum storage per Azure subscription	1 GB

Resource	Limit
Maximum size per file upload	100 MB

For more information on the Azure Maps pricing tiers, see Azure Maps pricing.

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.

Version 2

Resource	Default limit	Maximum limit
Total number of entities, such as pipelines, data sets, triggers, linked services, Private Endpoints, and integration runtimes, within a data factory	5,000	Find out how to request a quota increase from support.
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 does not 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 does not 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 does not 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 1[2]: 6,000 Region group 2[2]: 3,000 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	Find out how to request a quota increase from support.
Maximum activities per pipeline, which includes inner activities for containers	40	40
Maximum number of linked integration runtimes that can be created against a single self-hosted integration runtime	100	100
Maximum number of node that can be created against a single self-hosted integration runtime	4	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
Maximum queued runs per pipeline	100	100

Resource	Default limit	Maximum limit
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 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	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 factory	3	3
Data Flow Azure IR TTL limit	4 hrs	4 hrs
Meta Data Entity Size limit in a factory	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 Data Factory pricing.

[2] Azure Integration Runtime is globally available to ensure data compliance, efficiency, and reduced network egress costs.

Region group	Regions
Region group 1	Central US, East US, East US 2, North Europe, West Europe, West US, West US 2
Region group 2	Australia East, Australia Southeast, Brazil South, Central India, Japan East, North Central US, South Central US, Southeast Asia, West Central US
Region group 3	Other regions

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.

Version 1

Resource	Default limit	Maximum limit
Pipelines within a data factory	2,500	Find out how to request a quota increase from support.
Data sets within a data factory	5,000	Find out how to request a quota increase from support.
Concurrent slices per data set	10	10
Bytes per object for pipeline objects[1]	200 KB	200 KB
Bytes per object for data set and linked service objects[1]	100 KB	2,000 KB
Azure HDInsight on-demand cluster cores within a subscription[2]	60	Find out how to request a quota increase from support.
Cloud data movement units per copy activity run[3]	32	32
Retry count for pipeline activity runs	1,000	MaxInt (32 bit)

[1] Pipeline, data set, and linked service objects represent a logical grouping of your workload. Limits for these objects don't relate to the amount of data you can move and process with Azure Data Factory. Data Factory is designed to scale to handle petabytes of data.

[2] On-demand HDInsight cores are allocated out of the subscription that contains the data factory. As a result, the previous limit is the Data Factory-enforced core limit for on-demand HDInsight cores. It's different from the core limit that's associated with your Azure subscription.

[3] The cloud data movement unit (DMU) for version 1 is used in a cloud-to-cloud copy operation, learn more from Cloud data movement units (version 1). For information on billing, see Azure Data Factory pricing.

Resource	Default lower limit	Minimum limit
Scheduling interval	15 minutes	15 minutes
Interval between retry attempts	1 second	1 second
Retry timeout value	1 second	1 second

Web service call limits

Azure Resource Manager has limits for API calls. You can make API calls at a rate within the Azure Resource Manager API limits.

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.

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

Where	What	Maximum count
Initiative definition	Policies	1000
Initiative definition	Parameters	300
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 additional limits to the number of conditions and their complexity. See 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

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

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).

Area	Resource	Limit
Azure role assignments		

Area	Resource	Limit
	Azure role assignments per Azure subscription	4,000
	Azure role assignments per management group	500
	Size of description for Azure role assignments	2 KB
	Size of condition for Azure role assignments	8 KB
Azure custom roles		
	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

Resource	Default limit	Maximum limit
Azure SignalR Service units per instance for Free tier	1	1
Azure SignalR Service units per instance for Standard tier	100	100
Azure SignalR Service units per subscription per region for Free tier	5	5
Total Azure SignalR Service unit counts per subscription per region	150	Unlimited
Concurrent connections per unit for Free tier	20	20
Concurrent connections per unit for Standard tier	1,000	1,000
Included messages per unit per day for Free tier	20,000	20,000
Additional messages per unit per day for Free tier	0	0
Included messages per unit per day for Standard tier	1,000,000	1,000,000
Additional messages per unit per day for Standard tier	Unlimited	Unlimited

To request an update to your subscription's default limits, open a support ticket.

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 scale SignalR Service with multiple instances?.

Azure Spring Apps limits

To learn more about the limits for Azure Spring Apps, see Quotas and service plans 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**.

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, file shares, tables, queues, entities, or messages per storage account.	No limit
Default maximum request rate per storage account	20,000 requests per second[2]
Default maximum ingress per general-purpose v2 and Blob storage account in the following regions (LRS/GRS): • Australia East • Central US • East Asia • East US 2 • Japan East • Korea Central • North Europe • South Central US • Southeast Asia • UK South • West Europe • West US	60 Gbps[2]
Default maximum ingress per general-purpose v2 and Blob storage account in the following regions (ZRS): • Australia East • Central US • East US • East US 2 • Japan East • North Europe • South Central US • Southeast Asia • UK South • West Europe • West US 2	60 Gbps[2]
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]

Resource	Limit
Default maximum egress for general-purpose v2 and Blob storage accounts in the following regions (LRS/GRS): • Australia East • Central US • East Asia • East US 2 • Japan East • Korea Central • North Europe • South Central US • Southeast Asia • UK South • West Europe • West US	120 Gbps[2]
Default maximum egress for general-purpose v2 and Blob storage accounts in the following regions (ZRS): • Australia East • Central US • East US • East US 2 • Japan East • North Europe • South Central US • Southeast Asia • UK South • West Europe • West US 2	120 Gbps[2]
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	200
Maximum number of virtual network rules per storage account	200
Maximum number of resource instance rules per storage account	200
Maximum number of private endpoints per storage account	200

[1] 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. The limits apply per region of the resource in the request.

Resource	Limit
Storage account management operations (read)	800 per 5 minutes
Storage account management operations (write)	10 per second / 1200 per hour
Storage account management operations (list)	100 per 5 minutes

Azure Blob Storage limits

Resource	Target
Maximum size of single blob container	Same as maximum storage account capacity
Maximum number of blocks in a block blob or append blob	50,000 blocks
Maximum size of a block in a block blob	4000 MiB
Maximum size of a block blob	50,000 X 4000 MiB (approximately 190.7 TiB)
Maximum size of a block in an append blob	4 MiB
Maximum size of an append blob	50,000 x 4 MiB (approximately 195 GiB)
Maximum size of a page blob	8 TiB[2]
Maximum number of stored access policies per blob container	5
Target request rate for a single blob	Up to 500 requests per second
Target throughput for a single page blob	Up to 60 MiB per second[2]
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, upload larger blobs or blocks. Specifically, call the Put Blob or Put Block operation with a blob or block size that is greater than 4 MiB for standard storage accounts. For premium block blob or for Data Lake Storage Gen2 storage accounts, use a block or blob size that is greater than 256 KiB.

[2] Page blobs 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.

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

Resource	Target
Maximum size of a single queue	500 TiB
Maximum size of a message in a queue	64 KiB
Maximum number of stored access policies per queue	5
Maximum request rate per storage account	20,000 messages per second, which assumes a 1-KiB message size
Target throughput for a single queue (1-KiB messages)	Up to 2,000 messages per second

Azure Table storage limits

The following table describes capacity, scalability, and performance targets for Table storage.

Resource	Target
Number of tables in an Azure storage account	Limited only by the capacity of the storage account

Resource	Target
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, **PartitionKey**, **RowKey**, and **Timestamp**)
Maximum total size of an individual property in an entity	Varies by property type. For more information, see **Property Types** in Understanding the Table Service Data Model.
Size of the **PartitionKey**	A string up to 1 KiB in size
Size of the **RowKey**	A string up to 1 KiB in size
Size of an entity group transaction	A transaction can include at most 100 entities and the payload must be less than 4 MiB in size. An entity group transaction can include an update to an entity only once.
Maximum number of stored access policies per table	5
Maximum request rate per storage account	20,000 transactions per second, which assumes a 1-KiB entity size
Target throughput for a single table partition (1 KiB-entities)	Up to 2,000 entities per second

Azure subscription creation limits

To learn more about the creation limits for Azure subscriptions, see Billing accounts and scopes in the Azure portal.

Azure Virtual Desktop Service limits

The following table describes the maximum limits for Azure Virtual Desktop.

Azure Virtual Desktop Object	Per Parent Container Object	Service Limit
Workspace	Azure Active Directory Tenant	1300
HostPool	Workspace	400
Application group	Azure Active Directory Tenant	500[1]
RemoteApp	Application group	500
Role Assignment	Any Azure Virtual Desktop Object	200
Session Host	HostPool	10,000

[1]If 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.

Resource	Limit
vSphere clusters per private cloud	12
Minimum number of ESXi hosts per cluster	3 (hard-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 details, 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 port speed	10 Gbps (use Ultra Performance Gateway SKU with FastPath enabled) The virtual network gateway used determines the actual bandwidth. For more details, see About ExpressRoute virtual network gateways
Maximum number of Azure Public IPv4 addresses assigned to NSX-T Data Center	2,000
Maximum number of Azure VMware Solution Interconnects per private cloud	10
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*.

For other VMware-specific limits, use the VMware configuration maximum tool.

Backup limits

For a summary of Azure Backup support settings and limitations, see Azure Backup Support Matrices.

Batch limits

Resource	Default limit	Maximum limit
Azure Batch accounts per region per subscription	1-3	50
Dedicated cores per Batch account	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]

Resource	Default limit	Maximum limit
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

If you use classic deployment model instead of the Azure Resource Manager deployment model, the following limits apply.

Resource	Default limit	Maximum limit
vCPUs per subscription [1]	20	10,000
Coadministrators per subscription	200	200
Storage accounts per subscription[2]	100	100
Cloud services per subscription	20	200
Local networks per subscription	10	500
DNS servers per subscription	9	100
Reserved IPs per subscription	20	100
Affinity groups per subscription	256	256
Subscription name length (characters)	64	64

[1]Extra small instances count as one vCPU toward the vCPU limit despite using a partial CPU core.

[2]The storage account limit includes both Standard and Premium storage accounts.

Container Instances limits

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

Resource	Actual Limit
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]

[1]To 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.
[2]Default limit for Pay-As-You-Go subscription. Limit may differ for other category types.

Container Registry limits

The following table details the features and limits of the Basic, Standard, and Premium service tiers.

Resource	Basic	Standard	Premium
Included storage[1] (GiB)	10	100	500
Storage limit (TiB)	20	20	20
Maximum image layer size (GiB)	200	200	200
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.

Content Delivery Network limits

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

A Content Delivery Network subscription can contain one or more Content Delivery Network profiles. A Content Delivery Network profile can contain one or more Content Delivery Network endpoints. You might want to use multiple profiles to organize your Content Delivery Network endpoints by internet domain, web application, or some other criteria.

Data Lake Analytics limits

Azure Data Lake Analytics makes the complex task of managing distributed infrastructure and complex code easy. It dynamically provisions resources, and you can use it to do analytics on exabytes of data. When the job completes, it winds down resources automatically. You pay only for the processing power that was used. As you increase or decrease the size of data stored or the amount of compute used, you don't have to rewrite code. To raise the default limits for your subscription, contact support.

Resource	Limit	Comments
Maximum number of concurrent jobs	20	
Maximum number of analytics units (AUs) per account	250	Use any combination of up to a maximum of 250 AUs across 20 jobs. To increase this limit, contact Microsoft Support.
Maximum script size for job submission	3 MB	
Maximum number of Data Lake Analytics accounts per region per subscription	5	To increase this limit, contact Microsoft Support.

Data Factory limits

Azure Data Factory is a multitenant service that has the following default limits in place to make sure customer subscriptions are protected from each other's workloads. To raise the limits up to the maximum for your subscription, contact support.

Version 2

Resource	Default limit	Maximum limit
Total number of entities, such as pipelines, data sets, triggers, linked services, Private Endpoints, and integration runtimes, within a data factory	5,000	Find out how to request a quota increase from support.
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 does not 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 does not 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 does not 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 1[2]: 6,000 Region group 2[2]: 3,000 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	Find out how to request a quota increase from support.
Maximum activities per pipeline, which includes inner activities for containers	40	40
Maximum number of linked integration runtimes that can be created against a single self-hosted integration runtime	100	100
Maximum number of node that can be created against a single self-hosted integration runtime	4	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
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 Data Factory.

Resource	Default limit	Maximum limit
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	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 factory	3	3
Data Flow Azure IR TTL limit	4 hrs	4 hrs
Meta Data Entity Size limit in a factory	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 Data Factory pricing.

[2] Azure Integration Runtime is globally available to ensure data compliance, efficiency, and reduced network egress costs.

Region group	Regions
Region group 1	Central US, East US, East US 2, North Europe, West Europe, West US, West US 2
Region group 2	Australia East, Australia Southeast, Brazil South, Central India, Japan East, North Central US, South Central US, Southeast Asia, West Central US
Region group 3	Other regions

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.

Version 1

Resource	Default limit	Maximum limit
Pipelines within a data factory	2,500	Find out how to request a quota increase from support.
Data sets within a data factory	5,000	Find out how to request a quota increase from support.
Concurrent slices per data set	10	10
Bytes per object for pipeline objects[1]	200 KB	200 KB
Bytes per object for data set and linked service objects[1]	100 KB	2,000 KB
Azure HDInsight on-demand cluster cores within a subscription[2]	60	Find out how to request a quota increase from support.
Cloud data movement units per copy activity run[3]	32	32
Retry count for pipeline activity runs	1,000	MaxInt (32 bit)

[1] Pipeline, data set, and linked service objects represent a logical grouping of your workload. Limits for these objects don't relate to the amount of data you can move and process with Azure Data Factory. Data Factory is designed to scale to handle petabytes of data.

[2] On-demand HDInsight cores are allocated out of the subscription that contains the data factory. As a result, the previous limit is the Data Factory-enforced core limit for on-demand HDInsight cores. It's different from the core limit that's associated with your Azure subscription.

[3] The cloud data movement unit (DMU) for version 1 is used in a cloud-to-cloud copy operation, learn more from Cloud data movement units (version 1). For information on billing, see Azure Data Factory pricing.

Resource	Default lower limit	Minimum limit
Scheduling interval	15 minutes	15 minutes
Interval between retry attempts	1 second	1 second
Retry timeout value	1 second	1 second

Web service call limits

Azure Resource Manager has limits for API calls. You can make API calls at a rate within the Azure Resource Manager API limits.

Data Lake Storage limits

Azure Data Lake Storage Gen2 is not a dedicated service or storage account type. It is the latest release of capabilities that are dedicated to big data analytics. These capabilities are available in a general-purpose v2 or BlockBlobStorage storage account, and you can obtain them by enabling the **Hierarchical namespace** feature of the account. For scale targets, see these articles.

- Scale targets for Blob storage.
- Scale targets for standard storage accounts.

Azure Data Lake Storage Gen1 is a dedicated service. It's an enterprise-wide hyper-scale repository for big data analytic workloads. You can use Data Lake Storage Gen1 to capture data of any size, type, and ingestion speed in one single place for operational and exploratory analytics. There's no limit to the amount of data you can store in a Data Lake Storage Gen1 account.

Resource	Limit	Comments
Maximum number of Data Lake Storage Gen1 accounts, per subscription, per region	10	To request an increase for this limit, contact support.
Maximum number of access ACLs, per file or folder	32	This is a hard limit. Use groups to manage access with fewer entries.
Maximum number of default ACLs, per file or folder	32	This is a hard limit. Use groups to manage access with fewer entries.

Data Share limits

Azure Data Share enables organizations to simply and securely share data with their customers and partners.

Resource	Limit
Maximum number of Data Share resources per Azure subscription	100
Maximum number of sent shares per Data Share resource	200
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

Resource	Limit
Maximum number of snapshot schedules per share	1

Database Migration Service Limits

Azure Database Migration Service is a fully managed service designed to enable seamless migrations from multiple database sources to Azure data platforms with minimal downtime.

Resource	Limit	Comments
Maximum number of services per subscription, per region	10	To request an increase for this limit, contact support.

Device Update for IoT Hub limits

> **Note**
>
> When a given resource or operation doesn't have adjustable limits, the default and the maximum limits are the same. When the limit can be adjusted, the following table includes both the default limit and maximum limit. The limit can be raised above the default limit but not above the maximum limit. Limits can only be adjusted for the Standard SKU. Limit adjustment requests are not accepted for Free SKU. Limit adjustment requests are evaluated on a case-by-case basis and approvals are not guaranteed.
>
> If you want to raise the limit or quota above the default limit, open an **online customer support request**.

This table provides the limits for the Device Update for IoT Hub resource in Azure Resource Manager:

Resource	Standard SKU Limit	Free SKU Limit	Adjustable?
Accounts per subscription	50	1	No
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

This table provides the various limits associated with the operations within Device Update for IoT Hub:

Operation	Standard SKU Limit	Free SKU Limit	Adjustable?
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 (includes 1 reserved deployment for Cancels)	5 (includes 1 reserved deployment for Cancels)	Yes
Number of total deployments per instance	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

Operation	Standard SKU Limit	Free SKU Limit	Adjustable?
Total data storage included per instance	100 GB	5 GB	No

Digital Twins limits

> **Note**
>
> Some areas of this service have adjustable limits, and others do not. This is represented in the following tables with the *Adjustable?* column. When the limit can be adjusted, the *Adjustable?* value is *Yes*.

Functional limits

The following table lists the functional limits of Azure Digital Twins.

Area	Capability	Default limit	Adjustable?
Azure resource	Number of Azure Digital Twins instances in a region, per subscription	10	Yes
Digital twins	Number of twins in an Azure Digital Twins instance	2,000,000	Yes
Digital twins	Number of digital twins that can be imported in a single 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 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 Jobs API)	250	No
Models	Number of models that can be imported in a single 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 `AND` / `OR` expressions in a query	50	Yes
Query	Number of array items in an `IN` / `NOT IN` clause	50	Yes
Query	Number of characters in a query	8,000	Yes
Query	Number of `JOINS` in a query	5	Yes

Rate limits

The following table reflects the rate limits of different APIs.

API	Capability	Default limit	Adjustable?
Jobs API	Number of requests per second	1	Yes
Jobs API	Number of bulk import jobs running concurrently	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
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 within its spec documentation in GitHub: Digital Twins Definition Language (DTDL) - version 2.

Query latency details are described in Query language. Limitations of particular query language features can be found in the query reference documentation.

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 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.

Namespace resource limits

Azure Event Grid namespaces is available in public preview and enables MQTT messaging, and HTTP pull delivery. The following limits apply to namespace resources in Azure Event Grid.

Limit description	Limit
Namespaces per Azure subscription	10
Maximum throughput units per namespace	20

See throughput units (TUs) for more information.

MQTT limits in namespace

The following limits apply to MQTT in Azure Event Grid namespace resource.

Limit description	Limit
MQTT connections per namespace	10,000 per TU
Sessions per namespace	10,000 per TU
MQTT inbound publish requests per namespace	Up to 1,000 messages per second or 1 MB per second per TU (whichever comes first)
MQTT inbound publish requests per connection	Up to 100 messages per second or 1 MB per second (whichever comes first)
MQTT outbound publish requests per namespace	Up to 1,000 messages per second or 1 MB per second per TU (whichever comes first)
MQTT outbound publish requests per connection	Up to 100 messages per second or 1 MB per second (whichever comes first)
Connect requests	200 requests per second per TU
Subscribe and unsubscribe requests	200 requests per second per TU
Max message size	512 KB
Topic size	256 B
Segments per topic filter	8
MQTTv5 topic aliases	10 per connection
Subscriptions per MQTT client session	50
Topic filters per MQTT SUBSCRIBE packet	10
Maximum keep-alive interval	1160
Registered client resources	10,000 clients per TU
CA certificates	2
Client groups	10
Topic spaces	10
Topic templates	10 per topic space
Permission bindings	100

Events limits in namespace

The following limits apply to events in Azure Event Grid namespace resource.

Limit description	Limit
Namespace topics	100 per TU
Event ingress	Up to 1,000 events per second or 1 MB per second per TU (whichever comes first)
Event egress	Up to 2,000 events per second or 2 MB per second per TU
Event duration period in topic	1 day
Subscriptions per topic	100
Connected clients per namespace (queue subscriptions)	1,000
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.

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 per second or 5 MB per second (whichever comes first)
Event size	1 MB 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.

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 per second or 5 MB per second (whichever comes first)
Private endpoint connections per domain	64
IP Firewall rules per topic	128

Event Hubs limits

The following tables provide quotas and limits specific to Azure Event Hubs. For information about Event Hubs pricing, see Event Hubs pricing.

Common limits for all tiers

The following limits are common across all tiers.

Limit	Notes	Value
Size of an event hub name	-	256 characters
Size of a consumer group name	Kafka protocol doesn't require the creation of a consumer group.	Kafka: 256 characters AMQP: 50 characters
Number of non-epoch receivers per consumer group	-	5

Limit	Notes	Value
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 may be 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**, 1 CU is assigned to the cluster. If you enable the **Support Scaling** option while creating the cluster, you'll be able to 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.

Limit	Basic	Standard	Premium	Dedicated
Maximum size of Event Hubs publication	256 KB	1 MB	1 MB	1 MB
Number of Event Hub consumer groups per event hub	1	20	100	1000 No limit per CU
Number of Kafka consumer groups per Namespace	NA	1000	1000	1000
Number of brokered connections per namespace	100	5,000	10000 per PU For example, if the namespace is assigned 3 PUs, the limit is 30000.	100, 000 per CU
Maximum retention period of event data	1 day	7 days	90 days	90 days
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.	1024 per event hub 2000 per CU
Number of namespaces per subscription	1000	1000	1000	1000 (50 per CU)
Number of event hubs per	10	10	100 per PU	1000

Limit	Basic	Standard	Premium	Dedicated
namespace				
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 mega bytes	N/A	25	100	1024
Number of schema groups in a schema registry or namespace	N/A	1 - excluding the default group	100 1 MB per schema	1000 1 MB per schema
Number of schema versions across all schema groups	N/A	25	1000	10000
Throughput per unit	Ingress - 1 MB/s or 1000 events per second Egress – 2 MB/s or 4096 events per second	Ingress - 1 MB/s or 1000 events per second Egress – 2 MB/s or 4096 events per second	No limits per PU *	No limits per CU *

* Depends on various factors such as resource allocation, number of partitions, storage, and so on.

> **Note**
>
> You can publish events individually or batched. The publication limit (according to SKU) applies regardless of whether it is a single event or a batch. Publishing events larger than the maximum threshold will be rejected.

IoT Central limits

IoT Central limits the number of applications you can deploy in a subscription to 100. If you need to increase this limit, contact Microsoft support. To learn more, see Azure IoT Central quota and limits.

IoT Hub limits

The following table lists the limits associated with the different service tiers S1, S2, S3, and F1. For information about the cost of each *unit* in each tier, see Azure IoT Hub pricing.

Resource	S1 Standard	S2 Standard	S3 Standard	F1 Free
Messages/day	400,000	6,000,000	300,000,000	8,000
Maximum units	200	200	10	1

> **Note**
>
> If you anticipate using more than 200 units with an S1 or S2 tier hub or 10 units with an S3 tier hub, contact Microsoft Support.

The following table lists the limits that apply to IoT Hub resources.

Resource	Limit
Maximum paid IoT hubs per Azure subscription	50
Maximum free IoT hubs per Azure subscription	1
Maximum number of characters in a device ID	128
Maximum number of device identities returned in a single call	1,000

Resource	Limit
IoT Hub message maximum retention for device-to-cloud messages	7 days
Maximum size of device-to-cloud message	256 KB
Maximum size of device-to-cloud batch	AMQP and HTTP: 256 KB for the entire batch MQTT: 256 KB for each message
Maximum messages in device-to-cloud batch	500
Maximum size of cloud-to-device message	64 KB
Maximum TTL for cloud-to-device messages	2 days
Maximum delivery count for cloud-to-device messages	100
Maximum cloud-to-device queue depth per device	50
Maximum delivery count for feedback messages in response to a cloud-to-device message	100
Maximum TTL for feedback messages in response to a cloud-to-device message	2 days
Maximum size of device twin	8 KB for tags section, and 32 KB for desired and reported properties sections each
Maximum length of device twin string key	1 KB
Maximum length of device twin string value	4 KB
Maximum depth of object in device twin	10
Maximum size of direct method payload	128 KB
Job history maximum retention	30 days
Maximum concurrent jobs	10 (for S3), 5 for (S2), 1 (for S1)
Maximum additional endpoints (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**
>
> If you need more than 50 paid IoT hubs in an Azure subscription, contact Microsoft Support.

> **Note**
>
> Currently, the total number of devices plus modules that can be registered to a single IoT hub is capped at 1,000,000. If you want to increase this limit, contact **Microsoft Support**.

IoT Hub throttles requests when the following quotas are exceeded.

Throttle	Per-hub value
Identity registry operations (create, retrieve, list, update, and delete), individual or bulk import/export	83.33/sec/unit (5,000/min/unit) (for S3). 1.67/sec/unit (100/min/unit) (for S1 and S2).
Device connections	6,000/sec/unit (for S3), 120/sec/unit (for S2), 12/sec/unit (for S1). Minimum of 100/sec.

Throttle	Per-hub value
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).
Jobs per-device operation throughput	50/sec/unit (for S3), maximum of 10/sec or 1/sec/unit (for S2), 10/sec (for S1).
Device stream initiation rate	5 new streams/sec (for S1, S2, S3, and F1 only).

IoT Hub Device Provisioning Service limits

> **Note**
>
> Some areas of this service have adjustable limits. This is represented in the tables below with the *Adjustable?* column. When the limit can be adjusted, the *Adjustable?* value is *Yes*.
>
> The actual value to which a limit can be adjusted may vary based on each customer's deployment. Multiple instances of DPS may be required for very large deployments.
>
> If your business requires raising an adjustable limit or quota above the default limit, you can submit a request for additional resources by opening a support ticket. **Requesting an increase does not guarantee that it will be granted, as it needs to be reviewed on a case-by-case basis. Please contact Microsoft support as early as possible during your implementation, to be able to determine if your request could be approved and plan accordingly.**

The following table lists the limits that apply to Azure IoT Hub Device Provisioning Service resources.

Resource	Limit	Adjustable?
Maximum device provisioning services per Azure subscription	10	Yes
Maximum number of registrations	1,000,000	Yes
Maximum number of individual enrollments	1,000,000	Yes
Maximum number of enrollment groups *(X.509 certificate)*	100	Yes
Maximum number of enrollment groups *(symmetric key)*	100	No
Maximum number of CAs	25	Yes
Maximum number of linked IoT hubs	50	No
Maximum size of message	96 KB	No

> **Tip**
>
> If the hard limit on symmetric key enrollment groups is a blocking issue, it is recommended to use individual enrollments as a workaround.

The Device Provisioning Service has the following rate limits.

Rate	Per-unit value	Adjustable?
Operations	1,000/min/service	Yes
Device registrations	1,000/min/service	Yes
Device polling operation	5/10 sec/device	No

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 region[1]):

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:

Transactions type	Maximum transactions allowed in 10 seconds, per vault per region[1]
Secret CREATE secret	300

Transactions type	Maximum transactions allowed in 10 seconds, per vault per region[1]
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

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

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)

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)

Operation	2048-bit	3072-bit	4096-bit
Create Key	1	1	1
Delete Key (soft-delete)	10	10	10
Purge Key	10	10	10
Backup Key	10	10	10
Restore Key	10	10	10
Get Key Information	1100	1100	1100
Encrypt	10000	10000	6000
Decrypt	1100	360	160
Wrap	10000	10000	6000
Unwrap	1100	360	160
Sign	1100	360	160
Verify	10000	10000	6000

EC key operations (number of operations per second per HSM instance)

This table describes number of operations per second for each curve type.

Operation	P-256	P-256K	P-384	P-521
Create Key	1	1	1	1
Delete Key (soft-delete)	10	10	10	10
Purge Key	10	10	10	10
Backup Key	10	10	10	10
Restore Key	10	10	10	10
Get Key Information	1100	1100	1100	1100
Sign	260	260	165	56
Verify	130	130	82	28

AES key operations (number of operations per second per HSM instance)

- Encrypt and Decrypt operations assume a 4KB packet size.
- Throughput limits for Encrypt/Decrypt apply to AES-CBC and AES-GCM algorithms.
- Throughput limits for Wrap/Unwrap apply to AES-KW algorithm.

Operation	128-bit	192-bit	256-bit
Create Key	1	1	1
Delete Key (soft-delete)	10	10	10
Purge Key	10	10	10
Backup Key	10	10	10
Restore Key	10	10	10
Get Key Information	1100	1100	1100
Encrypt	8000	8000	8000
Decrypt	8000	8000	8000
Wrap	9000	9000	9000
Unwrap	9000	9000	9000

Managed identity limits

- Each managed identity counts towards the object quota limit in an Azure AD tenant as described in Azure AD service limits and restrictions.
- The rate at which managed identities can be created have the following limits:
  1. Per Azure AD Tenant per Azure region: 400 create operations per 20 seconds.
  2. 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 :
  1. Per Azure AD Tenant per Azure region: 400 assignment operations per 20 seconds.
  2. Per Azure Subscription per Azure region : 300 assignment operations per 20 seconds.

Media Services limits

> **Note**
>
> For resources that aren't fixed, open a support ticket to ask for an increase in the quotas. Don't create additional Azure Media Services accounts in an attempt to obtain higher limits.

Account limits

Resource	Default Limit
Media Services accounts in a single subscription	100 (fixed)

Asset limits

Resource	Default Limit
Assets per Media Services account	1,000,000

Storage (media) limits

Resource	Default Limit
File size	In some scenarios, there is a limit on the maximum file size supported for processing in Media Services. (1)
Storage accounts	100(2) (fixed)

[1] The maximum size supported for a single blob is currently up to 5 TB in Azure Blob Storage. Additional limits apply in Media Services based on the VM sizes that are used by the service. The size limit applies to the files that you upload and also the files that get generated as a result of Media Services processing (encoding or analyzing). If your source file is larger than 260-GB, your Job will likely fail.

[2] The storage accounts must be from the same Azure subscription.

Jobs (encoding & analyzing) limits

Resource	Default Limit
Jobs per Media Services account	500,000 [(3)] (fixed)
Job inputs per Job	50 (fixed)
Job outputs per Job	20 (fixed)
Transforms per Media Services account	100 (fixed)
Transform outputs in a Transform	20 (fixed)
Files per job input	10 (fixed)

[3] This number includes queued, finished, active, and canceled Jobs. It does not include deleted Jobs.

Any Job record in your account older than 90 days will be automatically deleted, even if the total number of records is below the maximum quota.

Live streaming limits

Resource	Default Limit
Live Events [(4)] per Media Services account	5
Live Outputs per Live Event	3 [(5)]
Max Live Output duration	Size of the DVR window

[4] For detailed information about Live Event limitations, see Live Event types comparison and limitations.

[5] Live Outputs start on creation and stop when deleted.

