

Azure Container Apps documentation

Azure Container Apps allows you to run containerized applications without worrying about orchestration or infrastructure.

About Azure Container Apps

OVERVIEW

[What is Azure Container Apps?](#)

[Compare container options in Azure](#)

[Learn to develop with .NET](#)

[Learn to develop with Java](#)

Get started

QUICKSTART

[Deploy your first container app - Azure CLI](#)

[Deploy your first container app - Azure portal](#)

CONCEPT

[Environments](#)

[Containers](#)

[Revisions](#)

[Application lifecycle management](#)

[Microservices](#)

[Observability](#)

[Jobs](#)

Common tasks

HOW-TO GUIDE

[Set scaling rules](#)

[Manage secrets](#)

[Manage revisions](#)

[Set up ingress](#)

[Connect multiple apps](#)

[Use a custom VNET](#)

[Quotas](#)

 CONCEPT

[What's new in Dapr?](#)

[Dapr integration](#)

Apps and microservices

 TUTORIAL

[Microservices with Dapr](#)

Azure Container Apps overview

Article • 11/15/2023

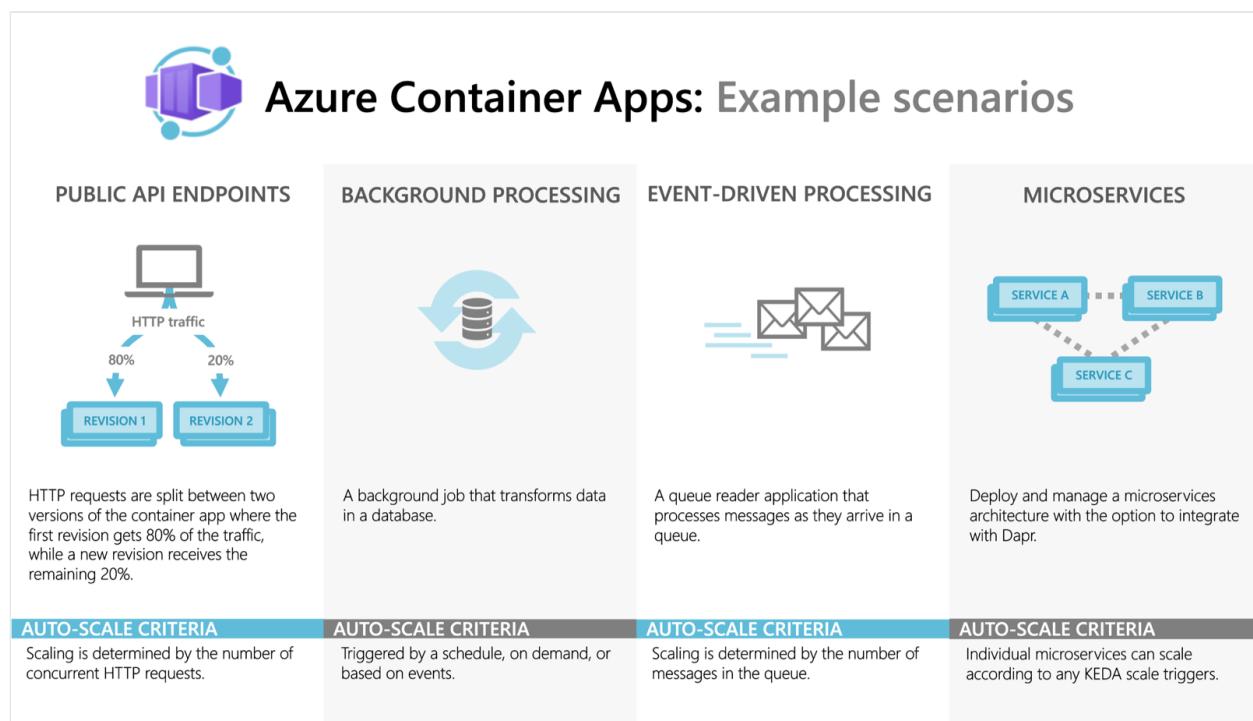
Azure Container Apps is a serverless platform that allows you to maintain less infrastructure and save costs while running containerized applications. Instead of worrying about server configuration, container orchestration, and deployment details, Container Apps provides all the up-to-date server resources required to keep your applications stable and secure.

Common uses of Azure Container Apps include:

- Deploying API endpoints
- Hosting background processing jobs
- Handling event-driven processing
- Running microservices

Additionally, applications built on Azure Container Apps can dynamically scale based on the following characteristics:

- HTTP traffic
- Event-driven processing
- CPU or memory load
- Any [KEDA-supported scaler](#) ↗



To begin working with Container Apps, select the description that best describes your situation.

Description	Resource
I'm new to containers	<p>Start here if you have yet to build your first container, but are curious how containers can serve your development needs.</p>
I'm using serverless containers	<p>Container Apps provides automatic scaling, reduces operational complexity, and allows you to focus on your application rather than infrastructure.</p> <p>Start here if you're interested in management, scalability, and pay-per-use features of cloud computing.</p>

Features

With Azure Container Apps, you can:

- **Use the Azure CLI extension, Azure portal or ARM templates** to manage your applications.
- **Enable HTTPS or TCP ingress** without having to manage other Azure infrastructure.
- **Build microservices with Dapr** and **access its rich set of APIs**.
- **Run jobs** on-demand, on a schedule, or based on events.
- Add **Azure Functions** and **Azure Spring Apps** to your Azure Container Apps environment.
- **Use specialized hardware** for access to increased compute resources.
- **Run multiple container revisions** and manage the container app's application lifecycle.
- **Autoscale** your apps based on any KEDA-supported scale trigger. Most applications can scale to zero¹.
- **Split traffic** across multiple versions of an application for Blue/Green deployments and A/B testing scenarios.
- **Use internal ingress and service discovery** for secure internal-only endpoints with built-in DNS-based service discovery.

- **Run containers from any registry**, public or private, including Docker Hub and Azure Container Registry (ACR).
- **Provide an existing virtual network** when creating an environment for your container apps.
- **Securely manage secrets** directly in your application.
- **Monitor logs** using Azure Log Analytics.
- **Generous quotas** which can be overridden to increase limits on a per-account basis.

¹ Applications that [scale on CPU or memory load](#) can't scale to zero.

Introductory video

<https://www.youtube-nocookie.com/embed/b3dopSTnSRg> ↗

Next steps

[Deploy your first container app](#)

Introduction to containers on Azure

Article • 11/30/2023

As you develop and deploy applications, you quickly run into challenges common to any production-grade system. For instance, you might ask yourself questions like:

- How can I be confident that what works on my machine works in production?
- How can I manage settings between different environments?
- How do I reliably deploy my application?

Some organizations choose to use virtual machines to deal with these problems.

However, virtual machines can be costly, sometimes slow, and too large to move around the network.

Instead of using a fully virtualized environment, some developers turn to containers.

What is a container?

Think for a moment about goods traveling around in a shipping container. When you see large metal boxes on cargo ships, you notice they're all the same size and shape. These containers make it easy to stack and move goods all around the world, regardless of what's inside.

Software containers work the same way, but in the digital world. Just like how a shipping container can hold toys, clothes, or electronics, a software container packages up everything an application needs to run. Whether on your computer, in a test environment, or in production a cloud service like Microsoft Azure, a container works the same way in diverse contexts.

Benefits of using containers

Containers package your applications in an easy-to-transport unit. Here are a few benefits of using containers:

- **Consistency:** Goods in a shipping container remain safe and unchanged during transport. Similarly, a software container guarantees consistent application behavior among different environments.
- **Flexibility:** Despite the diverse contents of a shipping container, transportation methods remain standardized. Software containers encapsulate different apps and technologies, but maintain are maintained in a standardized fashion.

- **Efficiency:** Just as shipping containers optimize transport by allowing efficient stacking on ships and trucks, software containers optimize the use of computing resources. This optimization allows multiple containers to operate simultaneously on a single server.
- **Simplicity:** Moving shipping containers requires specific, yet standardized tools. Similarly, Azure Container Apps simplifies how you use containers, which allows you focus on app development without worrying about the details of container management.

[Use serverless containers](#)

Use serverless containers on Azure

Article • 11/30/2023

Serverless computing offers services that manage and maintain servers, which relieve you of the burden of physically operating servers yourself. Azure Container Apps is a serverless platform that handles scaling, security, and infrastructure management for you - all while reducing costs. Once freed from server-related concerns, you're able to spend your time focusing on your application code.

Container Apps make it easy to manage:

- 1. Automatic scaling:** As requests for your applications fluctuate, Container Apps keeps your systems running even during seasons of high demand. Container Apps meets the demand for your app at any level by [automatically creating new copies](#) (called replicas) of your container. As demand falls, the runtime removes unneeded replicas on your behalf.
- 2. Security:** Application security is enforced throughout many layers. From [authentication and authorization](#) to [network-level security](#), Container Apps allows you to be explicit about the users and requests allowed into your system.
- 3. Monitoring:** Easily monitor your container app's health using [observability tools](#) in Container Apps.
- 4. Deployment flexibility:** You can deploy from GitHub, Azure DevOps, or from your local machine.
- 5. Changes:** As your containers evolve, Container Apps catalogs changes as [revisions](#) to your containers. If you're experiencing a problem with a container, you can easily roll back to an older version.

Where to go next

Use the following table to help you get acquainted with Azure Container Apps.

⋮ [Expand table](#)

Action	Description
Build the app	Deploy your first app, then create an event driven app to process a message queue.

Action	Description
Scale the app	Learn how Containers Apps handles meeting variable levels of demand.
Enable public access	Enable ingress on your container app to accept request from the public web.
Observe app behavior	Use log streaming, your apps console, application logs, and alerts to observe the state of your container app.
Configure the virtual network	Learn to set up your virtual network to secure your containers and communicate between applications.
Run a process that executes and exits	Find out how jobs can help you run tasks that have finite beginning and end.

Quickstart: Deploy your first container app using the Azure portal

Article • 07/24/2024

Azure Container Apps enables you to run microservices and containerized applications on a serverless platform. With Container Apps, you enjoy the benefits of running containers while leaving behind the concerns of manually configuring cloud infrastructure and complex container orchestrators.

In this quickstart, you create a secure Container Apps environment and deploy your first container app using the Azure portal.

Prerequisites

- An Azure account with an active subscription is required. If you don't already have one, you can [create an account for free](#).
- Register the `Microsoft.App` resource provider.

Setup

Begin by signing in to the [Azure portal](#).

Create a container app

To create your container app, start at the Azure portal home page.

1. Search for **Container Apps** in the top search bar.
2. Select **Container Apps** in the search results.
3. Select the **Create** button.

Basics tab

In the *Basics* tab, do the following actions.

1. Enter the following values in the *Project details* section.

[] [Expand table](#)

Setting	Action
Subscription	Select your Azure subscription.
Resource group	Select Create new and enter my-container-apps .
Container app name	Enter my-container-app .
Deployment source	Select Container image .

2. Enter the following values in the "Container Apps Environment" section.

 Expand table

Setting	Action
Region	Select a region near you.
Container Apps Environment	Use the default value.

3. Select the **Container** tab.

4. Select *Use quickstart image*.

Deploy the container app

1. Select **Review and create** at the bottom of the page.

If no errors are found, the *Create* button is enabled.

If there are errors, any tab containing errors is marked with a red dot. Navigate to the appropriate tab. Fields containing an error are highlighted in red. Once all errors are fixed, select **Review and create** again.

2. Select **Create**.

A page with the message *Deployment is in progress* is displayed. Once the deployment is successfully completed, you see the message: *Your deployment is complete*.

Verify deployment

Select **Go to resource** to view your new container app.

Select the link next to *Application URL* to view your application. The following message appears in your browser.

Welcome to Azure Container App

myapp.happyhill-70162bb9.centraluseuap....

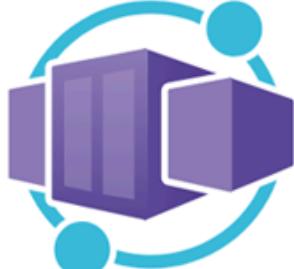
Microsoft Azure

Your container app is running with a Hello World image

Azure Container Apps is a serverless container solution for apps and microservices that helps you:

- Simplify your container deployments
- Manage less infrastructure
- Scale automatically on demand

[Learn more.](#)



Next steps

Explore sample templates you can leverage for your container apps.

Follow our Quickstart guide and deploy your own app.

[Sample apps](#)

[Quickstart](#)

Clean up resources

If you're not going to continue to use this application, you can delete the container app and all the associated services by removing the resource group.

1. Select the **my-container-apps** resource group from the *Overview* section.
2. Select the **Delete resource group** button at the top of the resource group *Overview*.
3. Enter the resource group name **my-container-apps** in the *Are you sure you want to delete "my-container-apps"* confirmation dialog.
4. Select **Delete**.

The process to delete the resource group could take a few minutes to complete.

 Tip

Having issues? Let us know on GitHub by opening an issue in the [Azure Container Apps repo](#).

Next steps

[Communication between microservices](#)

Feedback

Was this page helpful?

 Yes

 No

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

Quickstart: Deploy your first container app with containerapp up

Article • 08/04/2024

The Azure Container Apps service enables you to run microservices and containerized applications on a serverless platform. With Container Apps, you enjoy the benefits of running containers while you leave behind the concerns of manually configuring cloud infrastructure and complex container orchestrators.

In this quickstart, you create and deploy your first container app using the `az containerapp up` command.

Prerequisites

- An Azure account with an active subscription.
 - If you don't have one, you [can create one for free ↗](#).
- Install the [Azure CLI](#).

Setup

To sign in to Azure from the CLI, run the following command and follow the prompts to complete the authentication process.

```
Bash
az login
```

To ensure you're running the latest version of the CLI, run the upgrade command.

```
Bash
az upgrade
```

Next, install or update the Azure Container Apps extension for the CLI.

If you receive errors about missing parameters when you run `az containerapp` commands in Azure CLI or cmdlets from the `Az.App` module in Azure PowerShell, be sure you have the latest version of the Azure Container Apps extension installed.

Bash

Azure CLI

```
az extension add --name containerapp --upgrade
```

 **Note**

Starting in May 2024, Azure CLI extensions no longer enable preview features by default. To access Container Apps [preview features](#), install the Container Apps extension with `--allow-preview true`.

Azure CLI

```
az extension add --name containerapp --upgrade --allow-preview true
```

Now that the current extension or module is installed, register the `Microsoft.App` and `Microsoft.OperationalInsights` namespaces.

Bash

Azure CLI

```
az provider register --namespace Microsoft.App
```

Azure CLI

```
az provider register --namespace Microsoft.OperationalInsights
```

Create an Azure resource group

Create a resource group to organize the services related to your container app deployment.

Bash

Azure CLI

```
az group create \
--name my-container-apps \
--location centralus
```

Create and deploy the container app

Create and deploy your first container app with the `containerapp up` command. This command will:

- Create the Container Apps environment
- Create the Log Analytics workspace
- Create and deploy the container app using a public container image

Note that if any of these resources already exist, the command will use them instead of creating new ones.

Bash

Azure CLI

```
az containerapp up \
--name my-container-app \
--resource-group my-container-apps \
--location centralus \
--environment 'my-container-apps' \
--image mcr.microsoft.com/k8se/quickstart:latest \
--target-port 80 \
--ingress external \
--query properties.configuration.ingress.fqdn
```

ⓘ Note

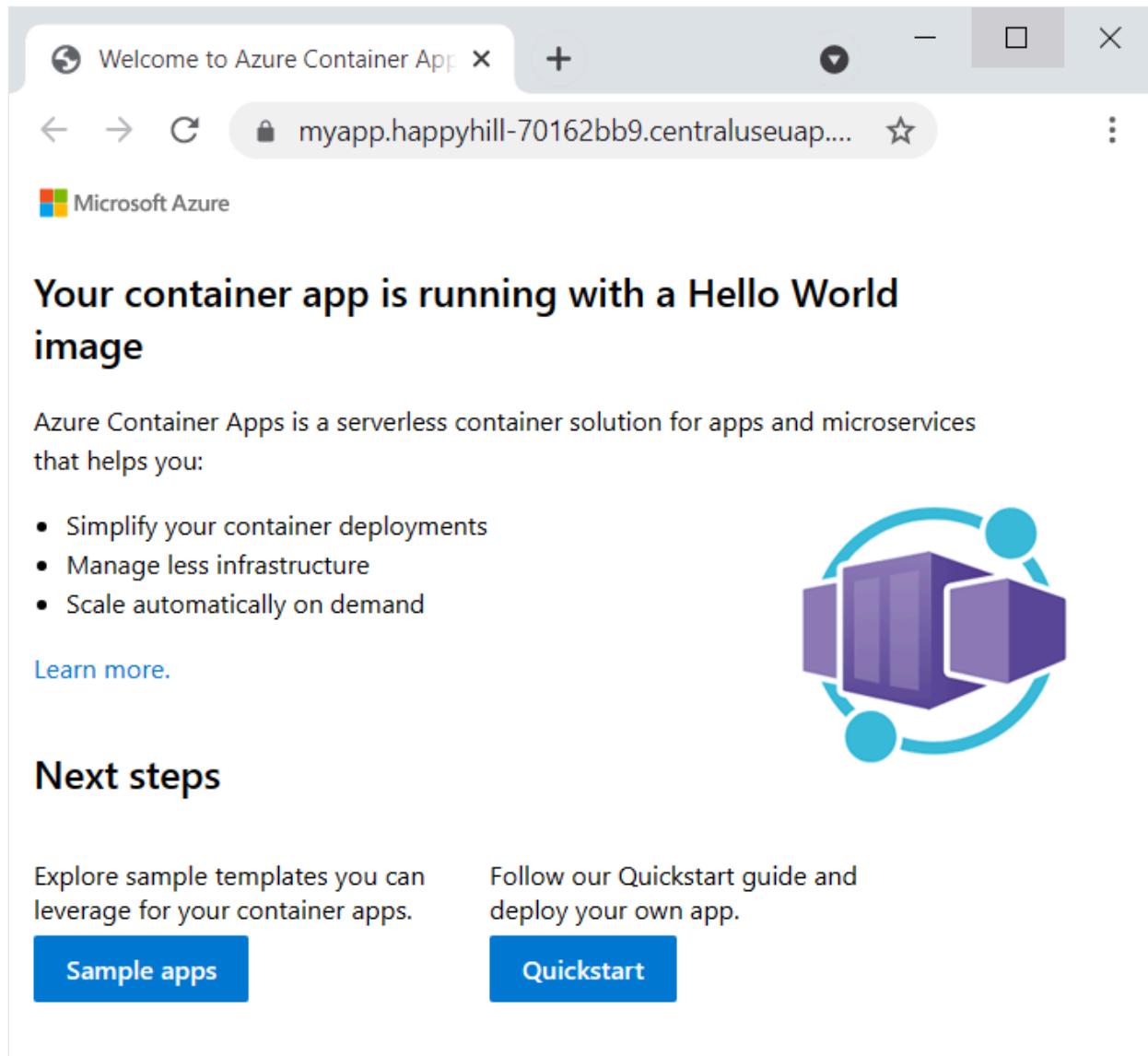
Make sure the value for the `--image` parameter is in lower case.

By setting `--ingress` to `external`, you make the container app available to public requests.

Verify deployment

The `up` command returns the fully qualified domain name for the container app. Copy this location to a web browser.

The following message is displayed when the container app is deployed:



A screenshot of a web browser window. The title bar says "Welcome to Azure Container App". The address bar shows "myapp.happyhill-70162bb9.centraluseuap...". Below the address bar, the Microsoft Azure logo is visible. The main content area displays the message: "Your container app is running with a Hello World image". Below this message, a paragraph states: "Azure Container Apps is a serverless container solution for apps and microservices that helps you:". A bulleted list follows: • Simplify your container deployments • Manage less infrastructure • Scale automatically on demand. A "Learn more." link is present. To the right of the text, there is a circular icon featuring three purple 3D cubes connected by blue lines, forming a network or container-like structure. At the bottom left, a blue button labeled "Sample apps" is shown. At the bottom right, another blue button labeled "Quickstart" is shown. The entire browser window is set against a light gray background.

Clean up resources

If you're not going to continue to use this application, run the following command to delete the resource group along with all the resources created in this quickstart.

 Caution

The following command deletes the specified resource group and all resources contained within it. If resources outside the scope of this quickstart exist in the specified resource group, they will also be deleted.

Azure CLI

```
az group delete --name my-container-apps
```

💡 Tip

Having issues? Let us know on GitHub by opening an issue in the [Azure Container Apps repo](#).

Next steps

[Communication between microservices](#)

Feedback

Was this page helpful?

 Yes

 No

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

Quickstart: Build and deploy from local source code to Azure Container Apps

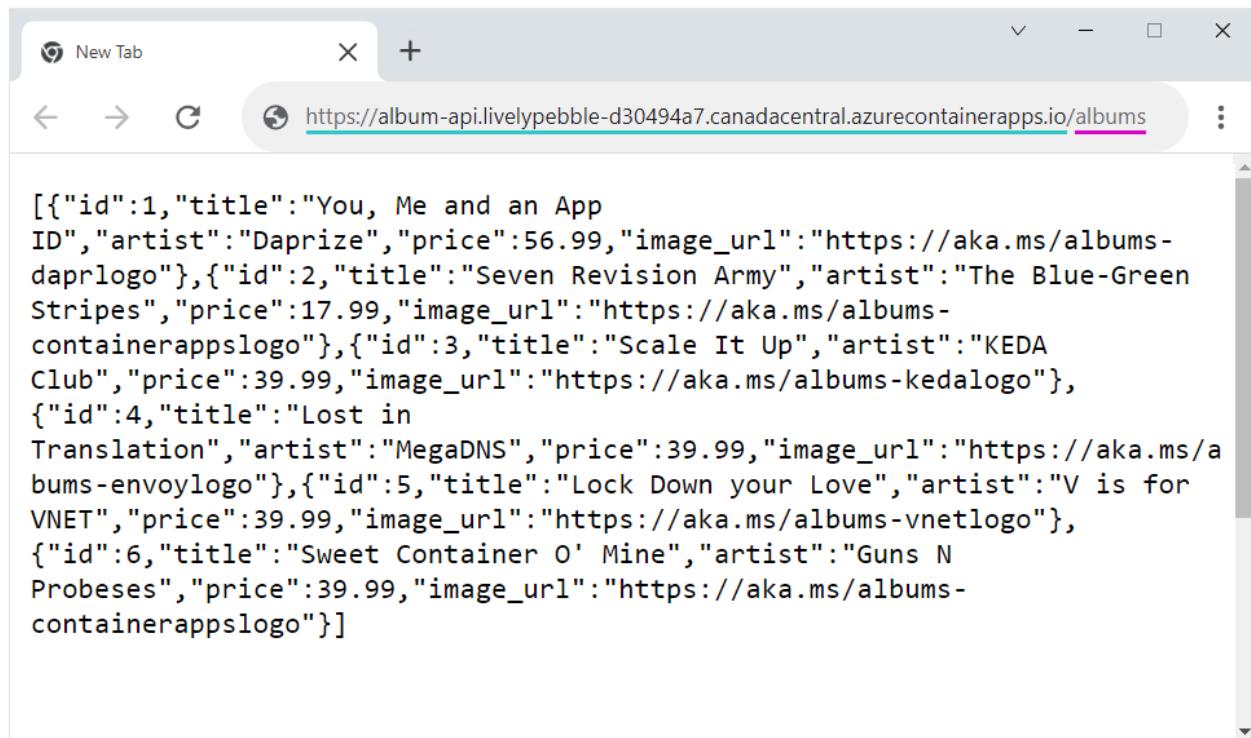
Article • 09/09/2024

This article demonstrates how to build and deploy a microservice to Azure Container Apps from local source code using the programming language of your choice. In this quickstart, you create a backend web API service that returns a static collection of music albums.

ⓘ Note

This sample application is available in two versions. One version where the source contains a Dockerfile. The other version has no Dockerfile. Select the version that best reflects your source code. If you are new to containers, select the **No Dockerfile** option at the top.

The following screenshot shows the output from the album API service you deploy.



A screenshot of a Microsoft Edge browser window. The address bar shows the URL <https://album-api.livelypebble-d30494a7.canadacentral.azurecontainerapps.io/albums>. The page content displays a JSON array of album data:

```
[{"id":1,"title":"You, Me and an App ID","artist":"Daprize","price":56.99,"image_url":"https://aka.ms/albums-daprllogo"}, {"id":2,"title":"Seven Revision Army","artist":"The Blue-Green Stripes","price":17.99,"image_url":"https://aka.ms/albums-containerappslogo"}, {"id":3,"title":"Scale It Up","artist":"KEDA Club","price":39.99,"image_url":"https://aka.ms/albums-kedalogo"}, {"id":4,"title":"Lost in Translation","artist":"MegaDNS","price":39.99,"image_url":"https://aka.ms/albums-envoylogo"}, {"id":5,"title":"Lock Down your Love","artist":"V is for VNET","price":39.99,"image_url":"https://aka.ms/albums-vnetlogo"}, {"id":6,"title":"Sweet Container O' Mine","artist":"Guns N Probeses","price":39.99,"image_url":"https://aka.ms/albums-containerappslogo"}]
```

Prerequisites

To complete this project, you need the following items:

Requirement	Instructions
Azure account	If you don't have one, create an account for free . You need the <i>Contributor</i> or <i>Owner</i> permission on the Azure subscription to proceed. Refer to Assign Azure roles using the Azure portal for details.
Azure CLI	Install the Azure CLI .

Setup

To sign in to Azure from the CLI, run the following command and follow the prompts to complete the authentication process.

```
Bash
```
Azure CLI
 az login
```

```

To ensure you're running the latest version of the CLI, run the upgrade command.

```
Bash
```
Azure CLI
 az upgrade
```

```

Next, install or update the Azure Container Apps extension for the CLI.

If you receive errors about missing parameters when you run `az containerapp` commands in Azure CLI or cmdlets from the `Az.App` module in Azure PowerShell, be sure you have the latest version of the Azure Container Apps extension installed.

```
Bash
```
Azure CLI
 az extension add --name containerapp --upgrade
```

```

ⓘ Note

Starting in May 2024, Azure CLI extensions no longer enable preview features by default. To access Container Apps [preview features](#), install the Container Apps extension with `--allow-preview true`.

Azure CLI

```
az extension add --name containerapp --upgrade --allow-preview true
```

Now that the current extension or module is installed, register the `Microsoft.App` and `Microsoft.OperationalInsights` namespaces.

ⓘ Note

Azure Container Apps resources have migrated from the `Microsoft.Web` namespace to the `Microsoft.App` namespace. Refer to [Namespace migration from Microsoft.Web to Microsoft.App in March 2022](#) for more details.

Bash

Azure CLI

```
az provider register --namespace Microsoft.App
```

Azure CLI

```
az provider register --namespace Microsoft.OperationalInsights
```

Create environment variables

Now that your Azure CLI setup is complete, you can define the environment variables that are used throughout this article.

Bash

Define the following variables in your bash shell.

Azure CLI

```
export RESOURCE_GROUP="album-containerapps"
export LOCATION="canadacentral"
export ENVIRONMENT="env-album-containerapps"
export API_NAME="album-api"
```

Get the sample code

Download and extract the API sample application in the language of your choice.

C#

[Download the source code ↗](#) to your machine.

Extract the download and change into the `containerapps-albumapi-csharp-main/src` folder.

Build and deploy the container app

Build and deploy your first container app with the `containerapp up` command. This command will:

- Create the resource group
- Create an Azure Container Registry
- Build the container image and push it to the registry
- Create the Container Apps environment with a Log Analytics workspace
- Create and deploy the container app using the built container image

The `up` command uses the Dockerfile in the root of the repository to build the container image. The `EXPOSE` instruction in the Dockerfile defined the target port, which is the port used to send ingress traffic to the container.

In the following code example, the `.` (dot) tells `containerapp up` to run in the `src` directory of the extracted sample API application.

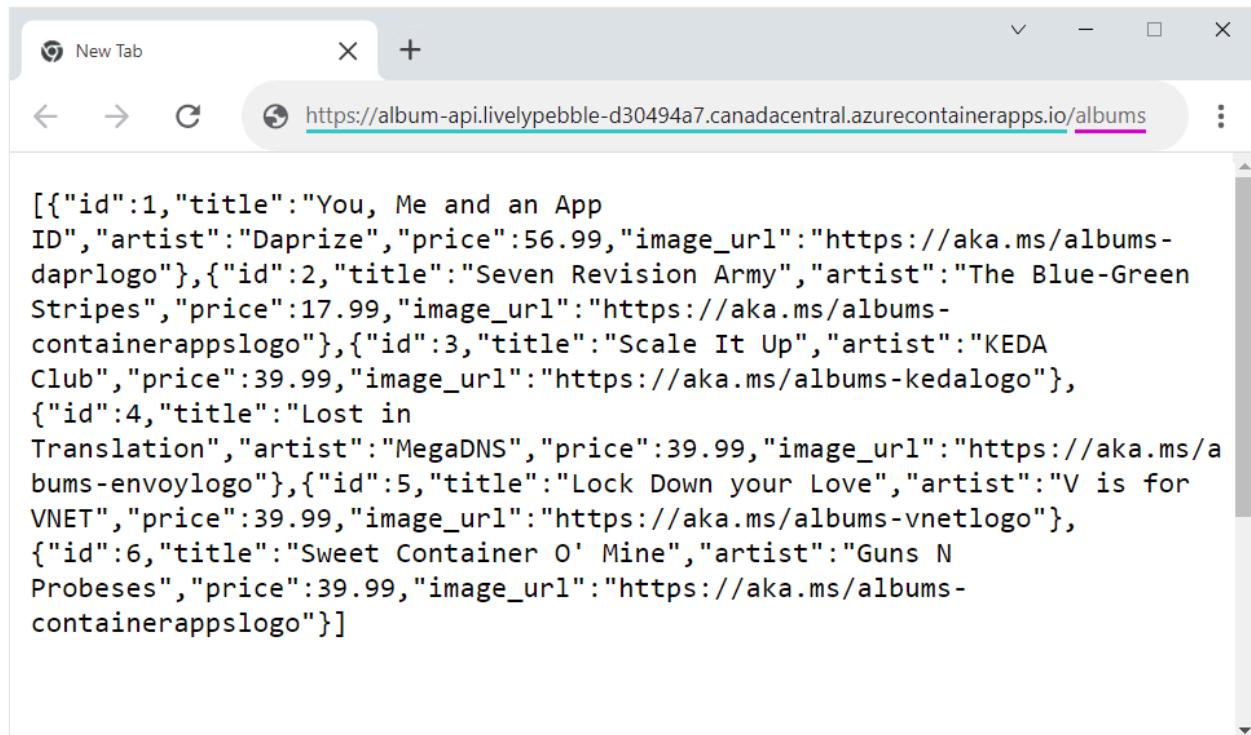
Bash

Azure CLI

```
az containerapp up \
--name $API_NAME \
--location $LOCATION \
--environment $ENVIRONMENT \
--source .
```

Verify deployment

Copy the FQDN to a web browser. From your web browser, go to the `/albums` endpoint of the FQDN.



Limits

The maximum size for uploading source code is 200MB. If the upload goes over the limit, error 413 is returned.

Clean up resources

If you're not going to continue on to the [Deploy a frontend](#) tutorial, you can remove the Azure resources created during this quickstart with the following command.

⊗ Caution

The following command deletes the specified resource group and all resources contained within it. If the group contains resources outside the scope of this quickstart, they are also deleted.

Bash

Azure CLI

```
az group delete --name $RESOURCE_GROUP
```



Having issues? Let us know on GitHub by opening an issue in the [Azure Container Apps repo](#).

Next steps

After completing this quickstart, you can continue to [Tutorial: Communication between microservices in Azure Container Apps](#) to learn how to deploy a front end application that calls the API.

[Tutorial: Communication between microservices](#)

Feedback

Was this page helpful?

Yes

No

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

Quickstart: Build and deploy from a repository to Azure Container Apps

Article • 05/06/2024

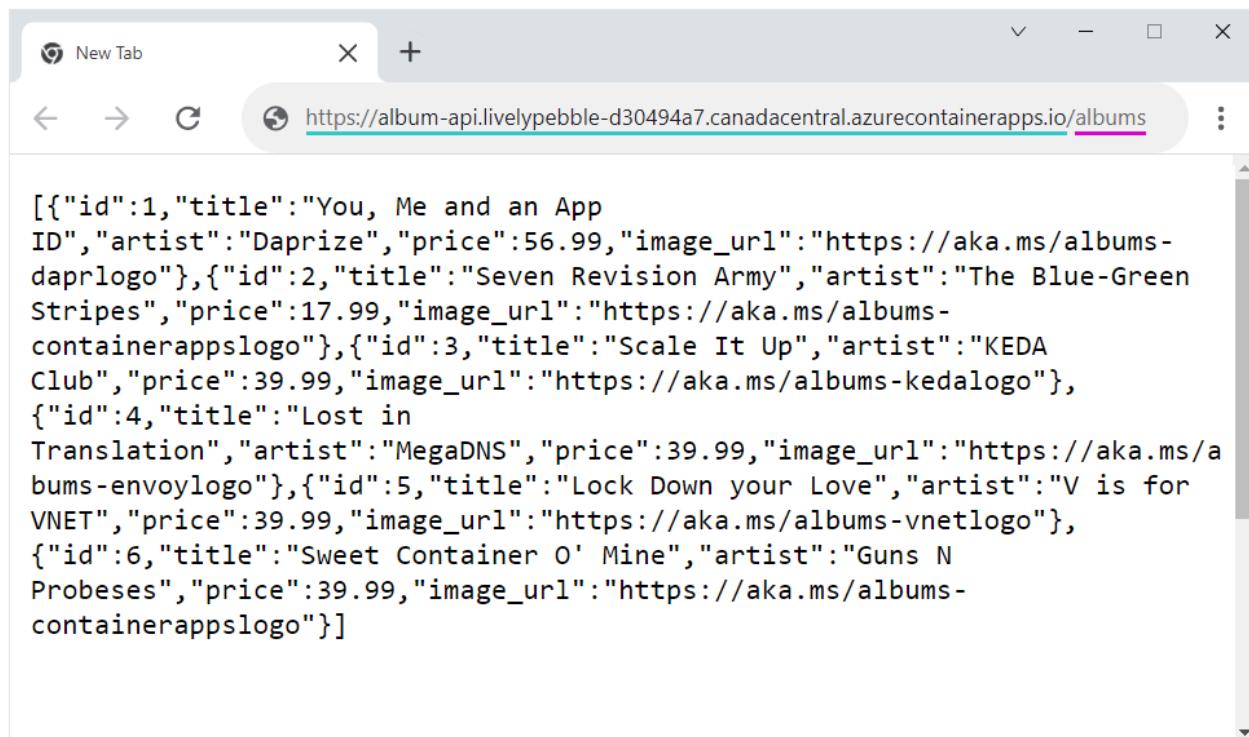
This article demonstrates how to build and deploy a microservice to Azure Container Apps from a GitHub repository using the programming language of your choice. In this quickstart, you create a sample microservice, which represents a backend web API service that returns a static collection of music albums.

This sample application is available in two versions. One version includes a container, where the source contains a Dockerfile. The other version has no Dockerfile. Select the version that best reflects your source code. If you're new to containers, select the **No Dockerfile** option at the top.

Note

You can also build and deploy this sample application from your local filesystem. For more information, see [Build from local source code and deploy your application in Azure Container Apps](#).

The following screenshot shows the output from the album API service you deploy.



Prerequisites

To complete this project, you need the following items:

[+] Expand table

Requirement	Instructions
Azure account	If you don't have one, create an account for free ↗ . You need the <i>Contributor</i> or <i>Owner</i> permission on the Azure subscription to proceed. Refer to Assign Azure roles using the Azure portal for details.
GitHub Account	Get one for free ↗ .
git	Install git ↗
Azure CLI	Install the Azure CLI .

Setup

To sign in to Azure from the CLI, run the following command and follow the prompts to complete the authentication process.

```
Bash
az login
```

To ensure you're running the latest version of the CLI, run the upgrade command.

```
Bash
az upgrade
```

Next, install or update the Azure Container Apps extension for the CLI.

```
Bash
az extension add --name acrcli
```

```
az extension add --name containerapp --upgrade
```

Now that the current extension or module is installed, register the `Microsoft.App` and `Microsoft.OperationalInsights` namespaces.