Packaging & delivery limits

Resource	Default Limit
Streaming Endpoints (stopped or running) per Media Services account	2
Dynamic Manifest Filters	100
Streaming Policies	100 [(6)]
Unique Streaming Locators associated with an Asset at one time	100[(7)] (fixed)

[6] When using a custom Streaming Policy, you should design a limited set of such policies for your Media Service account, and re-use them for your StreamingLocators whenever the same encryption options and protocols are needed. You should not be creating a new Streaming Policy for each Streaming Locator.

[7] Streaming Locators are not designed for managing per-user access control. To give different access rights to individual users, use Digital Rights Management (DRM) solutions.

Protection limits

Resource	Default Limit
Options per Content Key Policy	30

Resource	Default Limit
Licenses per month for each of the DRM types on Media Services key delivery service per account	1,000,000

Support ticket

For resources that are not fixed, you may ask for the quotas to be raised, by opening a support ticket. Include detailed information in the request on the desired quota changes, use-case scenarios, and regions required.
Do **not** create additional Azure Media Services accounts in an attempt to obtain higher limits.

Media Services v2 (legacy)

For limits specific to Media Services v2 (legacy), see Media Services v2 (legacy)

Mobile Services limits

Tier	Free	Basic	Standard
API calls	500,000	1.5 million per unit	15 million per unit
Active devices	500	Unlimited	Unlimited
Scale	N/A	Up to 6 units	Unlimited units
Push notifications	Azure Notification Hubs Free tier included, up to 1 million pushes	Notification Hubs Basic tier included, up to 10 million pushes	Notification Hubs Standard tier included, up to 10 million pushes
Real-time messaging/ 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.

Multi-Factor Authentication limits

Resource	Default limit	Maximum limit
Maximum number of trusted IP addresses or ranges per subscription	0	50
Remember my devices, number of days	14	60
Maximum number of app passwords	0	No limit
Allow **X** attempts during MFA call	1	99
Two-way text message timeout seconds	60	600
Default one-time bypass seconds	300	1,800
Lock user account after **X** consecutive MFA denials	Not set	99
Reset account lockout counter after **X** minutes	Not set	9,999
Unlock account after **X** minutes	Not set	9,999

Networking limits

Networking limits - Azure Resource Manager

The following limits apply only for networking resources managed through **Azure Resource Manager** per region per subscription. Learn how to view your current resource usage against your subscription limits.

> **Note**
>
> We 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**

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	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

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.
Public IP Prefixes	limited by number of Standard Public IPs in a subscription	Contact support.
Public IP prefix length	/28	Contact support.

[1]Default limits for Public IP addresses vary by offer category type, such as Free Trial, Pay-As-You-Go, CSP. For example, the default for Enterprise Agreement subscriptions is 1000.

[2]Public IP addresses limit refers to the total amount of Public IP addresses, including Basic and Standard.

Load balancer limits

The following limits apply only for networking resources managed through Azure Resource Manager per region per subscription. Learn how to view your current resource usage against your subscription limits.

Standard Load Balancer

Resource	Limit
Load balancers	1,000
Frontend IP configurations	600
Rules (Load Balancer + Inbound NAT) per resource	1,500
Rules per NIC (across all IPs on a NIC)	300
High-availability ports rule	1 per internal frontend
Outbound rules per Load Balancer	600
Backend pool size	5,000
Backend IP configurations per frontend [1]	10,000
Backend IP configurations across all frontends	500,000

[1] 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

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

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

Resource	Limit
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.

The following limits apply only for networking resources managed through the **classic** deployment model per subscription. Learn how to view your current resource usage against your subscription limits.

Resource	Default limit	Maximum limit
Virtual networks	100	100
Local network sites	20	50
DNS servers per virtual network	20	20
Private IP addresses per virtual network	4,096	4,096
Concurrent TCP or UDP flows per NIC of a virtual machine or role instance	500,000, up to 1,000,000 for two or more NICs.	500,000, up to 1,000,000 for two or more NICs.
Network Security Groups (NSGs)	200	200
NSG rules per NSG	200	1,000
User-defined route tables	200	200
User-defined routes per route table	400	400
Public IP addresses (dynamic)	500	500
Reserved public IP addresses	500	500
Public IP per deployment	5	Contact support
Private IP (internal load balancing) per deployment	1	1
Endpoint access control lists (ACLs)	50	50

Application Gateway limits

The following table applies to v1, v2, Standard, and WAF SKUs unless otherwise stated.

Resource	Limit	Note
Azure Application Gateway	1,000 per region per subscription	
Frontend IP configurations	2	1 public and 1 private
Frontend ports	100[1]	
Backend address pools	100	
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]	

Resource	Limit	Note
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 file upload size (Standard SKU)	V2 - 4 GB V1 - 2 GB	
Maximum file upload size (WAF SKU)	V1 Medium - 100 MB V1 Large - 500 MB V2 - 750 MB V2 (with CRS 3.2 or newer) - 4 GB[3]	
WAF body size limit (without files)	V1 or V2 (with CRS 3.1 and older) - 128 KB	

Resource	Limit	Note
	V2 (with CRS 3.2 or newer) - 2 MB[3]	
Maximum Private Link Configurations	2	1 for public IP, 1 for private IP
Maximum Private Link IP Configurations	8	
Maximum WAF custom rules	100	
WAF IP address ranges per match condition	540 600 - with CRS 3.2 or newer	
Maximum WAF exclusions per Application Gateway	40 200 - with CRS 3.2 or newer	
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 higher. 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.

Application Gateway for Containers limits

Resource	Limit
Associations	1 per gateway
Frontends	5 per gateway

Kubernetes Ingress and Gateway API configuration limits

Resource	Limit
Total Rules	200 per cluster
Total Services	100 per cluster
Total Endpoints	5000 per cluster

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.

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 zones

Resource	Limit
Public DNS zones per subscription	250 [1]
Record sets per public DNS zone	10,000 [1]
Records per record set in public DNS zone	20
Number of Alias records for a single Azure resource	20

[1]If you need to increase these limits, contact Azure Support.

Private DNS zones

Resource	Limit
Private DNS zones per subscription	1000
Record sets per private DNS zone	25000
Records per record set for private DNS zones	20
Virtual Network Links per private DNS zone	1000
Virtual Networks Links per private DNS zones with 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

Azure-provided DNS resolver

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]

[1]These limits are applied to every individual virtual machine and not at the virtual network level. DNS queries exceeding these limits are dropped.

DNS private resolver[1]

Resource	Limit
DNS private resolvers per subscription	15
Inbound endpoints per DNS private resolver	5
Outbound endpoints per DNS private resolver	5
Forwarding rules per DNS forwarding ruleset	1000
Virtual network links per DNS forwarding ruleset	500
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

[1]Different 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

Resource	Limit
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** = sum of (unique source addresses * unique destination addresses for each rule) 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 60 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).
Minimum AzureFirewallSubnet size	/26
Port range in network and application rules	1 - 65535
Public IP addresses	250 maximum. All public IP addresses can be used in DNAT rules and they all contribute to available SNAT ports.
IP addresses in IP Groups	Maximum of 100 IP Groups per firewall. Maximum 5000 individual IP addresses or IP prefixes per each IP Group.
Route table	By default, AzureFirewallSubnet has a 0.0.0.0/0 route with the NextHopType value set to **Internet**. 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. 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.
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.

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	50[1]

Resource	Classic tier limit
Back ends per back-end pool	100
Path patterns to match for a routing rule	25
URLs in a single cache purge call	100
Custom web application firewall rules per policy	100
Web application firewall policy per subscription	100
Web application firewall match conditions per custom rule	10
Web application firewall IP address ranges per 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 size inspected	128 KB
Web application firewall custom response body length	32 KB

[1]To 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.

> **Note**
>
> Resources with * are new limits for Azure Front Door Standard and Premium.

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
Path patterns to match for a routing rule	25	50

Resource	Standard tier limit	Premium tier limit
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
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
WAF exclusion per policy	100	100
WAF HTTP request body size inspected	128 KB	128 KB
WAF custom response body length	32 KB	32 KB

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

	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

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

Resource	Limit
Number of BGP peers supported	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 [2]	4,000

[1] If your NVA advertises more routes than the limit, the BGP session gets dropped.

[2] The number of VMs that Azure Route Server can support isn't a hard limit and it depends on the availability and performance of the underlying infrastructure.

> **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.

ExpressRoute limits

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

Resource	Limit
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	30,000

Route advertisement limits

Resource	Local / Standard SKU	Premium SKU
Maximum number of IPv4 routes advertised to Azure private peering	4,000	10,000
Maximum number of IPv6 routes advertised to Azure private peering	100	100
Maximum number of IPv4 routes advertised from Azure private peering from the VNet address space	1,000	1,000
Maximum number of IPv6 routes advertised from Azure private peering from the VNet address space	100	100
Maximum number of IPv4 routes advertised to Microsoft peering	200	200
Maximum number of IPv6 routes advertised to Microsoft peering	200	200

Virtual networks links allowed for each ExpressRoute circuit limit

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 table shows the gateway types and the estimated performance scale numbers. These numbers are derived from the following testing conditions and represent the max support limits. Actual performance may vary, depending on how closely traffic replicates these testing conditions.

Testing conditions

Gateway SKU	Traffic sent from on-premises	Number of routes advertised by gateway	Number of routes learned by gateway
Standard/ERGw1Az	1 Gbps	500	4000
High Performance/ERGw2Az	2 Gbps	500	9,500
Ultra Performance/ErGw3Az	10 Gbps	500	9,500

Performance results

This table applies to both the Resource Manager and classic deployment models.

Gateway SKU	Connections per second	Mega-Bits per second	Packets per second	Supported number of VMs in the Virtual Network
Standard/ERGw1Az	7,000	1,000	100,000	2,000
High Performance/ERGw2Az	14,000	2,000	250,000	4,500
Ultra Performance/ErGw3Az	16,000	10,000	1,000,000	11,000

Important

- Application performance depends on multiple factors, such as end-to-end latency, and the number of traffic flows the application opens. The numbers in the table represent the upper limit that the application can theoretically achieve in an ideal environment. Additionally, Microsoft performs 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 may experience intermittent connectivity issues to private endpoint resources.

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.

Resource	Limit
Public IP addresses	16 per NAT gateway
Subnets	800 per NAT gateway
Data throughput	50 Gbps
NAT gateways	1,000 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

Private Link limits

The following limits apply to Azure private link:

Resource	Limit
Number of private endpoints per virtual network	1000
Number of private endpoints per subscription	64000

Resource	Limit
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

Traffic Manager limits

Resource	Limit
Profiles per subscription	200 [1]
Endpoints per profile	200

[1]If you need to increase these limits, contact Azure Support.

Virtual Network Gateway limits

Resource	Limit
VNet Address Prefixes	600 per VPN gateway
Aggregate BGP routes	4,000 per VPN gateway
Local Network Gateway address prefixes	1000 per local network gateway
S2S connections	Depends on the gateway SKU
P2S connections	Depends on the gateway SKU
P2S route limit - IKEv2	256 for non-Windows / 25 for Windows
P2S route limit - OpenVPN	1000
Max. flows	100K for VpnGw1/AZ / 512K for VpnGw2-4/AZ
Traffic Selector Policies	100

Virtual WAN limits

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

Resource	Limit
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).

Notification Hubs limits

Tier	Free	Basic	Standard
Included pushes	1 million	10 million	10 million
Active devices	500	200,000	10 million
Tag quota per installation or registration	60	60	60

For more information on limits and pricing, see Notification Hubs pricing.

Microsoft Dev Box limits

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

The latest values for Microsoft Purview quotas can be found in the Microsoft Purview quota page.

Microsoft Sentinel limits

For Microsoft Sentinel limits, see Service limits for Microsoft Sentinel

Service Bus limits

The following table lists quota information specific to Azure Service Bus messaging. For information about pricing and other quotas for Service Bus, see Service Bus pricing.

Quota name	Scope	Value	Notes
Maximum number of namespaces per Azure subscription	Namespace	1000 (default and maximum)	This limit is based on the `Microsoft.ServiceBus` 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	Defined upon creation/updation of the queue or topic. Subsequent incoming messages are rejected, and an exception is received by the calling code.

Quota name	Scope	Value	Notes
		queue or topic size is 80 GB. Total size limit for a premium namespace is 1 TB per messaging unit. Total size of all entities in a namespace can't exceed this limit.	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 **QuotaExceededException** is received by the calling code. If you want to have more partitioned entities in a basic or a standard tier namespace, create additional namespaces.
Maximum size of any messaging entity path: queue or topic	Entity	260 characters.	
Maximum size of any messaging entity name: namespace, subscription, or subscription rule	Entity	50 characters.	
Maximum size of a message ID	Entity	128	
Maximum size of a message session ID	Entity	128	
Message size for a queue, topic, or subscription entity	Entity	256 KB for Standard tier 100 MB for 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.	Incoming messages that exceed these quotas are rejected, and an exception is received by the calling code.
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. Maximum number of header properties in property bag: **byte/int.MaxValue**.	The exception `SerializationException` is generated.

Quota name	Scope	Value	Notes
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). Maximum number of expressions per rule action: 32.	Subsequent requests for creation of additional filters are rejected, and an exception is received by the calling code.
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 **Send()** and **SendAsync()** 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.
Number of virtual network and IP filter rules	Namespace	128	

Site Recovery limits

The following limits apply to Azure Site Recovery.

Limit identifier	Limit
Number of vaults per subscription	500
Number of servers per Recovery Services vault	250
Number of protection groups per Recovery Services vault	No limit
Number of recovery plans per Recovery Services vault	No limit
Number of servers per protection group	No limit
Number of servers per recovery plan	100

SQL Database limits

For SQL Database limits, see SQL Database resource limits for single databases, SQL Database resource limits for elastic pools and pooled databases, and SQL Database resource limits for SQL Managed Instance.

The maximum number of private endpoints per Azure SQL Database logical server is 250.

Azure Synapse Analytics limits

Azure Synapse Analytics has the following default limits to ensure customer's subscriptions are protected from each other's workloads. To raise the limits to the maximum for your subscription, contact support.

Azure Synapse limits for workspaces

For Pay-As-You-Go, Free Trial, Azure Pass, and Azure for Students subscription offer types:

Resource	Default limit	Maximum limit
Synapse workspaces in an Azure subscription	2	2

For other subscription offer types:

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:

Resource	Memory Optimized cores	GPU cores
Spark cores in a Synapse workspace	12	48

For other subscription offer types:

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

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]:	Region group 1[2]: 6,000 Region group 2[2]: 3,000 Region group 3[2]: 1,500

Resource	Default limit	Maximum limit
	3,000 Region group 3[2]: 1,500 Managed virtual network[2]: 2,400	Managed virtual network: Find out how to request a quota increase from support.
Maximum activities per pipeline, which includes inner activities for containers	40	40
Maximum number of linked integration runtimes that can be created against a single self-hosted integration runtime	100	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
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.

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	Regions
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.

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.

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

[1]An individual disk can have 500 incremental snapshots.

[2]This 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.

For more information, see Virtual machine sizes.

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.

For more information, see VM Applications.

Disk encryption sets

There's a limitation of 1000 disk encryption sets per region, per subscription. For more information, see the encryption documentation for Linux or Windows virtual machines. If you need to increase the quota, contact Azure support.

Managed virtual machine disks

Standard HDD managed disks

Standard Disk Type	S4	S6	S10	S15	S20	S30	S40	S50	S60	S70	S80
Disk size in GiB	32	64	128	256	512	1,024	2,048	4,096	8,192	16,384	32,767
Base IOPS per disk	Up to 500	Up to 500	Up to 500	Up to 500	Up to 500	Up to 500	Up to 500	Up to 500	Up to 1,300	Up to 2,000	Up to 2,000
*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 60 MB/s	Up to 60 MB/s	Up to 60 MB/s	Up to 60 MB/s	Up to 60 MB/s	Up to 60 MB/s	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	Up to 500 MB/s	Up to 500 MB/s	Up to 500 MB/s

* Only applies to disks with performance plus (preview) enabled.

Standard SSD managed disks

Standard SSD sizes	E1	E2	E3	E4	E6	E10	E15	E20	E30	E40	E50	E60	E70	E80
Disk size in GiB	4	8	16	32	64	128	256	512	1,024	2,048	4,096	8,192	16,384	32,767
Base IOPS per disk	Up to 500	Up to 500	Up to 500	Up to 500	Up to 500	Up to 500	Up to 500	Up to 500	Up to 500	Up to 500	Up to 500	Up to 2,000	Up to 4,000	Up to 6,000
*Expanded IOPS per disk	N/A	N/A	N/A	N/A	N/A	N/A	N/A	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 60 MB/s	Up to 60 MB/s	Up to 60 MB/s	Up to 60 MB/s	Up to 60 MB/s	Up to 60 MB/s	Up to 60 MB/s	Up to 60 MB/s	Up to 60 MB/s	Up to 60 MB/s	Up to 60 MB/s	Up to 400 MB/s	Up to 600 MB/s	Up to 750 MB/s
*Expanded throughput per disk	N/A	N/A	N/A	N/A	N/A	N/A	N/A	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	1000					
Max burst throughput per disk	150 MB/s	150 MB/s	150 MB/s	150 MB/s	150 MB/s	150 MB/s	150 MB/s	150 MB/s	250 MB/s					
Max burst duration	30 min	30 min	30 min	30 min	30 min	30 min	30 min	30 min	30 min					

* Only applies to disks with performance plus (preview) enabled.

Premium SSD managed disks: Per-disk limits

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 Throughput per disk	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
**Expanded provisioned	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	300 MB/s	600 MB/s	900 MB/s	900 MB/s	900 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														
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	170 MB/s	170 MB/s	170 MB/s	170 MB/s	170 MB/s	170 MB/s	170 MB/s	1,000 MB/s*	1,000 MB/s*	1,000 MB/s*	1,000 MB/s*	1,000 MB/s*	1,000 MB/s*
Max burst duration	30 min	30 min	30 min	30 min	30 min	30 min	30 min	30 min	Unlimited*	Unlimited*	Unlimited*	Unlimited*	Unlimited*	Unlimited*
Eligible for reservation	No	No	No	No	No	No	No	No	Yes, up to one year	Yes, up to one year	Yes, up to one year	Yes, up to one year	Yes, up to one year	Yes, up to one year

*Applies only to disks with on-demand bursting enabled.

** Only applies to disks with performance plus (preview) enabled.

Premium SSD managed disks: Per-VM limits

Resource	Limit
Maximum IOPS Per VM	80,000 IOPS with GS5 VM
Maximum throughput per VM	2,000 MB/s with GS5 VM

Unmanaged virtual machine disks

Standard unmanaged virtual machine disks: Per-disk limits

VM tier	Basic tier VM	Standard tier VM
Disk size	4,095 GB	4,095 GB
Maximum 8-KB IOPS per persistent disk	300	500
Maximum number of disks that perform the maximum IOPS	66	40

Premium unmanaged virtual machine disks: Per-account limits

Resource	Limit
Total disk capacity per account	35 TB
Total snapshot capacity per account	10 TB
Maximum bandwidth per account (ingress + egress)[1]	<=50 Gbps

[1]*Ingress* refers to all data from requests that are sent to a storage account. *Egress* refers to all data from responses that are received from a storage account.

Premium unmanaged virtual machine disks: Per-disk limits

Premium storage disk type	P10	P20	P30	P40	P50
Disk size	128 GiB	512 GiB	1,024 GiB (1 TB)	2,048 GiB (2 TB)	4,095 GiB (4 TB)
Maximum IOPS per disk	500	2,300	5,000	7,500	7,500
Maximum throughput per disk	100 MB/sec	150 MB/sec	200 MB/sec	250 MB/sec	250 MB/sec
Maximum number of disks per storage account	280	70	35	17	8

Premium unmanaged virtual machine disks: Per-VM limits

Resource	Limit
Maximum IOPS per VM	80,000 IOPS with GS5 VM
Maximum throughput per VM	2,000 MB/sec with GS5 VM

StorSimple System limits

Limit identifier	Limit	Comments
Maximum number of storage account credentials	64	
Maximum number of volume containers	64	
Maximum number of volumes	255	
Maximum number of schedules per bandwidth template	168	A schedule for every hour, every day of the week.
Maximum size of a tiered volume on physical devices	64 TB for StorSimple 8100 and StorSimple 8600	StorSimple 8100 and StorSimple 8600 are physical devices.
Maximum size of a tiered volume on virtual devices in Azure	30 TB for StorSimple 8010 64 TB for StorSimple 8020	StorSimple 8010 and StorSimple 8020 are virtual devices in Azure that use Standard storage and Premium storage, respectively.
Maximum size of a locally pinned volume on physical devices	9 TB for StorSimple 8100 24 TB for StorSimple 8600	StorSimple 8100 and StorSimple 8600 are physical devices.
Maximum number of iSCSI connections	512	
Maximum number of iSCSI connections from initiators	512	
Maximum number of access control records per device	64	
Maximum number of volumes per backup policy	24	
Maximum number of backups retained per backup policy	64	
Maximum number of schedules per backup policy	10	
Maximum number of snapshots of any type that can be retained per volume	256	This amount includes local snapshots and cloud snapshots.
Maximum number of snapshots that can be present in any device	10,000	
Maximum number of volumes that can be processed in parallel for backup, restore, or clone	16	• If there are more than 16 volumes, they're processed sequentially as processing slots become available. • New backups of a cloned or a restored tiered volume can't occur until the operation is finished. For a local volume, backups are allowed after the volume is online.

Limit identifier	Limit	Comments
Restore and clone recover time for tiered volumes	<2 minutes	• 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	• The volume is made available within 2 minutes of the restore operation, regardless of the volume size. • The volume performance might initially be slower than normal as most of the data and metadata still resides in the cloud. Performance might increase as data flows from the cloud to the StorSimple device. • The total time to download metadata depends on the allocated volume size. Metadata is automatically brought into the device in the background at the rate of 5 minutes per TB of allocated volume data. This rate might be affected by Internet bandwidth to the cloud. • Unlike tiered volumes, if there are locally pinned volumes, the volume data is also downloaded locally on the device. The restore operation is complete when all the volume data has been brought to the device. • The restore operations might be long and the total time to complete the restore will depend on the size of the provisioned local volume, your Internet bandwidth, and the existing data on the device. Backup operations on the locally pinned volume are allowed while the restore operation is in progress.
Thin-restore availability	Last failover	
Maximum client read/write throughput, when served from the SSD tier*	920/720 MB/sec with a single 10-gigabit Ethernet network interface	Up to two times with MPIO and two network interfaces.
Maximum client read/write throughput, when served from the HDD tier*	120/250 MB/sec	
Maximum client read/write throughput, when served from the cloud tier*	11/41 MB/sec	Read throughput depends on clients generating and maintaining sufficient I/O queue depth.

*Maximum throughput per I/O type was measured with 100 percent read and 100 percent write scenarios. Actual throughput might be lower and depends on I/O mix and network conditions.

Stream Analytics limits

Limit identifier	Limit	Comments
Maximum number of streaming units per subscription per region	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.

Limit identifier	Limit	Comments
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.

Virtual Machines limits

Virtual Machines limits

Resource	Limit
Virtual machines per cloud service [1]	50
Input endpoints per cloud service [2]	150

[1] Virtual machines created by using the classic deployment model instead of Azure Resource Manager are automatically stored in a cloud service. You can add more virtual machines to that cloud service for load balancing and availability.

[2] Input endpoints allow communications to a virtual machine from outside the virtual machine's cloud service. Virtual machines in the same cloud service or virtual network can automatically communicate with each other.

Virtual Machines limits - Azure Resource Manager

The following limits apply when you use Azure Resource Manager and Azure resource groups.

Resource	Limit
VMs per subscription	25,000[1] per region.
VM total cores per subscription	20[1] per region. Contact support to increase limit.
Azure Spot VM total cores per subscription	20[1] per region. Contact support to increase limit.
VM per series, such as Dv2 and F, cores per subscription	20[1] per region. Contact support to increase limit.
Availability sets per subscription	2,500 per region.
Virtual machines per availability set	200
Proximity placement groups per resource group	800
Certificates per availability set	199[2]
Certificates per subscription	Unlimited[3]

[1] Default limits vary by offer category type, such as Free Trial and Pay-As-You-Go, and by series, such as Dv2, F, and G. For example, the default for Enterprise Agreement subscriptions is 350. For security, subscriptions default to 20 cores to prevent large core deployments. If you need more cores, submit a support ticket.

[2] Properties such as SSH public keys are also pushed as certificates and count towards this limit. To bypass this limit, use the Azure Key Vault extension for Windows or the Azure Key Vault extension for Linux to install certificates.

[3] With Azure Resource Manager, certificates are stored in the Azure Key Vault. The number of certificates is unlimited for a subscription. There's a 1-MB limit of certificates per deployment, which consists of either a single VM or an availability set.

> **Note**
>
> Virtual machine cores have a regional total limit. They also have a limit for regional per-size series, such as Dv2 and F. These limits are separately enforced. For example, consider a subscription with a US East total VM core limit of 30, an A series core limit of 30, and a D series core limit of 30. This subscription can deploy 30 A1 VMs, or 30 D1 VMs, or a combination of the two not to exceed a total of 30 cores. An example of a combination is 10 A1 VMs and 20 D1 VMs.

Compute Gallery limits

There are limits, per subscription, for deploying resources using Compute Galleries:

- 100 compute galleries, per subscription, per region
- 1,000 image definitions, per subscription, per region
- 10,000 image versions, per subscription, per region

Virtual Machine Scale Sets limits

Resource	Limit
Maximum number of VMs in a scale set	1,000
Maximum number of VMs based on a custom VM image in a scale set	600
Maximum number of scale sets per subscription per region	2,500
Maximum number of nodes supported in VMSS for IB cluster	100

Dev tunnels limits

The following limits apply to dev tunnels. The limits reset monthly.

Resource	Limit
Bandwidth	2 GB per user
Tunnels	5 per user
Active connections	20 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

To request higher usage limits for dev tunnels, open an issue in our GitHub repo. In the issue, include which limit you'd like increased and why.

See also

- Understand Azure limits and increases
- Virtual machine and cloud service sizes for Azure
- Sizes for Azure Cloud Services
- Naming rules and restrictions for Azure resources

Routine (planned) maintenance for App Service

Article • 02/10/2023

Routine maintenance covers behind the scenes updates to the Azure App Service platform. Types of maintenance can be performance improvements, bug fixes, new features, or security updates. App Service maintenance can be on App Service itself or the underlying operating system.

> **Important**
>
> A breaking change or deprecation of functionality is not a part of routine maintenance (see **Modern Lifecycle Policy - Microsoft Lifecycle | Microsoft Learn** for deprecation topic for details).

Our service quality and uptime guarantees continue to apply during maintenance periods. Maintenance periods are mentioned to help customers to get visibility into platform changes.

What to expect

Like security updates on personal computers, mobile phones and other devices, even machines in the cloud need the latest updates. Unlike physical devices, cloud solutions like Azure App Service provide ways to overcome these routines with more ease. There's no need to "stop working" for a certain period 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 are made monthly, but can vary on the needs and other factors.

Since a typical cloud solution consists of multiple applications, databases, storage accounts, functions, and other resources, various parts of your solutions can be undergoing maintenance at different times. Some of this coordination is related to geography, region, data centers, and availability zones. It can also be due to the cloud where not everything is touched simultaneously.

Safe deployment practices - Azure DevOps | Microsoft Learn

In order from top to bottom we see:

- A descriptive title of the maintenance event
- Impacted regions and subscriptions
- Expected maintenance window

Frequently Asked Questions

Why is the maintenance taking so long?

The maintenance fundamentally represents delivering latest updates to the platform and service. It's difficult to predict when individual apps would be affected down to a specific time, so more generic notifications are sent out. The time ranges in those notifications don't reflect the experiences at the app level, but the overall operation across all resources. Apps which undergo maintenance instantly restart on freshly updated machines and continue working. There's no downtime when requests/traffic aren't served.

Why am I getting so many notifications?

A typical scenario is that customers have multiple applications, and they are upgraded at different times. To avoid sending notifications for each of them, a more generic notification is sent that captures multiple resources. The notification is sent at the

beginning and throughout the maintenance window. Due to the time window being longer, you can receive multiple reminders for the same rollout so you can easier correlate any restart/interruption/issue in case it is needed.

How is routine maintenance related to SLA?

Platform maintenance isn't expected to impact application uptime or availability. Applications continue to stay online while platform maintenance occurs. Platform maintenance may cause applications to be cold started on new virtual machines, which can lead to cold start delays. An application is still considered to be online, even while cold-starting. For best practices to minimize/avoid cold starts, consider using local cache for Windows apps as well as Health check. It's not expected that sites would incur any SLA violation during maintenance windows.

How does the upgrade work how does it ensure the smooth operation of my apps?

Azure App Service represents a fleet of scale units, which provide hosting of web applications/solutions to the customers. Each scale unit is further divided into smaller pieces and sliced into a concept of upgrade domains and availability zones. This is to optimize placements of bigger App Service Plans and smooth deployments since not all machines in each scale unit are updated at once. Fleet upgrades machines iteratively while monitoring the health of the fleet so any time there is an issue, the system can stop the rollout. This process is described in detail at Demystifying the magic behind App Service OS updates - Azure App Service.

Are business hours reflected?

Maintenance operations are optimized to start outside standard business hours (9-5pm) as statistically that is a better timing for any interruptions and restarts of workloads as there is a less stress on the system (in customer applications and transitively also on the platform itself). For App Service Plan and App Service Environment v2, maintenance can continue into business hours during longer maintenance events.

What are my options to control routine maintenance?

If you run your workloads in Isolated SKU via App Service Environment v3, you can also schedule the upgrades when needed. This is described with details at Control and automate planned maintenance for App Service Environment v3 - Azure App Service.

Can I prepare my apps better for restarts?

If your applications need extra time during restarts to come online (a typical pattern would be heavy dependency on external resources during application warm-up/start-up), consider using Health Check. You can use this to communicate with the platform that your application is not ready to receive requests yet and the system can use that information to route requests to other instances in your App Service Plan. For such case, it's recommended to have at least two instances in the plan.

My applications have been online, but since these notifications started showing up things are worse. 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 increases. However, there is an increased level of visibility into all changes which can cause the perception that more changes are being made.

Next steps

Control and automate planned maintenance for App Service Environment v3 - Azure App Service

Demystifying the magic behind App Service OS updates - Azure App Service

Routine Planned Maintenance Notifications for Azure App Service - Azure App Service

App Service language runtime support policy

Article • 09/11/2023

This document describes the App Service language runtime support policy for updating existing stacks and retiring process for upcoming end-of-life stacks. This policy is to clarify existing practices and doesn't represent a change to customer commitments.