ⓘ Note

Azure Container Apps resources have migrated from the `Microsoft.Web` namespace to the `Microsoft.App` namespace. Refer to [Namespace migration from Microsoft.Web to Microsoft.App in March 2022](#) for more details.

Bash

Azure CLI

```
az provider register --namespace Microsoft.App
```

Azure CLI

```
az provider register --namespace Microsoft.OperationalInsights
```

Create environment variables

Now that your Azure CLI setup is complete, you can define the environment variables that are used throughout this article.

Bash

Define the following variables in your bash shell.

Azure CLI

```
export RESOURCE_GROUP="album-containerapps"
export LOCATION="canadacentral"
export ENVIRONMENT="env-album-containerapps"
export API_NAME="album-api"
export GITHUB_USERNAME=<YOUR_GITHUB_USERNAME>"
```

Before you run this command, make sure to replace `<YOUR_GITHUB_USERNAME>` with your GitHub username.

Next, define a container registry name unique to you.

Azure CLI

```
export ACR_NAME="acaalbums"$GITHUB_USERNAME
```

Prepare the GitHub repository

In a browser window, go to the GitHub repository for your preferred language and fork the repository.

C#

Select the **Fork** button at the top of the [album API repo](#) to fork the repo to your account. Then copy the repo URL to use it in the next step.

Build and deploy the container app

Build and deploy your first container app from your forked GitHub repository with the `containerapp up` command. This command will:

- Create the resource group
- Create the Container Apps environment with a Log Analytics workspace
- Create an Azure Container Registry
- Create a GitHub Action workflow to build and deploy the container app

When you push new code to the repository, the GitHub Action will:

- Build the container image and push it to the Azure Container Registry
- Deploy the container image to the created container app

The `up` command uses the Dockerfile in the root of the repository to build the container image. The `EXPOSE` instruction in the Dockerfile defines the target port. A Docker file isn't required to build a container app.

In the following command, replace the `<YOUR_GITHUB_REPOSITORY_NAME>` with your GitHub repository name in the form of `https://github.com/<OWNER>/<REPOSITORY-NAME>` or

```
<OWNER>/<REPOSITORY-NAME>.
```

Bash

Azure CLI

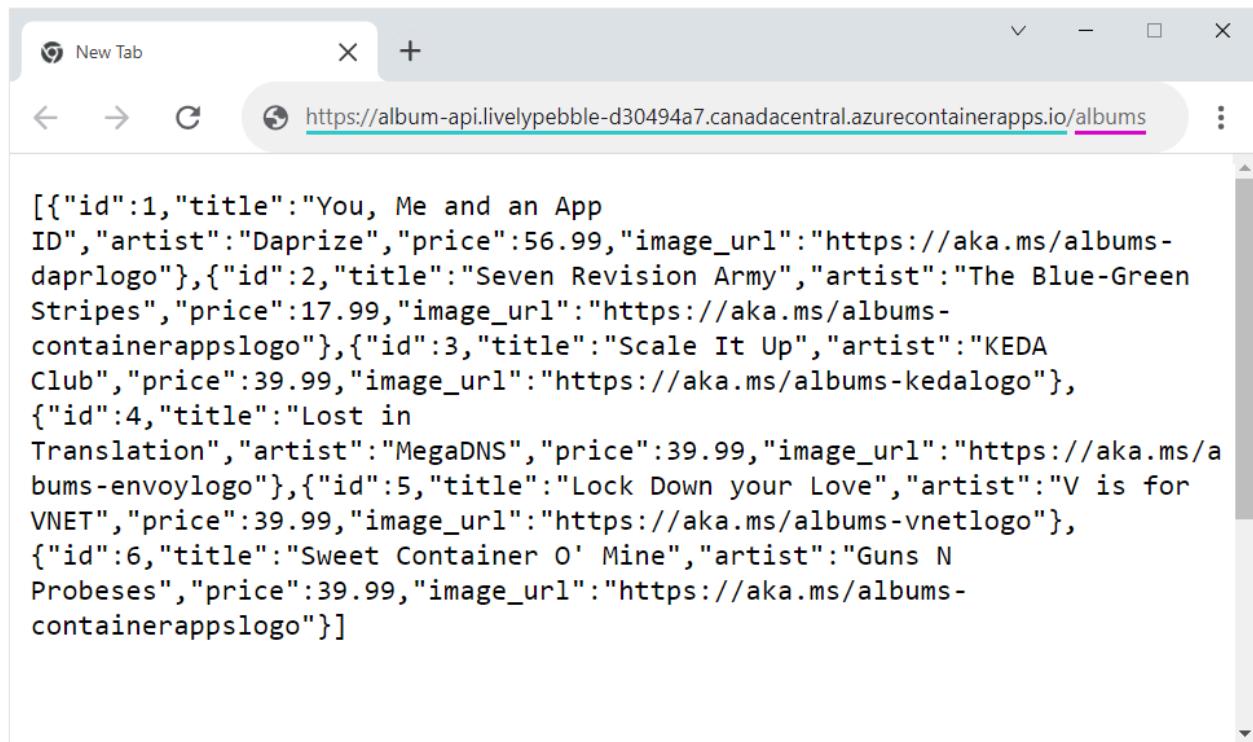
```
az containerapp up \
--name $API_NAME \
--resource-group $RESOURCE_GROUP \
--location $LOCATION \
--environment $ENVIRONMENT \
--context-path ./src \
--repo <YOUR_GITHUB_REPOSITORY_NAME>
```

Using the URL and the user code displayed in the terminal, go to the GitHub device activation page in a browser and enter the user code to the page. Follow the prompts to authorize the Azure CLI to access your GitHub repository.

The `up` command creates a GitHub Action workflow in your repository's `.github/workflows` folder. The workflow is triggered to build and deploy your container app when you push changes to the repository.

Verify deployment

Copy the domain name returned by the `containerapp up` to a web browser. From your web browser, go to the `/albums` endpoint of the URL.



A screenshot of a web browser window displaying a JSON array of album data. The URL in the address bar is <https://album-api.livelypebble-d30494a7.canadacentral.azurecontainerapps.io/albums>. The JSON response contains six albums, each with an ID, title, artist, price, and image URL.

```
[{"id":1,"title":"You, Me and an App ID","artist":"Daprize","price":56.99,"image_url":"https://aka.ms/albums-daprllogo"}, {"id":2,"title":"Seven Revision Army","artist":"The Blue-Green Stripes","price":17.99,"image_url":"https://aka.ms/albums-containerappslogo"}, {"id":3,"title":"Scale It Up","artist":"KEDA Club","price":39.99,"image_url":"https://aka.ms/albums-kedalogo"}, {"id":4,"title":"Lost in Translation","artist":"MegaDNS","price":39.99,"image_url":"https://aka.ms/albums-envoylogo"}, {"id":5,"title":"Lock Down your Love","artist":"V is for VNET","price":39.99,"image_url":"https://aka.ms/albums-vnetlogo"}, {"id":6,"title":"Sweet Container O' Mine","artist":"Guns N Probeses","price":39.99,"image_url":"https://aka.ms/albums-containerappslogo"}]
```

Clean up resources

If you're not going to continue on to the [Deploy a frontend](#) tutorial, you can remove the Azure resources created during this quickstart with the following command.

⊗ Caution

The following command deletes the specified resource group and all resources contained within it. If the group contains resources outside the scope of this quickstart, they are also deleted.

Bash

Azure CLI

```
az group delete --name $RESOURCE_GROUP
```

💡 Tip

Having issues? Let us know on GitHub by opening an issue in the [Azure Container Apps repo](#).

Next steps

After completing this quickstart, you can continue to [Tutorial: Communication between microservices in Azure Container Apps](#) to learn how to deploy a front end application that calls the API.

[Tutorial: Communication between microservices](#)

Tutorial: Deploy to Azure Container Apps using Visual Studio

Article • 10/16/2024

Azure Container Apps enables you to run microservices and containerized applications on a serverless platform. With Container Apps, you enjoy the benefits of running containers while leaving behind the concerns of manually configuring cloud infrastructure and complex container orchestrators.

In this tutorial, you deploy a containerized ASP.NET Core 8.0 application to Azure Container Apps using Visual Studio. The steps below also apply to earlier versions of ASP.NET Core.

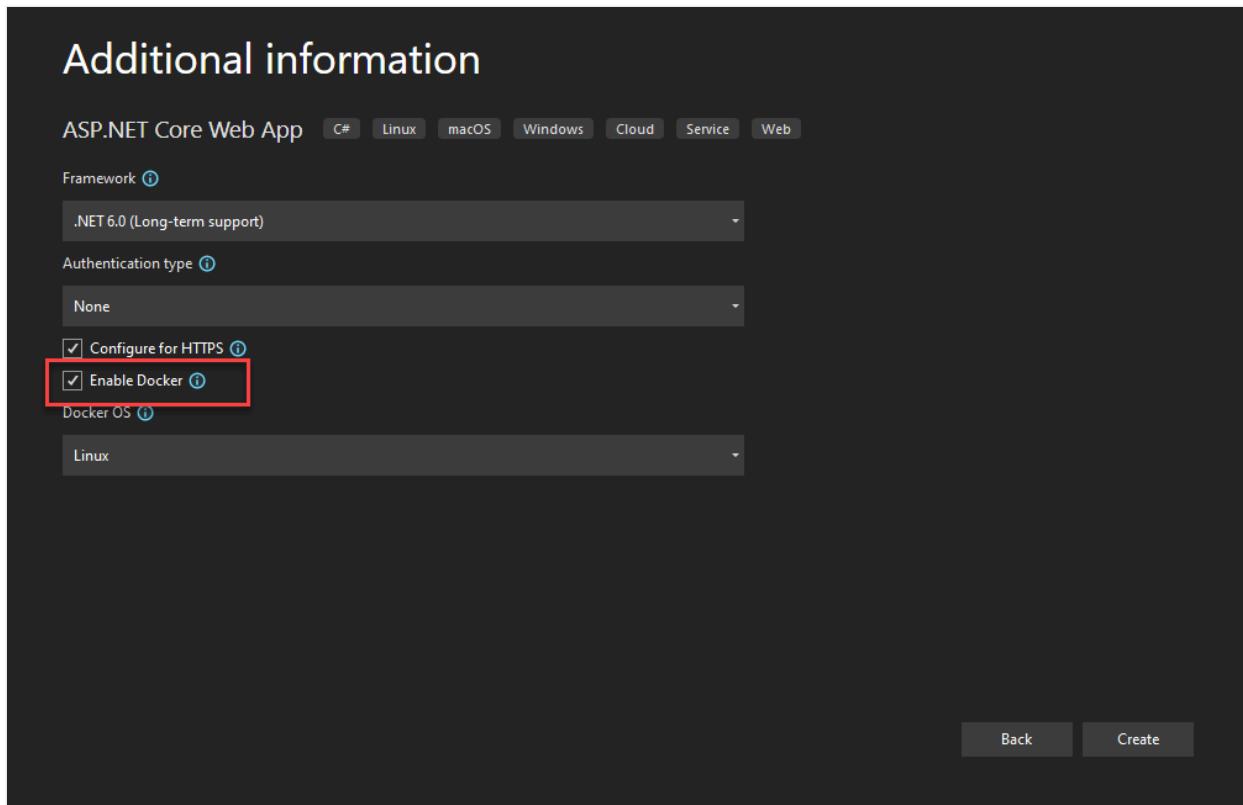
Prerequisites

- An Azure account with an active subscription is required. If you don't already have one, you can [create an account for free](#).
- Visual Studio 2022 version 17.2 or higher, available as a [free download](#).

Create the project

Begin by creating the containerized ASP.NET Core application.

1. In Visual Studio, select **File** and then choose **New => Project**.
2. In the dialog window, search for **ASP.NET**, and then choose **ASP.NET Core Web App** and select **Next**.
3. In the **Project Name** field, name the application *MyContainerApp* and then select **Next**.
4. On the **Additional Information** screen, make sure to select **Enable Docker**, and then make sure **Linux** is selected for the **Docker OS** setting. Azure Container Apps currently doesn't support Windows containers. This selection ensures the project template supports containerization by default. While enabled, the project uses a container as it is running or building.
5. Click **Create** and Visual Studio creates and loads the project.



Deploy to Azure Container Apps

The application includes a Dockerfile because the project template had the *Enable Docker* setting selected. Visual Studio uses the Dockerfile to build the container image that the Azure Container Apps run.

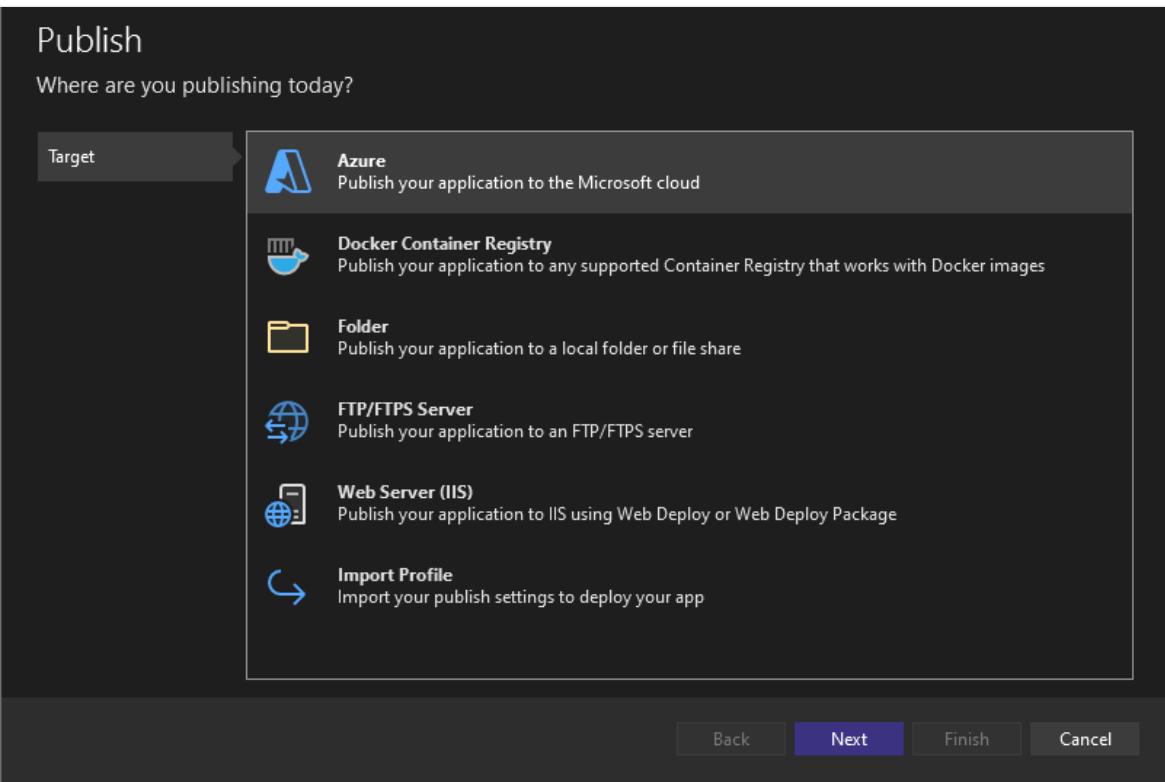
Refer to [How Visual Studio builds containerized apps](#) if you'd like to learn more about the specifics of this process.

You're now ready to deploy to the application to Azure Containers Apps.

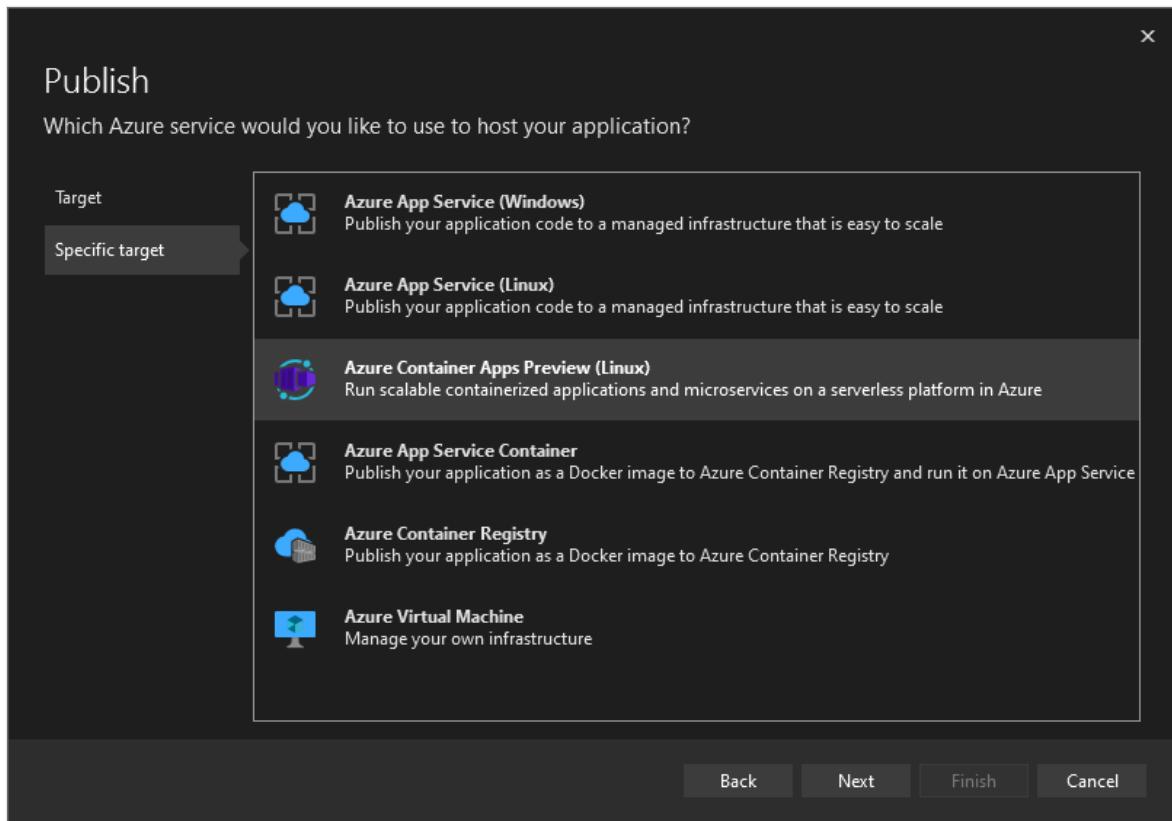
Create the resources

The publish dialog windows in Visual Studio help you choose existing Azure resources, or allow you to create new ones for deployment. This process also builds the container image, pushes the image to Azure Container Registry (ACR), and deploys the new container app image.

1. Right-click the **MyContainerApp** project node and select **Publish**.
2. In the dialog, choose **Azure** from the list of publishing options, and then select **Next**.



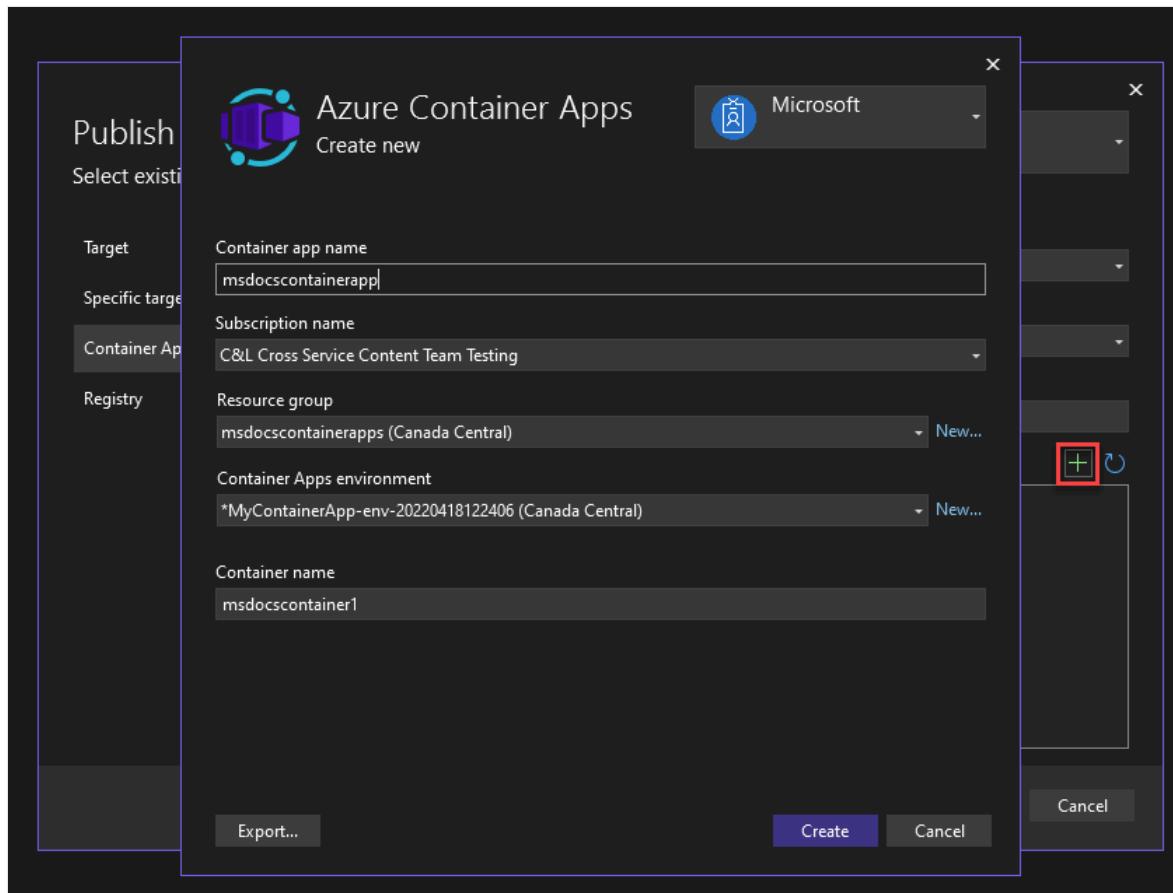
3. On the **Specific target** screen, choose **Azure Container Apps (Linux)**, and then select **Next** again.



4. Next, create an Azure Container App to host the project. Select the green plus icon on the right to open the *Create new* dialog. In the *Create new* dialog, enter the following values:

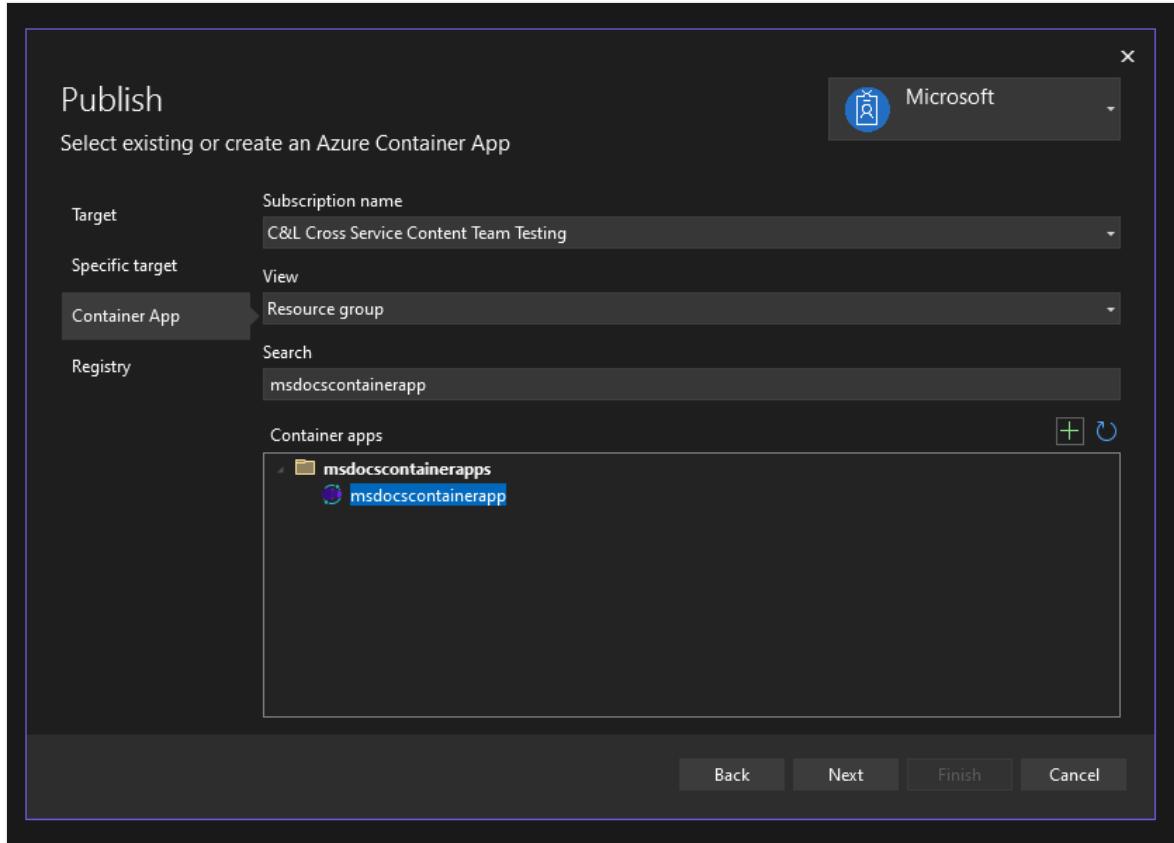
- **Container App name:** Enter a name of `msdocscontainerapp`.

- **Subscription name:** Choose the subscription where you would like to host your app.
- **Resource group:** A resource group acts as a logical container to organize related resources in Azure. You can either select an existing resource group, or select **New** to create one with a name of your choosing, such as `msdocscontainerapps`.
- **Container Apps Environment:** Container Apps Environment: Every container app must be part of a container app environment. An environment provides an isolated network for one or more container apps, making it possible for them to easily invoke each other. Click **New** to open the Create new dialog for your container app environment. Leave the default values and select **OK** to close the environment dialog.
- **Container Name:** This is the friendly name of the container that runs for this container app. Use the name `msdocscontainer1` for this quickstart. A container app typically runs a single container, but there are times when having more than one container is needed. One such example is when a sidecar container is required to perform an activity such as specialized logging or communications.



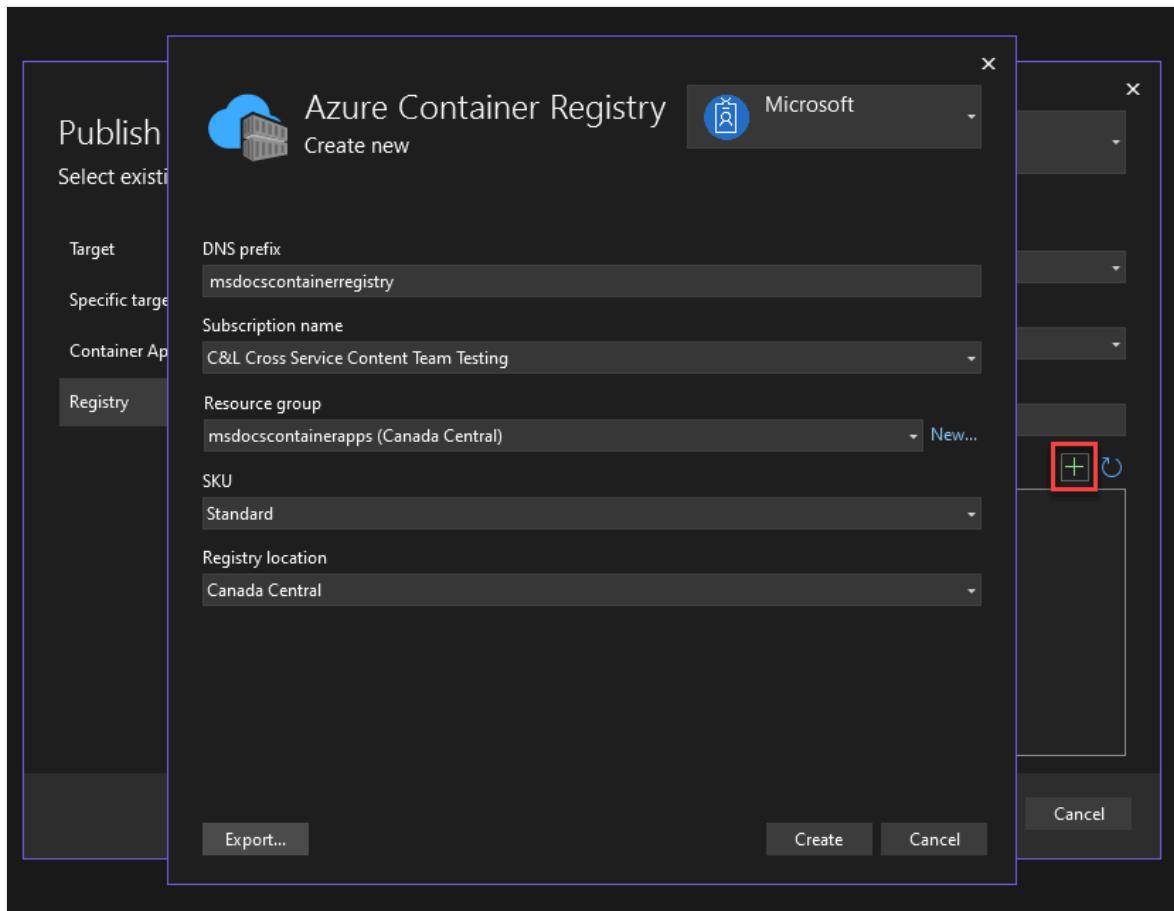
5. Select **Create** to finalize the creation of your container app. Visual Studio and Azure create the needed resources on your behalf. This process may take a couple minutes, so allow it to run to completion before moving on.

6. Once the resources are created, choose **Next**.

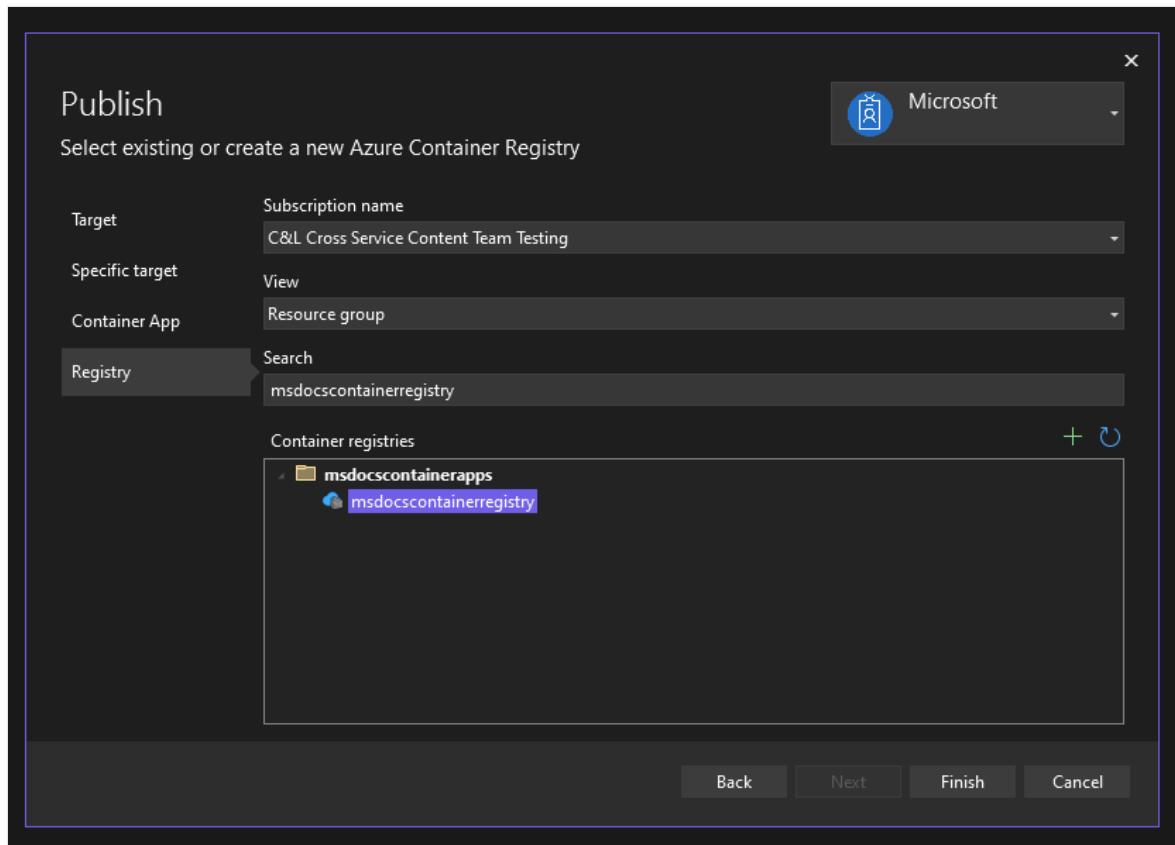


7. On the **Registry** screen, you can either select an existing Registry if you have one, or create a new one. To create a new one, click the green + icon on the right. On the *Create new registry* screen, fill in the following values:

- **DNS prefix:** Enter a value of `msdocscontainerregistry` or a name of your choosing.
- **Subscription Name:** Select the subscription you want to use - you might only have one to choose from.
- **Resource Group:** Choose the msdocs resource group you created previously.
- **Sku:** Select **Standard**.
- **Registry Location:** Select a region that is geographically close to you.



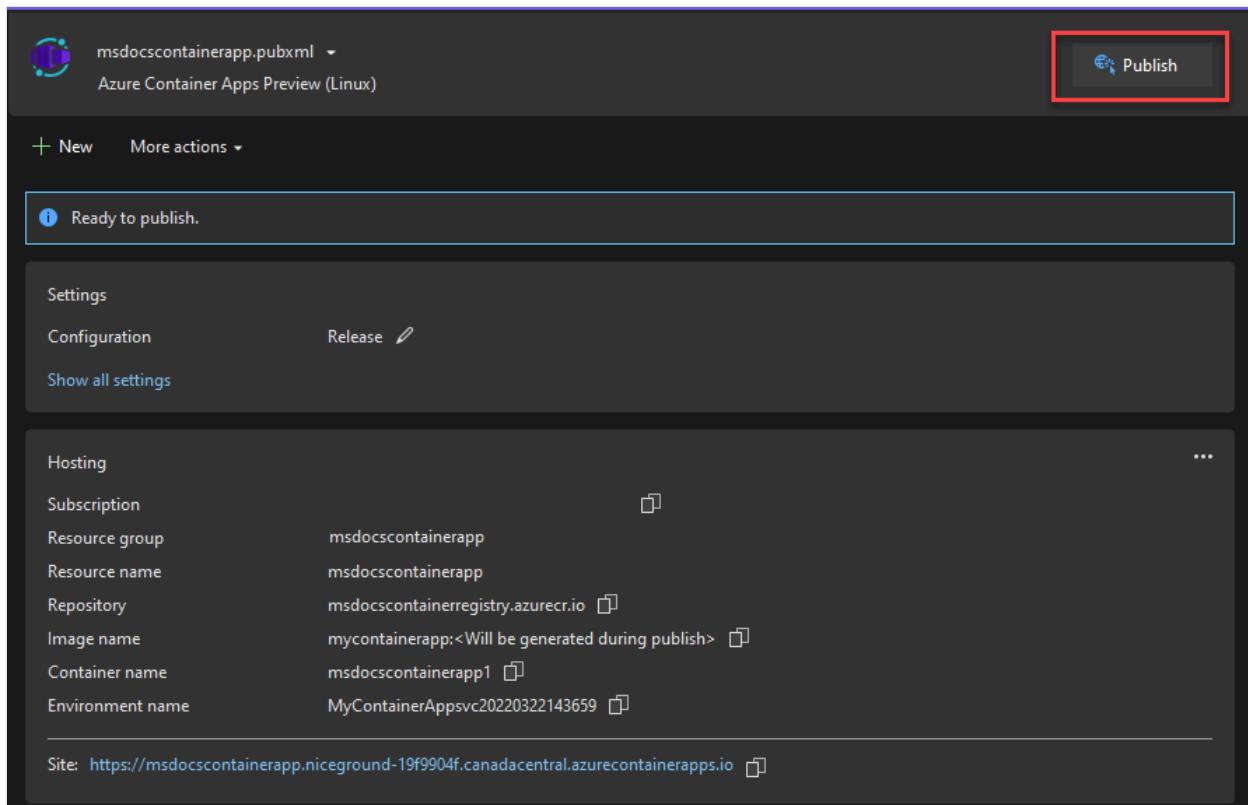
8. Once you populate these values, select **Create**. Visual Studio and Azure take a moment to create the registry.
9. Once the container registry is created, make sure it's selected, and then choose **Finish**. Visual Studio takes a moment to create the publish profile. This publish profile is where Visual Studio stores the publish options and resources you chose so you can quickly publish again whenever you want. You can close the dialog once it finishes.