Updates to existing stacks

App Service will update existing stacks after they become available from each community. App Service will update major versions of stacks but can't guarantee any specific minor or patch versions. Minor and patch versions are controlled by the platform, and it's not possible for App Service to pin a specific minor or patch version. For example, Node 18 will be updated by App Service but a specific Node 18.x.x version won't be guaranteed. 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. Once community support for a given language reaches end-of-life, your applications will continue to run unchanged. However, App Service can't provide security patches or related customer support for that runtime version past its end-of-life date. If your application has any issues past the end-of-life date for that version, you should move up to a supported version to receive the latest security patches and features.

> **Important**
>
> You're encouraged to upgrade the language version of your affected apps to a currently supported version. If you're running apps using an unsupported language version, you'll be required to upgrade before receiving support for your app.

Notifications

End-of-life dates for runtime versions are determined independently by their respective stacks and are outside the control of App Service. App Service will send reminder

notifications to subscription owners for upcoming end-of-life runtime versions when they become available for each language.

Those who receive notifications include account administrators, service administrators, and co-administrators. Contributors, readers, or other roles won't directly receive notifications, unless they opt-in to receive notification emails, using Service Health Alerts.

Language runtime version support timelines

To learn more about specific language support policy timelines, visit 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 your App Service application language versions, see the following resources:

- .NET
- Node
- Java
- Python
- PHP

Operating system functionality on Azure App Service

Article • 06/19/2023

This article describes the common baseline operating system functionality that is available to all Windows apps running on Azure App Service. This functionality includes file, network, and registry access, and diagnostics logs and events.

> **Note**
>
> **Linux apps** in App Service run in their own containers. You have root access to the container but no access to the host operating system is allowed. Likewise, for **apps running in Windows containers**, you have administrative access to the container but no access to the host operating system.

App Service plan tiers

App Service runs customer apps in a multi-tenant hosting environment. Apps deployed in the **Free** and **Shared** tiers run in worker processes on shared virtual machines, while apps deployed in the **Standard** and **Premium** tiers run on virtual machine(s) dedicated specifically for the apps associated with a single customer.

> **Note**
>
> App Service Free and Shared (preview) 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 to be used only for development and testing purposes.

Because App Service supports a seamless scaling experience between different tiers, the security configuration enforced for App Service apps remains the same. This ensures that apps don't suddenly behave differently, failing in unexpected ways, when 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 a variety of development frameworks, including ASP.NET, classic ASP, Node.js, PHP, and Python. In order to simplify and normalize security configuration, App Service apps typically run the various development frameworks with their default settings. The frameworks and runtime components provided by the platform are updated regularly to satisfy security and compliance requirements, for this reason we don't guarantee specific minor/patch versions and recommend customers target major version 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 PaaS (platform as a service) infrastructure. As a result, the local drives that are "attached" to a virtual machine are the same drive types available to any worker role running in Azure. This includes:

- An operating system drive (`%SystemDrive%`), whose size varies depending on the size of the VM.
- A resource drive (`%ResourceDrive%`) used by App Service 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:\` will continue to work because the App Service platform automatically remaps `d:\` to instead point at `c:\`. As noted above, it's highly recommended to 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. If the disk quota is reached, it can have adverse effects to your application. For example:

- The app may throw an error indicating not enough space on the disk.

- You may see disk errors when browsing to the Kudu console.
- Deployment from Azure DevOps or Visual Studio may fail with `ERROR_NOT_ENOUGH_DISK_SPACE: Web deployment task failed. (Web Deploy detected insufficient space on disk)`.
- Your app may suffer slow performance.

Network drives (UNC shares)

One of the unique aspects of App Service that makes 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, there is a number of UNC shares created in each data center. A percentage of the user content for all customers in each data center is allocated to each UNC share. Each customer's subscription has a reserved directory structure on a specific UNC share within a data center. A customer may have multiple apps created within a specific data center, so all of the directories belonging to a single customer subscription are created on the same UNC share.

Due to how Azure services work, the specific virtual machine responsible for hosting a UNC share will change over time. It is guaranteed that UNC shares will be mounted by different virtual machines as they'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. This faux absolute path provides a portable, app-and-user-agnostic method for referring to one's own app. By using `%HOME%\site`, one can transfer shared files from app to app without having to configure a new absolute path for each transfer.

Types of file access granted to an app

The `%HOME%` directory in an app maps to a content share in Azure Storage dedicated for that app, and its size is defined by your pricing tier. It may include directories such as those for content, error and diagnostic logs, and earlier versions of the app created by source control. 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/write access to its own temporary local storage, that storage really isn't intended to be used directly by the application code. Rather, the intent is to provide temporary file storage for IIS and web application frameworks. App Service also limits the amount of 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. See the following table for other tiers:

SKU	Local file storage
B1/S1/P1	11GB
B2/S2/P2	15GB
B3/S3/P3	58GB
P0v3	11GB
P1v2/P1v3/P1mv3/Isolated1/Isolated1v2	21GB
P2v2/P2v3/P2mv3/Isolated2/Isolated2v2	61GB
P3v2/P3v3/P3mv3/Isolated3/Isolated3v2	140GB
Isolated4v2	276GB
P4mv3	280GB
Isolated5v2	552GB
P5mv3	560GB
Isolated6v2	1104GB

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 (as well as others) are remapped in App Service to per-app temporary local storage. This remapping ensures that functionality continues as expected.

Each app in App Service runs as a random unique low-privileged worker process identity called the "application pool identity", described further in the IIS Application Pool Identities documentation. Application code uses this identity for basic read-only access to the operating system drive. This means application code can list common directory structures and read common files on operating system drive. Although this might

appear to be a somewhat broad level of access, the same directories and files are accessible when you provision a worker role in an Azure hosted service and read the drive contents.

File access across multiple instances

The 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. So, 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, neither is it 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 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 exists primarily to enable apps that listen on local loopback sockets as part of their functionality. Each app sees a "private" loopback connection. App "A" cannot listen to a local loopback socket established by app "B".

Named pipes are also supported as an inter-process communication (IPC) mechanism between different processes that collectively run an app. For example, the IIS FastCGI module relies on named pipes to coordinate the individual processes that run PHP pages.

Code execution, processes, and memory

As noted earlier, apps run inside of low-privileged worker processes using a random application pool identity. Application code has access to the memory space associated with the worker process, as well as any child processes that may be spawned by CGI processes or other applications. However, one app cannot access the memory or data of another app even if it'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 on App Service run in "full" trust as opposed to a more restricted trust mode. Web frameworks, including both classic ASP and ASP.NET, can call in-process COM components (but not out of process COM components) like ADO (ActiveX Data Objects) that are registered by default on the Windows operating system.

Apps can spawn and run arbitrary code. It's allowable for an app to do things like spawn a command shell or run a PowerShell script. However, even though arbitrary code and processes can be spawned from an app, executable programs and scripts are still restricted to the privileges granted to the parent application pool. For example, an app can spawn an executable that makes an outbound HTTP call, but that same executable cannot attempt to unbind the IP address of a virtual machine from its NIC. Making an outbound network call is allowed to low-privileged code, but attempting to reconfigure network settings on a virtual machine requires administrative privileges.

Diagnostics logs and events

Log information is another set of data that some apps attempt to access. The types of log information available to code running in App Service includes diagnostic and log information generated by an app that is also easily accessible to the app.

For example, W3C HTTP logs generated by an active app are available either on a log directory in the network share location created for the app, or available in blob storage if a customer has set up W3C logging to storage. The latter option enables large quantities of logs to be gathered without the risk of exceeding the file storage limits associated with a network share.

In a similar vein, real-time diagnostics information from .NET apps can also be logged using the .NET tracing and diagnostics infrastructure, with options to write the trace information to either the app's network share, or alternatively to a blob storage location.

Areas of diagnostics logging and tracing that aren't available to apps are Windows ETW events and common Windows event logs (for example, System, Application, and Security event logs). Since ETW trace information can potentially be viewable machine-wide (with the right ACLs), read and write access to ETW events are blocked. Developers might notice that API calls to read and write ETW events and common Windows event logs appear to work, but that is because App Service is "faking" the calls so that they appear to succeed. In reality, the application code has no access to this event data.

Registry access

Apps have read-only access to much (though not all) of the registry of the virtual machine they're running on. In practice, this means registry keys that allow read-only access to the local Users group are accessible by apps. One area of the registry that is currently not supported for either read or write access is the HKEY_CURRENT_USER hive.

Write-access to the registry is blocked, including access to any per-user registry keys. From the app's perspective, write access to the registry should never be relied upon in the Azure environment since apps can (and do) get migrated across different virtual machines. The only persistent writeable storage that can be depended on by an app is the per-app content directory structure stored on the App Service UNC shares.

Remote desktop access

App Service doesn't provide remote desktop access to the VM instances.

More information

Azure App Service sandbox - The most up-to-date information about the execution environment of App Service. This page is maintained directly by the App Service development team.

Kudu service overview

Article • 05/13/2022

Kudu is the engine behind a number of features in Azure App Service 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:

- App not in Isolated tier: `https://<app-name>.scm.azurewebsites.net`
- Internet-facing app in Isolated tier (App Service Environment): `https://<app-name>.scm.<ase-name>.p.azurewebsites.net`
- Internal app in Isolated tier (ILB App Service Environment): `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 other features, such as:

- 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`.
- Generates custom deployment scripts.
- Allows access with REST API.

RBAC permissions required to access Kudu

To access Kudu in the browser with Azure Active Directory authentication, you need to be a member of a built-in or custom role.

- If using a built-in role, you must be a member of Website Contributor, Contributor, or Owner.
- If using a custom role, you need the resource provider operation: `Microsoft.Web/sites/publish/Action`.

More Resources

Kudu is an open source project, and has its documentation at Kudu Wiki.

App Service, Functions, and Logic Apps on Azure Arc (Preview)

Article • 03/15/2023

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.

Limitation	Details
Supported Azure regions	East US, West Europe
Cluster networking requirement	Must support `LoadBalancer` service type
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

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?

- 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.

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 Stack HCI, 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
```

Next steps

Create an App Service Kubernetes environment (Preview)

Set up an Azure Arc-enabled Kubernetes cluster to run App Service, Functions, and Logic Apps (Preview)

Article • 03/24/2023

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.

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 below 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

   ```bash
   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

   ```bash
   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
```

2. 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

1. 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 will be 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

```bash
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:

Parameter	Description
`Microsoft.CustomLocation.ServiceAccount`	The service account that should be created for the custom location that will be created. It is recommended that this be set to the value `default`.
`appsNamespace`	The namespace to provision the app definitions and pods. **Must** match that of the extension release namespace.
`clusterName`	The name of the App Service Kubernetes environment that will be created against this extension.
`keda.enabled`	Whether KEDA should be installed on the Kubernetes cluster. Accepts `true` or `false`.
`buildService.storageClassName`	The name of the storage class for the build service to store build artifacts. A value like `default` specifies a class named `default`, and not any class that is marked as default. Default is a valid storage class for AKS and AKS HCI but it may not be for other distrubtions/platforms.
`buildService.storageAccessMode`	The access mode to use with the named storage class above. Accepts `ReadWriteOnce` or `ReadWriteMany`.
`customConfigMap`	The name of the config map that will be set by the App Service Kubernetes environment. Currently, it must be `<namespace>/kube-environment-config`, replacing `<namespace>` with the value of `appsNamespace` above.

Parameter	Description
`envoy.annotations.service.beta.kubernetes.io/azure-load-balancer-resource-group`	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.
`logProcessor.appLogs.destination`	Optional. Accepts `log-analytics` or `none`, choosing none disables platform logs.
`logProcessor.appLogs.logAnalyticsConfig.customerId`	Required only when `logProcessor.appLogs.destination` is set to `log-analytics`. The base64-encoded Log analytics workspace ID. This parameter should be configured as a protected setting.
`logProcessor.appLogs.logAnalyticsConfig.sharedKey`	Required only when `logProcessor.appLogs.destination` is set to `log-analytics`. 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

```bash
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 that have been 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 \
       --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 an Azure Active Directory 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

Create an App Service app on Azure Arc (Preview)

Article • 02/16/2023

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.

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 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 `-`). To see all supported runtimes, run az webapp list-runtimes --os linux.

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/node:14-lts
```

To update the image after the app is create, 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 Linux Ruby app
- Configure a custom container

Run background tasks with WebJobs in Azure App Service

Article • 07/30/2023

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, API app, or mobile app. There's no extra cost to use WebJobs.

You can use the Azure WebJobs SDK with WebJobs to simplify many programming tasks. WebJobs aren't supported for App Service on Linux yet. For more information, see What is the WebJobs SDK.

Azure Functions provides another way to run programs and scripts. For a comparison between WebJobs and Functions, see Choose between Flow, Logic Apps, Functions, and WebJobs.

WebJob types

The following table describes the differences between *continuous* and *triggered* WebJobs.

Continuous	Triggered
Starts immediately when the WebJob is created. To keep the job from ending, the program or script typically does its work inside an endless loop. If the job does end, you can restart it. 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.

Continuous	Triggered
Code is deployed under `\site\wwwroot\app_data\Jobs\Continuous`.	Code is deployed under `\site\wwwroot\app_data\Jobs\Triggered`.

> **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.

Supported file types for scripts or programs

The following file types are supported:

- .cmd, .bat, .exe (using Windows cmd)
- .ps1 (using PowerShell)
- .sh (using Bash)
- .php (using PHP)
- .py (using Python)
- .js (using Node.js)
- .jar (using Java)

Create a continuous WebJob

> **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**.

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 "_".

Setting	Sample value	Description
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	Continuous	The WebJob types are described earlier in this article.
Scale	Multi Instance	Available only for Continuous WebJobs. Determines whether the program or script runs on all instances or just one instance. The option to run on multiple instances doesn't apply to the Free or Shared pricing tiers.

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.

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**.

3. Fill in the **Add WebJob** settings as specified in the table, then select **Create Webjob**.

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.

Setting	Sample value	Description
		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.

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**.

3. Fill in the **Add WebJob** settings as specified in the table, then select **Create Webjob**.

Setting	Sample value	Description
Name	mySchedulerWebJob	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

Setting	Sample value	Description
		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.

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. Just 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**.

2. In the **WebJob Details** page, select a time to see details for one run.

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

Below is a list of common WebJob statuses:

- **Initializing** The app has just 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 a number 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.

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.

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.

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
`is_in_place`	All	Allows the WebJob to run in place without first being copied to a temporary folder. For more information, see WebJob working directory.
`is_singleton`	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
`schedule`	Triggered	Run the WebJob on a CRON-based schedule. For more information, see NCRONTAB expressions.
`stopping_wait_time`	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 • 06/01/2022

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 Core 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 .NET Core console app that runs on .NET 6.0.

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>
   ```

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:

```
C#

using System.Threading.Tasks;
using Microsoft.Extensions.Hosting;
```

2. Also under **Program.cs**, add the following code:

```
C#

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 `<6_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
   <6_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:

   C#

   ```
   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.

   C#

   ```
   builder.ConfigureLogging((context, b) =>
   {
       b.AddConsole();
   });
   ```

   The `Main` method now looks like this:

   C#

   ```
   static async Task Main()
   {
       var builder = new HostBuilder();
       builder.ConfigureWebJobs(b =>
               {
                   b.AddAzureStorageCoreServices();
               });
       builder.ConfigureLogging((context, b) =>
               {
                   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.

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. 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:

```
C#

builder.ConfigureWebJobs(b =>
{
    b.AddAzureStorageCoreServices();
    b.AddAzureStorageQueues();
});
```

5. 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.ConfigureLogging((context, b) =>
    {
```

```
        b.AddConsole();
    });
    builder.ConfigureWebJobs(b =>
    {
        b.AddAzureStorageCoreServices();
        b.AddAzureStorageQueues();
    });
    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:

```C#
using Microsoft.Azure.WebJobs;
using Microsoft.Extensions.Logging;

namespace WebJobsSDKSample
{
    public class Functions
    {
        public static void ProcessQueueMessage([QueueTrigger("queue")]
string message, ILogger logger)
        {
            logger.LogInformation(message);
        }
    }
}
```

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.

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.

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.

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.

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 **Settings**, choose **Application Insights**, and select **Turn on Application Insights**.

3. Verify the generated **Resource name** for the instance and the **Location**, and select **Apply**.

4. Under **Settings**, choose **Configuration** 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.AddAzureStorage();
            });
    builder.ConfigureLogging((context, b) =>
            {
                b.AddConsole();

                // If the key exists in settings, use it to enable
Application Insights.
                string instrumentationKey =
context.Configuration["APPINSIGHTS_INSTRUMENTATIONKEY"];
                if (!string.IsNullOrEmpty(instrumentationKey))
                {
                    b.AddApplicationInsightsWebJobs(o =>
o.InstrumentationKey = instrumentationKey);
                }
```

```
            });
    var host = builder.Build();
    using (host)
    {
        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.

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**, choose **Application Insights**, and select **View Application Insights data**.
5. Select **Search** and then select **See all data in the last 24 hours**.

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`:

```
C#
```

```
using System.IO;
```

2. Replace the `ProcessQueueMessage` method with the following code:

```C#
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:

```C#
builder.ConfigureWebJobs(b =>
{
    b.AddAzureStorageCoreServices();
    b.AddAzureStorageQueues();
    b.AddAzureStorageBlobs();
});
```

4. Create a blob container in your storage account.

a. In the Azure portal, navigate to the **Containers** tab below **Data storage** and select **+ Container**

b. 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.

   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.

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

How to use the Azure WebJobs SDK for event-driven background processing

Article • 10/12/2022

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 connection strings

The WebJobs SDK looks for Azure Storage and Azure Service Bus connection strings in the local.settings.json file when you run locally or in the environment of the WebJob when you run in Azure. By default, the WebJobs SDK requires a storage connection string setting with the name `AzureWebJobsStorage`.

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:

C#

```
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();
}
```

> **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:

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:

C#

```
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`.

C#

```
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:

```C#
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}`).

> **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:

C#

```
[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*

C#

```
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*

```
C#

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:

C#

```
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:

C#

```
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:

C#

```
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.

C#

```
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:

C#

```
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:

C#

```
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:

C#

```
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:

C#

```
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*

C#

```
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*

```
C#

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:

```
C#

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:

```C#
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*

```C#
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*

```
C#

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.

```
C#

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:

C#

```
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:

C#

```
using System.Configuration;
```

Your `NameResolver` class gets the queue name from app settings, as shown here:

C#

```
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:

```
C#

using Microsoft.Extensions.DependencyInjection;
```

You add the resolver by calling the ConfigureServices extension method on HostBuilder, as in this example:

```
C#

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:

```
C#

 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.

C#

```
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*.

C#

```
[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.

C#

```
[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:

C#

```
[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:

LogLevel	Code
Trace	0
Debug	1
Information	2

LogLevel	Code
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:

C#

```
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.

C#

```
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("MySpecificFunctionNa
me"),
                    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:

C#

```
using Microsoft.ApplicationInsights.Extensibility;
using Microsoft.ApplicationInsights.Channel;
```

The following custom implementation of ITelemetryInitializer lets you add your own ITelemetry to the default TelemetryConfiguration.

C#

```
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.

C#

```
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 .

Best Practices for Azure App Service

Article • 04/15/2022

This article summarizes best practices for using Azure App Service.

Colocation

When Azure resources composing a solution such as a web app and a database are located in different regions, it can have the following effects:

- Increased latency in communication between resources
- Monetary charges for outbound data transfer cross-region as noted on the Azure pricing page.

Colocation in the same region is best for Azure resources composing a solution such as a web app and a database or storage account used to hold content or data. When creating resources, make sure they are in the same Azure region unless you have specific business or design reason for them not to be. You can move an App Service app to the same region as your database by using the App Service cloning feature currently available for Premium App Service Plan apps.

Certificate Pinning

Applications should never have a hard dependency or pin to the default *.azurewebsites.net TLS certificate because the *.azurewebsites.net TLS certificate could be rotated anytime given the nature of App Service as a Platform as a Service (PaaS). Certificate pinning is a practice where an application only allows a specific list of acceptable Certificate Authorities (CAs), public keys, thumbprints, or any part of the certificate hierarchy. In the event that the service rotates the App Service 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 *.azurewebsites.net TLS certificate is rotated is also not guaranteed since the rotation frequency can change at any time.

Note that applications which rely on certificate pinning should also not 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 is best practice to provide a custom TLS certificate for applications that rely on certificate pinning.

If an application needs to rely on certificate pinning behavior, it is recommended to add a custom domain to a web app and provide a custom TLS certificate for the domain which can then be relied on for certificate pinning.

When apps consume more memory than expected

When you notice an app consumes more memory than expected as indicated via monitoring or service recommendations, consider the App Service Auto-Healing feature. One of the options for the Auto-Healing feature is taking custom actions based on a memory threshold. Actions span the spectrum from email notifications to investigation via memory dump to on-the-spot mitigation by recycling the worker process. Auto-healing can be configured via web.config and via a friendly user interface as described at in this blog post for the App Service Support Site Extension.

When apps consume more CPU than expected

When you notice an app consumes more CPU than expected or experiences repeated CPU spikes as indicated via monitoring or service recommendations, consider scaling up or scaling out the App Service plan. If your application is stateful, scaling up is the only option, while if your application is stateless, scaling out gives you more flexibility and higher scale potential.

For more information about App Service scaling and autoscaling options, see Scale a Web App in Azure App Service.

When socket resources are exhausted

A common reason for exhausting outbound TCP connections is the use of client libraries, which are not implemented to reuse TCP connections, or when a higher-level protocol such as HTTP - Keep-Alive is not used. Review the documentation for each of the libraries referenced by the apps in your App Service Plan to ensure they are configured or accessed in your code for efficient reuse of outbound connections. Also follow the library documentation guidance for proper creation and release or cleanup to avoid leaking connections. While such client libraries investigations are in progress, impact may be mitigated by scaling out to multiple instances.

Node.js and outgoing http requests

When working with Node.js and many outgoing http requests, dealing with HTTP - Keep-Alive is important. You can use the agentkeepalive `npm` package to make it easier in your code.

Always handle the `http` response, even if you do nothing in the handler. If you don't handle the response properly, your application gets stuck eventually because no more sockets are available.

For example, when working with the `http` or `https` package:

JavaScript

```javascript
const request = https.request(options, function(response) {
    response.on('data', function() { /* do nothing */ });
});
```

If you are running on App Service on Linux on a machine with multiple cores, another best practice is to use PM2 to start multiple Node.js processes to execute your application. You can do it by specifying a startup command to your container.

For example, to start four instances:

```
pm2 start /home/site/wwwroot/app.js --no-daemon -i 4
```

When your app backup starts failing

The two most common reasons why app backup fails are: invalid storage settings and invalid database configuration. These failures typically happen when there are changes to storage or database resources, or changes for how to access these resources (for example, credentials updated for the database selected in the backup settings). Backups typically run on a schedule and require access to storage (for outputting the backed-up files) and databases (for copying and reading contents to be included in the backup). The result of failing to access either of these resources would be consistent backup failure.

When backup failures happen, review most recent results to understand which type of failure is happening. For storage access failures, review and update the storage settings used in the backup configuration. For database access failures, review and update your connections strings as part of app settings; then proceed to update your backup

configuration to properly include the required databases. For more information on app backups, see Back up a web app in Azure App Service.

When new Node.js apps are deployed to Azure App Service

Azure App Service default configuration for Node.js apps is intended to best suit the needs of most common apps. If configuration for your Node.js app would benefit from personalized tuning to improve performance or optimize resource usage for CPU/memory/network resources, see Best practices and troubleshooting guide for Node applications on Azure App Service. This article describes the iisnode settings you may need to configure for your Node.js app, describes the various scenarios or issues that your app may be facing, and shows how to address these issues.

When Internet of Things (IoT) devices are connected to apps on App Service

There are a few scenarios where you can improve your environment when running Internet of Things (IoT) devices that are connected to App Service. One very 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 nor to an App Service Managed Certificate. If your system needs to rely on certificate pinning behavior, it is recommended to add a custom domain to a web app and provide a custom TLS certificate for the domain which can then be relied on for certificate pinning. You can refer to the certificate pinning section of this article for more information.

To increase resiliency in your environment, you should not rely on a single endpoint for all your devices. You should at least host your web apps in two different regions to avoid a single point of failure and be ready to failover traffic. On App Service, you can add identical custom domain to different web apps as long as these web apps are hosted in different regions. This ensures that if you need to pin certificates, you can also pin on the custom TLS certificate that you provided. Another option would be to use a load balancer in front of the web apps, such as Azure Front Door or Traffic Manager, to ensure high availability for your web apps. You can refer to Quickstart: Create a Front Door for a highly available global web application or Controlling Azure App Service traffic with Azure Traffic Manager for more information.

Next Steps

For more information on best practices, visit App Service Diagnostics to find out actionable best practices specific to your resource.

- Navigate to your Web App in the Azure portal.
- Click on **Diagnose and solve problems** in the left navigation, which opens App Service Diagnostics.
- Choose **Best Practices** homepage tile.
- Click **Best Practices for Availability & Performance** or **Best Practices for Optimal Configuration** to view the current state of your app in regards to these best practices.

You can also use this link to directly open App Service Diagnostics for your resource: `https://portal.azure.com/?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 • 06/29/2023

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.

Ready for a migration assessment? Select one of the following options to get started.

- Self-service assessment
- Partner assessment

Want to learn more?

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

More resources to migrate .NET apps to the cloud
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

How-tos
Discover web apps and SQL Server instances
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 • 07/14/2023

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:

Requirement	Details
Identify a machine on which to install the tool	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.) 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. Ensure that 6 GB is available on the Windows machine running the Azure Migrate App Containerization tool. This space is for storing application artifacts. The Windows machine should have internet access, directly or via a proxy. 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.
Application servers	Enable PowerShell remoting on the application servers: sign in to the application server and follow these instructions to turn on PowerShell remoting. Ensure that PowerShell 5.1 is installed on the application server. Follow the instructions here to download and install PowerShell 5.1 on the application server. 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.
ASP.NET application	The tool currently supports: • 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. The tool currently doesn't support: • Applications that require Windows authentication. (AKS doesn't currently support gMSA.)

Requirement	Details
	• 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 Azure Active Directory 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:

1. In the Azure portal, search for "subscriptions." Under **Services**, select **Subscriptions**:

2. On the **Subscriptions** page, select the subscription in which you want to create an Azure Migrate project.

3. In the subscription, on the left pane, select **Access control (IAM)**.

4. On the **Check access** tab, search for the relevant user account.

5. Under **Add a role assignment**, select **Add**:

6. On the **Add role assignment** page, select the **Owner** role, and then select the account (**azmigrateuser** in our example). Then select **Save**.

Your Azure account also needs permissions to register Azure Active Directory apps.

7. In the Azure portal, go to **Azure Active Directory** > **Users** > **User Settings**.

8. In **User settings**, verify that Azure AD users can register applications. (This option is set to **Yes** by default.)

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 Azure Active Directory 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
```

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**:

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**:

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:

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**:

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**:

> **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:

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

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 Spring Boot applications to Azure App Service

Article • 05/30/2023

This guide describes what you should be aware of when you want to migrate an existing Spring Boot application to Azure App Service.

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 executable JAR applications to containers on Azure Kubernetes Service (guidance planned)
- Migrate executable JAR Applications to Azure Virtual Machines (guidance planned)

Switch to a supported platform

App Service offers specific versions of Java SE. To ensure compatibility, migrate your application to one of the supported versions of 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

Inventory external resources

Identify external resources, such as data sources, JMS message brokers, and URLs of other services. In Spring Boot applications, you can typically find the configuration for such resources in the *src/main/directory* folder, in a file typically called *application.properties* or *application.yml*. Additionally, check the production deployment's environment variables for any pertinent configuration settings.

Databases

For any SQL database, identify the connection string.

For a Spring Boot application, connection strings typically appear in configuration files.

Here's an example from an *application.properties* file:

properties

```properties
spring.datasource.url=jdbc:mysql://localhost:3306/mysql_db
spring.datasource.username=dbuser
spring.datasource.driver-class-name=com.mysql.jdbc.Driver
```

Here's an example from an *application.yaml* file:

YAML

```yaml
spring:
  data:
    mongodb:
      uri: mongodb://mongouser:deepsecret@mongoserver.contoso.com:27017
```

See Spring Data documentation for more possible configuration scenarios:

- JPA Repositories
- JDBC Repositories
- Cassandra Repositories
- MongoDB Repositories

JMS message brokers

Identify the broker or brokers in use by looking in the build manifest (typically, a *pom.xml* or *build.gradle* file) for the relevant dependencies.

For example, a Spring Boot application using ActiveMQ would typically contain this dependency in its *pom.xml* file:

```XML
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-activemq</artifactId>
</dependency>
```

Spring Boot applications using commercial brokers typically contain dependencies directly on the brokers' JMS driver libraries. Here's an example from a *build.gradle* file:

```JSON
    dependencies {
      ...
      compile("com.ibm.mq:com.ibm.mq.allclient:9.0.4.0")
      ...
    }
```

After you've identified the broker or brokers in use, find the corresponding settings. In Spring Boot applications, you can typically find them in the *application.properties* and *application.yml* files in the application directory.

Here's an ActiveMQ example from an *application.properties* file:

```properties
spring.activemq.brokerurl=broker:
(tcp://localhost:61616,network:static:tcp://remotehost:61616)?
persistent=false&useJmx=true
spring.activemq.user=admin
spring.activemq.password=tryandguess
```

For more information on ActiveMQ configuration, see the Spring Boot messaging documentation.

Here's an IBM MQ example from an *application.yaml* file:

```YAML
```

```yaml
ibm:
  mq:
    queueManager: qm1
    channel: dev.ORDERS
    connName: localhost(14)
    user: admin
    password: big$ecr3t
```

For more information on IBM MQ configuration, see the IBM MQ Spring components documentation.

Identify external caches

Identify any external caches in use. Frequently, Redis is used via Spring Data Redis. For configuration information, see the Spring Data Redis documentation.

Determine whether session data is being cached via Spring Session by searching for the respective configuration (in Java or XML).

Identity providers

Identify any identity provider(s) used by your application. For information on how identity providers may be configured, consult the following:

- For OAuth2 configuration, see the Spring Security reference.
- For Auth0 Spring Security configuration, see the Auth0 Spring Security documentation.
- For PingFederate Spring Security configuration, see the Auth0 PingFederate instructions.

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 every external dependency of your application can be satisfied after an App Service migration.

Inventory secrets

Passwords and secure strings

Check all properties and configuration files and all environment variables on the production deployment(s) for any secret strings and passwords. In a Spring Boot application, such strings will likely be found in *application.properties* or *application.yml*.

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. You can also directly deploy the static content to an app in the Azure Spring Apps Enterprise plan. For more information, see Deploy web static files.

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. You can also directly deploy the static content to an app in the Azure Spring Apps Enterprise plan. For more information, see Deploy web static files.

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 process.

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.

Identify all outside processes/daemons running on the production server(s)

Processes running outside of Application Server, such as monitoring daemons, will need to be migrated elsewhere or eliminated.

Identify handling of non-HTTP requests or multiple ports

App Service supports only a single HTTP endpoint on a single port. If your application listens on multiple ports or accepts requests using protocols other than HTTP, don't use Azure App Service.

Migration

Parameterize the configuration

Ensure that all external resource coordinates (such as database connection strings) and other customizable settings can be read from environment variables. If you're migrating a Spring Boot Application, all configuration settings should already be externalizable.

For more information, see Externalized Configuration in the Spring Boot documentation.

Here's an example that references a `SERVICEBUS_CONNECTION_STRING` environment variable from an *application.properties* file:

```properties
spring.jms.servicebus.connection-string=${SERVICEBUS_CONNECTION_STRING}
spring.jms.servicebus.topic-client-id=contoso1
spring.jms.servicebus.idle-timeout=10000
```

Provision an App Service plan

From the list of available service plans, 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.

Create the App Service plan.

Create and Deploy Web App(s)

You'll need to create a Web App on your App Service Plan (choosing "Java SE" as the runtime stack) for every executable JAR file you intend to run.

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 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

Once the Web App has been created, use one of the available deployment mechanisms to deploy your application. If possible, your application should be uploaded to */home/site/wwwroot/app.jar*. 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. Then paste the full 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`).

Migrate JVM runtime options

If your application requires specific runtime options, use the most appropriate mechanism to specify them.

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 external resource coordinates and other settings

Follow these steps to migrate connection strings and other settings.

> **Note**

For any Spring Boot application settings parameterized with variables in the **Parameterize the configuration** section, those environment variables must be defined in the application configuration. Any Spring Boot application settings not explicitly parameterized with environment variables can still be overridden by them via Application Configuration. For example:

properties

```
spring.jms.servicebus.connection-string=${CUSTOMCONNSTR_SERVICE_BUS}
spring.jms.servicebus.topic-client-id=contoso1
spring.jms.servicebus.idle-timeout=1800000
```

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.

Migrate and enable the identity provider

If your application requires authentication or authorization, ensure they're configured to access the identity provider by using the following guidance:

- If the identity provider is Azure Active Directory, no changes should be necessary.
- If the identity provider is an on-premises Active Directory forest, consider implementing a hybrid identity solution with Azure Active Directory. For more information, see the Hybrid identity documentation.
- If the identity provider is another on-premises solution, such as PingFederate, consult the Custom installation of Azure AD Connect topic to configure federation with Azure Active Directory. Alternatively, consider using Spring Security to use your identity provider through OAuth2/OpenID Connect or SAML.

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 Azure Key Vault and/or parameter injection with application settings where possible.
- 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.

Migrate Tomcat applications to Tomcat on Azure App Service

Article • 04/27/2023

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. You can also directly deploy the static content to an app in the Azure Spring Apps Enterprise plan. For more information, see Deploy web static files.

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. You can also directly deploy the static content to an app in the Azure Spring Apps Enterprise plan. For more information, see Deploy web static files.

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 Serve content from Azure Storage in App Service on Linux.

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'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 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.

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="t00secure2gue$$"
/>
```

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.digester.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 JASPIC)

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.
- 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.

Migrate JBoss EAP applications to JBoss EAP on Azure App Service

Article • 01/16/2023

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. You can also directly deploy the static content to an app in the Azure Spring Apps Enterprise plan. For more information, see Deploy web static files.

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. You can also directly deploy the static content to an app in the Azure Spring Apps Enterprise plan. For more information, see Deploy web static files.

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 Choose a solution for connecting 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 JMS with Azure Service Bus 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.
- **Transactions**: The application instances are run in a stateless manner, so the Transactions API isn't currently supported. 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 March 15th, 2022, clustered deployment of JBoss EAP is supported in Public Preview. 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 in JBoss EAP 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**.

Azure CLI

```
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.

```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.PGXADataSource)

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
SWORD: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.PostgreSQLExceptio
nSorter --jta=true --use-java-context=true --valid-connection-checker-
class-
name=org.jboss.jca.adapters.jdbc.extensions.postgres.PostgreSQLValidCon
nectionChecker
```

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.

Migrate WebLogic Server applications to JBoss EAP on Azure App Service

Article • 05/30/2023

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 Choose a solution for connecting 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 JMS with Azure Service Bus 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 our 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, seeDeploy 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 JDNI 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 Deploy to a slot section of Deploy an Azure Web App.
- 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.

Migrate WebSphere applications to JBoss EAP on Azure App Service

Article • 05/30/2023

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. You can also directly deploy the static content to an app in the Azure Spring Apps Enterprise plan. For more information, see Deploy web static files.

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. You can also directly deploy the static content to an app in the Azure Spring Apps Enterprise plan. For more information, see Deploy web static files.

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 Serve content from Azure Storage in App Service on Linux.

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 Choose a solution for connecting 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 JMS with Azure Service Bus 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'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 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, seeDeploy 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.

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 JDNI resources. For more information, see the JBoss EAP section of Configure a Java app for 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 Deploy to a slot section of Deploy an Azure Web App.
- 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.

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

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

Basic web application

Azure App Service | Azure Key Vault | Azure Monitor | Azure SQL Database

This architecture shows the fundamental components of a basic web application. You can use the architecture to build a web application and then customize the application to your needs.

Architecture

Download a Visio file of this architecture.

Components

- Azure App Service is a fully managed platform for creating and deploying cloud applications. It lets you define a set of compute resources for a web app to run, deploy web apps, and configure deployment slots.
- Deployment slots lets you stage a deployment and then swap it with the production deployment. That way, you avoid deploying directly into production. See the release engineering and deployment section below for specific recommendations.
- IP address: The App Service app has a public IP address and a domain name. The domain name is a subdomain of `azurewebsites.net`, such as `contoso.azurewebsites.net`.

- Azure DNS is a hosting service for DNS domains, providing name resolution using Microsoft Azure infrastructure. By hosting your domains in Azure, you can manage your DNS records using the same credentials, APIs, tools, and billing as your other Azure services. To use a custom domain name (such as `contoso.com`), create DNS records that map the custom domain name to the IP address. For more information, see Configure a custom domain name in Azure App Service.
- Azure SQL Database is a relational database-as-a-service in the cloud. SQL Database shares its code base with the Microsoft SQL Server database engine. Depending on your application requirements, you can also use Azure Database for MySQL or Azure Database for PostgreSQL. These alternatives are fully managed database services based on the open-source MySQL Server and Postgres database engines.
- Azure Active Directory is a cloud-based identity and access management service that lets employees access cloud apps developed for your organization.
- Azure Monitor is a solution for collecting, analyzing, and acting on logs and metrics across your environments.
- Azure Key Vault supports secrets management, key management, and certificate management. It can store application secrets like database connection strings.

Recommendations

Your requirements might differ from the architecture described and given in the code. The code deploys with production configurations. Use the recommendations to customize your deployment to meet your needs.

App Service plan

The App Service plan has different pricing tiers. Each pricing tier supports several *instance sizes* that differ by the number of cores and memory. You can change the pricing tier after deployment by selecting "Scale up (App Service Plan)" on the left navigation. Here are some App Service recommendations:

- Run your production workload on the *Basic*, *Standard*, and *Premium* pricing tiers. In these three tiers, the app runs on dedicated virtual machine instances and has allocated resources that can scale out.
- Use the *Standard* and *Premier* tiers if you need autoscale and TLS/SSL.
- Create a different App Service plan for testing and development. Use the *Free* and *Shared* (preview) tiers for testing and development for cost efficiency. But don't use the *Free* and *Shared* tiers for production workloads. Shared resources can't scale out.

- Make sure to delete plans that you aren't using, such as testing deployments. App Service plans are billed on a per-second basis. You're charged for the instances in the App Service plan even if the app is stopped. For more information about App Service plans and billing, see:
  - App Service Pricing.
  - How much does my App Service plan cost?

The ARM template below deploys to the Standard pricing tier.

SQL Database

- Use Azure SQL Database to reduce management overhead. Azure SQL Database creates a logical construct that acts as a central administrative point for a collection of databases. This logical construct reduces management overhead. Each database within the group is deployed with a specific service tier. Within each group, the databases can't share resources. There are no compute costs for the server, but you need to specify the tier for each database. Therefore, the performance might be better because of the dedicated resources, but the cost can be higher.
- Perform capacity planning and choose a tier and performance level that meets your requirements. SQL Database supports Basic, Standard, and Premium service tiers, with multiple performance levels within each tier measured in Database Transaction Units (DTUs).

Region

- Create the App Service plan and the SQL Database in the same region to minimize network latency. Generally, choose the region closest to your users.
- The resource group also has a region. It specifies where deployment metadata is stored. Put the resource group and its resources in the same region to improve availability during deployment.
- Use the pricing calculator to estimate costs.
- For more information, see the cost section in Microsoft Azure Well-Architected Framework.

Considerations

These considerations implement the pillars of the Azure Well-Architected Framework. The pillars are a set of guiding tenets that improve the quality of a workload. For more information, see Microsoft Azure Well-Architected Framework.

Performance efficiency

A major benefit of Azure App Service is the ability to scale your application based on load. Here are some considerations to keep in mind when planning to scale your application.

Scaling the App Service app

There are two ways to scale an App Service app:

- *Scale up* means changing the instance size. The instance size determines the memory, number of cores, and storage on each VM instance. You can scale up manually by changing the instance size or the plan tier.
- *Scale out* means adding instances to handle increased load. Each pricing tier has a maximum number of instances. You can scale out by manually changing the instance count or by configuring autoscaling to have Azure automatically add or remove instances based on a schedule and/or performance metrics. Each scale operation happens quickly, typically within seconds.

To enable autoscaling, create an autoscale *profile* that defines the minimum and maximum number of instances. You can set up schedule-based profiles to trigger scale events. For example, you can create separate profiles for weekdays and weekends. A profile can contain rules for when to add or remove instances. For example, adding two instances if CPU usage is above 70% for 5 minutes.

Recommendations for scaling a web app:

- Limit scaling up and down as much as possible. It can trigger an application restart. Instead, scale out. Select a tier and size that meet your performance requirements under typical load and then scale out the instances to handle changes in traffic volume.
- Enable autoscaling. If your application has a predictable, regular workload, create profiles to schedule the instance counts ahead of time. If the workload isn't predictable, use rule-based autoscaling to react to changes in load as they occur. You can combine both approaches.
- Use CPU usage for autoscaling rules. CPU usage is generally a good metric for autoscale rules. However, you should load test your application, identify potential bottlenecks, and base your autoscale rules on that data.
- Set a shorter cool-down period for adding instances and a longer cool-down period for removing instances. Autoscale rules include a *cool-down* period. The *cool-down* period is the interval to wait after a scale action has been completed before starting a new scale action. The cool-down period lets the system stabilize

before scaling again. For example, set 5 minutes to add an instance, but 60 minutes to remove an instance. It's better to add new instances quickly under heavy load to handle the extra traffic and then gradually scale back.

Scaling SQL databases

Scale up individual databases with no application downtime if you need a higher service tier or performance level for SQL Database.

For more information, see Scale single database resources in Azure SQL Database.

Reliability

At the time of writing, the service level agreement (SLA) for App Service is 99.95%. The App Service SLA applies to both single and multiple instances. The SLA for SQL Database is 99.99% for Basic, Standard, and Premium tiers.

Backups

SQL Database provides point-in-time restore and geo-restore to restore data loss. These features are available in all tiers and are automatically enabled. You don't need to schedule or manage the backups.

- Use point-in-time restore. You can recover from human error by returning the database to an earlier point in time.
- Use geo-restore. You can recover from a service outage by restoring a database from a geo-redundant backup.
- Consider using App Service backup and restore. Backup and restore is feature for your application files. However, the backed-up files include app settings in plain text, such as connection strings.
- Avoid using the App Service backup feature to back up your SQL databases. It exports the database to a SQL BACPAC file, consuming DTUs. Instead, use SQL Database point-in-time restore described above. For more information, see cloud business continuity and database disaster recovery with SQL Database.

Operational excellence

Create separate resource groups for production, development, and test environments. Separating environments makes it easier to manage deployments, delete test deployments, and assign access rights.

When assigning resources to resource groups, consider the following features:

- *Lifecycle*. In general, put resources with the same lifecycle into the same resource group.
- *Access*. You can use Azure role-based access control (RBAC) to apply access policies to the resources in a group.
- *Billing*. You can view the rolled-up costs for the resource group.

For more information, see Azure Resource Manager overview.

App configurations

- Store configuration settings as app settings. Define the app settings in your Resource Manager templates or using PowerShell. At runtime, app settings are available to the application as environment variables.
- Never check passwords, access keys, or connection strings into source control. Instead, pass secrets as parameters to a deployment script that stores these values as app settings.
- When you swap a deployment slot, the app settings are swapped by default. If you need different production and staging settings, you can create app settings that stick to a slot and don't get swapped.

Diagnostics and monitoring

- Enable diagnostics logging, including application logging and web server logging. Configure logging to use Azure Log Analytics. For more detailed guidance on logging, see Monitoring and diagnostics guidance.
- Use a service such as New Relic or Application Insights to monitor application performance and behavior under load. Be aware of the data rate limits for Application Insights.
- Perform load testing, using a tool such as Azure DevOps or Azure DevOps Server. For a general overview of performance analysis in cloud applications, see Performance Analysis Primer.

DevOps

- Use ARM templates to deploy Azure resources and their dependencies. The accompanying ARM template deploys a single web application. All the resources are isolated in the same basic workload. This isolation makes it easier to associate the workload's specific resources to a team. The team can then independently

manage all aspects of those resources. This isolation enables the DevOps team to perform continuous integration and continuous delivery (CI/CD).

- Use different ARM Templates and integrate them with Azure DevOps services. This setup lets you create different environments in minutes. For example, you can replicate production-like scenarios or load testing environments only when needed and save on cost.
- Provision multiple instances of the web application. You don't want your web app to depend on a single instance and potentially create a single point of failure. Multiple instances improve resiliency and scalability.

For more information, see the DevOps section in Azure Well-Architected Framework.

Release engineering and deployment

- Use Azure Resource Manager templates to provision Azure resources. Templates make it easier to automate deployments via PowerShell or the Azure CLI.
- Deploy the application (code, binaries, and content files). You have several options, including deploying from a local Git repository, using Visual Studio, or continuous deployment from cloud-based source control. See Deploy your app to Azure App Service.

An App Service app always has one deployment slot named `production`. The production slot represents the live production site. We recommend creating a staging slot for deploying updates. The benefits of using a staging slot include:

- You can verify the deployment succeeded before swapping it into production.
- Deploying to a staging slot ensures that all instances are warmed up before being swapped into production. Many applications have a significant warmup and cold-start time.
- Create a third slot to hold the last-known-good deployment. After you swap staging and production, move the previous production deployment (which is now in staging) into the last-known-good slot. That way, if you discover a problem later, you can quickly revert to the last-known-good version.

- If you revert to a previous version, make sure any database schema changes are backward compatible.
- Don't use slots on your production deployment for testing because all apps within the same App Service plan share the same VM instances. For example, load tests might degrade the live production site. Instead, create separate App Service plans for production and test. By putting test deployments into a separate plan, you isolate them from the production version.

Security

This section lists security considerations that are specific to the Azure services described in this article. It's not a complete list of security best practices. For some other security considerations, see Secure an app in Azure App Service.

SQL Database auditing

Auditing can help you maintain regulatory compliance and get insight into discrepancies and irregularities that could indicate business concerns or suspected security violations. See Get started with SQL database auditing.

Deployment slots

Each deployment slot has a public IP address. Secure the nonproduction slots using the Azure Active Directory login so that only members of your development and DevOps teams can reach those endpoints.

Logging

Logs should never record users' passwords or other information that might be used to commit identity fraud. Scrub those details from the data before storing it.

SSL

An App Service app includes an SSL endpoint on a subdomain of `azurewebsites.net` at no extra cost. The SSL endpoint includes a wildcard certificate for the `*.azurewebsites.net` domain. If you use a custom domain name, you must provide a certificate that matches the custom domain. The simplest approach is to buy a certificate directly through the Azure portal. You can also import certificates from other certificate authorities. For more information, see buy and configure an SSL certificate for your Azure App Service.

HTTPS isn't enabled by default in the ARM template deployment. As a security best practice, your app should enforce HTTPS by redirecting HTTP requests. You can implement HTTPS inside your application or use a URL rewrite rule as described in enable HTTPS for an app in Azure App Service.

Authentication

We recommend authenticating through an identity provider (IDP), such as Azure AD, Facebook, Google, or Twitter. Use OAuth 2 or OpenID Connect (OIDC) for the authentication flow. Azure AD provides functionality to manage users and groups, create application roles, integrate your on-premises identities, and consume backend services such as Microsoft 365 and Skype for Business.

Avoid having the application manage user logins and credentials directly. It creates a potential attack surface. At a minimum, you would need to have an email confirmation, password recovery, and multi-factor authentication, validate password strength, and store password hashes securely. The large identity providers handle all of those things for you and are constantly monitoring and improving their security practices.

Consider using App Service authentication to implement the OAuth or OIDC authentication flow. The benefits of App Service authentication include:

- Easy to configure.
- No code is required for simple authentication scenarios.
- Supports delegated authorization using OAuth access tokens to consume resources on behalf of the user.
- Provides a built-in token cache.

Some limitations of App Service authentication:

- Limited customization options.
- Delegated authorization is restricted to one backend resource per login session.
- If you use more than one IDP, there's no built-in mechanism for home realm discovery.
- For multi-tenant scenarios, the application must implement the logic to validate the token issuer.

Deploy this scenario

This architecture includes an Azure App Service plan and an empty application. It uses Azure SQL Database, Azure Key Vault for storing the database connection string, and Azure Monitor for logging, monitoring, and alerting.

Azure CLI

Use the following command to create a resource group for the deployment. Select the **Try it** button to use an embedded shell.

Azure CLI

```
az group create --name basic-web-app --location eastus
```

Run the following command to deploy the web application and supporting infrastructure. When prompted, enter a user name and password. These values are used for accessing the Azure SQL Database instance.

Azure CLI

```
az deployment group create --resource-group basic-web-app  \
    --template-uri
https://raw.githubusercontent.com/mspnp/samples/master/solutions/basic-
web-app/azuredeploy.json
```

For detailed information and more deployment options, see the ARM Templates used to deploy this solution.

Basic web application ARM Template

Next steps

Tips for troubleshooting your application:

- Use the troubleshoot blade in the Azure portal to find solutions to common problems.
- Enable log streaming to see logging information in near-real-time.
- The Kudu dashboard has several tools for monitoring and debugging your application. For more information, see Azure Websites online tools you should know about (blog post). You can reach the Kudu dashboard from the Azure portal. Open the blade for your app and select **Tools**, then select **Kudu**.
- If you use Visual Studio, see the article Troubleshoot a web app in Azure App Service using Visual Studio for debugging and troubleshooting tips.

Product documentation:

- About Azure Key Vault

- App Service overview
- Azure Monitor overview
- Azure App Service plan overview
- Overview of Log Analytics in Azure Monitor
- What is Azure Active Directory?
- What is Azure DNS?
- What is Azure SQL Database?

Microsoft Learn modules:

- Configure and manage Azure Monitor
- Configure Azure Active Directory
- 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 Azure Active Directory

Related resources

- Ten design principles for Azure applications
- Baseline zone-redundant web application
- Highly available multi-region web application
- Multi-region web app with private connectivity to database

Use existing runbooks and modules

Article • 04/23/2023

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.

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**.

8. Alternatively, Select **Browse Gallery** in the **Runbooks** page to browse the available runbooks.

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.

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**.

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**.

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.

> **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.

> **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.

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.

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.

Frequently asked questions about creating or deleting resources in Azure App Service

FAQ

This article answers common questions about creating and deleting web apps in Azure App Service.

If your Azure issue is not addressed in this article, visit the Azure forums on MSDN and Stack Overflow. You can post your issue in these forums, or post to @AzureSupport on Twitter. You also can submit an Azure support request. To submit a support request, on the Azure support page, select **Get support**.

Create a web app

What if I create a web app with the same name as another one?

We cannot create a web app with a name that already exists in the Azure. The web app name is part of the web app's URL, so it must be unique among all Azure App Service web apps.

How can I create a web app in a region that's unavailable in the subscription?

Certain Azure regions require customers to go through a request process to gain access. For more information, see the Azure region access request process and Azure App Service available by region.

What if the requested SKU of the App Service Plan isn't available in the Resource Group?

Your App Service deployments may not have specific SKUs available due to constraints (for example, the SKU isn't available in the region or country).

If the requested SKU isn't available in the Resource Group, you must create a new App Service Plan in a new Resource Group in the same region or in the same Resource Group using a different region.

If you want to reuse the same Resource Group and region, you must delete all the App Services, App Service Plans, and `Microsoft.Web/Certificates` resources in this Resource Group, then create the desired SKU in this Resource Group. Note that migrating App Services originating in this Resource Group to another Resource Group won't unblock the creation.

Delete or restore a web app

If I delete all my web apps, will I still be charged?

Yes, charges will still apply unless you delete the App Service plan that the web app runs on and its all web apps.

To stop all billing associated with your App Service, you need to delete the App Service Plan or scale the App Service Plan to the free tier. For more information, see How much does my App Service Plan cost? and Plan and manage costs for Azure App Service.

I cannot create or delete a web app due to a permission error. What the permissions do I need to create or delete a web app?

You would need minimum Contributor access on the Resource Group to deploy App Services. If you have Contributor access only on App Service Plan and web app, it won't allow you to create the app service in the Resource Group.

For more information, see Azure built-in roles.

I'm trying to delete my App Service Plan, but I'm getting the following error "Storage usage quota exceeded. Can't update or delete a server farm. Please make sure your file system storage is below the limit of the target pricing tier". How

and where do I check the file system storage limit?

During the deletion process, we calculate the usage of the remaining App Service Plans. If they are above the remaining limit, then this error appears.

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 can't 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.

For more information, see App Service limits - Storage and App Service pricing.

Is there a way to list the deleted web apps for my Subscription?

You can run `Get-AzDeletedWebApp` to get the list of the web apps that were deleted within the last 30 days in your Subscription ID. Deleted apps are purged from the system 30 days after the initial deletion. After an app is purged, it can't be recovered. For more information, see List deleted apps.

How do I restore a deleted web app or a deleted App Service Plan?

If the web app was deleted within the last 30 days, you can restore it using `Restore-AzDeletedWebApp`. For more information, see Restore deleted app and Restore Deleted web apps.

Contact us for help

If you have questions or need help, create a support request, or ask Azure community support. You can also submit product feedback to Azure community support.

Feedback

Was this page helpful? Yes No

Application performance FAQs for Web Apps in Azure

Article • 10/04/2022

> **Note**
>
> Some of the below guidelines might only work on Windows or Linux App Services. For example, Linux App Services run in 64-bit mode by default.

This article has answers to frequently asked questions (FAQs) about application performance issues for the Web Apps feature of Azure App Service.

If your Azure issue is not addressed in this article, visit the Azure forums on MSDN and Stack Overflow. You can post your issue in these forums, or post to @AzureSupport on Twitter. You also can submit an Azure support request. To submit a support request, on the Azure support page, select **Get support**.

Why is my App Service Plan displaying CPU/Memory usage even when all Web Apps are stopped?

Azure App Service requires continuous system processes that handle several platform operations and features, such as security updates, availability of the SCM console, application monitoring, authentication, and many other vital features of your Web App.

System processes will run on App Service Plans even if there are no Web Apps running or if the App Service Plan contains no Web Apps.

The platform processes will consume a minimum amount of resources (such as CPU, Memory and Disk space), and the same should be accounted during the capacity planning, monitoring, and auto-scaling trigger configuration of an App Service Plan.

Why is my app slow?

Multiple factors might contribute to slow app performance. For detailed troubleshooting steps, see Troubleshoot slow web app performance.

How do I troubleshoot a high CPU-consumption scenario?

In some high CPU-consumption scenarios, your app might truly require more computing resources. In that case, consider scaling to a higher service tier so the application gets all the resources it needs. Other times, high CPU consumption might be caused by a bad loop or by a coding practice. Getting insight into what's triggering increased CPU consumption is a two-part process. First, create a process dump, and then analyze the process dump. For more information, see Capture and analyze a dump file for high CPU consumption for Web Apps.

How do I troubleshoot a high memory-consumption scenario?

In some high memory-consumption scenarios, your app might truly require more computing resources. In that case, consider scaling to a higher service tier so the application gets all the resources it needs. Other times, a bug in the code might cause a memory leak. A coding practice also might increase memory consumption. Getting insight into what's triggering high memory consumption is a two-part process. First, create a process dump, and then analyze the process dump. Crash Diagnoser from the Azure Site Extension Gallery can efficiently perform both these steps. For more information, see Capture and analyze a dump file for intermittent high memory for Web Apps.

How do I automate App Service web apps by using PowerShell?

You can use PowerShell cmdlets to manage and maintain App Service web apps. In our blog post Automate web apps hosted in Azure App Service by using PowerShell, we describe how to use Azure Resource Manager-based PowerShell cmdlets to automate common tasks. The blog post also has sample code for various web apps management tasks. For descriptions and syntax for all App Service web apps cmdlets, see Az.Websites.

How do I view my web app's event logs?

To view your web app's event logs:

1. Sign in to your **Kudu website** (`https://*yourwebsitename*.scm.azurewebsites.net`).

2. In the menu, select **Debug Console** > **CMD**.
3. Select the **LogFiles** folder.
4. To view event logs, select the pencil icon next to **eventlog.xml**.
5. To download the logs, run the PowerShell cmdlet `Save-AzureWebSiteLog -Name webappname`.

How do I capture a user-mode memory dump of my web app?

To capture a user-mode memory dump of your web app:

1. Sign in to your **Kudu website** (`https://*yourwebsitename*.scm.azurewebsites.net`).
2. Select the **Process Explorer** menu.
3. Right-click the **w3wp.exe** process or your WebJob process.
4. Select **Download Memory Dump** > **Full Dump**.

How do I view process-level info for my web app?

You have two options for viewing process-level information for your web app:

- In the Azure portal:

    1. Open the **Process Explorer** for the web app.
    2. To see the details, select the **w3wp.exe** process.

- In the Kudu console:

    1. Sign in to your **Kudu website** (`https://*yourwebsitename*.scm.azurewebsites.net`).
    2. Select the **Process Explorer** menu.
    3. For the **w3wp.exe** process, select **Properties**.

When I browse to my app, I see "Error 403 - This web app is stopped." How do I resolve this?

Three conditions can cause this error:

- The web app has reached a billing limit and your site has been disabled.

- The web app has been stopped in the portal.
- The web app has reached a resource quota limit that might apply to a Free or Shared scale service plan.

To see what is causing the error and to resolve the issue, follow the steps in Web Apps: "Error 403 – This web app is stopped".

Where can I learn more about quotas and limits for various App Service plans?

For information about quotas and limits, see App Service limits.

How do I decrease the response time for the first request after idle time?

By default, web apps are unloaded if they're idle for a set period of time. This way, the system can conserve resources. The downside is that the response to the first request after the web app is unloaded is longer, to allow the web app to load and start serving responses. In Basic and Standard service plans, you can turn on the **Always On** setting to keep the app always loaded. This eliminates longer load times after the app is idle. To change the **Always On** setting:

1. In the Azure portal, go to your web app.
2. Select **Configuration**
3. Select **General settings**.
4. For **Always On**, select **On**.

How do I turn on failed request tracing?

To turn on failed request tracing, follow these steps:

1. In the Azure portal, go to your web app.

2. Select **All Settings** > **Diagnostics Logs**.

3. For **Failed Request Tracing**, select **On**.

4. Select **Save**.

5. On the web app blade, select **Tools**.

6. Select **Visual Studio Online**.

7. If the setting isn't **On**, select **On**.

8. Select **Go**.

9. Select **Web.config**.

10. In system.webServer, add the following configuration (to capture a specific URL):

```xml
<system.webServer>
<tracing> <traceFailedRequests>
<remove path="*api*" />
<add path="*api*">
<traceAreas>
<add provider="ASP" verbosity="Verbose" />
<add provider="ASPNET" areas="Infrastructure,Module,Page,AppServices"
verbosity="Verbose" />
<add provider="ISAPI Extension" verbosity="Verbose" />
<add provider="WWW Server"
areas="Authentication,Security,Filter,StaticFile,CGI,Compression,
Cache,RequestNotifications,Module,FastCGI" verbosity="Verbose" />
</traceAreas>
<failureDefinitions statusCodes="200-999" />
</add> </traceFailedRequests>
</tracing>
```

11. To troubleshoot slow-performance issues, add this configuration (if the capturing request is taking more than 30 seconds):

```xml
<system.webServer>
<tracing> <traceFailedRequests>
<remove path="*" />
<add path="*">
<traceAreas> <add provider="ASP" verbosity="Verbose" />
<add provider="ASPNET" areas="Infrastructure,Module,Page,AppServices"
verbosity="Verbose" />
<add provider="ISAPI Extension" verbosity="Verbose" />
<add provider="WWW Server"
areas="Authentication,Security,Filter,StaticFile,CGI,Compression,
Cache,RequestNotifications,Module,FastCGI" verbosity="Verbose" />
</traceAreas>
<failureDefinitions timeTaken="00:00:30" statusCodes="200-999" />
</add> </traceFailedRequests>
</tracing>
```

12. To download the failed request traces, in the portal, go to your website.

13. Select **Tools** > **Kudu** > **Go**.
14. In the menu, select **Debug Console** > **CMD**.
15. Select the **LogFiles** folder, and then select the folder with a name that starts with **W3SVC**.
16. To see the XML file, select the pencil icon.

I see the message "Worker Process requested recycle due to 'Percent Memory' limit." How do I address this issue?

The maximum available amount of memory for a 32-bit process (even on a 64-bit operating system) is 2 GB. By default, the worker process is set to 32-bit in App Service (for compatibility with legacy web applications).

Consider switching to 64-bit processes so you can take advantage of the additional memory available in your Web Worker role. This action triggers a web app restart, so schedule accordingly.

Also note that a 64-bit environment requires a Basic or Standard service plan. Free and Shared plans always run in a 32-bit environment.

For more information, see Configure web apps in App Service.

Why does my request time out after 230 seconds?

Azure Load Balancer has a default idle timeout setting of four minutes. This setting is generally a reasonable response time limit for a web request. so, App Service returns a timeout to the client if your application does not return a response within approximately 240 seconds (230 seconds on Windows app, 240 seconds on Linux app). If your web app requires background processing, we recommend using Azure WebJobs. The Azure web app can call WebJobs and be notified when background processing is finished. You can choose from multiple methods for using WebJobs, including queues and triggers.

WebJobs is designed for background processing. You can do as much background processing as you want in a WebJob. For more information about WebJobs, see Run background tasks with WebJobs.

ASP.NET Core applications that are hosted in App Service sometimes stop responding. How do I fix this issue?