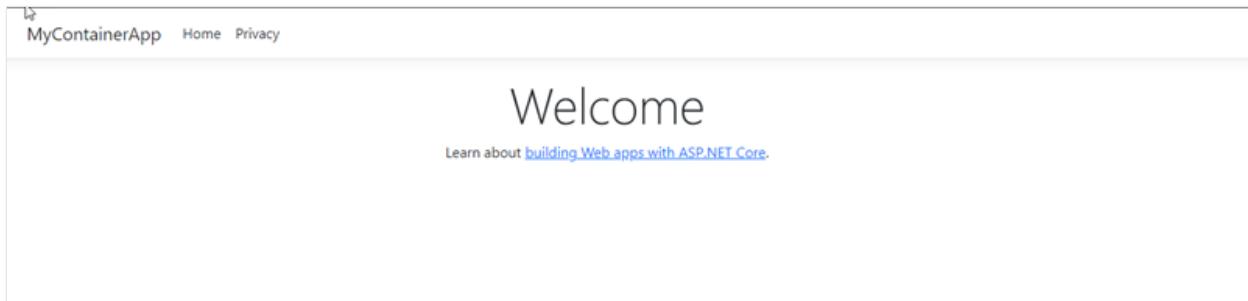
Publish the app using Visual Studio

While the resources and publishing profile are created, you still need to publish and deploy the app to Azure.

Choose **Publish** in the upper right of the publishing profile screen to deploy to the container app you created in Azure. This process might take a moment, so wait for it to complete.



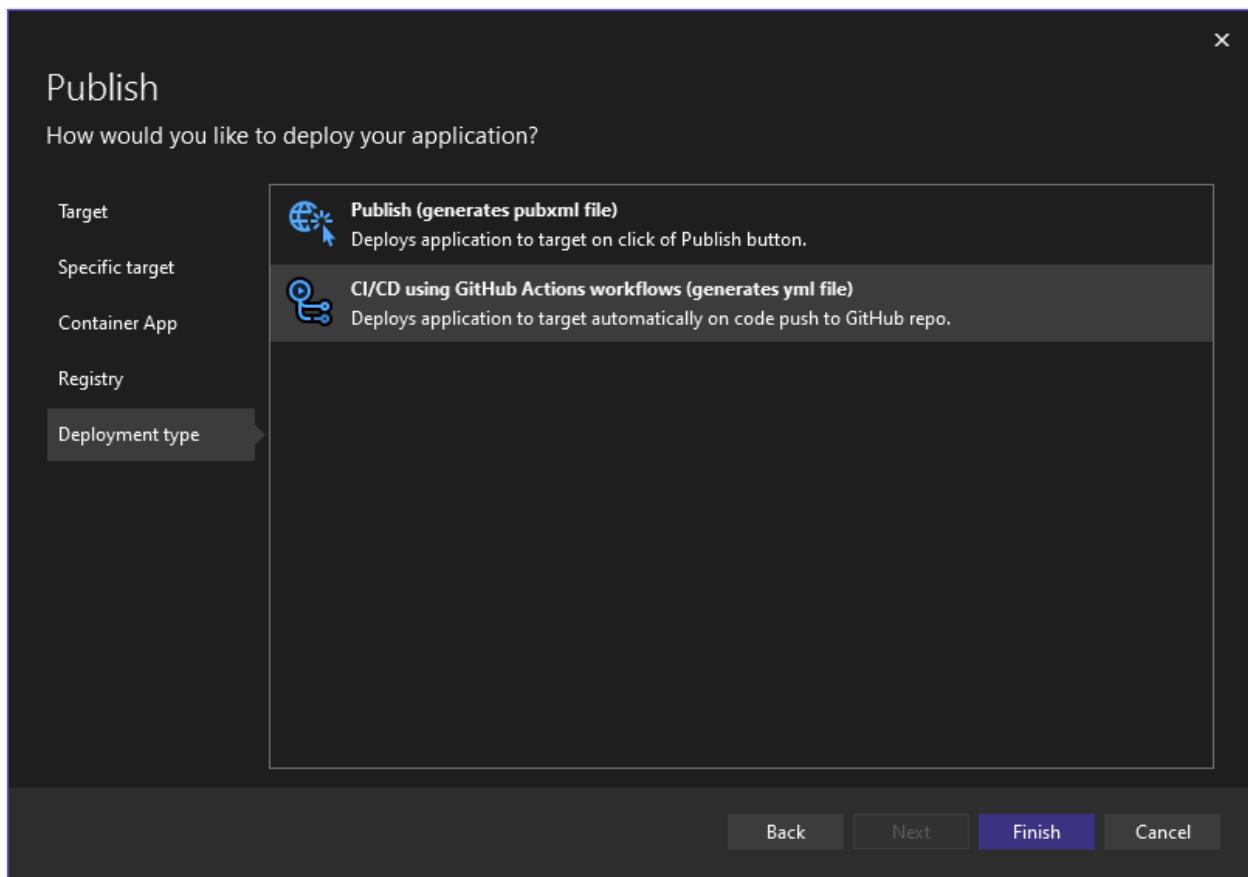
When the app finishes deploying, Visual Studio opens a browser to the URL of your deployed site. This page might initially display an error if all of the proper resources don't finish provisioning. You can continue to refresh the browser periodically to check if the deployment fully completes.



Publish the app using GitHub Actions

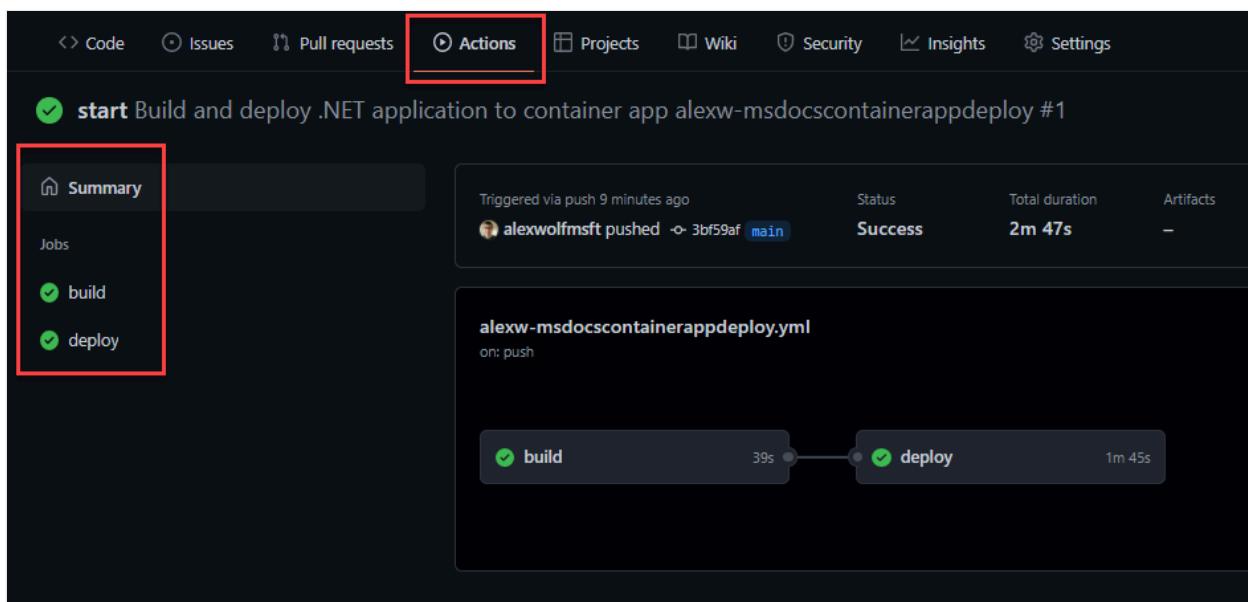
Container Apps can also be deployed using CI/CD through [GitHub Actions](#). GitHub Actions is a powerful tool for automating, customizing, and executing development workflows directly through the GitHub repository of your project.

If Visual Studio detects the project you're publishing is hosted in GitHub, the publish flow presents an additional **Deployment type** step. This stage allows developers to choose whether to publish directly through Visual Studio using the steps shown earlier in the quickstart, or through a GitHub Actions workflow.



If you select the GitHub Actions workflow, Visual Studio creates a `.github` folder to the root directory of the project, along with a generated YAML file inside of it. The YAML file contains GitHub Actions configurations to build and deploy your app to Azure every time you push your code.

After you make a change and push your code, you can see the progress of the build and deploy process in GitHub under the **Actions** tab. This page provides detailed logs and indicators regarding the progress and health of the workflow.



The workflow is complete when you see a green checkmark next to the build and deploy jobs. When you browse to your Container Apps site, you should see the latest changes

applied. You can always find the URL for your container app using the Azure portal page.

Clean up resources

If you no longer plan to use this application, you can delete the Azure Container Apps instance and all the associated services by removing the resource group.

To remove the resources you created, follow these steps in the Azure portal:

1. Select the **msdocscontainerapps** resource group from the *Overview* section.
2. Select the **Delete resource group** button at the top of the resource group *Overview*.
3. Enter the resource group name **msdocscontainerapps** in the *Are you sure you want to delete "my-container-apps"* confirmation dialog.
4. Select **Delete**.

The process to delete the resource group might take a few minutes to complete.

Tip

Having issues? Let us know on GitHub by opening an issue in the [Azure Container Apps repo](#).

Next steps

[Environments in Azure Container Apps](#)

Feedback

Was this page helpful?

 Yes

 No

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

Quickstart: Deploy to Azure Container Apps using Visual Studio Code

Article • 03/06/2024

Azure Container Apps enables you to run microservices and containerized applications on a serverless platform. With Container Apps, you enjoy the benefits of running containers while leaving behind the concerns of manually configuring cloud infrastructure and complex container orchestrators.

In this tutorial, you'll deploy a containerized application to Azure Container Apps using Visual Studio Code.

Prerequisites

- An Azure account with an active subscription is required. If you don't already have one, you can [create an account for free](#).
- Visual Studio Code, available as a [free download](#).
- The following Visual Studio Code extensions installed:
 - The [Azure Account extension](#)
 - The [Azure Container Apps extension](#)
 - The [Docker extension](#)

Clone the project

1. Open a new Visual Studio Code window.
2. Select `F1` to open the command palette.
3. Enter `Git: Clone` and press enter.
4. Enter the following URL to clone the sample project:

```
git
```

```
https://github.com/Azure-Samples/containerapps-albumapi-javascript.git
```

ⓘ Note

This tutorial uses a JavaScript project, but the steps are language agnostic.

5. Select a folder to clone the project into.
6. Select **Open** to open the project in Visual Studio Code.

Sign in to Azure

1. Select **F1** to open the command palette.
2. Select **Azure: Sign In** and follow the prompts to authenticate.
3. Once signed in, return to Visual Studio Code.

Create and deploy to Azure Container Apps

The Azure Container Apps extension for Visual Studio Code enables you to choose existing Container Apps resources, or create new ones to deploy your applications to. In this scenario, you create a new Container App environment and container app to host your application. After installing the Container Apps extension, you can access its features under the Azure control panel in Visual Studio Code.

1. Select **F1** to open the command palette and run the **Azure Container Apps: Deploy Project from Workspace** command.
2. Enter the following values as prompted by the extension.

 Expand table

Prompt	Value
Select subscription	Select the Azure subscription you want to use.
Select a container apps environment	Select Create new container apps environment . You're only asked this question if you have existing Container Apps environments.
Enter a name for the new container app resource(s)	Enter my-container-app .
Select a location	Select an Azure region close to you.
Would you like to save your deployment configuration?	Select Save .

The Azure activity log panel opens and displays the deployment progress. This process might take a few minutes to complete.

3. Once this process finishes, Visual Studio Code displays a notification. Select **Browse** to open the deployed app in a browser.

In the browser's location bar, append the `/albums` path at the end of the app URL to view data from a sample API request.

Congratulations! You successfully created and deployed your first container app using Visual Studio Code.

Clean up resources

If you're not going to continue to use this application, you can delete the Azure Container Apps instance and all the associated services at once by removing the resource group.

Follow these steps in the Azure portal to remove the resources you created:

1. Select the **my-container-app** resource group from the *Overview* section.
2. Select the **Delete resource group** button at the top of the resource group *Overview*.
3. Enter the resource group name **my-container-app** in the *Are you sure you want to delete "my-container-apps"* confirmation dialog.
4. Select **Delete**. The process to delete the resource group might take a few minutes to complete.

Tip

Having issues? Let us know on GitHub by opening an issue in the [Azure Container Apps repo](#).

Next steps

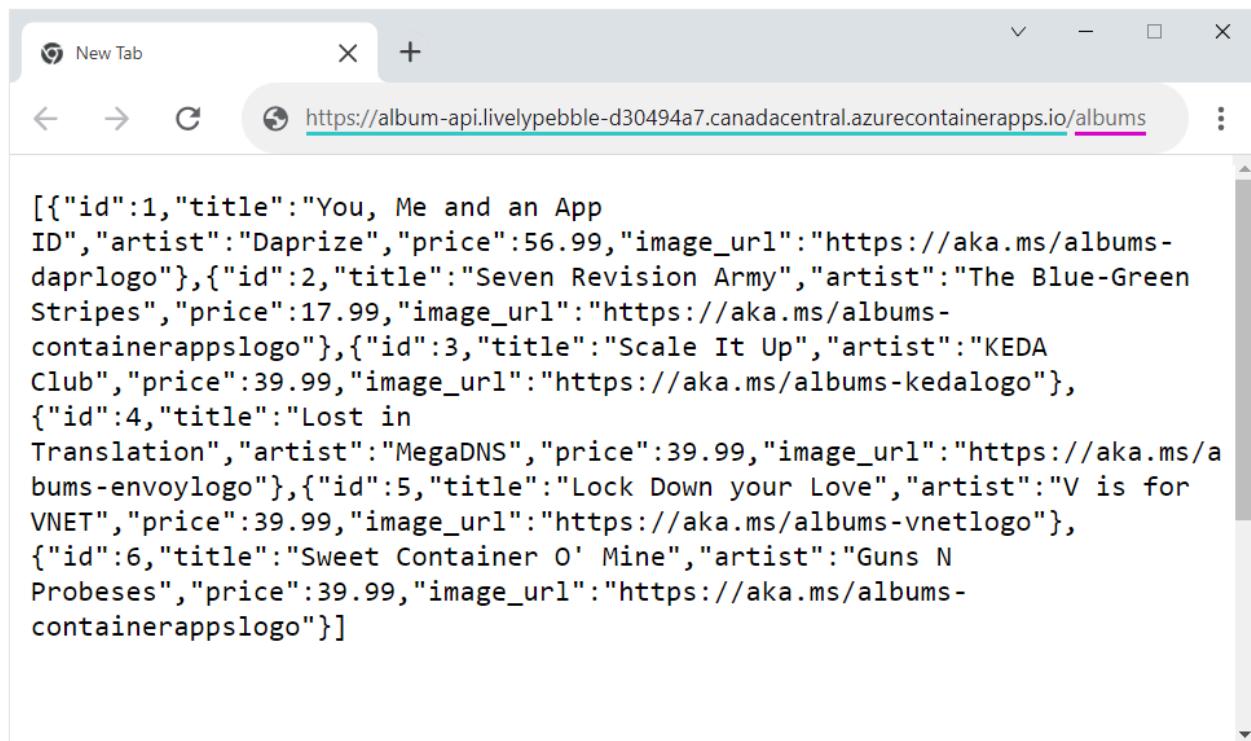
[Environments in Azure Container Apps](#)

Quickstart: Deploy an artifact file to Azure Container Apps (preview)

Article • 08/04/2024

In this quickstart, you learn to deploy a container app from a prebuilt artifact file. The example in this article deploys a Java application using a JAR file, which includes a Java-specific manifest file. Your job is to create a backend web API service that returns a static collection of music albums. After completing this quickstart, you can continue to [Communication between microservices](#) to learn how to deploy a front end application that calls the API.

The following screenshot shows the output from the album API service you deploy.



Prerequisites

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

Requirement	Instructions
Azure account	If you don't have one, create an account for free . You need the <i>Contributor</i> or <i>Owner</i> permission on the Azure subscription to proceed. Refer to Assign Azure roles using the Azure portal for details.

Requirement	Instructions
GitHub Account	Get one for free ↗ .
git	Install git ↗
Azure CLI	Install the Azure CLI .
Java	Install the JDK , recommend 17, or later
Maven	Install the Maven ↗ .

Setup

To sign in to Azure from the CLI, run the following command and follow the prompts to complete the authentication process.

```
Bash
az login
```

To ensure you're running the latest version of the CLI, run the upgrade command.

```
Bash
az upgrade
```

Next, install or update the Azure Container Apps extension for the CLI.

If you receive errors about missing parameters when you run `az containerapp` commands in Azure CLI or cmdlets from the `Az.App` module in Azure PowerShell, be sure you have the latest version of the Azure Container Apps extension installed.

```
Bash
az extension add --name azcontainerapp
```

```
az extension add --name containerapp --upgrade
```

ⓘ Note

Starting in May 2024, Azure CLI extensions no longer enable preview features by default. To access Container Apps [preview features](#), install the Container Apps extension with `--allow-preview true`.

Azure CLI

```
az extension add --name containerapp --upgrade --allow-preview true
```

Now that the current extension or module is installed, register the `Microsoft.App` and `Microsoft.OperationalInsights` namespaces.

Bash

Azure CLI

```
az provider register --namespace Microsoft.App
```

Azure CLI

```
az provider register --namespace Microsoft.OperationalInsights
```

Create environment variables

Now that your Azure CLI setup is complete, you can define the environment variables that are used throughout this article.