A known issue with an earlier Kestrel version might cause an ASP.NET Core 1.0 app that's hosted in App Service to intermittently stop responding. You also might see this message: "The specified CGI Application encountered an error and the server terminated the process."

This issue is fixed in Kestrel version 1.0.2. This version is included in the ASP.NET Core 1.0.3 update. To resolve this issue, make sure you update your app dependencies to use Kestrel 1.0.2. Alternatively, you can use one of two workarounds that are described in the blog post ASP.NET Core 1.0 slow perf issues in App Service web apps.

I can't find my log files in the file structure of my web app. How can I find them?

If you use the Local Cache feature of App Service, the folder structure of the LogFiles and Data folders for your App Service instance are affected. When Local Cache is used, subfolders are created in the storage LogFiles and Data folders. The subfolders use the naming pattern "unique identifier" + time stamp. Each subfolder corresponds to a VM instance in which the web app is running or has run.

To determine whether you're using Local Cache, check your App Service **Application settings** tab. If Local Cache is being used, the app setting `WEBSITE_LOCAL_CACHE_OPTION` is set to `Always`.

If you aren't using Local Cache and are experiencing this issue, submit a support request.

I see the message "An attempt was made to access a socket in a way forbidden by its access permissions." How do I resolve this error?

This error typically occurs if the outbound TCP connections on the VM instance are exhausted. In App Service, limits are enforced for the maximum number of outbound connections that can be made for each VM instance. For more information, see Cross-VM numerical limits.

This error also might occur if you try to access a local address from your application. For more information, see Local address requests.

For more information about outbound connections in your web app, see the blog post about outgoing connections to Azure websites.

How do I use Visual Studio to remote debug my App Service web app?

For a detailed walkthrough that shows you how to debug your web app by using Visual Studio, see Remote debug your App Service web app.

Contact us for help

If you have questions or need help, create a support request, or ask Azure community support. You can also submit product feedback to Azure community support.

Feedback

Was this page helpful? Yes No

Provide product feedback

Azure App Service Compliance with PCI Standards 3.0 and 3.1

Article • 06/23/2022

Original product version: Web App (Windows)
Original KB number: 3124528

The Azure App Service is currently in compliance with PCI DSS version 3.0 Level 1. We have also noted customer requests that make reference to PCI DSS version 3.1, and specifically the change from version 3.0 to 3.1, which states that SSL and "early TLS versions " will no longer be considered valid security options from June 30, 2018. From June 30th, 2018, the multi-tenant hosting model for Azure App Service, will be accepting TLS 1.2 with an option for customers to select their required TLS encryption level.

What this means

PCI DSS version 3.1 certification requires disabling TLS 1.0. If you are using App Service Environments or are willing to migrate your workload to App Service Environments, you can get greater control of your environment including disabling TLS 1.0, for more information, see Custom configuration settings for App Service Environments.

More information

Microsoft regularly reviews standards compliance procedures and will periodically update compliance baselines as standards bodies update and change their requirements. As part of Microsoft's Fiscal 2017 compliance planning, PCI standards will again be re-reviewed and technical determinations will be made. To view the current certifications, technical determinations will be made. To view the current certifications, visit the Microsoft Azure Trust Center: Compliance site .

Contact us for help

If you have questions or need help, create a support request , or ask Azure community support. You can also submit product feedback to Azure community support.

Feedback

Was this page helpful? Yes No

Provide product feedback

Backing up, restoring, and cloning Microsoft Azure App Service

Article • 06/23/2022

In this video, you will learn about different kinds of backup and restore procedures (traditional versus snapshot), the limitations of each, the requirements to complete the action (snapshot), and how to clone the app service to another ASP/ region.
https://www.youtube-nocookie.com/embed/P2BRPuQYPSo

Related content

For more troubleshooting tips, see My backups are failing - Azure App Service

Contact us for help

If you have questions or need help, create a support request, or ask Azure community support. You can also submit product feedback to Azure community support.

Feedback

Was this page helpful? Yes No

Provide product feedback

Azure App Service on Linux FAQ

FAQ

With the release of App Service on Linux, we're working on adding features and making improvements to our platform. This article provides answers to questions that our customers have been asking us recently.

If you have a question, comment on this article.

Built-in images

I want to fork the built-in Docker containers that the platform provides. Where can I find those files?

You can find all Docker files on GitHub.

What are the expected values for the Startup File section when I configure the runtime stack?

Stack	Expected Value
Java SE	the command to start your JAR app (for example, `java -jar /home/site/wwwroot/app.jar --server.port=80`)
Tomcat	the location of a script to perform any necessary configurations (for example, `/home/site/deployments/tools/startup_script.sh`)
Node.js	the PM2 configuration file or your script file
.NET Core	the compiled DLL name as `dotnet <myapp>.dll`
PHP	optional custom startup
Python	optional startup script

Stack	Expected Value
Ruby	the Ruby script that you want to initialize your app with

These commands or scripts are executed after the built-in Docker container is started, but before your application code is started.

Management

What happens when I press the restart button in the Azure portal?

This action is the same as a Docker restart.

Can I use Secure Shell (SSH) to connect to the app container virtual machine (VM)?

Yes, you can do that through the source control management (SCM) site.

> **Note**
>
> You can also connect to the app container directly from your local development machine using SSH, SFTP, or Visual Studio Code (for live debugging Node.js apps). For more information, see **Remote debugging and SSH in App Service on Linux**.

How can I create a Linux App Service plan through an SDK or an Azure Resource Manager template?

Set the **reserved** field of the app service to *true*.

Continuous integration and deployment

My web app still uses an old Docker container image after I've updated the image on Docker

Hub. Do you support continuous integration and deployment of custom containers?

Yes, to set up continuous integration/deployment for Azure Container Registry or DockerHub, by following Continuous Deployment with Web App for Containers. For private registries, you can refresh the container by stopping and then starting your web app. Or you can change or add a dummy application setting to force a refresh of your container.

Do you support staging environments?

Yes.

Can I use 'WebDeploy/MSDeploy' to deploy my web app?

Yes, you need to set an app setting called `WEBSITE_WEBDEPLOY_USE_SCM` to *false*.

Git deployment of my application fails when using Linux web app. How can I work around the issue?

If Git deployment fails to your Linux web app, choose one of the following options to deploy your application code:

- Use the Continuous Delivery (Preview) feature: You can store your app's source code in an Azure DevOps Git repo or GitHub repo to use Azure Continuous Delivery. For more information, see How to configure Continuous Delivery for Linux web app.

- Use the ZIP deploy API: To use this API, SSH into your web app and go to the folder where you want to deploy your code. Run the following code:

  Bash

  ```
  curl -X POST -u <user> --data-binary @<zipfile> https://{your-sitename}.scm.azurewebsites.net/api/zipdeploy
  ```

  If you get an error that the `curl` command is not found, make sure you install curl by using `apt-get install curl` before you run the previous `curl` command.

Language support

I want to use web sockets in my Node.js application, any special settings, or configurations to set?

Yes, disable `perMessageDeflate` in your server-side Node.js code. For example, if you are using socket.io, use the following code:

Node.js

```
const io = require('socket.io')(server,{
  perMessageDeflate :false
});
```

Do you support uncompiled .NET Core apps?

Yes.

Do you support Composer as a dependency manager for PHP apps?

Yes, during a Git deployment, Kudu should detect that you're deploying a PHP application (thanks to the presence of a composer.lock file), and Kudu will then trigger a composer install.

Custom containers

Can I use Managed Identities with App Service when pulling images form ACR?

Yes, this functionality is available from the Azure CLI. You can use system-assigned or user-assigned identities. This functionality isn't currently supported in the Azure portal.

I'm using my own custom container. I want the platform to mount an SMB share to the `/home/`

directory.

If `WEBSITES_ENABLE_APP_SERVICE_STORAGE` setting is **unspecified** or set to *false*, the `/home/` directory **will not be shared** across scale instances, and files written **will not persist** across restarts. Explicitly setting `WEBSITES_ENABLE_APP_SERVICE_STORAGE` to *true* will enable the mount. Once this is set to true, if you wish to disable the mount, you need to explicitly set `WEBSITES_ENABLE_APP_SERVICE_STORAGE` to *false*.

My container fails to start with "no space left on device". What does this error mean?

App Service on Linux uses two different types of storage:

- File system storage: The file system storage is included in the App Service plan quota. It's used when files are saved to the persistent storage that's rooted in the `/home` directory.
- Host disk space: The host disk space is used to store container images. It's managed by the platform through the docker storage driver.

The host disk space is separate from the file system storage quota. It's not expandable and there is a 15 GB limit for each instance. It's used to store any custom images on the worker. You might be able to use larger than 15 GBs depending on the exact availability of host disk space, but this isn't guaranteed.

If the container's writable layer saves data outside of the `/home` directory or a mounted azure storage path, the host disk space will also be consumed.

The platform routinely cleans the host disk space to remove unused containers. If the container writes a large quantity of data outside of the `/home` directory or Bring Your Own Storage (BYOS), it will result in startup failures or runtime exceptions once the host disk space limit is exceeded.

We recommend that you keep your container images as small as possible and write data to the persistent storage or BYOS when running on Linux App Service. If not possible, you have to split the App Service plan because the host disk space is fixed and shared between all containers in the App Service Plan.

My custom container takes a long time to start, and the platform restarts the container before it finishes starting up.

You can configure the amount of time the platform will wait before it restarts your container. To do so, set the `WEBSITES_CONTAINER_START_TIME_LIMIT` app setting to the value you want. The default value is 230 seconds, and the maximum value is 1800 seconds.

What is the format for the private registry server URL?

Provide the full registry URL, including `http://` or `https://`.

What is the format for the image name in the private registry option?

Add the full image name, including the private registry URL (for example, myacr.azurecr.io/dotnet:latest). Image names that use a custom port cannot be entered through the portal. To set `docker-custom-image-name`, use the az command-line tool.

Can I expose more than one port on my custom container image?

We don't support exposing more than one port.

Can I bring my own storage?

Yes, bring your own storage is in preview.

Why can't I browse my custom container's file system or running processes from the SCM site?

The SCM site runs in a separate container. You can't check the file system or running processes of the app container.

Do I need to implement HTTPS in my custom container?

No, the platform handles HTTPS termination at the shared front ends.

Do I need to use WEBSITES_PORT for custom containers?

Yes, this is required for custom containers. To manually configure a custom port, use the EXPOSE instruction in the Dockerfile and the app setting, WEBSITES_PORT, with a port value to bind on the container.

Can I use ASPNETCORE_URLS in the Docker image?

Yes, overwrite the environmental variable before .NET core app starts. E.g. In the init.sh script: export ASPNETCORE_URLS={Your value}

Multi-container with Docker Compose

How do I configure Azure Container Registry (ACR) to use with multi-container?

In order to use ACR with multi-container, **all container images** need to be hosted on the same ACR registry server. Once they are on the same registry server, you will need to create application settings and then update the Docker Compose configuration file to include the ACR image name.

Create the following application settings:

- DOCKER_REGISTRY_SERVER_USERNAME
- DOCKER_REGISTRY_SERVER_URL (full URL, ex: `https://<server-name>.azurecr.io`)
- DOCKER_REGISTRY_SERVER_PASSWORD (enable admin access in ACR settings)

Within the configuration file, reference your ACR image like the following example:

YAML

```yaml
image: <server-name>.azurecr.io/<image-name>:<tag>
```

How do I know which container is internet accessible?

- Only one container can be open for access

- Only port 80 and 8080 is accessible (exposed ports)

Here are the rules for determining which container is accessible - in the order of precedence:

- Application setting `WEBSITES_WEB_CONTAINER_NAME` set to the container name
- The first container to define port 80 or 8080
- If neither of the above is true, the first container defined in the file will be accessible (exposed)

How do I use depends_on?

The `depends_on` option is *unsupported* on App Service and it'll be ignored. Just as the control startup and shutdown recommendation from Docker, App Service Multi-container apps should check dependencies through application code - both at startup and disconnection. The example code below shows a Python app checking to see if a Redis container is running.

Python

```python
import time
import redis
from flask import Flask
app = Flask(__name__)
cache = redis.Redis(host='redis', port=6379)
def get_hit_count():
    retries = 5
    while True:
        try:
            return cache.incr('hits')
        except redis.exceptions.ConnectionError as exc:
            if retries == 0:
                raise exc
            retries -= 1
            time.sleep(0.5)
@app.route('/')
def hello():
    count = get_hit_count()
    return 'Hello from Azure App Service team! I have been seen {}
times.\n'.format(count)
if __name__ == "__main__":
    app.run(host="0.0.0.0", port=80, debug=True)
```

Web Sockets

Web Sockets are supported on Linux apps. The `webSocketsEnabled` ARM setting does not apply to Linux apps since Web Sockets are always enabled for Linux.

> **Important**
>
> Web Sockets are now supported for Linux apps on Free App Service plans. We support up to five web socket connections on Free App Service plans. Exceeding this limit results in an HTTP 429 (Too Many Requests) response.

Pricing and SLA

What is the pricing, now that the service is generally available?

Pricing varies by SKU and region but you can see more details at our pricing page: App Service Pricing.

Other questions

How does the container warmup request work?

When Azure App Services starts your container, the warmup request sends an HTTP request to the /robots933456.txt endpoint of your application. This is simply a dummy endpoint, but your application needs to reply with any non-5XX status code. If your application logic doesn't reply with any HTTP status code to nonexistent endpoints, the warmup request can't receive a response and it perpetually restarts your container. The warmup request also might fail due to port misconfiguration.

To ensure that the port is correctly configured on Azure App Services, see the question *How do I specify port in my Linux container?*

Is it possible to increase the container warmup request timeout?

The warmup request by default fails after waiting 240 seconds for a reply from the container. You may increase the container warmup request timeout by adding the application setting `WEBSITES_CONTAINER_START_TIME_LIMIT` with a value between 240 and 1800 seconds.

How do I specify port in my Linux container?

Container type	Description	How to set/use port
Built-in containers	If you select a language/framework version for a Linux app, a predefined container is selected for you.	To point your app code to the right port, use the PORT environment variable.
Custom containers	You have full control over the container.	App Service has no control about which port your container listens on. What it does need is to know which port to forward requests to. If your container listens to port 80 or 8080, App Service is able to automatically detect it. If it listens to any other port, you need to set the WEBSITES_PORT app setting to the port number, and App Service forwards requests to that port in the container. The WEBSITES_PORT app setting does not have any effect within the container, and you can't access it as an environment variable within the container.

Can I use a file based database (like SQLite) with my Linux Webapp?

The file system of your application is a mounted network share. This enables scale out scenarios where your code needs to be executed across multiple hosts. Unfortunately this blocks the use of file-based database providers like SQLite since it's not possible to acquire exclusive locks on the database file. We recommend a managed database service: Azure SQL, Azure Database for MySQL or Azure Database for PostgreSQL

What are the supported characters in application settings names?

You can use only letters (A-Z, a-z), numbers (0-9), and the underscore character (_) for application settings.

Where can I request new features?

You can submit your idea at the Web Apps feedback forum. Add "[Linux]" to the title of your idea.

Next steps

- What is Azure App Service on Linux?
- Set up staging environments in Azure App Service
- Continuous Deployment with Web App for Containers
- Things You Should Know: Web Apps and Linux
- Environment variables and app settings reference

Contact us for help

If you have questions or need help, create a support request, or ask Azure community support. You can also submit product feedback to Azure community support.

Feedback

Was this page helpful? Yes No

Frequently asked questions about creating or deleting resources in Azure App Service

FAQ

This article answers common questions about creating and deleting web apps in Azure App Service.

If your Azure issue is not addressed in this article, visit the Azure forums on MSDN and Stack Overflow. You can post your issue in these forums, or post to @AzureSupport on Twitter. You also can submit an Azure support request. To submit a support request, on the Azure support page, select **Get support**.

Create a web app

What if I create a web app with the same name as another one?

We cannot create a web app with a name that already exists in the Azure. The web app name is part of the web app's URL, so it must be unique among all Azure App Service web apps.

How can I create a web app in a region that's unavailable in the subscription?

Certain Azure regions require customers to go through a request process to gain access. For more information, see the Azure region access request process and Azure App Service available by region.

What if the requested SKU of the App Service Plan isn't available in the Resource Group?

Your App Service deployments may not have specific SKUs available due to constraints (for example, the SKU isn't available in the region or country).

If the requested SKU isn't available in the Resource Group, you must create a new App Service Plan in a new Resource Group in the same region or in the same Resource Group using a different region.

If you want to reuse the same Resource Group and region, you must delete all the App Services, App Service Plans, and `Microsoft.Web/Certificates` resources in this Resource Group, then create the desired SKU in this Resource Group. Note that migrating App Services originating in this Resource Group to another Resource Group won't unblock the creation.

Delete or restore a web app

If I delete all my web apps, will I still be charged?

Yes, charges will still apply unless you delete the App Service plan that the web app runs on and its all web apps.

To stop all billing associated with your App Service, you need to delete the App Service Plan or scale the App Service Plan to the free tier. For more information, see How much does my App Service Plan cost? and Plan and manage costs for Azure App Service.

I cannot create or delete a web app due to a permission error. What the permissions do I need to create or delete a web app?

You would need minimum Contributor access on the Resource Group to deploy App Services. If you have Contributor access only on App Service Plan and web app, it won't allow you to create the app service in the Resource Group.

For more information, see Azure built-in roles.

I'm trying to delete my App Service Plan, but I'm getting the following error "Storage usage quota exceeded. Can't update or delete a server farm. Please make sure your file system storage is below the limit of the target pricing tier". How

and where do I check the file system storage limit?

During the deletion process, we calculate the usage of the remaining App Service Plans. If they are above the remaining limit, then this error appears.

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 can't 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.

For more information, see App Service limits - Storage and App Service pricing.

Is there a way to list the deleted web apps for my Subscription?

You can run `Get-AzDeletedWebApp` to get the list of the web apps that were deleted within the last 30 days in your Subscription ID. Deleted apps are purged from the system 30 days after the initial deletion. After an app is purged, it can't be recovered. For more information, see List deleted apps.

How do I restore a deleted web app or a deleted App Service Plan?

If the web app was deleted within the last 30 days, you can restore it using `Restore-AzDeletedWebApp`. For more information, see Restore deleted app and Restore Deleted web apps.

Contact us for help

If you have questions or need help, create a support request, or ask Azure community support. You can also submit product feedback to Azure community support.

Feedback

Was this page helpful? Yes No

Deployment FAQs for Web Apps in Azure

Article • 06/23/2022

This article has answers to frequently asked questions (FAQs) about deployment issues for the Web Apps feature of Azure App Service.

If your Azure issue is not addressed in this article, visit the Azure forums on MSDN and Stack Overflow. You can post your issue in these forums, or post to @AzureSupport on Twitter. You also can submit an Azure support request. To submit a support request, on the Azure support page, select **Get support**.

I'm just getting started with App Service web apps. How do I publish my code?

Here are some options for publishing your web app code:

- Deploy by using Visual Studio. If you have the Visual Studio solution, right-click the web application project, and then select **Publish**.
- Deploy by using an FTP client. In the Azure portal, download the publish profile for the web app that you want to deploy your code to. Then, upload the files to \site\wwwroot by using the same publish profile FTP credentials.

For more information, see Deploy your app to App Service.

I see an error message when I try to deploy from Visual Studio. How do I resolve this error?

If you see the following message, you might be using an older version of the SDK:

> Error during deployment for resource 'YourResourceName' in resource group 'YourResourceGroup': MissingRegistrationForLocation: The subscription is not registered for the resource type 'components' in the location 'Central US'. Re-register for this provider in order to have access to this location.

To resolve this error, upgrade to the latest SDK. If you see this message and you have the latest SDK, submit a support request.

How do I deploy an ASP.NET application from Visual Studio to App Service?

The tutorial Create your first ASP.NET web app in Azure in five minutes shows you how to deploy an ASP.NET web application to a web app in App Service by using Visual Studio.

What are the different types of deployment credentials?

App Service supports two types of credentials for local Git deployment and FTP/S deployment. For more information about how to configure deployment credentials, see Configure deployment credentials for App Service.

What is the file or directory structure of my App Service web app?

For information about the file structure of your App Service app, see File structure in Azure.

How do I resolve "FTP Error 550 - There isn't enough space on the disk" when I try to FTP my files?

If you see this message, it's likely that you're running into a disk quota in the service plan for your web app. You might need to scale up to a higher service tier based on your disk space needs. For more information about pricing plans and resource limits, see App Service pricing.

How do I set up continuous deployment for my App Service web app?

You can set up continuous deployment from several resources, including Azure DevOps, OneDrive, GitHub, Bitbucket, Dropbox, and other Git repositories. These options are available in the portal. Continuous deployment to App Service is a helpful tutorial that explains how to set up continuous deployment.

How do I troubleshoot issues with continuous deployment from GitHub and Bitbucket?

For help investigating issues with continuous deployment from GitHub or Bitbucket, see Investigating continuous deployment.

I can't FTP to my site and publish my code. How do I resolve this issue?

To resolve FTP issues, follow these steps:

1. Verify that you're entering the correct host name and credentials. For detailed information about different types of credentials and how to use them, see Deployment credentials.
2. Verify that the FTP ports aren't blocked by a firewall. The ports should have these settings:
    - FTP control connection port: 21
    - FTP data connection port: 989, 10001-10300

How do I publish my code to App Service?

The Azure Quickstart is designed to help you deploy your app by using the deployment stack and method of your choice. To use the Quickstart, in the Azure portal, go to your app service, under **Deployment**, select **Quickstart**.

Why does my app sometimes restart after deployment to App Service?

To learn about the circumstances under which an application deployment might result in a restart, see Deployment vs. runtime issues. As the article describes, App Service deploys files to the wwwroot folder. It never directly restarts your app.

How do I integrate Azure DevOps code with App Service?

You have two options for using continuous deployment with Azure DevOps:

- Use a Git project. Connect via App Service by using the Deployment Center.
- Use a Team Foundation Version Control (TFVC) project. Deploy by using the build agent for App Service.

Continuous code deployment for both these options depends on existing developer workflows and check-in procedures. For more information, see these articles:

- Implement continuous deployment of your app to an Azure website
- Set up an Azure DevOps organization so it can deploy to a web app

How do I use FTP or FTPS to deploy my app to App Service?

For information about using FTP or FTPS to deploy your web app to App Service, see Deploy your app to App Service by using FTP/S.

Contact us for help

If you have questions or need help, create a support request, or ask Azure community support. You can also submit product feedback to Azure community support.

Feedback

Was this page helpful? Yes No

Provide product feedback

Frequently asked questions about scaling web apps in Azure App Service

FAQ

This article answers common questions about scaling web apps in Azure App Service.

If your Azure issue is not addressed in this article, visit the Azure forums on MSDN and Stack Overflow. You can post your issue in these forums, or post to @AzureSupport on Twitter. You also can submit an Azure support request. To submit a support request, on the Azure support page, select **Get support**.

How do I scale up a Web App?

You can scale up a Web App by using the Azure portal. On your Web App page, select **Scale Up (App Service Plan)** from the left menu. For more information, see Scale up an app in Azure App Service.

Is there any further action I need to perform before scaling up a Free App Service Plan?

Before you switch an App Service Plan from the Free tier, you must remove the spending limits in place for your Azure subscription. To view or change options for your Microsoft Azure App Service subscription, see Microsoft Azure Subscriptions - Remove spending limits.

Is there any instance limitation when scaling a Web App?

Yes, the limitation depends on the tier of the App Service Plan. For more information, see App Service limits.

Can I scale a Standard App Service Plan for more than 10 instances?

The Standard App Service Plan tier doesn't support more than 10 instances. You can move to a Premium App Service Plan and get the benefit of having 30 instances (in selected regions) per App Service Plan. To check the instance limit per pricing tier, see App Service limits.

Can I configure App Services on the same App Service Plan with a different number of instances?

You can configure it by enabling Per-App Scaling, and changing App Service's **numberOfWorkers** property to the desired instance count. For more information, see Per-App Scaling.

Why does scale up for my Web App also trigger scale up for another Web App?

Scaling happens on the entire App Service Plan. If you host multiple App Services in the same App Service Plan, all App Services in this App Service Plan will be scaled up.

Why isn't autoscale working as expected?

You might be running into a scenario in which we intentionally choose not to scale to avoid an infinite loop due to "flapping." This behavior usually happens when there isn't an adequate margin between the scale-out and scale-in thresholds. For more information about how to avoid "flapping" and other autoscale best practices, see Autoscale best practices.

How do I determine when an autoscale rule triggered scaling?

You can retrieve scale history from the activity log. Whenever your resource is scaled up or down, an event is logged in the activity log. You can view the scale history of your resource for the past 24 hours by switching to the **Run history** tab. To view the complete scale history (for up to 90 days), select **Click here to see more details**. For more information, see View the scale history of your resource.

Why does autoscale sometimes scale only partially?

Autoscale is triggered when metrics exceed preconfigured boundaries. Sometimes, you might notice that the capacity is only partially filled compared to what you expected. This behavior might occur when the number of instances you want isn't available. In that scenario, autoscale partially fills in with the available number of instances. Autoscale then runs the rebalance logic to get more capacity. It allocates the remaining instances, and this allocation might take a few minutes. If you don't see the expected number of instances after a few minutes, it might be because the partial refill was enough to bring the metrics within the boundaries. Or, autoscale might have scaled down because it reached the lower metrics boundary.

When I scale up an App Service Plan to a Premium V3 tier, the "Premium V3 is not supported for this scale unit. Please consider redeploying or cloning your app." error occurs. What should I do?

The Premium V3 feature requires the site to run on the newest hardware infrastructure. To scale up an App Service Plan to Premium V3, the Web App must be running in an App Service deployment that supports PremiumV3. For more information, see Scale up from an unsupported resource group and region combination.

I'm unable to scale up/scale down the App Service Plan due to the "You have exceeded the maximum amount of scale

changes within the past hour (XX changes and limit is XX)" error. What should I do?

To avoid this issue, don't perform scaling operations that release more than XX instances in an hour. Every time you release an instance during a scale-down operation, the instance is rebooted to ensure the next App Service Plan can get a clean instance. When you perform too many scaling operations in quick succession, instance reboots can cause performance issues for other App Services. Therefore we intentionally put a throttling mechanism for scaling that prevents you from executing scaling operations more than the acceptable limit in quick succession.

My Web App is using the Diagnostic setting "AppServiceFileAuditLogs" and I'm unable to scale the App Service Plan from Premium V2 to the Basic tier. What should I do?

The file change audit logs "AppServiceFileAuditLogs" are only available for App Services in Premium, PremiumV2, and Isolated App Service Plans. If you need "AppServiceFileAuditLogs," you won't be able to scale down to the Basic tier. To have these audit logs available, configure your App Service Plan for Premium or higher tier.

Contact us for help

If you have questions or need help, create a support request, or ask Azure community support. You can also submit product feedback to Azure community support.

Feedback

Was this page helpful? Yes No

Open-source technologies FAQs for Web Apps in Azure

Article • 09/28/2022

This article has answers to frequently asked questions (FAQs) about issues with open-source technologies for the Web Apps feature of Azure App Service.

If your Azure issue is not addressed in this article, visit the Azure forums on MSDN and Stack Overflow. You can post your issue in these forums, or post to @AzureSupport on Twitter. You also can submit an Azure support request. To submit a support request, on the Azure support page, select **Get support**.

How do I turn on PHP logging to troubleshoot PHP issues?

To turn on PHP logging, follow these steps:

1. Sign in to your **Kudu website** (`https://*yourwebsitename*.scm.azurewebsites.net`).
2. In the top menu, select **Debug Console** > **CMD**.
3. Select the **Site** folder.
4. Select the **wwwroot** folder.
5. Select the + icon, and then select **New File**.
6. Set the file name to **.user.ini**.
7. Select the pencil icon next to **.user.ini**.
8. In the file, add this code: `log_errors=on`
9. Select **Save**.
10. Select the pencil icon next to **wp-config.php**.
11. Change the text to the following code:

```php
//Enable WP_DEBUG modedefine('WP_DEBUG', true);//Enable debug logging
to /wp-content/debug.logdefine('WP_DEBUG_LOG', true);
```

```
//Suppress errors and warnings to screendefine('WP_DEBUG_DISPLAY',
false);//Suppress PHP errors to screenini_set('display_errors', 0);
```

12. In the Azure portal, in the web app menu, restart your web app.

For more information, see Enable WordPress error logs.

How do I log Python application errors in apps that are hosted in App Service?

If Python encounters an error while starting your application, only a simple error page will be returned. For example,

> The page cannot be displayed because an internal server error has occurred.

To capture Python application errors, follow these steps:

1. In the Azure portal, in your web app, select **Settings**.
2. On the **Settings** tab, select **Application settings**.
3. Under **App settings**, enter the following key/value pair:
    - Key: *WSGI_LOG*
    - Value: *D:\home\site\wwwroot\logs.txt* (enter your choice of file name)

You should now see errors in the *logs.txt* file in the *wwwroot* folder.

How do I change the version of the Node.js application that is hosted in App Service?

To change the version of the Node.js application, you can use one of the following options:

- In the Azure portal, use **App settings**.
    1. In the Azure portal, go to your web app.
    2. On the **Settings** blade, select **Application settings**.
    3. In **App settings**, you can include WEBSITE_NODE_DEFAULT_VERSION as the key, and the version of Node.js you want as the value.

4. Go to your **Kudu console** (`https://*yourwebsitename*.scm.azurewebsites.net`).

5. To check the Node.js version, enter the following command:

```
Node.js

node -v
```

- Modify the *iisnode.yml* file. Changing the Node.js version in the *iisnode.yml* file only sets the runtime environment that iisnode uses. Your Kudu cmd and others still use the Node.js version that is set in **App settings** in the Azure portal.

  To set the *iisnode.yml* manually, create an *iisnode.yml* file in your app root folder. In the file, include the following line:

```yml
nodeProcessCommandLine: "D:\Program Files (x86)\nodejs\5.9.1\node.exe"
```

- Set the *iisnode.yml* file by using *package.json* during source control deployment. The Azure source control deployment process involves the following steps:

  1. Moves content to the Azure web app.

  2. Creates a default deployment script, if there isn't one (*deploy.cmd*, *.deployment* files) in the web app root folder.

  3. Runs a deployment script in which it creates an *iisnode.yml* file if you mention the Node.js version in the *package.json* file > engine `"engines": {"node": "5.9.1","npm": "3.7.3"}`

  4. The *iisnode.yml* file has the following line of code:

```yml
nodeProcessCommandLine: "D:\Program Files (x86)\nodejs\5.9.1\node.exe"
```

I see the message "Error establishing a database connection" in my WordPress app

that's hosted in App Service. How do I troubleshoot this error?

If you see this error in your Azure WordPress app, to enable *php_errors.log* and *debug.log*, complete the steps detailed in Enable WordPress error logs.

When the logs are enabled, reproduce the error, and then check the logs to see if you're running out of connections:

Output

```
[09-Oct-2015 00:03:13 UTC] PHP Warning: mysqli_real_connect(): (HY000/1226):
User ‘abcdefghijk79' has exceeded the ‘max_user_connections’ resource
(current value: 4) in D:\home\site\wwwroot\wp-includes\wp-db.php on line
1454
```

If you see this error in your *debug.log* or *php_errors.log* files, your app is exceeding the number of connections. If you're hosting on ClearDB, verify the number of connections that are available in your service plan.

How do I debug a Node.js app that's hosted in App Service?

1. Go to your **Kudu console** (`https://*yourwebsitename*.scm.azurewebsites.net/DebugConsole`).
2. Go to your application logs folder (*D:\home\LogFiles\Application*).
3. In the *logging_errors.txt* file, check for content.

How do I install native Python modules in an App Service web app or API app?

Some packages might not install by using pip in Azure. The package might not be available on the Python Package Index, or a compiler might be required (a compiler isn't available on the computer that is running the web app in App Service). For information about installing native modules in App Service web apps and API apps, see Install Python modules in App Service.

How do I deploy a Django app to App Service by using Git and the new version of Python?

For information about installing Django, see Deploying a Django app to App Service.

Where are the Tomcat log files located?

For Azure Marketplace and custom deployments:

- Folder location: *D:\home\site\wwwroot\bin\apache-tomcat-8.0.33\logs*
- Files of interest:
  - *catalina.<yyyy-mm-dd>.log*
  - *host-manager.<yyyy-mm-dd>.log*
  - *localhost.<yyyy-mm-dd>.log*
  - *manager.<yyyy-mm-dd>.log*
  - *site_access_log.<yyyy-mm-dd>.log*

For portal **App settings** deployments:

- Folder location: *D:\home\LogFiles*
- Files of interest:
  - *catalina.<yyyy-mm-dd>.log*
  - *host-manager.<yyyy-mm-dd>.log*
  - *localhost.<yyyy-mm-dd>.log*
  - *manager.<yyyy-mm-dd>.log*
  - *site_access_log.<yyyy-mm-dd>.log*

How do I troubleshoot JDBC driver connection errors?

You might see the following message in your Tomcat logs:

> The web application[ROOT] registered the JDBC driver [com.mysql.jdbc.Driver] but failed to unregister it when the web application was stopped. To prevent a memory leak,the JDBC Driver has been forcibly unregistered

To resolve the error, follow these steps:

1. Remove the *sqljdbc*.jar* file from your *app/lib* folder.

2. If you're using the custom Tomcat or Azure Marketplace Tomcat web server, copy this .jar file to the Tomcat lib folder.

3. If you're enabling Java from the Azure portal (select **Java 1.8** > **Tomcat server**), copy the *sqljdbc*.jar* file in the folder that's parallel to your app. Then, add the

following classpath setting to the *web.config* file:

```XML
<httpPlatform>
    <environmentVariables>
        <environmentVariablename ="JAVA_OPTS" value=" -
Djava.net.preferIPv4Stack=true
-Xms128M -classpath %CLASSPATH%;[Path to the sqljdbc*.jarfile]" />
    </environmentVariables>
</httpPlatform>
```

Why do I see errors when I attempt to copy live log files?

If you try to copy live log files for a Java app (for example, Tomcat), you might see this FTP error:

> Error transferring file [filename] Copying files from remote side failed.
>
> The process cannot access the file because it is being used by another process.

The error message might vary, depending on the FTP client.

All Java apps have this locking issue. Only Kudu supports downloading this file while the app is running.

Stopping the app allows FTP access to these files.

Another workaround is to write a WebJob that runs on a schedule and copies these files to a different directory. For a sample project, see the CopyLogsJob project.

Where do I find the log files for Jetty?

For Marketplace and custom deployments, the log file is in the *D:\home\site\wwwroot\bin\jetty-distribution-9.1.2.v20140210\logs* folder. The folder location depends on the version of Jetty you're using. For example, the path provided here is for Jetty 9.1.2. Look for *jetty_<YYYY_MM_DD>.stderrout.log*.

For portal App Setting deployments, the log file is in *D:\home\LogFiles*. Look for *jetty_<YYYY_MM_DD>.stderrout.log*.

Can I send email from my Azure web app?

App Service doesn't have a built-in email feature. For some good alternatives for sending email from your app, see this Stack Overflow discussion.

Why does my WordPress site redirect to another URL?

If you have recently migrated to Azure, WordPress might redirect to the old domain URL. This issue is caused by a setting in the MySQL database.

WordPress Buddy+ is an Azure Site Extension that you can use to update the redirection URL directly in the database. For more information about using WordPress Buddy+, see WordPress tools and MySQL migration with WordPress Buddy+.

Alternatively, if you prefer to manually update the redirection URL by using SQL queries or PHPMyAdmin, see WordPress: Redirecting to wrong URL.

How do I change my WordPress sign-in password?

If you have forgotten your WordPress sign-in password, you can use WordPress Buddy+ to update it. To reset your password, install the WordPress Buddy+ Azure Site Extension, and then complete the steps described in WordPress tools and MySQL migration with WordPress Buddy+.

I can't sign in to WordPress. How do I resolve this issue?

If you find yourself locked out of WordPress after recently installing a plugin, you might have a faulty plugin. WordPress Buddy+ is an Azure Site Extension that can help you disable plugins in WordPress. For more information, see WordPress tools and MySQL migration with WordPress Buddy+.

How do I migrate my WordPress database?

You have multiple options for migrating the MySQL database that's connected to your WordPress website:

- Developers: Use the command prompt or PHPMyAdmin
- Non-developers: Use WordPress Buddy+

How do I help make WordPress more secure?

To learn about security best practices for WordPress, see Best practices for WordPress security in Azure.

I'm trying to use PHPMyAdmin, and I see the message "Access denied." How do I resolve this issue?

You might experience this issue if the MySQL in-app feature isn't running yet in this App Service instance. To resolve the issue, try to access your website. This starts the required processes, including the MySQL in-app process. To verify that MySQL in-app is running, in Process Explorer, ensure that *mysqld.exe* is listed in the processes.

After you ensure that MySQL in-app is running, try to use PHPMyAdmin.

I get an HTTP 403 error when I try to import or export my MySQL in-app database by using PHPMyadmin. How do I resolve this issue?

If you're using an older version of Chrome, you might be experiencing a known bug. To resolve the issue, upgrade to a newer version of Chrome. Also try using a different browser, like Internet Explorer or Microsoft Edge, where the issue doesn't occur.

Contact us for help

If you have questions or need help, create a support request, or ask Azure community support. You can also submit product feedback to Azure community support.

Feedback

Was this page helpful? Yes No

Provide product feedback

How to troubleshoot temporary storage on Azure App Service

Article • 06/23/2022

If an application is reporting high resource consumption, the source of the problem might not be the site content. Slow performance can also be caused by excessive use of temporary storage.

This video demonstrates how to narrow the scope of your troubleshooting to temporary storage, and the next steps for investigating and troubleshooting performance issues.

> [!VIDEO https://www.youtube.com/embed/bk8h-VYaIXs]

Related content

For more information, see: Understanding the Azure App Service file system - projectkudu/kudu Wiki (github.com)

Contact us for help

If you have questions or need help, create a support request, or ask Azure community support. You can also submit product feedback to Azure community support.

Feedback

Was this page helpful? Yes No

Provide product feedback

How to troubleshoot instance-related issues on Azure App Service

Article • 02/01/2023

App services are hosted on Azure App Service plans that define the compute resources that you need to be able to run a web app. The compute resources, also known as instances, can occasionally become unavailable. This video helps you to identify such issues, and explains how to resolve them. From improving load balancing to moving to an entirely new set of instances, you'll be well equipped to resolve instance-related issues by applying these techniques.
https://www.youtube-nocookie.com/embed/Cg_muggc2m0

Related content

For more information about the methods that are described in this video, plus additional methods to maintain high availability for Azure App Services, see The Ultimate Guide to Running Healthy Apps in the Cloud - Azure App Service.

Contact us for help

If you have questions or need help, create a support request, or ask Azure community support. You can also submit product feedback to Azure community support.

Feedback

Was this page helpful? Yes No

Provide product feedback

Troubleshooting intermittent outbound connection errors in Azure App Service

Article • 06/28/2023

This article helps you troubleshoot intermittent connection errors and related performance issues in Azure App Service. It provides more information on, and troubleshooting methodologies for, exhaustion of source network address translation (SNAT) ports. If you require more help at any point in this article, contact the Azure experts at the MSDN Azure and the Stack Overflow forums. Alternatively, file an Azure support incident. Go to the Azure Support site and select **Get Support**.

Symptoms

Applications and Functions hosted on Azure App service may exhibit one or more of the following symptoms:

- Slow response times on all or some of the instances in a service plan.
- Intermittent 5xx or **Bad Gateway** errors
- Timeout error messages
- Couldn't connect to external endpoints (like SQLDB, Service Fabric, other App services etc.)

Cause

The major cause for intermittent connection issues is hitting a limit while making new outbound connections. The limits you can hit include:

- TCP Connections: There's a limit on the number of outbound connections that can be made. The limit on outbound connections is associated with the size of the worker used.
- SNAT ports: Outbound connections in Azure describes SNAT port restrictions and how they affect outbound connections. Azure uses source network address translation (SNAT) and Load Balancers (not exposed to customers) to communicate with public IP addresses. Each instance on Azure App service is initially given a preallocated number of **128** SNAT ports. The SNAT port limit affects opening connections to the same address and port combination. If your app creates connections to a mix of address and port combinations, you won't use up your SNAT ports. The SNAT ports are used up when you have repeated calls to the same address and port combination. Once a port has been released, the port is available

for reuse as needed. The Azure Network load balancer reclaims SNAT port from closed connections only after waiting for 4 minutes.

When applications or functions rapidly open a new connection, they can quickly exhaust their preallocated quota of the 128 ports. They're then blocked until a new SNAT port becomes available, either through dynamically allocating more SNAT ports, or through reuse of a reclaimed SNAT port. If your app runs out of SNAT ports, it will have intermittent outbound connectivity issues.

Avoiding the problem

There are a few solutions that let you avoid SNAT port limitations. They include:

- connection pools: By pooling your connections, you avoid opening new network connections for calls to the same address and port.
- service endpoints: You don't have a SNAT port restriction to the services secured with service endpoints.
- private endpoints: You don't have a SNAT port restriction to services secured with private endpoints.
- NAT gateway: With a NAT gateway, you have 64k outbound SNAT ports that are usable by the resources sending traffic through it.

To avoid the SNAT port problem, you prevent the creation of new connections repetitively to the same host and port. Connection pools are one of the more obvious ways to solve that problem.

If your destination is an Azure service that supports service endpoints, you can avoid SNAT port exhaustion issues by using regional VNet Integration and service endpoints or private endpoints. When you use regional VNet Integration and place service endpoints on the integration subnet, your app outbound traffic to those services won't have outbound SNAT port restrictions. Likewise, if you use regional VNet Integration and private endpoints, you won't have any outbound SNAT port issues to that destination.

If your destination is an external endpoint outside of Azure, using a NAT gateway gives you 64k outbound SNAT ports. It also gives you a dedicated outbound address that you don't share with anybody.

If possible, improve your code to use connection pools and avoid the entire situation. It isn't always possible to change code fast enough to mitigate this situation. For the cases where you can't change your code in time, take advantage of the other solutions. The best solution to the problem is to combine all of the solutions as best you can. Try to

use service endpoints and private endpoints to Azure services and the NAT gateway for the rest.

General strategies for mitigating SNAT port exhaustion are discussed in the Problem-solving section of the **Outbound connections of Azure** documentation. Of these strategies, the following are applicable to apps and functions hosted on Azure App service.

Modify the application to use connection pooling

- For pooling HTTP connections, review Pool HTTP connections with HttpClientFactory.
- For information on SQL Server connection pooling, review SQL Server Connection Pooling (ADO.NET).

Here's a collection of links for implementing Connection pooling by different solution stack.

Node

By default, connections for NodeJS aren't kept alive. Below are the popular databases and packages for connection pooling which contain examples for how to implement them.

- MySQL
- MongoDB
- PostgreSQL
- SQL Server

HTTP Keep-alive

- agentkeepalive
- Node.js v13.9.0 Documentation

Java

Below are the popular libraries used for JDBC connection pooling which contain examples for how to implement them: JDBC Connection Pooling.

- Tomcat 8
- C3p0
- HikariCP
- Apache DBCP

HTTP Connection Pooling

- Apache Connection Management
- Class PoolingHttpClientConnectionManager

PHP

Although PHP doesn't support connection pooling, you can try using persistent database connections to your back-end server.

- MySQL server
  - MySQLi connections for newer versions
  - mysql_pconnect for older versions of PHP
- Other data Sources
  - PHP Connection Management

Python

Below are the popular databases and modules for connection pooling which contain examples for how to implement them.

- MySQL
- MariaDB
- PostgreSQL
- Pyodbc
- SQLAlchemy

HTTP Connection Pooling

- Keep-alive and HTTP connection pooling are enabled by default in Requests module.
- Urllib3

Modify the application to reuse connections

- For more pointers and examples on managing connections in Azure functions, review Manage connections in Azure Functions.

Modify the application to use less aggressive retry logic

- For more guidance and examples, review Retry pattern.

Use keepalives to reset the outbound idle timeout

- For implementing keepalives for Node.js apps, review My node application is making excessive outbound calls.

More guidance specific to App Service:

- A load test should simulate real world data in a steady feeding speed. Testing apps and functions under real world stress can identify and resolve SNAT port exhaustion issues ahead of time.
- Ensure that the back-end services can return responses quickly. For troubleshooting performance issues with Azure SQL Database, review Troubleshoot Azure SQL Database performance issues with Intelligent Insights.
- Scale out the App Service plan to more instances. For more information on scaling, see Scale an app in Azure App Service. Each worker instance in an app service plan is allocated a number of SNAT ports. If you spread your usage across more instances, you might get the SNAT port usage per instance below the recommended limit of 100 outbound connections, per unique remote endpoint.
- Consider moving to App Service Environment (ASE), where you're allotted a single outbound IP address, and the limits for connections and SNAT ports are higher. In an ASE, the number of SNAT ports per instance is based on the Azure load balancer preallocation table. For example, an ASE with 1-50 worker instances has 1024 preallocated ports per instance, while an ASE with 51-100 worker instances has 512 preallocated ports per instance.

Avoiding the outbound TCP limits is easier to solve, as the limits are set by the size of your worker. You can see the limits in Sandbox Cross VM Numerical Limits - TCP Connections

Limit name	Description	Small (A1)	Medium (A2)	Large (A3)	Isolated tier (ASE)
Connections	Number of connections across entire VM	1920	3968	8064	16,000

To avoid outbound TCP limits, you can either increase the size of your workers, or scale out horizontally.

Troubleshooting

Knowing the two types of outbound connection limits, and what your app does, should make it easier to troubleshoot. If you know that your app makes many calls to the same

storage account, you might suspect a SNAT limit. If your app creates a great many calls to endpoints all over the internet, you would suspect you're reaching the VM limit.

If you don't know the application behavior enough to determine the cause quickly, there are some tools and techniques available in App Service to help with that determination.

Find SNAT port allocation information

You can use App Service Diagnostics to find SNAT port allocation information, and observe the SNAT ports allocation metric of an App Service site. To find SNAT port allocation information, follow the following steps:

1. To access App Service diagnostics, navigate to your App Service web app or App Service Environment in the Azure portal. In the left navigation, select **Diagnose and solve problems**.
2. Select Availability and Performance Category
3. Select SNAT Port Exhaustion tile in the list of available tiles under the category. The practice is to keep it below 128. If you do need it, you can still open a support ticket, and the support engineer will get the metric from back-end for you.

Since SNAT port usage isn't available as a metric, it isn't possible to either autoscale based on SNAT port usage, or to configure auto scale based on SNAT ports allocation metric.

TCP Connections and SNAT Ports

TCP connections and SNAT ports aren't directly related. A TCP connections usage detector is included in the Diagnose and Solve Problems management page of any App Service app. Search for the phrase "TCP connections" to find it.

- The SNAT Ports are only used for external network flows, while the total TCP Connections includes local loopback connections.
- A SNAT port can be shared by different flows, if the flows are different in either protocol, IP address or port. The TCP Connections metric counts every TCP connection.
- The TCP connections limit happens at the worker instance level. The Azure Network outbound load balancing doesn't use the TCP Connections metric for SNAT port limiting.
- The TCP connections limits are described in Sandbox Cross VM Numerical Limits - TCP Connections
- Existing TCP sessions fail when new outbound TCP sessions are added from Azure App Service source port. You can either use a single IP or reconfigure backend pool

members to avoid conflicts.

Limit name	Description	Small (A1)	Medium (A2)	Large (A3)	Isolated tier (ASE)
Connections	Number of connections across entire VM	1920	3968	8064	16,000

WebJobs and Database connections

If SNAT ports are exhausted, and WebJobs are unable to connect to SQL Database, there's no metric to show how many connections are opened by each individual web application process. To find the problematic WebJob, move several WebJobs out to another App Service plan to see if the situation improves, or if an issue remains in one of the plans. Repeat the process until you find the problematic WebJob.

Additional information

- SNAT with App Service
- Troubleshoot slow app performance issues in Azure App Service

Troubleshoot an app in Azure App Service using Visual Studio

Article • 06/05/2022

Overview

This tutorial shows how to use Visual Studio tools to help debug an app in App Service, by running in debug mode remotely or by viewing application logs and web server logs.

You'll learn:

- Which app management functions are available in Visual Studio.
- How to use Visual Studio remote view to make quick changes in a remote app.
- How to run debug mode remotely while a project is running in Azure, both for an app and for a WebJob.
- How to create application trace logs and view them while the application is creating them.
- How to view web server logs, including detailed error messages and failed request tracing.
- How to send diagnostic logs to an Azure Storage account and view them there.

If you have Visual Studio Ultimate, you can also use IntelliTrace for debugging. IntelliTrace is not covered in this tutorial.

Prerequisites

This tutorial works with the development environment, web project, and App Service app that you set up in Create an ASP.NET app in Azure App Service. For the WebJobs sections, you'll need the application that you create in Get Started with the Azure WebJobs SDK.

The code samples shown in this tutorial are for a C# MVC web application, but the troubleshooting procedures are the same for Visual Basic and Web Forms applications.

The tutorial assumes you're using Visual Studio 2019.

The streaming logs feature only works for applications that target .NET Framework 4 or later.

App configuration and management

Visual Studio provides access to a subset of the app management functions and configuration settings available in the Azure portal. In this section, you'll see what's available by using **Server Explorer**. To see the latest Azure integration features, try out **Cloud Explorer** also. You can open both windows from the **View** menu.

1. If you aren't already signed in to Azure in Visual Studio, right-click **Azure** and select Connect to **Microsoft Azure Subscription** in **Server Explorer**.

   An alternative is to install a management certificate that enables access to your account. If you choose to install a certificate, right-click the **Azure** node in **Server Explorer**, and then select **Manage and Filter Subscriptions** in the context menu. In the **Manage Microsoft Azure Subscriptions** dialog box, click the **Certificates** tab, and then click **Import**. Follow the directions to download and then import a subscription file (also called a *.publishsettings* file) for your Azure account.

   > **Note**
   >
   > If you download a subscription file, save it to a folder outside your source code directories (for example, in the Downloads folder), and then delete it once the import has completed. A malicious user who gains access to the subscription file can edit, create, and delete your Azure services.

   For more information about connecting to Azure resources from Visual Studio, see Assign Azure roles using the Azure portal.

2. In **Server Explorer**, expand **Azure** and expand **App Service**.

3. Expand the resource group that includes the app that you created in Create an ASP.NET app in Azure App Service, and then right-click the app node and click **View Settings**.

The **Azure Web App** tab appears, and you can see there the app management and configuration tasks that are available in Visual Studio.

In this tutorial, you'll use the logging and tracing drop-downs. You'll also use remote debugging but you'll use a different method to enable it.

For information about the App Settings and Connection Strings boxes in this window, see Azure App Service: How Application Strings and Connection Strings Work.

If you want to perform an app management task that can't be done in this window, click **Open in Management Portal** to open a browser window to the Azure portal.

Access app files in Server Explorer

You typically deploy a web project with the `customErrors` flag in the Web.config file set to `On` or `RemoteOnly`, which means you don't get a helpful error message when something goes wrong. For many errors, all you get is a page like one of the following ones:

Server Error in '/' Application:

An error occurred:

The website cannot display the page

Frequently the easiest way to find the cause of the error is to enable detailed error messages, which the first of the preceding screenshots explains how to do. That requires a change in the deployed Web.config file. You could edit the *Web.config* file in the project and redeploy the project, or create a Web.config transform and deploy a debug build, but there's a quicker way: in **Solution Explorer**, you can directly view and edit files in the remote app by using the *remote view* feature.

1. In **Server Explorer**, expand **Azure**, expand **App Service**, expand the resource group that your app is located in, and then expand the node for your app.

   You see nodes that give you access to the app's content files and log files.

2. Expand the **Files** node, and double-click the *Web.config* file.

Visual Studio opens the Web.config file from the remote app and shows [Remote] next to the file name in the title bar.

3. Add the following line to the `system.web` element:

```
<customErrors mode="Off"></customErrors>
```

4. Refresh the browser that is showing the unhelpful error message, and now you get a detailed error message, such as the following example:

(The error shown was created by adding the line shown in red to *Views\Home\Index.cshtml*.)

Editing the Web.config file is only one example of scenarios in which the ability to read and edit files on your App Service app make troubleshooting easier.

Remote debugging apps

If the detailed error message doesn't provide enough information, and you can't re-create the error locally, another way to troubleshoot is to run in debug mode remotely. You can set breakpoints, manipulate memory directly, step through code, and even change the code path.

Remote debugging does not work in Express editions of Visual Studio.

This section shows how to debug remotely using the project you create in Create an ASP.NET app in Azure App Service.

1. Open the web project that you created in Create an ASP.NET app in Azure App Service.

2. Open *Controllers\HomeController.cs*.

3. Delete the `About()` method and insert the following code in its place.

```
C#

public ActionResult About()
{
    string currentTime = DateTime.Now.ToLongTimeString();
    ViewBag.Message = "The current time is " + currentTime;
    return View();
}
```

4. Set a breakpoint on the `ViewBag.Message` line.

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

6. In the **Profile** drop-down list, select the same profile that you used in Create an ASP.NET app in Azure App Service. Then, click Settings.

7. In the **Publish** dialog, click the **Settings** tab, and then change **Configuration** to **Debug**, and then click **Save**.

8. Click **Publish**. After deployment finishes and your browser opens to the Azure URL of your app, close the browser.

9. In **Server Explorer**, right-click your app, and then click **Attach Debugger**.

The browser automatically opens to your home page running in Azure. You might have to wait 20 seconds or so while Azure sets up the server for debugging. This delay only happens the first time you run in debug mode on an app in a 48-hour period. When you start debugging again in the same period, there isn't a delay.

> ⓘ **Note**
>
> If you have any trouble starting the debugger, try to do it by using **Cloud Explorer** instead of **Server Explorer**.

10. Click **About** in the menu.

    Visual Studio stops on the breakpoint, and the code is running in Azure, not on your local computer.

11. Hover over the `currentTime` variable to see the time value.

The time you see is the Azure server time, which may be in a different time zone than your local computer.

12. Enter a new value for the `currentTime` variable, such as "Now running in Azure".

13. Press F5 to continue running.

    The About page running in Azure displays the new value that you entered into the currentTime variable.

Remote debugging WebJobs

This section shows how to debug remotely using the project and app you create in Get Started with the Azure WebJobs SDK.

The features shown in this section are available only in Visual Studio 2013 with Update 4 or later.

Remote debugging only works with continuous WebJobs. Scheduled and on-demand WebJobs don't support debugging.

1. Open the web project that you created in Get Started with the Azure WebJobs SDK.

2. In the ContosoAdsWebJob project, open *Functions.cs*.

3. Set a breakpoint on the first statement in the `GenerateThumbnail` method.

```
public class Functions
{
    0 references
    public static void GenerateThumbnail(
    [QueueTrigger("thumbnailrequest")] BlobInformation blobInfo,
    [Blob("images/{BlobName}", FileAccess.Read)] Stream input,
    [Blob("images/{BlobNameWithoutExtension}_thumbnail.jpg")] CloudBlockBlob outputBlob)
    {
        using (Stream output = outputBlob.OpenWrite())
```

4. In **Solution Explorer**, right-click the web project (not the WebJob project), and click **Publish**.

5. In the **Profile** drop-down list, select the same profile that you used in Get Started with the Azure WebJobs SDK.

6. Click the **Settings** tab, and change **Configuration** to **Debug**, and then click **Publish**.

   Visual Studio deploys the web and WebJob projects, and your browser opens to the Azure URL of your app.

7. In **Server Explorer**, expand **Azure** > **App Service** > **your resource group** > **your app** > **WebJobs** > **Continuous**, and then right-click **ContosoAdsWebJob**.

8. Click **Attach Debugger**.

   The browser automatically opens to your home page running in Azure. You might have to wait 20 seconds or so while Azure sets up the server for debugging. This delay only happens the first time you run in debug mode on an app in a 48-hour period. When you start debugging again in the same period, there isn't a delay.

9. In the web browser that is opened to the Contoso Ads home page, create a new ad.

   Creating an ad causes a queue message to be created, which is picked up by the WebJob and processed. When the WebJobs SDK calls the function to process the queue message, the code hits your breakpoint.

10. When the debugger breaks at your breakpoint, you can examine and change variable values while the program is running the cloud. In the following illustration, the debugger shows the contents of the blobInfo object that was passed to the `GenerateThumbnail` method.

```
public static void GenerateThumbnail(
[QueueTrigger("thumbnailrequest")] BlobInformation blobInfo,
[Blob("images/{BlobName}", FileAccess.Read)] Stream inp
```

AdId	3
BlobName	"229e33e7-b964-4ee4-b258-7a5cb5f52efd.jpg"
BlobNameWithoutExtension	"229e33e7-b964-4ee4-b258-7a5cb5f52efd"
BlobUri	{https://contosoads.blob.core.windows.net/images/229e33e7-b964-4ee4-b258-7a5cb5f52efd.jpg...

```
        ConvertImageToThumbnailJPG(input, output);
        outputBlob.Properties.ContentType = "image/jpeg";
    }
```

11. Press F5 to continue running.

    The `GenerateThumbnail` method finishes creating the thumbnail.

12. In the browser, refresh the Index page and you see the thumbnail.

13. In Visual Studio, press SHIFT+F5 to stop debugging.

14. In **Server Explorer**, right-click the ContosoAdsWebJob node and click **View Dashboard**.

15. Sign in with your Azure credentials, and then click the WebJob name to go to the page for your WebJob.

    The Dashboard shows that the `GenerateThumbnail` function executed recently.

    (The next time you click **View Dashboard**, you don't have to sign in, and the browser goes directly to the page for your WebJob.)

16. Click the function name to see details about the function execution.

If your function wrote logs, you could click **ToggleOutput** to see them.

Notes about remote debugging

- Running in debug mode in production is not recommended. If your production app is not scaled out to multiple server instances, debugging prevents the web server from responding to other requests. If you do have multiple web server instances, when you attach to the debugger, you get a random instance, and you have no way to ensure that subsequent browser requests go to the same instance. Also, you typically don't deploy a debug build to production, and compiler

optimizations for release builds might make it impossible to show what is happening line by line in your source code. For troubleshooting production problems, your best resource is application tracing and web server logs.

- Avoid long stops at breakpoints when remote debugging. Azure treats a process that is stopped for longer than a few minutes as an unresponsive process, and shuts it down.

- While you're debugging, the server is sending data to Visual Studio, which could affect bandwidth charges. For information about bandwidth rates, see Azure Pricing.

- Make sure that the `debug` attribute of the `compilation` element in the *Web.config* file is set to true. It is set to true by default when you publish a debug build configuration.

  XML

  ```xml
  <system.web>
    <compilation debug="true" targetFramework="4.5" />
    <httpRuntime targetFramework="4.5" />
  </system.web>
  ```

- If you find that the debugger doesn't step into the code that you want to debug, you might have to change the Just My Code setting. For more information, see Specify whether to debug only user code using Just My Code in Visual Studio.

- A timer starts on the server when you enable the remote debugging feature, and after 48 hours the feature is automatically turned off. This 48-hour limit is done for security and performance reasons. You can easily turn the feature back on as many times as you like. We recommend leaving it disabled when you are not actively debugging.

- You can manually attach the debugger to any process, not only the app process (w3wp.exe). For more information about how to use debug mode in Visual Studio, see Debugging in Visual Studio.

Diagnostic logs overview

An ASP.NET application that runs in an App Service app can create the following kinds of logs:

- **Application tracing logs**
  The application creates these logs by calling methods of the System.Diagnostics.Trace class.
- **Web server logs**
  The web server creates a log entry for every HTTP request to the app.
- **Detailed error message logs**
  The web server creates an HTML page with some additional information for failed HTTP requests (requests that result in status code 400 or greater).
- **Failed request tracing logs**
  The web server creates an XML file with detailed tracing information for failed HTTP requests. The web server also provides an XSL file to format the XML in a browser.

Logging affects app performance, so Azure gives you the ability to enable or disable each type of log as needed. For application logs, you can specify that only logs above a certain severity level should be written. When you create a new app, by default all logging is disabled.

Logs are written to files in a *LogFiles* folder in the file system of your app and are accessible via FTP. Web server logs and application logs can also be written to an Azure Storage account. You can retain a greater volume of logs in a storage account than is possible in the file system. You're limited to a maximum of 100 megabytes of logs when you use the file system. (File system logs are only for short-term retention. Azure deletes old log files to make room for new ones after the limit is reached.)

Create and view application trace logs

In this section, you do the following tasks:

- Add tracing statements to the web project that you created in Get started with Azure and ASP.NET.
- View the logs when you run the project locally.
- View the logs as they are generated by the application running in Azure.

For information about how to create application logs in WebJobs, see How to work with Azure queue storage using the WebJobs SDK - How to write logs. The following instructions for viewing logs and controlling how they're stored in Azure apply also to application logs created by WebJobs.

Add tracing statements to the application

1. Open *Controllers\HomeController.cs*, and replace the `Index`, `About`, and `Contact` methods with the following code in order to add `Trace` statements and a `using` statement for `System.Diagnostics`:

```
C#

public ActionResult Index()
{
    Trace.WriteLine("Entering Index method");
    ViewBag.Message = "Modify this template to jump-start your ASP.NET
MVC application.";
    Trace.TraceInformation("Displaying the Index page at " +
DateTime.Now.ToLongTimeString());
    Trace.WriteLine("Leaving Index method");
    return View();
}

public ActionResult About()
{
    Trace.WriteLine("Entering About method");
    ViewBag.Message = "Your app description page.";
    Trace.TraceWarning("Transient error on the About page at " +
DateTime.Now.ToShortTimeString());
    Trace.WriteLine("Leaving About method");
    return View();
}

public ActionResult Contact()
{
    Trace.WriteLine("Entering Contact method");
    ViewBag.Message = "Your contact page.";
    Trace.TraceError("Fatal error on the Contact page at " +
DateTime.Now.ToLongTimeString());
    Trace.WriteLine("Leaving Contact method");
    return View();
}
```

2. Add a `using System.Diagnostics;` statement to the top of the file.

View the tracing output locally

1. Press F5 to run the application in debug mode.

   The default trace listener writes all trace output to the **Output** window, along with other Debug output. The following illustration shows the output from the trace statements that you added to the `Index` method.