Bash

Define the following variables in your bash shell.

Azure CLI

```
RESOURCE_GROUP="album-containerapps"  
LOCATION="canadacentral"  
ENVIRONMENT="env-album-containerapps"
```

```
API_NAME="album-api"
SUBSCRIPTION=<YOUR_SUBSCRIPTION_ID>
```

If necessary, you can query for your subscription ID.

Azure CLI

```
az account list --output table
```

Prepare the GitHub repository

Begin by cloning the sample repository.

Use the following git command to clone the sample app into the *code-to-cloud* folder:

```
git
```

```
git clone https://github.com/azure-samples/containerapps-albumapi-java code-to-cloud
```

```
git
```

```
cd code-to-cloud
```

Build a JAR file

ⓘ Note

The Java sample only supports a Maven build, which results in an executable JAR file. The build uses default settings as passing in environment variables is unsupported.

Build the project with [Maven ↗](#).

Bash

Azure CLI

```
mvn clean package -DskipTests
```

Run the project locally

Bash

Azure CLI

```
java -jar target\containerapps-albumapi-java-0.0.1-SNAPSHOT.jar
```

To verify application is running, open a browser and go to

<http://localhost:8080/albums>. The page returns a list of the JSON objects.

Deploy the artifact

Build and deploy your first container app from your local JAR file with the `containerapp up` command.

This command:

- Creates the resource group
- Creates an Azure Container Registry
- Builds the container image and push it to the registry
- Creates the Container Apps environment with a Log Analytics workspace
- Creates and deploys the container app using a public container image

The `up` command uses the Docker file in the root of the repository to build the container image. The `EXPOSE` instruction in the Docker file defines the target port. A Docker file, however, isn't required to build a container app.

ⓘ Note

Note: When using `containerapp up` in combination with a Docker-less code base, use the `--location` parameter so that application runs in a location other than US East.

Bash

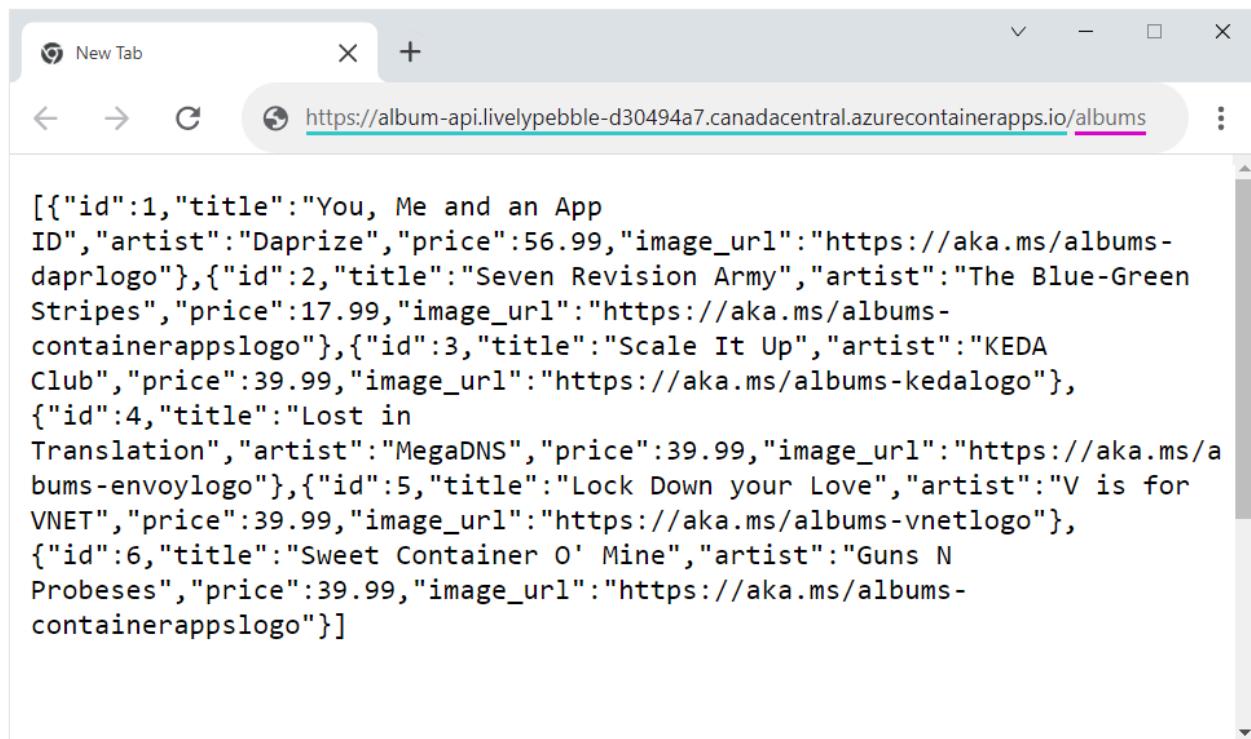
Azure CLI

```
az containerapp up \
--name $API_NAME \
```

```
--resource-group $RESOURCE_GROUP \
--location $LOCATION \
--environment $ENVIRONMENT \
--artifact ./target/containerapps-albumapi-java-0.0.1-SNAPSHOT.jar \
--ingress external \
--target-port 8080 \
--subscription $SUBSCRIPTION
```

Verify deployment

Copy the FQDN to a web browser. From your web browser, go to the `/albums` endpoint of the FQDN.



Deploy a WAR file

You can also deploy your container app from a [WAR file](#).

Clean up resources

If you're not going to continue to use this application, you can delete the Azure Container Apps instance and all the associated services by removing the resource group.

Follow these steps to remove the resources you created:

Bash

Azure CLI

```
az group delete \
--resource-group $RESOURCE_GROUP
```



Having issues? Let us know on GitHub by opening an issue in the [Azure Container Apps repo](#).

Next steps

[Learn more about developing in Java on Container Apps](#)

Feedback

Was this page helpful?

Yes

No

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

Create a job with Azure Container Apps using the Azure portal

Article • 08/29/2023

Azure Container Apps [jobs](#) allow you to run containerized tasks that execute for a finite duration and exit. You can trigger a job manually, schedule their execution, or trigger their execution based on events.

Jobs are best suited to for tasks such as data processing, machine learning, or any scenario that requires on-demand processing.

In this quickstart, you create a scheduled job. To learn how to create an event-driven job, see [Deploy an event-driven job with Azure Container Apps](#).

Prerequisites

An Azure account with an active subscription is required. If you don't already have one, you can [create an account for free](#). Also, please make sure to have the Resource Provider "Microsoft.App" registered.

Setup

Begin by signing in to the [Azure portal](#).

Create a container app

To create your Container Apps job, start at the Azure portal home page.

1. Search for **Container App Jobs** in the top search bar.
2. Select **Container App Jobs** in the search results.
3. Select the **Create** button.

Basics tab

In the *Basics* tab, do the following actions.

1. Enter the following values in the *Project details* section.

Setting	Action
Subscription	Select your Azure subscription.
Resource group	Select Create new and enter jobs-quickstart .
Container job name	Enter my-job .

Create an environment

Next, create an environment for your container app.

1. Select the appropriate region.

Setting	Value
Region	Select Central US .

2. In the *Create Container Apps environment* field, select the **Create new** link.
3. In the *Create Container Apps Environment* page, on the *Basics* tab, enter the following values:

Setting	Value
Environment name	Enter my-environment .
Type	Enter Workload Profile .
Zone redundancy	Select Disabled

4. Select the **Create** button at the bottom of the *Create Container App Environment* page.

Deploy the job

1. In *Job details*, select **Scheduled** for the *Trigger type*.

In the *Cron expression* field, enter `*/1 * * * *`.

This expression starts the job every minute.

2. Select the **Next: Container** button at the bottom of the page.
3. In the *Container* tab, enter the following values:

Setting	Value
Name	Enter main-container .
Image source	Select Docker Hub or other registries .
Image type	Select Public .
Registry login server	Enter mcr.microsoft.com .
Image and tag	Enter k8se/quickstart-jobs:latest .
Workload profile	Select Consumption .
CPU and memory	Select 0.25 and 0.5Gi .

4. Select the **Review and create** button at the bottom of the page.

As the settings in the job are verified, if no errors are found, the *Create* button is enabled.

Any errors appear on a tab marked with a red dot. If you encounter errors, navigate to the appropriate tab and you'll find fields containing errors highlighted in red. Once all errors are fixed, select **Review and create** again.

5. Select **Create**.

A page with the message *Deployment is in progress* is displayed. Once the deployment is successfully completed, you'll see the message: *Your deployment is complete*.

Verify deployment

1. Select **Go to resource** to view your new Container Apps job.

2. Select the **Execution history** tab.

The *Execution history* tab displays the status of each job execution. Select the **Refresh** button to update the list. Wait up to a minute for the scheduled job execution to start. Its status changes from *Pending* to *Running* to *Succeeded*.

3. Select **View logs**.

The logs show the output of the job execution. It may take a few minutes for the logs to appear.

Clean up resources

If you're not going to continue to use this application, you can delete the Azure Container Apps instance and all the associated services by removing the resource group.

1. Select the **jobs-quickstart** resource group from the *Overview* section.
2. Select the **Delete resource group** button at the top of the resource group *Overview*.
3. Enter the resource group name **jobs-quickstart** in the *Are you sure you want to delete "jobs-quickstart"* confirmation dialog.
4. Select **Delete**.

The process to delete the resource group may take a few minutes to complete.

💡 Tip

Having issues? Let us know on GitHub by opening an issue in the [Azure Container Apps repo](#) ↗.

Next steps

[Container Apps jobs](#)

Create a job with Azure Container Apps

Article • 08/09/2024

Azure Container Apps [jobs](#) allow you to run containerized tasks that execute for a finite duration and exit. You can trigger a job manually, schedule their execution, or trigger their execution based on events.

Jobs are best suited to for tasks such as data processing, machine learning, resource cleanup, or any scenario that requires on-demand processing.

In this quickstart, you create a manual or scheduled job. To learn how to create an event-driven job, see [Deploy an event-driven job with Azure Container Apps](#).

Prerequisites

- An Azure account with an active subscription.
 - If you don't have one, you [can create one for free ↗](#).
- Install the [Azure CLI](#).
- Refer to [jobs restrictions](#) for a list of limitations.

Setup

1. To sign in to Azure from the CLI, run the following command and follow the prompts to complete the authentication process.

```
Azure CLI
```

```
az login
```

2. Ensure you're running the latest version of the CLI via the upgrade command.

```
Azure CLI
```

```
az upgrade
```

3. Install the latest version of the Azure Container Apps CLI extension.

```
Azure CLI
```

```
az extension add --name containerapp --upgrade
```

4. Register the `Microsoft.App` and `Microsoft.OperationalInsights` namespaces if you haven't already registered them in your Azure subscription.

```
Azure CLI
```

```
az provider register --namespace Microsoft.App  
az provider register --namespace Microsoft.OperationalInsights
```

5. Now that your Azure CLI setup is complete, you can define the environment variables that are used throughout this article.

```
Azure CLI
```

```
RESOURCE_GROUP="jobs-quickstart"  
LOCATION="northcentralus"  
ENVIRONMENT="env-jobs-quickstart"  
JOB_NAME="my-job"
```

Create a Container Apps environment

The Azure Container Apps environment acts as a secure boundary around container apps and jobs so they can share the same network and communicate with each other.

1. Create a resource group using the following command.

```
Azure CLI
```

```
az group create \  
  --name "$RESOURCE_GROUP" \  
  --location "$LOCATION"
```

2. Create the Container Apps environment using the following command.

```
Azure CLI
```

```
az containerapp env create \  
  --name "$ENVIRONMENT" \  
  --resource-group "$RESOURCE_GROUP" \  
  --location "$LOCATION"
```

Create and run a manual job

To use manual jobs, you first create a job with trigger type `Manual` and then start an execution. You can start multiple executions of the same job and multiple job executions can run concurrently.

1. Create a job in the Container Apps environment using the following command.

Azure CLI

```
az containerapp job create \
--name "$JOB_NAME" --resource-group "$RESOURCE_GROUP" --
environment "$ENVIRONMENT" \
--trigger-type "Manual" \
--replica-timeout 1800 \
--image "mcr.microsoft.com/k8se/quickstart-jobs:latest" \
--cpu "0.25" --memory "0.5Gi"
```

Manual jobs don't execute automatically. You must start an execution of the job.

2. Start an execution of the job using the following command.

Azure CLI

```
az containerapp job start \
--name "$JOB_NAME" \
--resource-group "$RESOURCE_GROUP"
```

The command returns details of the job execution, including its name.

List recent job execution history

Container Apps jobs maintain a history of recent executions. You can list the executions of a job.

Azure CLI

```
az containerapp job execution list \
--name "$JOB_NAME" \
--resource-group "$RESOURCE_GROUP" \
--output table \
--query '[].{Status: properties.status, Name: name, StartTime: properties.startTime}'
```

Executions of scheduled jobs appear in the list as they run.

Console

Status	Name	StartTime
Succeeded	my-job-jvsgub6	2023-05-08T21:21:45+00:00

Query job execution logs

Job executions output logs to the logging provider that you configured for the Container Apps environment. By default, logs are stored in Azure Log Analytics.

1. Save the Log Analytics workspace ID for the Container Apps environment to a variable.

Azure CLI

```
LOG_ANALYTICS_WORKSPACE_ID=$(az containerapp env show \
    --name "$ENVIRONMENT" \
    --resource-group "$RESOURCE_GROUP" \
    --query
    "properties.appLogsConfiguration.logAnalyticsConfiguration.customerId"
    \
    --output tsv)
```

2. Save the name of the most recent job execution to a variable.

Azure CLI

```
JOB_EXECUTION_NAME=$(az containerapp job execution list \
    --name "$JOB_NAME" \
    --resource-group "$RESOURCE_GROUP" \
    --query "[0].name" \
    --output tsv)
```

3. Run a query against Log Analytics for the job execution using the following command.

Azure CLI

```
az monitor log-analytics query \
    --workspace "$LOG_ANALYTICS_WORKSPACE_ID" \
    --analytics-query "ContainerAppConsoleLogs_CL | where
    ContainerGroupName_s startswith '$JOB_EXECUTION_NAME' | order by
    _timestamp_d asc" \
    --query "[].Log_s"
```

(!) Note

Until the `ContainerAppConsoleLogs_CL` table is ready, the command returns no results or with an error: `BadArgumentError: The request had some invalid properties.` Wait a few minutes and run the command again.

The following output is an example of the logs printed by the job execution.

JSON

```
[  
    "2023/04/24 18:38:28 This is a sample application that demonstrates  
    how to use Azure Container Apps jobs",  
    "2023/04/24 18:38:28 Starting processing...",  
    "2023/04/24 18:38:33 Finished processing. Shutting down!"  
]
```

Clean up resources

If you're not going to continue to use this application, run the following command to delete the resource group along with all the resources created in this quickstart.

(⊗) Caution

The following command deletes the specified resource group and all resources contained within it. If resources outside the scope of this quickstart exist in the specified resource group, they will also be deleted.

Azure CLI

```
az group delete --name "$RESOURCE_GROUP"
```

💡 Tip

Having issues? Let us know on GitHub by opening an issue in the [Azure Container Apps repo](#).

Next steps

Feedback

Was this page helpful?

 Yes

 No

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

Comparing Container Apps with other Azure container options

Article • 07/12/2024

There are many options for teams to build and deploy cloud native and containerized applications on Azure. This article helps you understand which scenarios and use cases are best suited for Azure Container Apps and how it compares to other container options on Azure including:

- [Azure Container Apps](#)
- [Azure App Service](#)
- [Azure Container Instances](#)
- [Azure Kubernetes Service](#)
- [Azure Functions](#)
- [Azure Spring Apps](#)
- [Azure Red Hat OpenShift](#)

There's no perfect solution for every use case and every team. The following explanation provides general guidance and recommendations as a starting point to help find the best fit for your team and your requirements.

Container option comparisons

Azure Container Apps

[Azure Container Apps](#) enables you to build serverless microservices and jobs based on containers. Distinctive features of Container Apps include:

- Optimized to run general purpose containers, especially for applications that span many microservices deployed in containers.
- Powered by Kubernetes and open-source technologies like [Dapr](#), [KEDA](#), and [envoy](#).
- Supports Kubernetes-style apps and microservices with features like [service discovery](#) and [traffic splitting](#).
- Enables event-driven application architectures by supporting scale based on traffic and pulling from [event sources like queues](#), including [scale to zero](#).
- Supports running on demand, scheduled, and event-driven [jobs](#).

Azure Container Apps doesn't provide direct access to the underlying Kubernetes APIs. If you require access to the Kubernetes APIs and control plane, you should use [Azure](#)

[Kubernetes Service](#). However, if you would like to build Kubernetes-style applications and don't require direct access to all the native Kubernetes APIs and cluster management, Container Apps provides a fully managed experience based on best-practices. For these reasons, many teams prefer to start building container microservices with Azure Container Apps.

You can get started building your first container app [using the quickstarts](#).

Azure App Service

[Azure App Service](#) provides fully managed hosting for web applications including websites and web APIs. You can deploy these web applications using code or containers. Azure App Service is optimized for web applications. Azure App Service is integrated with other Azure services including Azure Container Apps or Azure Functions. When building web apps, Azure App Service is an ideal option.

Azure Container Instances

[Azure Container Instances \(ACI\)](#) provides a single pod of Hyper-V isolated containers on demand. It can be thought of as a lower-level "building block" option compared to Container Apps. Concepts like scale, load balancing, and certificates aren't provided with ACI containers. For example, to scale to five container instances, you create five distinct container instances. Azure Container Apps provide many application-specific concepts on top of containers, including certificates, revisions, scale, and environments. Users often interact with Azure Container Instances through other services. For example, Azure Kubernetes Service can layer orchestration and scale on top of ACI through [virtual nodes](#). If you need a less "opinionated" building block that doesn't align with the scenarios Azure Container Apps is optimizing for, Azure Container Instances is an ideal option.

Azure Kubernetes Service

[Azure Kubernetes Service \(AKS\)](#) provides a fully managed Kubernetes option in Azure. It supports direct access to the Kubernetes API and runs any Kubernetes workload. The full cluster resides in your subscription, with the cluster configurations and operations within your control and responsibility. Teams looking for a fully managed version of Kubernetes in Azure, Azure Kubernetes Service is an ideal option.

Azure Functions

[Azure Functions](#) is a serverless Functions-as-a-Service (FaaS) solution. It's optimized for running event-driven applications using the functions programming model. It shares many characteristics with Azure Container Apps around scale and integration with events, but optimized for ephemeral functions deployed as either code or containers. The Azure Functions programming model provides productivity benefits for teams looking to trigger the execution of your functions on events and bind to other data sources. When building FaaS-style functions, Azure Functions is the ideal option. The Azure Functions programming model is available as a base container image, making it portable to other container based compute platforms allowing teams to reuse code as environment requirements change.

Azure Spring Apps

[Azure Spring Apps](#) is a fully managed service for Spring developers. If you want to run Spring Boot, Spring Cloud or any other Spring applications on Azure, Azure Spring Apps is an ideal option. The service manages the infrastructure of Spring applications so developers can focus on their code. Azure Spring Apps provides lifecycle management using comprehensive monitoring and diagnostics, configuration management, service discovery, CI/CD integration, blue-green deployments, and more.

Azure Red Hat OpenShift

[Azure Red Hat OpenShift](#) is an integrated product with Red Hat and Microsoft jointly engineered, operated, and supported. This collaboration provides an integrated product and support experience for running Kubernetes-powered OpenShift. With Azure Red Hat OpenShift, teams can choose their own registry, networking, storage, and CI/CD solutions. Alternatively, they can use the built-in solutions for automated source code management, container and application builds, deployments, scaling, health management, and more from OpenShift. If your team or organization is using OpenShift, Azure Red Hat OpenShift is an ideal option.

Next steps

[Deploy your first container app](#)

Feedback

Was this page helpful?

 Yes

 No

Provide product feedback  | Get help at Microsoft Q&A

Azure Container Apps plan types

Article • 10/08/2024

Azure Container Apps features two different plan types.

[+] Expand table

Plan type	Description
Dedicated	Fully managed environment with support for scale-to-zero and pay only for resources your apps use. Optionally, run apps with customized hardware and increased cost predictability using Dedicated workload profiles environment .
Consumption	Serverless environment with support for scale-to-zero and pay only for resources your apps use.

Dedicated

The Dedicated plan consists of a series of workload profiles that range from the default consumption profile to profiles that feature dedicated hardware customized for specialized compute needs.

You can select from general purpose or specialized compute [workload profiles](#) that provide larger amounts of CPU and memory or GPU enabled features. You pay per instance of the workload profile, versus per app, and workload profiles can scale in and out as demand rises and falls.

Use the Dedicated plan when you need any of the following in a single environment:

- **Compute isolation:** Dedicated workload profiles provide access to dedicated hardware with a single tenant guarantee.
- **Customized compute:** Select from many types and sizes of workload profiles based on your apps requirements. You can deploy many apps to each workload profile. Each workload profile can scale independently as more apps are added or removed or as apps scale their replicas up or down.

The Dedicated plan can be more cost effective when you're running higher scale deployments with steady throughput.

ⓘ Note

When configuring your cluster with a user defined route for egress, you must explicitly send egress traffic to a network virtual appliance such as Azure Firewall.

Consumption

The Consumption plan features a serverless architecture that allows your applications to scale in and out on demand. Applications can scale to zero, and you only pay for running apps.

Use the Consumption plan when you don't have specific hardware requirements for your container app.

Next steps

Deploy an app with:

- [Consumption plan](#)
- [Dedicated plan](#)

Feedback

Was this page helpful?

 Yes

 No

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

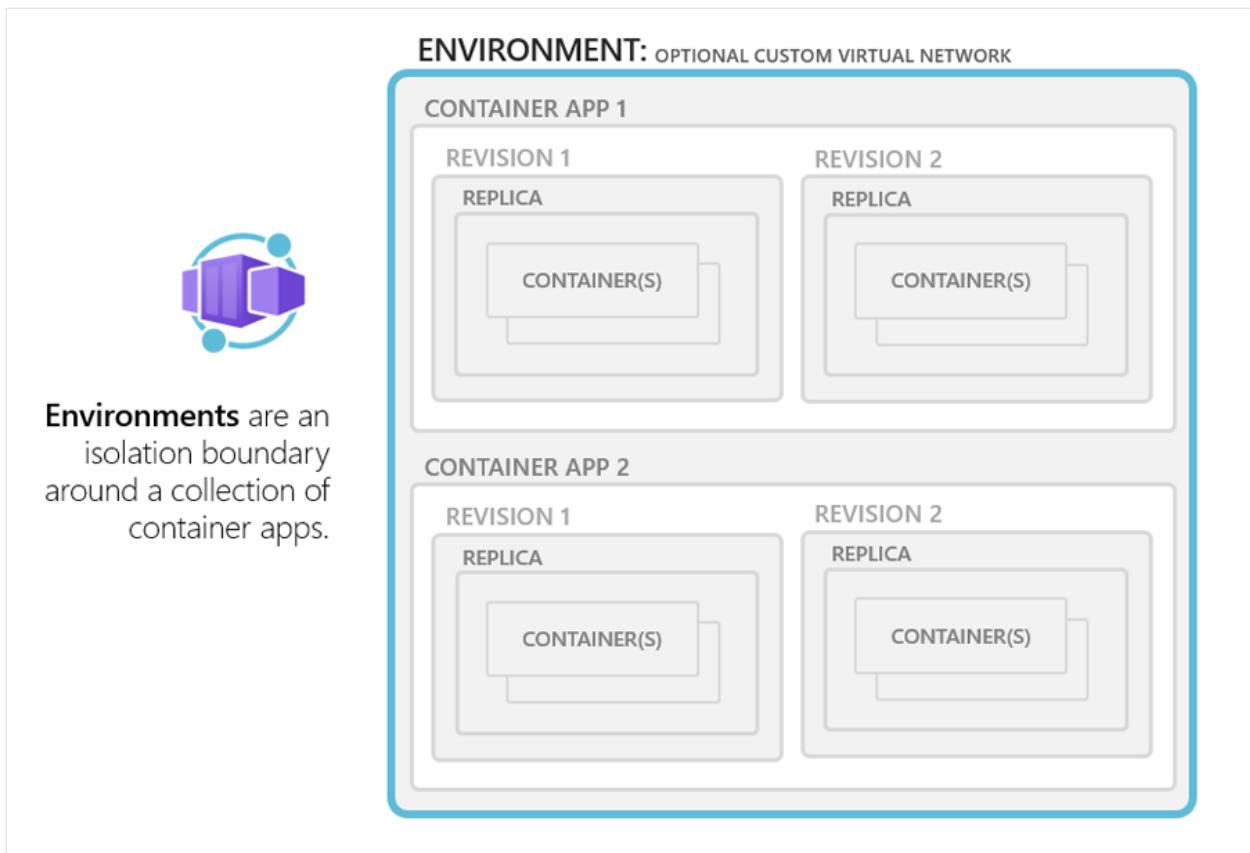
Azure Container Apps environments

Article • 10/25/2023

A Container Apps environment is a secure boundary around one or more container apps and jobs. The Container Apps runtime manages each environment by handling OS upgrades, scale operations, failover procedures, and resource balancing.

Environments include the following features:

Feature	Description
Type	There are two different types of Container Apps environments: Workload profiles environments and Consumption only environments. Workload profiles environments support both the Consumption and Dedicated plans whereas Consumption only environments support only the Consumption plan .
Virtual network	A virtual network supports each environment, which enforces the environment's secure boundaries. As you create an environment, a virtual network that has limited network capabilities is created for you, or you can provide your own. Adding an existing virtual network gives you fine-grained control over your network.
Multiple container apps	When multiple container apps are in the same environment, they share the same virtual network and write logs to the same logging destination.
Multi-service integration	You can add Azure Functions and Azure Spring Apps to your Azure Container Apps environment.



Environments are an isolation boundary around a collection of container apps.

Depending on your needs, you may want to use one or more Container Apps environments. Use the following criteria to help you decide if you should use a single or multiple environments.

Single environment

Use a single environment when you want to:

- Manage related services
- Deploy different applications to the same virtual network
- Instrument Dapr applications that communicate via the Dapr service invocation API
- Have applications share the same Dapr configuration
- Have applications share the same log destination

Multiple environments

Use more than one environment when you want two or more applications to:

- Never share the same compute resources
- Not communicate via the Dapr service invocation API
- Be isolated due to team or environment usage (for example, test vs. production)

Types

Type	Description	Plan	Billing considerations
Workload profile	Run serverless apps with support for scale-to-zero and pay only for resources your apps use with the consumption profile. You can also run apps with customized hardware and increased cost predictability using dedicated workload profiles.	Consumption and Dedicated	You can choose to run apps under either or both plans using separate workload profiles. The Dedicated plan has a fixed cost for the entire environment regardless of how many workload profiles you're using.
Consumption only	Run serverless apps with support for scale-to-zero and pay only for resources your apps use.	Consumption only	Billed only for individual container apps and their resource usage. There's no cost associated with the Container Apps environment.

Logs

Settings relevant to the Azure Container Apps environment API resource.

Property	Description
<code>properties.appLogsConfiguration</code>	Used for configuring the Log Analytics workspace where logs for all apps in the environment are published.
<code>properties.containerAppsConfiguration.daprAIInstrumentationKey</code>	App Insights instrumentation key provided to Dapr for tracing

Policies

Azure Container Apps environments are automatically deleted if one of the following conditions is detected for longer than 90 days:

- In an idle state
- In a failed state due to VNet or Azure Policy configuration
- Blocks infrastructure updates due to VNet or Azure Policy configuration

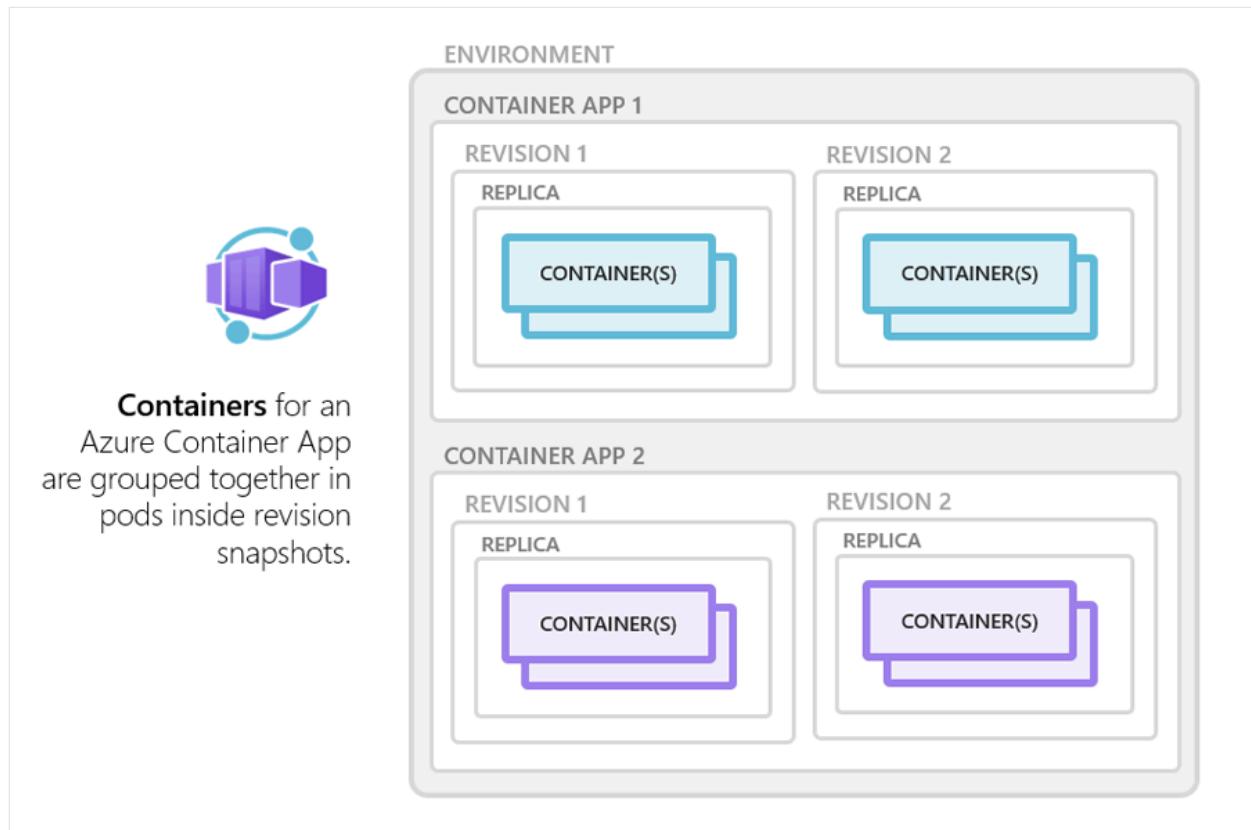
Next steps

Containers

Containers in Azure Container Apps

Article • 09/23/2024

Azure Container Apps manages the details of Kubernetes and container orchestration for you. Containers in Azure Container Apps can use any runtime, programming language, or development stack of your choice.



Azure Container Apps supports:

- Any Linux-based x86-64 (`linux/amd64`) container image
- Containers from any public or private container registry
- Optional [sidecar](#) and [init](#) containers

Features also include:

- Apps use the `template` configuration section to define the container image and other settings. Changes to the `template` configuration section trigger a new [container app revision](#).
- If a container crashes, it automatically restarts.

Jobs features include:

- Job executions use the `template` configuration section to define the container image and other settings when each execution starts.

- If a container exits with a non-zero exit code, the job execution is marked as failed. You can configure a job to retry failed executions.

Configuration

Most container apps have a single container. In advanced scenarios, an app may also have sidecar and init containers. In a container app definition, the main app and its sidecar containers are listed in the `containers` array in the `properties.template` section, and init containers are listed in the `initContainers` array. The following excerpt shows the available configuration options when setting up an app's containers.

JSON

```
{
  "properties": {
    "template": {
      "containers": [
        {
          "name": "main",
          "image": "[parameters('container_image')]",
          "env": [
            {
              "name": "HTTP_PORT",
              "value": "80"
            },
            {
              "name": "SECRET_VAL",
              "secretRef": "mysecret"
            }
          ],
          "resources": {
            "cpu": 0.5,
            "memory": "1Gi"
          },
          "volumeMounts": [
            {
              "mountPath": "/appsettings",
              "volumeName": "appsettings-volume"
            }
          ],
          "probes": [
            {
              "type": "liveness",
              "httpGet": {
                "path": "/health",
                "port": 8080,
                "httpHeaders": [
                  {
                    "name": "Custom-Header",
                    "value": "liveness probe"
                  }
                ]
              }
            }
          ]
        }
      ]
    }
  }
}
```

```
        }
      ],
    },
    "initialDelaySeconds": 7,
    "periodSeconds": 3
  },
  {
    "type": "readiness",
    "tcpSocket": {
      "port": 8081
    },
    "initialDelaySeconds": 10,
    "periodSeconds": 3
  },
  {
    "type": "startup",
    "httpGet": {
      "path": "/startup",
      "port": 8080,
      "httpHeaders": [
        {
          "name": "Custom-Header",
          "value": "startup probe"
        }
      ]
    },
    "initialDelaySeconds": 3,
    "periodSeconds": 3
  }
]
}
],
},
"initContainers": [
{
  "name": "init",
  "image": "[parameters('init_container_image')]",
  "resources": {
    "cpu": 0.25,
    "memory": "0.5Gi"
  },
  "volumeMounts": [
    {
      "mountPath": "/appsettings",
      "volumeName": "appsettings-volume"
    }
  ]
}
]
...
}
...
}
```

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

Setting	Description	Remarks
<code>image</code>	The container image name for your container app.	This value takes the form of <code>repository/<IMAGE_NAME>:<TAG></code> .
<code>name</code>	Friendly name of the container.	Used for reporting and identification.
<code>command</code>	The container's startup command.	Equivalent to Docker's entrypoint field.
<code>args</code>	Start up command arguments.	Entries in the array are joined together to create a parameter list to pass to the startup command.
<code>env</code>	An array of name/value pairs that define environment variables.	Use <code>secretRef</code> instead of the <code>value</code> field to refer to a secret.
<code>resources.cpu</code>	The number of CPUs allocated to the container.	See vCPU and memory allocation requirements
<code>resources.memory</code>	The amount of RAM allocated to the container.	See vCPU and memory allocation requirements
<code>volumeMounts</code>	An array of volume mount definitions.	You can define a temporary or permanent storage volumes for your container. For more information about storage volumes, see Use storage mounts in Azure Container Apps .
<code>probes</code>	An array of health probes enabled in the container.	For more information about probes settings, see Health probes in Azure Container Apps .

vCPU and memory allocation requirements

When you use the Consumption plan, the total CPU and memory allocated to all the containers in a container app must add up to one of the following combinations.

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

vCPUs (cores)	Memory
<code>0.25</code>	<code>0.5Gi</code>

vCPUs (cores)	Memory
0.5	1.0Gi
0.75	1.5Gi
1.0	2.0Gi
1.25	2.5Gi
1.5	3.0Gi
1.75	3.5Gi
2.0	4.0Gi
2.25	4.5Gi
2.5	5.0Gi
2.75	5.5Gi
3.0	6.0Gi
3.25	6.5Gi
3.5	7.0Gi
3.75	7.5Gi
4.0	8.0Gi

ⓘ Note

Apps using the Consumption plan in a *Consumption only* environment are limited to a maximum of 2 cores and 4Gi of memory.

Multiple containers

In advanced scenarios, you can run multiple containers in a single container app. Use this pattern only in specific instances where your containers are tightly coupled.

For most microservice scenarios, the best practice is to deploy each service as a separate container app.

Multiple containers in the same container app share hard disk and network resources and experience the same [application lifecycle](#).

There are two ways to run additional containers in a container app: [sidecar containers](#) and [init containers](#).

Sidecar containers

You can define multiple containers in a single container app to implement the [sidecar pattern](#).

Examples of sidecar containers include:

- An agent that reads logs from the primary app container on a [shared volume](#) and forwards them to a logging service.
- A background process that refreshes a cache used by the primary app container in a shared volume.

These scenarios are examples, and don't represent the only ways you can implement a sidecar.

To run multiple containers in a container app, add more than one container in the `containers` array of the container app template.