The following steps show how to view trace output in a web page, without compiling in debug mode.

2. Open the application Web.config file (the one located in the project folder) and add a `<system.diagnostics>` element at the end of the file just before the closing `</configuration>` element:

XML

```xml
<system.diagnostics>
<trace>
  <listeners>
    <add name="WebPageTraceListener"
        type="System.Web.WebPageTraceListener,
        System.Web,
        Version=4.0.0.0,
        Culture=neutral,
        PublicKeyToken=b03f5f7f11d50a3a" />
  </listeners>
</trace>
</system.diagnostics>
```

The `WebPageTraceListener` lets you view trace output by browsing to `/trace.axd`.

1. Add a trace element under `<system.web>` in the Web.config file, such as the following example:

XML

```xml
<trace enabled="true" writeToDiagnosticsTrace="true" mostRecent="true"
pageOutput="false" />
```

2. Press CTRL+F5 to run the application.

3. In the address bar of the browser window, add *trace.axd* to the URL, and then press Enter (the URL is similar to `http://localhost:53370/trace.axd`).

4. On the **Application Trace** page, click **View Details** on the first line (not the BrowserLink line).

The **Request Details** page appears, and in the **Trace Information** section you see the output from the trace statements that you added to the `Index` method.

By default, `trace.axd` is only available locally. If you wanted to make it available from a remote app, you could add `localOnly="false"` to the `trace` element in the *Web.config* file, as shown in the following example:

```xml
<trace enabled="true" writeToDiagnosticsTrace="true" localOnly="false"
mostRecent="true" pageOutput="false" />
```

However, enabling `trace.axd` in a production app is not recommended for security reasons. In the following sections, you'll see an easier way to read tracing logs in an App Service app.

View the tracing output in Azure

1. In **Solution Explorer**, right-click the web project and click **Publish**.

2. In the **Publish Web** dialog box, click **Publish**.

   After Visual Studio publishes your update, it opens a browser window to your home page (assuming you didn't clear **Destination URL** on the **Connection** tab).

3. In **Server Explorer**, right-click your app and select **View Streaming Logs**.

   The **Output** window shows that you are connected to the log-streaming service, and adds a notification line each minute that goes by without a log to display.

4. In the browser window that shows your application home page, click **Contact**.

Within a few seconds, the output from the error-level trace you added to the `Contact` method appears in the **Output** window.

Visual Studio is only showing error-level traces because that is the default setting when you enable the log monitoring service. When you create a new App Service app, all logging is disabled by default, as you saw when you opened the settings page earlier:

However, when you selected **View Streaming Logs**, Visual Studio automatically changed **Application Logging(File System)** to **Error**, which means error-level logs get reported. In order to see all of your tracing logs, you can change this setting to **Verbose**. When you select a severity level lower than error, all logs for higher severity levels are also reported. So when you select verbose, you also see information, warning, and error logs.

5. In **Server Explorer**, right-click the app, and then click **View Settings** as you did earlier.

6. Change **Application Logging (File System)** to **Verbose**, and then click **Save**.

7. In the browser window that is now showing your **Contact** page, click **Home**, then click **About**, and then click **Contact**.

   Within a few seconds, the **Output** window shows all of your tracing output.

   In this section, you enabled and disabled logging by using app settings. You can also enable and disable trace listeners by modifying the Web.config file. However, modifying the Web.config file causes the app domain to recycle, while enabling logging via the app configuration doesn't do that. If the problem takes a long time to reproduce, or is intermittent, recycling the app domain might "fix" it and force you to wait until it happens again. Enabling diagnostics in Azure lets you start capturing error information immediately without recycling the app domain.

Output window features

The **Microsoft Azure Logs** tab of the **Output** Window has several buttons and a text box:

These perform the following functions:

- Clear the **Output** window.
- Enable or disable word wrap.
- Start or stop monitoring logs.
- Specify which logs to monitor.
- Download logs.
- Filter logs based on a search string or a regular expression.
- Close the **Output** window.

If you enter a search string or regular expression, Visual Studio filters logging information at the client. That means you can enter the criteria after the logs are displayed in the **Output** window and you can change filtering criteria without having to regenerate the logs.

View web server logs

Web server logs record all HTTP activity for the app. In order to see them in the **Output** window, you must enable them for the app and tell Visual Studio that you want to monitor them.

1. In the **Azure Web App Configuration** tab that you opened from **Server Explorer**, change Web Server Logging to **On**, and then click **Save**.

2. In the **Output** Window, click the **Specify which Microsoft Azure logs to monitor** button.

3. In the **Microsoft Azure Logging Options** dialog box, select **Web server logs**, and then click **OK**.

4. In the browser window that shows the app, click **Home**, then click **About**, and then click **Contact**.

   The application logs generally appear first, followed by the web server logs. You might have to wait a while for the logs to appear.

```
Output
Show output from: Windows Azure Logs - example1z
Connecting to Application logs ...
2013-07-08T19:50:34  Welcome, you are now connected to log-streaming service.
Application: 2013-07-08T19:51:34  No new trace in the past 1 min(s).
Application: 2013-07-08T19:52:34  No new trace in the past 2 min(s).
Connecting to Web server logs ...
2013-07-08T19:52:36  Welcome, you are now connected to log-streaming service.
Application: 2013-07-08T19:53:34  No new trace in the past 3 min(s).
Web server: 2013-07-08T19:53:36  No new trace in the past 1 min(s).
Application: 2013-07-08T19:54:34  No new trace in the past 4 min(s).
Web server: 2013-07-08T19:54:36  No new trace in the past 2 min(s).
Application: 2013-07-08T19:55:06  PID[3268] Information DotNetOpenAuth.Core,
Version=4.0.0.0, Culture=neutral, PublicKeyToken=2780ccd10d57b246 (official)
Application: 2013-07-08T19:55:06  PID[3268] Information Reporting will use
isolated storage with scope: Domain, Assembly, Machine
Application: 2013-07-08T19:55:10  PID[3268] Verbose     Entering Contact method
Application: 2013-07-08T19:55:10  PID[3268] Verbose     Entering Index method
Application: 2013-07-08T19:55:10  PID[3268] Verbose     Entering About method
Application: 2013-07-08T19:55:10  PID[3268] Warning     Transient error on the
About page at 7:55 PM
Application: 2013-07-08T19:55:10  PID[3268] Verbose     Leaving About method
Application: 2013-07-08T19:55:10  PID[3268] Error       Fatal error on the
Contact page at 7:55:10 PM
Application: 2013-07-08T19:55:10  PID[3268] Verbose     Leaving Contact method
Application: 2013-07-08T19:55:10  PID[3268] Information Displaying the Index page
 at 7:55:10 PM
Application: 2013-07-08T19:55:10  PID[3268] Verbose     Leaving Index method
Web server: 2013-07-08T19:55:36  No new trace in the past 3 min(s).
Web server: 2013-07-08 19:55:13 EXAMPLE1Z GET /Home/Contact X-ARR-LOG-
ID=1f7e79df-bfbc-40c6-9573-cfcb04612f93 80 - 131.107.0.112 Mozilla/5.0
+(compatible;+MSIE+10.0;+Windows+NT+6.2;+WOW64;+Trident/6.0)
ARRAffinity=948219045391acdeaa3c34903baa60b2db150a12c5d3f2589134a811deb4a38c;
+WAWebSiteSID=e0d53e3f499942b699075cd5f0881106 http://
example1z.azurewebsites.net/Home/Contact example1z.azurewebsites.net 200 0 0 3307
 623 7549
Web server: 2013-07-08 19:55:13 EXAMPLE1Z GET /Home/About X-ARR-LOG-
ID=8a35806d-8e92-479e-bad7-55d001c2fab2 80 - 131.107.0.112 Mozilla/5.0
+(compatible;+MSIE+10.0;+Windows+NT+6.2;+WOW64;+Trident/6.0)
ARRAffinity=948219045391acdeaa3c34903baa60b2db150a12c5d3f2589134a811deb4a38c;
+WAWebSiteSID=e0d53e3f499942b699075cd5f0881106 http://
example1z.azurewebsites.net/Home/Contact example1z.azurewebsites.net 200 0 0 2845
 619 8480
Web server: 2013-07-08 19:55:13 EXAMPLE1Z GET / X-ARR-LOG-ID=1e9c3f42-7f32-4a23-
a15b-037f2b69f870 80 - 131.107.0.112 Mozilla/5.0+(compatible;+MSIE+10.0;+Windows
+NT+6.2;+WOW64;+Trident/6.0)
ARRAffinity=948219045391acdeaa3c34903baa60b2db150a12c5d3f2589134a811deb4a38c;
+WAWebSiteSID=e0d53e3f499942b699075cd5f0881106 http://
example1z.azurewebsites.net/Home/Contact example1z.azurewebsites.net 200 0 0 4137
 599 9261
Application: 2013-07-08T19:56:34  No new trace in the past 1 min(s).
Application: 2013-07-08T19:57:34  No new trace in the past 2 min(s).
Web server: 2013-07-08T19:57:36  No new trace in the past 1 min(s).
Web Publish Activity  Output
```

By default, when you first enable web server logs by using Visual Studio, Azure writes the logs to the file system. As an alternative, you can use the Azure portal to specify that web server logs should be written to a blob container in a storage account.

If you use the portal to enable web server logging to an Azure storage account, and then disable logging in Visual Studio, when you re-enable logging in Visual Studio your storage account settings are restored.

View detailed error message logs

Detailed error logs provide some additional information about HTTP requests that result in error response codes (400 or above). In order to see them in the **Output** window, you have to enable them for the app and tell Visual Studio that you want to monitor them.

1. In the **Azure Web App Configuration** tab that you opened from **Server Explorer**, change **Detailed Error Messages** to **On**, and then click **Save**.

2. In the **Output** Window, click the **Specify which Microsoft Azure logs to monitor** button.

3. In the **Microsoft Azure Logging Options** dialog box, click **All logs**, and then click **OK**.

4. In the address bar of the browser window, add an extra character to the URL to cause a 404 error (for example, `http://localhost:53370/Home/Contactx`), and press Enter.

After several seconds, the detailed error log appears in the Visual Studio **Output** window.

Control+click the link to see the log output formatted in a browser:

Download file system logs

Any logs that you can monitor in the **Output** window can also be downloaded as a *.zip* file.

1. In the **Output** window, click **Download Streaming Logs**.

File Explorer opens to your *Downloads* folder with the downloaded file selected.

2. Extract the *.zip* file, and you see the following folder structure:

- Application tracing logs are in *.txt* files in the *LogFiles\Application* folder.
- Web server logs are in *.log* files in the *LogFiles\http\RawLogs* folder. You can use a tool such as Log Parser to view and manipulate these files.
- Detailed error message logs are in *.html* files in the *LogFiles\DetailedErrors* folder.

  (The *deployments* folder is for files created by source control publishing; it doesn't have anything related to Visual Studio publishing. The *Git* folder is for traces related to source control publishing and the log file streaming service.)

View failed request tracing logs

Failed request tracing logs are useful when you need to understand the details of how IIS is handling an HTTP request, in scenarios such as URL rewriting or authentication problems.

App Service apps use the same failed request tracing functionality that has been available with IIS 7.0 and later. You don't have access to the IIS settings that configure

which errors get logged, however. When you enable failed request tracing, all errors are captured.

You can enable failed request tracing by using Visual Studio, but you can't view them in Visual Studio. These logs are XML files. The streaming log service only monitors files that are deemed readable in plain text mode: *.txt*, *.html*, and *.log* files.

You can view failed request tracing logs in a browser directly via FTP or locally after using an FTP tool to download them to your local computer. In this section, you'll view them in a browser directly.

1. In the **Configuration** tab of the **Azure Web App** window that you opened from **Server Explorer**, change **Failed Request Tracing** to **On**, and then click **Save**.

2. In the address bar of the browser window that shows the app, add an extra character to the URL and click Enter to cause a 404 error.

   This causes a failed request tracing log to be created, and the following steps show how to view or download the log.

3. In Visual Studio, in the **Configuration** tab of the **Azure Web App** window, click **Open in Management Portal**.

4. In the Azure portal **Settings** page for your app, click **Deployment credentials**, and then enter a new user name and password.

> **Note**
>
> When you log in, you have to use the full user name with the app name prefixed to it. For example, if you enter "myid" as a user name and the site is "myexample", you log in as "myexample\myid".

5. In a new browser window, go to the URL that is shown under **FTP hostname** or **FTPS hostname** in the **Overview** page for your app.

6. Sign in using the FTP credentials that you created earlier (including the app name prefix for the user name).

   The browser shows the root folder of the app.

7. Open the *LogFiles* folder.

8. Open the folder that is named W3SVC plus a numeric value.

The folder contains XML files for any errors that have been logged after you enabled failed request tracing, and an XSL file that a browser can use to format the XML.

9. Click the XML file for the failed request that you want to see tracing information for.

   The following illustration shows part of the tracing information for a sample error.

http://example1z.azurewe

ftp://waws-prod-bay-003.ftp.azurewebsites.windows.net/Log

Request Diagnostics for GET http://example1z.azurewebsites.net:80/Home/Contactx

– Request Summary

Site	1261305712
Process	3268
Failure Reason	STATUS_CODE
Trigger Status	404
Final Status	404
Time Taken	5125 msec

Url	http://example1z.azurewebsites.net:80/Home/Contactx
App Pool	example1z
Authentication	anonymous
User from token	IIS APPPOOL\example1z
Activity ID	{00000000-0000-0000-7369-0180000000F1}

– Errors & Warnings

No.↓	Severity	Event	Module Name
191. view trace	Warning	– MODULE_SET_RESPONSE_ERROR_STATUS	ManagedPipelineHa

ModuleName	ManagedPipelineHandler
Notification	EXECUTE_REQUEST_HANDLER
HttpStatus	404
HttpReason	Not Found
HttpSubStatus	0
ErrorCode	The operation completed successfully. (0x0)
ConfigExceptionInfo	

See all events for the request

No.	EventName	Details	Time
1.	GENERAL_REQUEST_START	SiteId="1261305712", AppPoolId="example1z", ConnId="1610705266", RawConnId="0", RequestURL="http://example1z.azurewebsites.net:80/Home/Contactx", RequestVerb="GET"	21:05:24.691
2.	PRE_BEGIN_REQUEST_START	ModuleName="FailedRequestsTracingModule"	21:05:24.722
3.	PRE_BEGIN_REQUEST_END	ModuleName="FailedRequestsTracingModule", NotificationStatus="NOTIFICATION_CONTINUE"	21:05:24.722
4.	PRE_BEGIN_REQUEST_START	ModuleName="RequestMonitorModule"	21:05:24.722
5.	PRE_BEGIN_REQUEST_END	ModuleName="RequestMonitorModule", NotificationStatus="NOTIFICATION_CONTINUE"	21:05:24.722
6.	PRE_BEGIN_REQUEST_START	ModuleName="IsapiFilterModule"	21:05:24.722
7.	FILTER_PREPROC_HEADERS_START		21:05:24.722
8	FILTER_START	FilterName="D:\Windows\	21:05:24.722

Next Steps

You've seen how Visual Studio makes it easy to view logs created by an App Service app. The following sections provide links to more resources on related topics:

- App Service troubleshooting
- Debugging in Visual Studio
- Remote debugging in Azure
- Tracing in ASP.NET applications
- Analyzing web server logs
- Analyzing failed request tracing logs
- Debugging Cloud Services

App Service troubleshooting

For more information about troubleshooting apps in Azure App Service, see the following resources:

- How to monitor apps
- Investigating Memory Leaks in Azure App Service with Visual Studio 2013. Microsoft ALM blog post about Visual Studio features for analyzing managed memory issues.
- Azure App Service online tools you should know about. Blog post by Amit Apple.

For help with a specific troubleshooting question, start a thread in one of the following forums:

- The Azure forum on the ASP.NET site.
- The Azure forum on Microsoft Q&A.
- StackOverflow.com.

Debugging in Visual Studio

For more information about how to use debug mode in Visual Studio, see Debugging in Visual Studio and Debugging Tips with Visual Studio 2010.

Remote debugging in Azure

For more information about remote debugging for App Service apps and WebJobs, see the following resources:

- Introduction to Remote Debugging Azure App Service.

- Introduction to Remote Debugging Azure App Service part 2 - Inside Remote debugging
- Introduction to Remote Debugging on Azure App Service part 3 - Multi-Instance environment and GIT
- WebJobs Debugging (video)

If your app uses an Azure Web API or Mobile Services back-end and you need to debug that, see Debugging .NET Backend in Visual Studio.

Tracing in ASP.NET applications

There are no thorough and up-to-date introductions to ASP.NET tracing available on the Internet. The best you can do is get started with old introductory materials written for Web Forms because MVC didn't exist yet, and supplement that with newer blog posts that focus on specific issues. Some good places to start are the following resources:

- Monitoring and Telemetry (Building Real-World Cloud Apps with Azure).
  E-book chapter with recommendations for tracing in Azure cloud applications.

- ASP.NET Tracing
  Old but still a good resource for a basic introduction to the subject.

- Trace Listeners
  Information about trace listeners but doesn't mention the WebPageTraceListener.

- Walkthrough: Integrating ASP.NET Tracing with System.Diagnostics Tracing
  This article is also old, but includes some additional information that the introductory article doesn't cover.

- Tracing in ASP.NET MVC Razor Views
  Besides tracing in Razor views, the post also explains how to create an error filter in order to log all unhandled exceptions in an MVC application. For information about how to log all unhandled exceptions in a Web Forms application, see the Global.asax example in Complete Example for Error Handlers on MSDN. In either MVC or Web Forms, if you want to log certain exceptions but let the default framework handling take effect for them, you can catch and rethrow as in the following example:

  C#

  ```
  try
  {
     // Your code that might cause an exception to be thrown.
  }
  ```

```
catch (Exception ex)
{
    Trace.TraceError("Exception: " + ex.ToString());
    throw;
}
```

- Streaming Diagnostics Trace Logging from the Azure Command Line (plus Glimpse!)
  How to use the command line to do what this tutorial shows how to do in Visual Studio. Glimpse is a tool for debugging ASP.NET applications.

- Using Web Apps Logging and Diagnostics - with David Ebbo and Streaming Logs from Web Apps - with David Ebbo
  Videos by Scott Hanselman and David Ebbo.

For error logging, an alternative to writing your own tracing code is to use an open-source logging framework such as ELMAH. For more information, see Scott Hanselman's blog posts about ELMAH.

Also, you don't need to use ASP.NET or `System.Diagnostics` tracing to get streaming logs from Azure. The App Service app streaming log service streams any *.txt*, *.html*, or *.log* file that it finds in the *LogFiles* folder. Therefore, you could create your own logging system that writes to the file system of the app, and your file is automatically streamed and downloaded. All you have to do is write application code that creates files in the *d:\home\logfiles* folder.

Analyzing web server logs

For more information about analyzing web server logs, see the following resources:

- LogParser
  A tool for viewing data in web server logs (*.log* files).
- Troubleshooting IIS Performance Issues or Application Errors using LogParser
  An introduction to the Log Parser tool that you can use to analyze web server logs.
- Blog posts by Robert McMurray on using LogParser
- The HTTP status code in IIS 7.0, IIS 7.5, and IIS 8.0

Analyzing failed request tracing logs

The Microsoft TechNet website includes a Using Failed Request Tracing section, which may be helpful for understanding how to use these logs. However, this documentation focuses mainly on configuring failed request tracing in IIS, which you can't do in Azure App Service.

Best practices and troubleshooting guide for node applications on Azure App Service Windows

Article • 03/23/2022

In this article, you learn best practices and troubleshooting steps for Windows Node.js applications running on Azure App Service (with iisnode).

> ⚠ **Warning**
>
> Use caution when using troubleshooting steps on your production site. Recommendation is to troubleshoot your app on a non-production setup for example your staging slot and when the issue is fixed, swap your staging slot with your production slot.

IISNODE configuration

This schema file shows all the settings that you can configure for iisnode. Some of the settings that are useful for your application:

nodeProcessCountPerApplication

This setting controls the number of node processes that are launched per IIS application. The default value is 1. You can launch as many node.exes as your VM vCPU count by changing the value to 0. The recommended value is 0 for most applications so you can use all of the vCPUs on your machine. Node.exe is single-threaded so one node.exe consumes a maximum of 1 vCPU. To get maximum performance out of your node application, you want to use all vCPUs.

nodeProcessCommandLine

This setting controls the path to the node.exe. You can set this value to point to your node.exe version.

maxConcurrentRequestsPerProcess

This setting controls the maximum number of concurrent requests sent by iisnode to each node.exe. On Azure App Service, the default value is Infinite. You can configure the value depending on how many requests your application receives and how fast your application processes each request.

maxNamedPipeConnectionRetry

This setting controls the maximum number of times iisnode retries making the connection on the named pipe to send the requests to node.exe. This setting in combination with namedPipeConnectionRetryDelay determines the total timeout of each request within iisnode. The default value is 200 on Azure App Service. Total Timeout in seconds = (maxNamedPipeConnectionRetry * namedPipeConnectionRetryDelay) / 1000

namedPipeConnectionRetryDelay

This setting controls the amount of time (in ms) iisnode waits between each retry to send the request to node.exe over the named pipe. The default value is 250 ms. Total Timeout in seconds = (maxNamedPipeConnectionRetry * namedPipeConnectionRetryDelay) / 1000

By default, the total timeout in iisnode on Azure App Service is 200 * 250 ms = 50 seconds.

logDirectory

This setting controls the directory where iisnode logs stdout/stderr. The default value is iisnode, which is relative to the main script directory (directory where main server.js is present)

debuggerExtensionDll

This setting controls what version of node-inspector iisnode uses when debugging your node application. Currently, iisnode-inspector-0.7.3.dll and iisnode-inspector.dll are the only two valid values for this setting. The default value is iisnode-inspector-0.7.3.dll. The iisnode-inspector-0.7.3.dll version uses node-inspector-0.7.3 and uses web sockets. Enable web sockets on your Azure webapp to use this version.

flushResponse

The default behavior of IIS is that it buffers response data up to 4 MB before flushing, or until the end of the response, whichever comes first. iisnode offers a configuration setting to override this behavior: to flush a fragment of the response entity body as soon as iisnode receives it from node.exe, you need to set the iisnode/@flushResponse attribute in web.config to 'true':

```XML
<configuration>
    <system.webServer>
        <!-- ... -->
        <iisnode flushResponse="true" />
    </system.webServer>
</configuration>
```

Enable the flushing of every fragment of the response entity body adds performance overhead that reduces the throughput of the system by ~5% (as of v0.1.13). The best to scope this setting only to endpoints that require response streaming (for example, using the `<location>` element in the web.config)

In addition to this, for streaming applications, you must also set responseBufferLimit of your iisnode handler to 0.

```XML
<handlers>
    <add name="iisnode" path="app.js" verb="\*" modules="iisnode"
responseBufferLimit="0"/>
</handlers>
```

watchedFiles

A semi-colon separated list of files that are watched for changes. Any change to a file causes the application to recycle. Each entry consists of an optional directory name as well as a required file name, which are relative to the directory where the main application entry point is located. Wild cards are allowed in the file name portion only. The default value is `*.js;iisnode.yml`

recycleSignalEnabled

The default value is false. If enabled, your node application can connect to a named pipe (environment variable IISNODE_CONTROL_PIPE) and send a “recycle” message. This causes the w3wp to recycle gracefully.

idlePageOutTimePeriod

The default value is 0, which means this feature is disabled. When set to some value greater than 0, iisnode will page out all its child processes every 'idlePageOutTimePeriod' in milliseconds. See documentation to understand what page out means. This setting is useful for applications that consume a high amount of memory and want to page out memory to disk occasionally to free up RAM.

> ⚠ **Warning**
>
> Use caution when enabling the following configuration settings on production applications. The recommendation is to not enable them on live production applications.

debugHeaderEnabled

The default value is false. If set to true, iisnode adds an HTTP response header `iisnode-debug` to every HTTP response it sends the `iisnode-debug` header value is a URL. Individual pieces of diagnostic information can be obtained by looking at the URL fragment, however, a visualization is available by opening the URL in a browser.

loggingEnabled

This setting controls the logging of stdout and stderr by iisnode. Iisnode captures stdout/stderr from node processes it launches and writes to the directory specified in the 'logDirectory' setting. Once this is enabled, your application writes logs to the file system and depending on the amount of logging done by the application, there could be performance implications.

devErrorsEnabled

The default value is false. When set to true, iisnode displays the HTTP status code and Win32 error code on your browser. The win32 code is helpful in debugging certain types of issues.

debuggingEnabled (do not enable on live production site)

This setting controls debugging feature. Iisnode is integrated with node-inspector. By enabling this setting, you enable debugging of your node application. Upon enabling this setting, iisnode creates node-inspector files in 'debuggerVirtualDir' directory on the first debug request to your node application. You can load the node-inspector by sending a request to `http://yoursite/server.js/debug`. You can control the debug URL segment with 'debuggerPathSegment' setting. By default, debuggerPathSegment='debug'. You can set `debuggerPathSegment` to a GUID, for example, so that it is more difficult to be discovered by others.

Read Debug Node.js applications on Windows for more details on debugging.

Scenarios and recommendations/troubleshooting

My node application is making excessive outbound calls

Many applications would want to make outbound connections as part of their regular operation. For example, when a request comes in, your node app would want to contact a REST API elsewhere and get some information to process the request. You would want to use a keep alive agent when making http or https calls. You could use the agentkeepalive module as your keep alive agent when making these outbound calls.

The agentkeepalive module ensures that sockets are reused on your Azure webapp VM. Creating a new socket on each outbound request adds overhead to your application. Having your application reuse sockets for outbound requests ensures that your application doesn't exceed the maxSockets that are allocated per VM. The recommendation on Azure App Service is to set the agentKeepAlive maxSockets value to a total of (4 instances of node.exe * 32 maxSockets/instance) 128 sockets per VM.

Example agentKeepALive configuration:

Node.js

```
let keepaliveAgent = new Agent({
    maxSockets: 32,
    maxFreeSockets: 10,
    timeout: 60000,
    keepAliveTimeout: 300000
});
```

Important

This example assumes you have 4 node.exe running on your VM. If you have a different number of node.exe running on the VM, you must modify the maxSockets setting accordingly.

My node application is consuming too much CPU

You may receive a recommendation from Azure App Service on your portal about high cpu consumption. You can also set up monitors to watch for certain metrics. When checking the CPU usage on the Azure portal Dashboard, check the MAX values for CPU so you don't miss the peak values. If you believe your application is consuming too much CPU and you cannot explain why, you can profile your node application to find out.

Profiling your node application on Azure App Service with V8-Profiler

For example, let's say you have a hello world app that you want to profile as follows:

Node.js

```
const http = require('http');
function WriteConsoleLog() {
    for(let i=0;i<99999;++i) {
        console.log('hello world');
    }
}

function HandleRequest() {
    WriteConsoleLog();
}

http.createServer(function (req, res) {
    res.writeHead(200, {'Content-Type': 'text/html'});
    HandleRequest();
    res.end('Hello world!');
}).listen(process.env.PORT);
```

Go to the Debug Console site `https://yoursite.scm.azurewebsites.net/DebugConsole`

Go into your site/wwwroot directory. You see a command prompt as shown in the following example:

Run the command `npm install v8-profiler`.

This command installs the v8-profiler under node_modules directory and all of its dependencies. Now, edit your server.js to profile your application.

Node.js

```
const http = require('http');
const profiler = require('v8-profiler');
const fs = require('fs');

function WriteConsoleLog() {
    for(let i=0;i<99999;++i) {
        console.log('hello world');
    }
}

function HandleRequest() {
    profiler.startProfiling('HandleRequest');
    WriteConsoleLog();
    fs.writeFileSync('profile.cpuprofile',
JSON.stringify(profiler.stopProfiling('HandleRequest')));
}

http.createServer(function (req, res) {
    res.writeHead(200, {'Content-Type': 'text/html'});
    HandleRequest();
    res.end('Hello world!');
}).listen(process.env.PORT);
```

The preceding code profiles the WriteConsoleLog function and then writes the profile output to the 'profile.cpuprofile' file under your site wwwroot. Send a request to your application. You see a 'profile.cpuprofile' file created under your site wwwroot.

Download this file and open it with Chrome F12 Tools. Press F12 on Chrome, then choose the **Profiles** tab. Choose the **Load** button. Select your profile.cpuprofile file that you downloaded. Click on the profile you just loaded.

You can see that 95% of the time was consumed by the WriteConsoleLog function. The output also shows you the exact line numbers and source files that caused the issue.

My node application is consuming too much memory

If your application is consuming too much memory, you see a notice from Azure App Service on your portal about high memory consumption. You can set up monitors to watch for certain metrics. When checking the memory usage on the Azure portal Dashboard, be sure to check the MAX values for memory so you don't miss the peak values.

Leak detection and Heap Diff for Node.js

You could use node-memwatch to help you identify memory leaks. You can install `memwatch` just like v8-profiler and edit your code to capture and diff heaps to identify the memory leaks in your application.

My node.exe's are getting killed randomly

There are a few reasons why node.exe is shut down randomly:

1. Your application is throwing uncaught exceptions – Check d:\home\LogFiles\Application\logging-errors.txt file for the details on the exception thrown. This file has the stack trace to help debug and fix your application.
2. Your application is consuming too much memory, which is affecting other processes from getting started. If the total VM memory is close to 100%, your node.exe's could be killed by the process manager. Process manager kills some processes to let other processes get a chance to do some work. To fix this issue, profile your application for memory leaks. If your application requires large amounts of memory, scale up to a larger VM (which increases the RAM available to the VM).

My node application does not start

If your application is returning 500 Errors when it starts, there could be a few reasons:

1. Node.exe is not present at the correct location. Check nodeProcessCommandLine setting.
2. Main script file is not present at the correct location. Check web.config and make sure the name of the main script file in the handlers section matches the main

script file.
3. Web.config configuration is not correct – check the settings names/values.
4. Cold Start – Your application is taking too long to start. If your application takes longer than (maxNamedPipeConnectionRetry * namedPipeConnectionRetryDelay) / 1000 seconds, iisnode returns a 500 error. Increase the values of these settings to match your application start time to prevent iisnode from timing out and returning the 500 error.

My node application crashed

Your application is throwing uncaught exceptions – Check `d:\\home\\LogFiles\\Application\\logging-errors.txt` file for the details on the exception thrown. This file has the stack trace to help diagnose and fix your application.

My node application takes too much time to start (Cold Start)

The common cause for long application start times is a high number of files in the node_modules. The application tries to load most of these files when starting. By default, since your files are stored on the network share on Azure App Service, loading many files can take time. Some solutions to make this process faster are:

1. Try to lazy load your node_modules and not load all of the modules at application start. To Lazy load modules, the call to require('module') should be made when you actually need the module within the function before the first execution of module code.
2. Azure App Service offers a feature called local cache. This feature copies your content from the network share to the local disk on the VM. Since the files are local, the load time of node_modules is much faster.

IISNODE http status and substatus

The `cnodeconstants` source file lists all of the possible status/substatus combinations iisnode can return due to an error.

Enable FREB for your application to see the win32 error code (be sure you enable FREB only on non-production sites for performance reasons).

Http Status	Http Substatus	Possible Reason?

Http Status	Http Substatus	Possible Reason?
500	1000	There was some issue dispatching the request to IISNODE – Check if node.exe was started. Node.exe could have crashed when starting. Check your web.config configuration for errors.
500	1001	- Win32Error 0x2 - App is not responding to the URL. Check the URL rewrite rules or check if your express app has the correct routes defined. - Win32Error 0x6d – named pipe is busy – Node.exe is not accepting requests because the pipe is busy. Check high cpu usage. - Other errors – check if node.exe crashed.
500	1002	Node.exe crashed – check d:\home\LogFiles\logging-errors.txt for stack trace.
500	1003	Pipe configuration Issue – The named pipe configuration is incorrect.
500	1004-1018	There was some error while sending the request or processing the response to/from node.exe. Check if node.exe crashed. check d:\home\LogFiles\logging-errors.txt for stack trace.
503	1000	Not enough memory to allocate more named pipe connections. Check why your app is consuming so much memory. Check maxConcurrentRequestsPerProcess setting value. If it's not infinite and you have many requests, increase this value to prevent this error.
503	1001	Request could not be dispatched to node.exe because the application is recycling. After the application has recycled, requests should be served normally.
503	1002	Check win32 error code for actual reason – Request could not be dispatched to a node.exe.
503	1003	Named pipe is too Busy – Verify if node.exe is consuming excessive CPU

NODE.exe has a setting called `NODE_PENDING_PIPE_INSTANCES`. On Azure App Service, this value is set to 5000. Meaning that node.exe can accept 5000 requests at a time on the named pipe. This value should be good enough for most node applications running on Azure App Service. You should not see 503.1003 on Azure App Service because of the high value for the `NODE_PENDING_PIPE_INSTANCES`

More resources

Follow these links to learn more about Node.js applications on Azure App Service.

- Get started with Node.js web apps in Azure App Service
- How to debug a Node.js web app in Azure App Service

- Using Node.js Modules with Azure applications
- Azure App Service Web Apps: Node.js
- Node.js Developer Center
- Exploring the Super Secret Kudu Debug Console

Troubleshoot HTTP errors of "502 bad gateway" and "503 service unavailable" in Azure App Service

Article • 02/14/2022

"502 bad gateway" and "503 service unavailable" are common errors in your app hosted in Azure App Service. This article helps you troubleshoot these errors.

If you need more help at any point in this article, you can contact the Azure experts on the MSDN Azure and the Stack Overflow forums. Alternatively, you can also file an Azure support incident. Go to the Azure Support site and click on **Get Support**.

Symptom

When you browse to the app, it returns a HTTP "502 Bad Gateway" error or a HTTP "503 Service Unavailable" error.

Cause

This problem is often caused by application level issues, such as:

- requests taking a long time
- application using high memory/CPU
- application crashing due to an exception.

Troubleshooting steps to solve "502 bad gateway" and "503 service unavailable" errors

Troubleshooting can be divided into three distinct tasks, in sequential order:

1. Observe and monitor application behavior
2. Collect data
3. Mitigate the issue

App Service gives you various options at each step.

1. Observe and monitor application behavior

Track Service health

Microsoft Azure publicizes each time there is a service interruption or performance degradation. You can track the health of the service on the Azure Portal. For more information, see Track service health.

Monitor your app

This option enables you to find out if your application is having any issues. In your app's blade, click the **Requests and errors** tile. The **Metric** blade will show you all the metrics you can add.

Some of the metrics that you might want to monitor for your app are

- Average memory working set
- Average response time
- CPU time
- Memory working set
- Requests

For more information, see:

- Monitor apps in Azure App Service
- Receive alert notifications

2. Collect data

Use the diagnostics tool

App Service provides an intelligent and interactive experience to help you troubleshoot your app with no configuration required. When you do run into issues with your app, the diagnostics tool will point out what's wrong to guide you to the right information to more easily and quickly troubleshoot and resolve the issue.

To access App Service diagnostics, navigate to your App Service app or App Service Environment in the Azure portal. In the left navigation, click on **Diagnose and solve problems**.

Use the Kudu Debug Console

App Service comes with a debug console that you can use for debugging, exploring, uploading files, as well as JSON endpoints for getting information about your environment. This is called the *Kudu Console* or the *SCM Dashboard* for your app.

You can access this dashboard by going to the link **https://<Your app name>.scm.azurewebsites.net/**.

Some of the things that Kudu provides are:

- environment settings for your application
- log stream
- diagnostic dump
- debug console in which you can run PowerShell cmdlets and basic DOS commands.

Another useful feature of Kudu is that, in case your application is throwing first-chance exceptions, you can use Kudu and the SysInternals tool Procdump to create memory dumps. These memory dumps are snapshots of the process and can often help you troubleshoot more complicated issues with your app.

For more information on features available in Kudu, see Azure Websites online tools you should know about.

3. Mitigate the issue

Scale the app

In Azure App Service, for increased performance and throughput, you can adjust the scale at which you are running your application. Scaling up an app involves two related actions: changing your App Service plan to a higher pricing tier, and configuring certain settings after you have switched to the higher pricing tier.

For more information on scaling, see Scale an app in Azure App Service.

Additionally, you can choose to run your application on more than one instance . This not only provides you with more processing capability, but also gives you some amount of fault tolerance. If the process goes down on one instance, the other instance will still continue serving requests.

You can set the scaling to be Manual or Automatic.

Use AutoHeal

AutoHeal recycles the worker process for your app based on settings you choose (like configuration changes, requests, memory-based limits, or the time needed to execute a request). Most of the time, recycle the process is the fastest way to recover from a problem. Though you can always restart the app from directly within the Azure Portal, AutoHeal will do it automatically for you. All you need to do is add some triggers in the root web.config for your app. Note that these settings would work in the same way even if your application is not a .NET one.

For more information, see Auto-Healing Azure Web Sites .

Restart the app

This is often the simplest way to recover from one-time issues. On the Azure Portal , on your app's blade, you have the options to stop or restart your app.

You can also manage your app using Azure PowerShell. For more information, see Using Azure PowerShell with Azure Resource Manager.

Troubleshoot slow app performance issues in Azure App Service

Article • 01/25/2023

This article helps you troubleshoot slow app performance issues in Azure App Service.

If you need more help at any point in this article, you can contact the Azure experts on the MSDN Azure and the Stack Overflow forums. Alternatively, you can also file an Azure support incident. Go to the Azure Support site and click on **Get Support**.

Symptom

When you browse the app, the pages load slowly and sometimes timeout.

Cause

This problem is often caused by application level issues, such as:

- network requests taking a long time
- application code or database queries being inefficient
- application using high memory/CPU
- application crashing due to an exception

Troubleshooting steps

Troubleshooting can be divided into three distinct tasks, in sequential order:

1. Observe and monitor application behavior
2. Collect data
3. Mitigate the issue

App Service gives you various options at each step.

1. Observe and monitor application behavior

Track Service health

Microsoft Azure publicizes each time there is a service interruption or performance degradation. You can track the health of the service on the Azure portal. For more

information, see Track service health.

Monitor your app

This option enables you to find out if your application is having any issues. In your app's blade, click the **Requests and errors** tile. The **Metric** blade shows you all the metrics you can add.

Some of the metrics that you might want to monitor for your app are

- Average memory working set
- Response time
- CPU time
- Memory working set
- Requests

For more information, see:

- Monitor apps in Azure App Service
- Receive alert notifications

Monitor web endpoint status

If you are running your app in the **Standard** pricing tier, App Service lets you monitor two endpoints from three geographic locations.

Endpoint monitoring configures web tests from geo-distributed locations that test response time and uptime of web URLs. The test performs an HTTP GET operation on

the web URL to determine the response time and uptime from each location. Each configured location runs a test every five minutes.

Uptime is monitored using HTTP response codes, and response time is measured in milliseconds. A monitoring test fails if the HTTP response code is greater than or equal to 400 or if the response takes more than 30 seconds. An endpoint is considered available if its monitoring tests succeed from all the specified locations.

To set it up, see Monitor apps in Azure App Service.

Also, see Keeping Azure Web Sites up plus Endpoint Monitoring - with Stefan Schackow for a video on endpoint monitoring.

Application performance monitoring using Extensions

You can also monitor your application performance by using a *site extension*.

Each App Service app provides an extensible management end point that allows you to use a powerful set of tools deployed as site extensions. Extensions include:

- Source code editors like Azure DevOps.
- Management tools for connected resources such as a MySQL database connected to an app.

Azure Application Insights is a performance monitoring site extension that's also available. To use Application Insights, you rebuild your code with an SDK. You can also install an extension that provides access to additional data. The SDK lets you write code to monitor the usage and performance of your app in more detail. For more information, see Monitor performance in web applications.

2. Collect data

App Service provides diagnostic functionality for logging information from both the web server and the web application. The information is separated into web server diagnostics and application diagnostics.

Enable web server diagnostics

You can enable or disable the following kinds of logs:

- **Detailed Error Logging** - Detailed error information for HTTP status codes that indicate a failure (status code 400 or greater). This may contain information that can help determine why the server returned the error code.

- **Failed Request Tracing** - Detailed information on failed requests, including a trace of the IIS components used to process the request and the time taken in each component. This can be useful if you are attempting to improve app performance or isolate what is causing a specific HTTP error.
- **Web Server Logging** - Information about HTTP transactions using the W3C extended log file format. This is useful when determining overall app metrics, such as the number of requests handled or how many requests are from a specific IP address.

Enable application diagnostics

There are several options to collect application performance data from App Service, profile your application live from Visual Studio, or modify your application code to log more information and traces. You can choose the options based on how much access you have to the application and what you observed from the monitoring tools.

Use Application Insights Profiler

You can enable the Application Insights Profiler to start capturing detailed performance traces. You can access traces captured up to five days ago when you need to investigate problems happened in the past. You can choose this option as long as you have access to the app's Application Insights resource on Azure portal.

Application Insights Profiler provides statistics on response time for each web call and traces that indicates which line of code caused the slow responses. Sometimes the App Service app is slow because certain code is not written in a performant way. Examples include sequential code that can be run in parallel and undesired database lock contentions. Removing these bottlenecks in the code increases the app's performance, but they are hard to detect without setting up elaborate traces and logs. The traces collected by Application Insights Profiler helps identifying the lines of code that slows down the application and overcome this challenge for App Service apps.

For more information, see Profiling live apps in Azure App Service with Application Insights.

Use Remote Profiling

In Azure App Service, web apps, API apps, mobile back ends, and WebJobs can be remotely profiled. Choose this option if you have access to the app resource and you know how to reproduce the issue, or if you know the exact time interval the performance issue happens.

Remote Profiling is useful if the CPU usage of the process is high and your process is running slower than expected, or the latency of HTTP requests are higher than normal, you can remotely profile your process and get the CPU sampling call stacks to analyze the process activity and code hot paths.

For more information, see Remote Profiling support in Azure App Service.

Set up diagnostic traces manually

If you have access to the web application source code, Application diagnostics enables you to capture information produced by a web application. ASP.NET applications can use the `System.Diagnostics.Trace` class to log information to the application diagnostics log. However, you need to change the code and redeploy your application. This method is recommended if your app is running on a testing environment.

For detailed instructions on how to configure your application for logging, see Enable diagnostics logging for apps in Azure App Service.

Use the diagnostics tool

App Service provides an intelligent and interactive experience to help you troubleshoot your app with no configuration required. When you do run into issues with your app, the diagnostics tool will point out what's wrong to guide you to the right information to more easily and quickly troubleshoot and resolve the issue.

To access App Service diagnostics, navigate to your App Service app or App Service Environment in the Azure portal. In the left navigation, click on **Diagnose and solve problems**.

Use the Kudu Debug Console

App Service comes with a debug console that you can use for debugging, exploring, uploading files, as well as JSON endpoints for getting information about your environment. This console is called the *Kudu Console* or the *SCM Dashboard* for your app.

You can access this dashboard by going to the link **https://<Your app name>.scm.azurewebsites.net/**.

Some of the things that Kudu provides are:

- environment settings for your application
- log stream

- diagnostic dump
- debug console in which you can run PowerShell cmdlets and basic DOS commands.

Another useful feature of Kudu is that, in case your application is throwing first-chance exceptions, you can use Kudu and the SysInternals tool Procdump to create memory dumps. These memory dumps are snapshots of the process and can often help you troubleshoot more complicated issues with your app.

For more information on features available in Kudu, see Azure DevOps tools you should know about.

3. Mitigate the issue

Scale the app

In Azure App Service, for increased performance and throughput, you can adjust the scale at which you are running your application. Scaling up an app involves two related actions: changing your App Service plan to a higher pricing tier, and configuring certain settings after you have switched to the higher pricing tier.

For more information on scaling, see Scale an app in Azure App Service.

Additionally, you can choose to run your application on more than one instance. Scaling out not only provides you with more processing capability, but also gives you some amount of fault tolerance. If the process goes down on one instance, the other instances continue to serve requests.

You can set the scaling to be Manual or Automatic.

Use AutoHeal

AutoHeal recycles the worker process for your app based on settings you choose (like configuration changes, requests, memory-based limits, or the time needed to execute a request). Most of the time, recycle the process is the fastest way to recover from a problem. Though you can always restart the app from directly within the Azure portal, AutoHeal does it automatically for you. All you need to do is add some triggers in the root web.config for your app. These settings would work in the same way even if your application is not a .NET app.

For more information, see Auto-Healing Azure Web Sites.

Restart the app

Restarting is often the simplest way to recover from one-time issues. On the Azure portal, on your app's blade, you have the options to stop or restart your app.

You can also manage your app using Azure PowerShell. For more information, see Using Azure PowerShell with Azure Resource Manager.

More resources

Tutorial: Run a load test to identify performance bottlenecks in a web app

Troubleshoot domain and TLS/SSL certificate problems in Azure App Service

Article • 05/23/2023

When you set up a domain or TLS/SSL certificate for your web apps in Azure App Service, you might encounter the following common problems. This article also includes the possible causes and solutions for these problems.

At any point in this article, you can get more help by contacting Azure experts on the Microsoft Q & A and Stack Overflow forums. Alternatively, to file an Azure support incident, go to the Azure Support site, and select **Get Support**.

> **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**.

Certificate problems

You can't add a TLS/SSL certificate binding to an app

Symptom

When you add a TLS binding, you receive the following error message:

"Failed to add SSL binding. Cannot set certificate for existing VIP because another VIP already uses that certificate."

Cause

This problem might happen if you have multiple IP-based TLS/SSL bindings for the same IP address across multiple apps. For example, app A has an IP-based TLS/SSL binding with an old certificate. App B has an IP-based TLS/SSL binding with a new certificate for the same IP address. When you update the app TLS binding with the new certificate, the update fails with this error because the same IP address is used for another app.

Solution

To resolve this problem, try one of the following methods:

- Delete the IP-based TLS/SSL binding on the app that uses the old certificate.
- Create a new IP-based TLS/SSL binding that uses the new certificate.

You can't delete a certificate

Symptom

When you try to delete a certificate, you receive the following error message:

"Unable to delete the certificate because it is currently being used in a TLS/SSL binding. The TLS binding must be removed before you can delete the certificate."

Cause

This problem might happen if another app uses the certificate.

Solution

1. Remove the TLS binding for that certificate from the apps.
2. Try to delete the certificate.
3. If you still can't delete the certificate, clear the internet browser cache, and reopen the Azure portal in a new browser window. Then try to delete the certificate.

You can't purchase an App Service certificate

Symptom

In the Azure portal, you can't purchase an Azure App Service certificate.

Cause and solution

This problem can happen for any of the following reasons:

- The App Service plan is a Free or Shared pricing tier. These tiers don't support TLS.

Solution: Upgrade the App Service plan to Standard.

- The subscription doesn't have a valid credit card.

  Solution: Add a valid credit card to your subscription.

- The subscription offer doesn't support purchasing an App Service certificate such as Microsoft Student.

  Solution: Upgrade your subscription.

- The subscription reached the limit on purchases that are allowed on a subscription.

  Solution: App Service certificates have a limit of 10 certificate purchases for the Pay-As-You-Go and EA subscription types. For other subscription types, the limit is 3. To increase the limit, contact Azure support.

- The App Service certificate was marked as fraud. You received the following error message: "Your certificate has been flagged for possible fraud. The request is currently under review. If the certificate does not become usable within 24 hours, contact Azure Support."

  Solution: If the certificate is marked as fraud and isn't resolved after 24 hours, follow these steps:

  1. Sign in to the Azure portal.
  2. Go to **App Service Certificates**, and select the certificate.
  3. Select **Certificate Configuration** > **Step 2: Verify** > **Domain Verification**. This step sends an email notice to the Azure certificate provider to resolve the problem.

An App Service certificate was renewed, but the app shows the old certificate

Symptom

The App Service certificate was renewed, but the app that uses the App Service certificate is still using the old certificate. Also, you may receive a warning that the HTTPS protocol is required.

Cause 1: Missing access policy permissions on the key vault

The Key Vault used to store the App Service Certificate is missing access policy permissions on the key vault for Microsoft.Azure.Websites and Microsoft.Azure.CertificateRegistration. The service principals and their required permissions for Key Vault access are:

Service Principal	Secret Permissions	Certificate Permissions
Microsoft Azure App Service	Get	Get
Microsoft Azure CertificateRegistration	Get, List, Delete	Get, List

Solution 1: Modify the access policies for the key vault

To modify the access polices for the key vault, follow these steps:

1. Sign in to the Azure portal. Select the Key Vault used by your App Service Certificate. Navigate to Access policies.
2. If you do not see the two Service Principals listed you will need to add them. If they are available, verify the permissions include the recommended secret and certificate permissions.
3. Add a Service Principal by selecting "Create". Then select the needed permissions for Secret and Certificate permissions.
4. For the Principal, enter the value(s) given above in the search box. Then select the principal.

Cause 2: The app service has not yet synced with the new certificate

The App Service automatically syncs your certificate within 48 hours. When you rotate or update a certificate, sometimes the application is still retrieving the old certificate and not the newly updated certificate. The reason is that the job to sync the certificate resource hasn't run yet. To resolve this problem, sync the certificate manually, which automatically updates the hostname bindings for the certificate in App Service without causing any downtime to your apps.

Solution 2: Force a sync for the certificate

To force a sync for the certificate, follow these steps:

1. Sign in to the Azure portal. Select **App Service Certificates**, and then select the certificate.

2. Select **Rekey and Sync**, and then select **Sync**. The sync takes some time to finish.
3. When the sync completes, the following notification appears: "Successfully updated all the resources with the latest certificate."

An App Service is showing the wrong certificate

Symptom

When browsing the App Service, it is presenting the wrong certificate

Cause

This problem can manifest when both IP SSL and SNI based bindings have been configured for the App Service. When non SNI clients hit the IP SSL endpoint, the IP SSL certificate gets cached. Now even if the SNI enabled clients hit the site, they will be presented with the IP SSL certificate causing an invalid cert to be presented.

Solution

Please ensure not to use SNI bindings along with IP SSL bindings and always browse to the website over custom domain URL if you have non SNI clients. In case you need to use SNI bindings, you need to ensure that the certificate that is bound to the IP SSL binding is issued to protect all configured URLs for the site (including the SNI bindings) and configure the same certificate against all other bindings. This behavior is by design.

Custom domain problems

A custom domain returns a 404 error

Symptom

When you browse to the site by using the custom domain name, you receive the following error message:

"Error 404 - Web app not found."

Cause and solution

Cause 1

Your configured custom domain is missing a "CNAME record" or an "A record".

Solution for cause 1

- If you added an "A record", make sure that a TXT record is also added. For more information, see Create the DNS records.
- If you don't have to use the root domain for your app, the recommendation is that you use a "CNAME record", rather than an "A record".
- Don't use both a "CNAME record" and an "A record" for the same domain. This issue can cause a conflict and prevent domain resolution.

Cause 2

The internet browser might still be caching the old IP address for your domain.

Solution for Cause 2

Clear the browser. For Windows devices, you can run the command `ipconfig /flushdns`. To check that your domain points to the app's IP address, use WhatsmyDNS.net.

You can't add a subdomain

Symptom

You can't add a new host name to an app to assign a subdomain.

Solution

- Make sure that you have permissions to add a host name to an app by checking with the subscription administrator.
- If you need more subdomains, we recommend that you change the domain hosting to Azure Domain Name Service (DNS). By using Azure DNS, you can add 500 host names to your app. For more information, see Add a subdomain.

DNS can't be resolved

Symptom

You received the following error message:

"The DNS record could not be located."

Cause

This problem happens for one of the following reasons:

- The time to live (TTL) period hasn't expired. To determine the TTL value, check your domain's DNS configuration, and wait for the period to expire.
- The DNS configuration is incorrect.

Solution

- Wait for 48 hours for this problem to resolve by itself.
- If you can change the TTL setting in your DNS configuration, try changing the value to 5 minutes, which might solve this problem.
- To verify that your domain points to the app's IP address, use WhatsmyDNS.net. If the domain doesn't point to the IP address, configure the "A record" to the app's correct IP address.

You need to restore a deleted domain

Symptom

Your domain is no longer visible in the Azure portal.

Cause

The subscription owner might have accidentally deleted the domain.

Solution

If your domain was deleted fewer than seven days ago, the domain hasn't started the deletion process. In this case, you can buy the same domain again on the Azure portal under the same subscription. (Be sure to type the exact domain name in the search box.) You won't be charged again for this domain. If the domain was deleted more than seven days ago, contact Azure support for help with restoring the domain.

Domain problems

You purchased a TLS/SSL certificate for the wrong domain

Symptom

You purchased an App Service certificate for the wrong domain. You can't update the certificate to use the correct domain.

Solution

Delete that certificate, and then buy a new certificate.

If the current certificate that uses the wrong domain is in the "Issued" state, you'll also be billed for that certificate. App Service certificates aren't refundable, but you can contact Azure support for other possible options.

Domain verification is not working

Symptom

The App Service certificate requires domain verification before the certificate is ready to use. When you select **Verify**, the process fails.

Solution

Manually verify your domain by adding a TXT record:

1. Go to the Domain Name Service (DNS) provider that hosts your domain name.
2. Add a TXT record for your domain that uses the value of the domain token that's shown in the Azure portal.
3. Wait a few minutes for DNS propagation to run, and then select the **Refresh** button to trigger the verification.

As an alternative, you can use the HTML webpage method to manually verify your domain. This method allows the certificate authority to confirm the domain ownership of the domain for which the certificate is issued.

1. Create an HTML file that's named **{domain verification token}.html**. The file content should contain the value of domain verification token.

2. Upload this file at the root of the web server that's hosting your domain.
3. Select **Refresh** to check the certificate status. It might take few minutes for verification to finish.

For example, if you're buying a standard certificate for azure.com with the domain verification token 1234abcd, a web request made to https://azure.com/1234abcd.html should return 1234abcd.

> **Important**
>
> A certificate purchase has only 15 days to complete the domain verification operation. After 15 days, the certificate authority denies the certificate, and you're not charged for the certificate. In this situation, delete this certificate and try again.>

You can't purchase a domain

Symptom

You can't buy an App Service domain in the Azure portal.

Cause and solution

This problem happens for one of the following reasons:

- There's no credit card on the Azure subscription, or the credit card is invalid.

  Solution: Add a valid credit card to your subscription.

- Your Azure subscription type does not support the purchase of an App Service domain.

  Solution: Upgrade your Azure subscription to another subscription type, such as a Pay-As-You-Go subscription.

- Depending on the subscription type, a sufficient payment history may be required prior to purchasing an App Service domain.

  Solution: Either purchase with a different subscription that has a payment history or wait until you have a payment history with current subscription.

- You're not the subscription owner, so you don't have permission to purchase a domain.

  Solution: Assign the Owner role to your account. Or, contact the subscription administrator to get permission to purchase a domain.

You can't add a host name to an app

Symptom

When you add a host name, the process fails to validate and verify the domain.

Cause

This problem happens for one of the following reasons:

- You don’t have permission to add a host name.

  Solution: Ask the subscription administrator to give you permission to add a host name.

- Your domain ownership could not be verified.

  Solution: Verify that your "CNAME record" or "A record" is correctly set up. To map a custom domain to an app, create either a "CNAME record" or an "A record". If you want to use a root domain, you must use an "A record" and "TXT record":

Record type	Host	Point to
A	@	IP address for an app
TXT	@	`<app-name>.azurewebsites.net`
CNAME	www	`<app-name>.azurewebsites.net`

FAQ

Do I have to configure my custom domain for my website once I buy it?

When you purchase a domain from the Azure portal, the App Service app is automatically configured to use that custom domain. You don’t have to take any further steps. For more information, watch Azure App Service Self Help: Add a Custom Domain Name on Channel9.

Can I use a domain purchased in the Azure portal to point to an Azure virtual machine instead?

Yes, you can point the domain to a virtual machine. For more information, see Use Azure DNS to provide custom domain settings for an Azure service.

Is my domain hosted by GoDaddy or Azure DNS?

App Service Domains use GoDaddy for domain registration and Azure DNS to host the domains.

I enabled auto-renew but still received a renewal notice for my domain via email. What should I do?

If you enabled auto-renew, you don't need to take any action. The renewal notice through email only informs you that the domain is close to expiration and if auto-renew isn't enabled, you have to manually renew.

Will I be charged for Azure DNS hosting my domain?

The initial cost of domain purchase applies to domain registration only. Along with the registration cost, Azure DNS incurs charges, based on your usage. For more information, see Azure DNS pricing.

I purchased my domain earlier from the Azure portal and want to move from GoDaddy hosting to Azure DNS hosting. How can I do this?

You're not required to migrate to Azure DNS hosting. If you want to migrate to Azure DNS, the domain management experience in the Azure portal provides information about the steps necessary to move to Azure DNS. If you bought the domain through App Service, migration from GoDaddy hosting to Azure DNS is a relatively seamless procedure.

I would like to purchase my domain from App Service Domain but can I host my domain on GoDaddy instead of Azure DNS?

Starting July 24, 2017, Azure hosts App Service domains purchased from the Azure portal on Azure DNS. If you prefer to use a different hosting provider, you must go to their website to obtain a domain hosting solution.

Do I have to pay for privacy protection for my domain?

When you purchase a domain through the Azure portal, you can choose to add privacy at no extra cost. This benefit is included with purchasing your domain through Azure App Service.

If I decide I no longer want my domain, can I get my money back?

When you purchase a domain, you're not charged for five days. During this time, you can decide whether to keep the domain. If you choose to not keep the domain within this duration, you're not charged. However, domains that end with `.uk` are the exception. If you purchase such a domain, you're immediately charged, and you can't get a refund.

Can I use the domain in another Azure App Service app in my subscription?

Yes, when you access the Custom Domains and TLS blade in the Azure portal, you see the domains that you purchased. You can configure your app to use any of those domains.

Can I transfer a domain from one subscription to another subscription?

Yes, you can move a domain to another subscription or resource group using the Move-AzResource PowerShell cmdlet.

How can I manage my custom domain if I don't currently have an Azure App Service app?

You can manage your domain even if you don't have an App Service web app. You can use the domain for Azure services such as Virtual Machines, Azure Storage, and so on. If you plan to use the domain for App Service web apps, you must include a web app that's not on a free App Service plan so that you can bind the domain to your web app.

Can I move a web app with a custom domain to another subscription or from App Service Environment v1 to V2?

Yes, you can move your web app across subscriptions. Follow the guidance in How to move resources in Azure. Some limitations apply when you move a web app. For more information, see Limitations for moving App Service resources.

After you move a web app, the host name bindings of the domains within the custom domains setting should stay the same. No extra steps are required to configure the host name bindings.

What file formats are returned when I download my App Service Certificate from its Key Vault?

When you select "Download as a certificate" for the App Service Certificate under its Key Vault/Secrets, the certificate file format will be .pfx. No password will be applied to the file.

What file format can I use to upload a certificate to my App Service?

The certificate file format must be a .pfx file with a password applied to the file. The certificate must also meet the certificate requirements mentioned here. If you have obtained your certificate from a 3rd party CA and the file format is a .PEM/.KEY format, you can use a tool like openSSL to convert the file(s) to a .pfx file format. The private key must be included during the conversion as it is required in the .pfx file format. Also, if your certificate authority gives you multiple certificates in the certificate chain, you have to merge the certificates following the same order. For more information, please see here.

How do I generate a certificate signing request (CSR) for an App Service Certificate?

For an App Service Certificate, you would purchase through the Azure portal or using a Powershell/CLI command. A CSR is not needed. However, Azure Key Vault supports storing digital certificates issued by any certificate authority (CA). It supports creating a certificate signing request (CSR) with a private/public key pair. The CSR can be signed by any CA (an internal enterprise CA or an external public CA). For more information, please see here.

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 • 03/03/2023

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 **SSL settings** in the left navigation.
5. In the TLS/SSL bindings section, select the host name record. In the editor that opens, choose **SNI SSL** on the **SSL Type** drop-down menu and click **Add Binding**. When you see the operation success message, the existing IP address has been released.
6. In the **SSL bindings** section, again select the same host name record with the certificate. In the editor that opens, this time choose **IP Based SSL** on the **SSL Type** drop-down menu and click **Add Binding**. When you see the operation success message, you've acquired a new IP address.
7. If an A record (DNS record pointing directly to your IP address) is configured in your Domain Registration Portal (third party DNS Provider or Azure DNS), replace the existing IP address with the newly generated one. You can find the new IP address by following the instructions in the next section.

Find the new SSL IP address in the Azure Portal

1. Wait a few minutes, and then open the Azure portal.
2. In the left-hand navigation menu, select **App Services**.
3. Select your App Service app from the list.
4. Under the **Settings** header, click **Properties** in the left navigation, and find the section labeled **Virtual IP address**.
5. Copy the IP address and reconfigure your domain record or IP mechanism.

Next steps

This article explained how to prepare for an IP address change that was initiated by Azure. For more information about IP addresses in Azure App Service, see Inbound and outbound IP addresses in Azure App Service.