Init containers

You can define one or more [init containers](#) in a container app. Init containers run before the primary app container and are used to perform initialization tasks such as downloading data or preparing the environment.

Init containers are defined in the `initContainers` array of the container app template. The containers run in the order they're defined in the array and must complete successfully before the primary app container starts.

Note

Init containers in apps using the Dedicated plan or running in a *Consumption only* environment can't access managed identity at run time.

Container registries

You can deploy images hosted on private registries by providing credentials in the Container Apps configuration.

To use a container registry, you define the registry in the `registries` array in the `properties.configuration` section of the container app resource template. The `passwordSecretRef` field identifies the name of the secret in the `secrets` array name where you defined the password.

JSON

```
{  
  ...  
  "registries": [  
    {"  
      "server": "docker.io",  
      "username": "my-registry-user-name",  
      "passwordSecretRef": "my-password-secret-name"  
    }]  
}
```

Saved credentials are used to pull a container image from the private registry as your app is deployed.

The following example shows how to configure Azure Container Registry credentials in a container app.

JSON

```
{  
  ...  
  "configuration": {  
    "secrets": [  
      {  
        "name": "docker-hub-password",  
        "value": "my-docker-hub-password"  
      }  
    ],  
    ...  
    "registries": [  
      {  
        "server": "docker.io",  
        "username": "someuser",  
        "passwordSecretRef": "docker-hub-password"  
      }  
    ]  
  }  
}
```

ⓘ Note

Docker Hub [limits](#) the number of Docker image downloads. When the limit is reached, containers in your app will fail to start. Use a registry with sufficient limits,

such as [Azure Container Registry](#) to avoid this problem.

Managed identity with Azure Container Registry

You can use an Azure managed identity to authenticate with Azure Container Registry instead of using a username and password. For more information, see [Managed identities in Azure Container Apps](#).

To use managed identity with a registry, the identity must be enabled in the app and it must be assigned `acrPull` role in the registry. To configure the registry, use the managed identity resource ID for a user-assigned identity, or `system` for the system-assigned identity in the `identity` property of the registry. Don't configure a username and password when using managed identity.

JSON

```
{  
    "identity": {  
        "type": "SystemAssigned,UserAssigned",  
        "userAssignedIdentities": {  
            "<IDENTITY1_RESOURCE_ID>": {}  
        }  
    }  
    "properties": {  
        "configuration": {  
            "registries": [  
                {  
                    "server": "myacr1.azurecr.io",  
                    "identity": "<IDENTITY1_RESOURCE_ID>"  
                },  
                {  
                    "server": "myacr2.azurecr.io",  
                    "identity": "system"  
                }  
            ]  
        }  
        ...  
    }  
}
```

For more information about configuring user-assigned identities, see [Add a user-assigned identity](#).

Limitations

Azure Container Apps has the following limitations:

- **Privileged containers:** Azure Container Apps doesn't allow privileged containers mode with host-level access.
- **Operating system:** Linux-based (`linux/amd64`) container images are required.
- **Maximum image size:**
 - Consumption workload profile supports container images totaling up to 8GB for each app or job replica.
 - Dedicated workload profiles support larger container images. Because a Dedicated workload profile can run multiple apps or jobs, multiple container images share the available disk space. The actual supported image size varies based on resources consumed by other apps and jobs.

Next steps

Revisions

Feedback

Was this page helpful?

 Yes

 No

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

Update and deploy changes in Azure Container Apps

Article • 02/01/2024

Change management can be challenging as you develop containerized applications in the cloud. Ultimately, you need the support to track changes, ensure uptime, and have mechanisms to handle smooth rollbacks.

Change management in Azure Container Apps is powered by revisions, which are a snapshot of each version of your container app.

Key characteristics of revisions include:

- **Immutable:** Once established, a revision remains unchangeable.
- **Versioned:** Revisions act as a record of the container app's versions, capturing its state at various stages.
- **Automatically provisioned:** When you deploy a container app for the first time, an initial revision is automatically created.
- **Scoped changes:** While revisions remain static, [application-scope](#) changes can affect all revisions, while [revision-scope](#) changes create a new revision.
- **Historical record:** By default, you have access to 100 inactive revisions, but you can [adjust this threshold manually](#).
- **Multiple revisions:** You can run multiple revisions concurrently. This feature is especially beneficial when you need to manage different versions of your app simultaneously.

Lifecycle

Each revision undergoes specific states, influenced by its status and availability. During its lifecycle, a container app goes through different provisioning, running, and an inactive status.

Provisioning status

When you create a new revision, the container app undergoes startup and readiness checks. During this phase, the provisioning status serves as a guide to track the

container app's progress.

 Expand table

Status	Description
Provisioning	The revision is in the verification process.
Provisioned	The revision has successfully passed all checks.
Provisioning failed	The revision encountered issues during verification.

Running status

After a container app is successfully provisioned, a revision enters its operating phase. The running status helps monitor a container app's health and functionality.

 Expand table

Status	Description
Provisioning	The revision is in the verification process.
Scale to 0	Zero running replicas, and not provisioning any new replicas. The container app can create new replicas if scale rules are triggered.
Activating	Zero running replicas, one replica being provisioned.
Activation failed	The first replica failed to provision.
Scaling / Processing	Scaling in or out is occurring. One or more replicas are running, while other replicas are being provisioned.
Running	One or more replicas are running. There are no issues to report.
Running (at max)	The maximum number of replicas (according to the scale rules of the revision) are running. There are no issues to report.
Deprovisioning	The revision is transitioning from active to inactive, and is removing any resources it has created.
Degraded	At least one replica in the revision is in a failed state. View running state details for specific issues.
Failed	Critical errors caused revisions to fail. The <i>running state</i> provides details. Common causes include: <ul style="list-style-type: none">TerminationExit code 137

Inactive status

Revisions can also enter an inactive state. These revisions don't possess provisioning or running states. However, Azure Container Apps maintains a list of these revisions, accommodating up to 100 inactive entries. You can activate a revision at any time.

Change inactive revision limit

You can use the `--max-inactive-revisions` parameter with the `containerapp create` or `containerapp update` commands to control the number of inactive revisions tracked by Container Apps.

This example demonstrates how to create a new container app that tracks 50 inactive revisions:

Azure CLI

```
az containerapp create --max-inactive-revisions 50
```

Revision modes

Azure Container Apps support two revision modes. Your choice of mode determines how many revisions of your app are simultaneously active.

[+] Expand table

Revision modes	Description	Default
Single	New revisions are automatically provisioned, activated, and scaled to the desired size. Once all the replicas are running as defined by the scale rule , then traffic is diverted from the old version to the new one. If an update fails, traffic remains pointed to the old revision. Old revisions are automatically deprovisioned.	Yes
Multiple	You can have multiple active revisions, split traffic between revisions, and choose when to deprovision old revisions. This level of control is helpful for testing multiple versions of an app, blue-green testing, or taking full control of app updates. Refer to traffic splitting for more detail.	

Labels

For container apps with external HTTP traffic, labels direct traffic to specific revisions. A label provides a unique URL that you can use to route traffic to the revision that the label is assigned.

To switch traffic between revisions, you can move the label from one revision to another.

- Labels keep the same URL when moved from one revision to another.
- A label can be applied to only one revision at a time.
- Allocation for traffic splitting isn't required for revisions with labels.
- Labels are most useful when the app is in *multiple revision mode*.
- You can enable labels, traffic splitting or both.

Labels are useful for testing new revisions. For example, when you want to give access to a set of test users, you can give them the label's URL. Then when you want to move your users to a different revision, you can move the label to that revision.

Labels work independently of traffic splitting. Traffic splitting distributes traffic going to the container app's application URL to revisions based on the percentage of traffic. When traffic is directed to a label's URL, the traffic is routed to one specific revision.

A label name must:

- Consist of lower case alphanumeric characters or dashes (-)
- Start with an alphabetic character
- End with an alphanumeric character

Labels must not:

- Have two consecutive dashes (--)
- Be more than 64 characters

You can manage labels from your container app's **Revision management** page in the Azure portal.

Name	Date created	Provision Status	Label	Traffic ↓	Active
album-api--v3	5/3/2022, 12:29...	Provisioned	label-1	0 %	<input checked="" type="checkbox"/>
album-api--v2	5/2/2022, 12:44...	Provisioned		100 %	<input checked="" type="checkbox"/>

The label URL is available in the revision details pane.

Revision details		X
Revision name	album-api--v3	
Revision URL	album-api--v3.grayplant-a453c292.canadacentral.azurecontainerapps.io	
Label URL	album-api---label-1.grayplant-a453c292.canadacentral.azurecontainerapps.io	
Status	Active	
Time created	5/3/2022, 12:29:11 PM	
Traffic	0%	
Containers	View details	
Scale	View details	
Logs	View details	

Zero downtime deployment

In *single revision mode*, Container Apps ensures your app doesn't experience downtime when creating a new revision. The existing active revision isn't deactivated until the new revision is ready.

If ingress is enabled, the existing revision continues to receive 100% of the traffic until the new revision is ready.

A new revision is considered ready when:

- The revision has provisioned successfully
- The revision has scaled up to match the previous revision's replica count (respecting the new revision's min and max replica count)
- All the replicas have passed their startup and readiness probes

In *multiple revision mode*, you can control when revisions are activated or deactivated and which revisions receive ingress traffic. If a [traffic splitting rule](#) is configured with `latestRevision` set to `true`, traffic doesn't switch to the latest revision until it's ready.

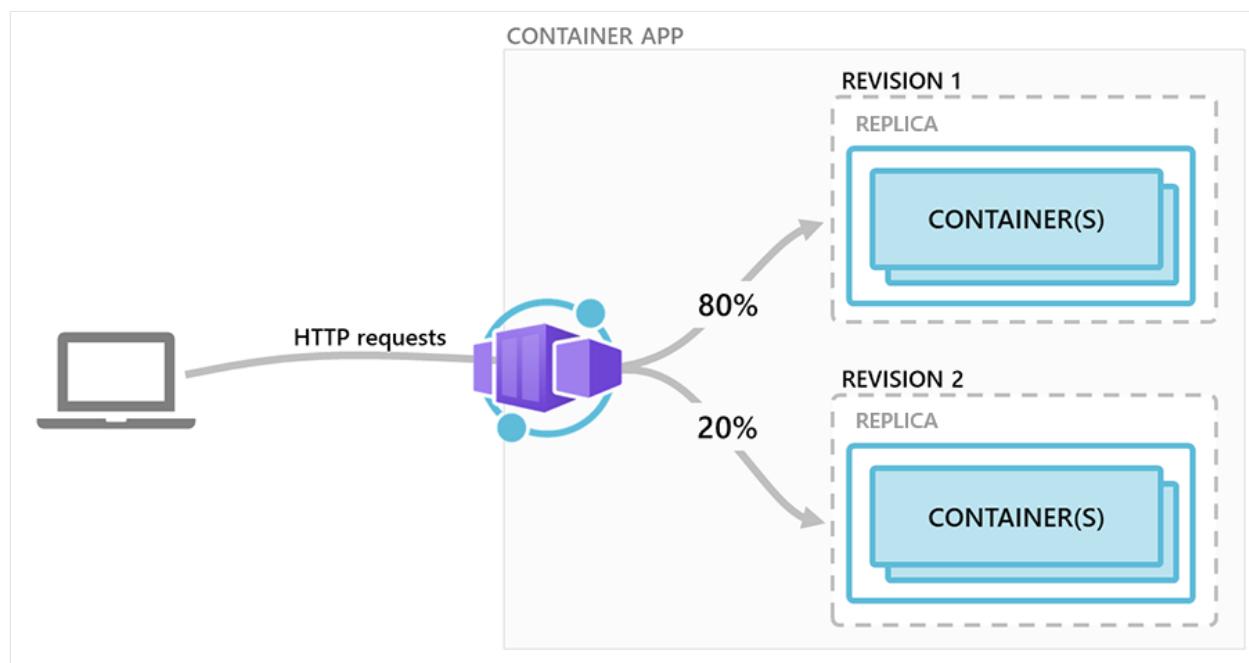
Work with multiple revisions

While single revision mode is the default, sometimes you might want to have full control over how your revisions are managed.

Multiple revision mode gives you the flexibility to manage your revision manually. For instance, using multiple revision mode allows you to decide exactly how much traffic is allocated to each revision.

Traffic splitting

The following diagram shows a container app with two revisions.



This scenario presumes the container app is in the following state:

- [Ingress](#) is enabled, making the container app available via HTTP or TCP.
- The first revision was deployed as *Revision 1*.
- After the container was updated, a new revision was activated as *Revision 2*.
- [Traffic splitting](#) rules are configured so that *Revision 1* receives 80% of the requests, and *Revision 2* receives the remaining 20%.

Direct revision access

Rather than using a routing rule to divert traffic to a revision, you might want to make a revision available to requests for a specific URL. Multiple revision mode can allow you to send all requests coming in to your domain to the latest revision, while requests for an older revision are available via [labels](#) for direct access.

Activation state

In multiple revision mode, you can activate or deactivate revisions as needed. Active revisions are operational and can handle requests, while inactive revisions remain dormant.

Container Apps doesn't charge for inactive revisions. However, there's a cap on the total number of available revisions, with the oldest ones being purged once you exceed a count of 100.

Change types

Changes to a container app fall under two categories: *revision-scope* or *application-scope* changes. *Revision-scope* changes trigger a new revision when you deploy your app, while *application-scope* changes don't.

Revision-scope changes

A new revision is created when a container app is updated with *revision-scope* changes. The changes are limited to the revision in which they're deployed, and don't affect other revisions.

A *revision-scope* change is any change to the parameters in the [properties.template](#) section of the container app resource template.

These parameters include:

- [Revision suffix](#)
- Container configuration and images
- Scale rules for the container application

Application-scope changes

When you deploy a container app with *application-scope* changes:

- The changes are globally applied to all revisions.
- A new revision isn't created.

Application-scope changes are defined as any change to the parameters in the [properties.configuration](#) section of the container app resource template.

These parameters include:

- [Secret values](#) (revisions must be restarted before a container recognizes new secret values)

- Revision mode
- Ingress configuration including:
 - Turning [ingress](#) on or off
 - [Traffic splitting rules](#)
 - Labels
- Credentials for private container registries
- Dapr settings

Customize revisions

You can customize the revision name and labels to better align with your naming conventions or versioning strategy.

Name suffix

Every revision in Container Apps is assigned a unique identifier. While names are automatically generated, you can personalize the revision name.

The typical format for a revision name is:

text

```
<CONTAINER_APP_NAME>-<REVISION_SUFFIX>
```

For example, if you have a container app named *album-api* and decide on the revision suffix *first-revision*, the complete revision name becomes *album-api-first-revision*.

A revision suffix name must:

- Consist of only lower case alphanumeric characters or dashes (-)
- Start with an alphabetic character
- End with an alphanumeric character

Names must not have:

- Two consecutive dashes (--)
- Be more than 64 characters

You can set the revision suffix in the [ARM template](#), through the Azure CLI `az containerapp create` and `az containerapp update` commands, or when creating a revision via the Azure portal.

Use cases

The following are common use cases for using revisions in container apps. This list isn't an exhaustive list of the purpose or capabilities of using Container Apps revisions.

Release management

Revisions streamline the process of introducing new versions of your app. When you're ready to roll out an update or a new feature, you can create a new revision without affecting the current live version. This approach ensures a smooth transition and minimizes disruptions for end-users.

Reverting to previous versions

Sometimes you need to quickly revert to a previous, stable version of your app. You can roll back to a previous revision of your container app if necessary.

A/B testing

When you want to test different versions of your app, revisions can support [A/B testing](#). You can route a subset of your users to a new revision, gather feedback, and make informed decisions based on real-world data.

Blue-green deployments

Revisions support the [blue-green deployment](#) strategy. By having two parallel revisions (blue for the live version and green for the new one), you can gradually phase in a new revision. Once you're confident in the new version's stability and performance, you can switch traffic entirely to the green environment.

Next steps

[Application lifecycle management](#)

Application lifecycle management in Azure Container Apps

Article • 05/28/2024

The Azure Container Apps application lifecycle revolves around [revisions](#).

When you deploy a container app, the first revision is automatically created. [More revisions are created](#) as [containers](#) change, or any adjustments are made to the [template](#) section of the configuration.

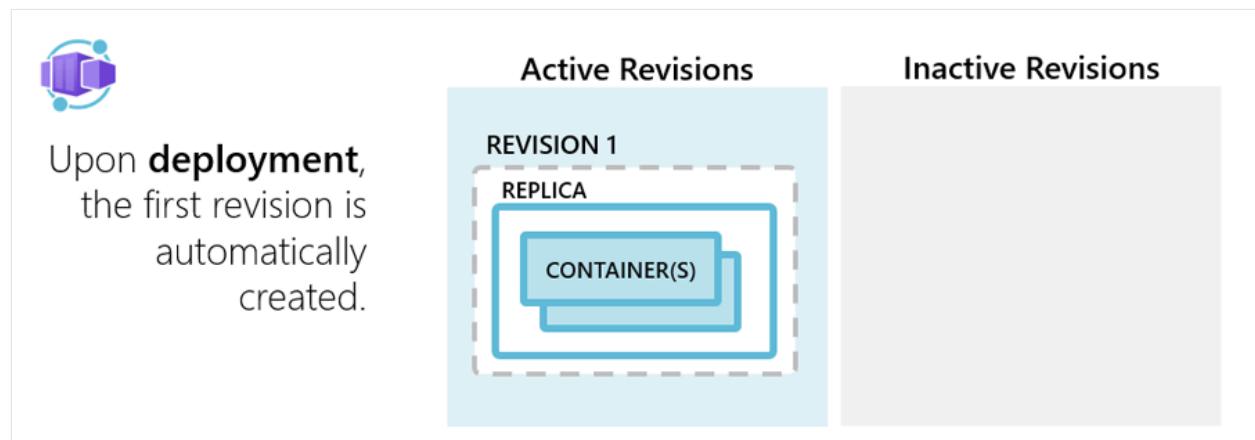
A container app flows through four phases: deployment, update, deactivation, and shut down.

ⓘ Note

[Azure Container Apps jobs](#) don't support revisions. Jobs are deployed and updated directly.

Deployment

As a container app is deployed, the first revision is automatically created.

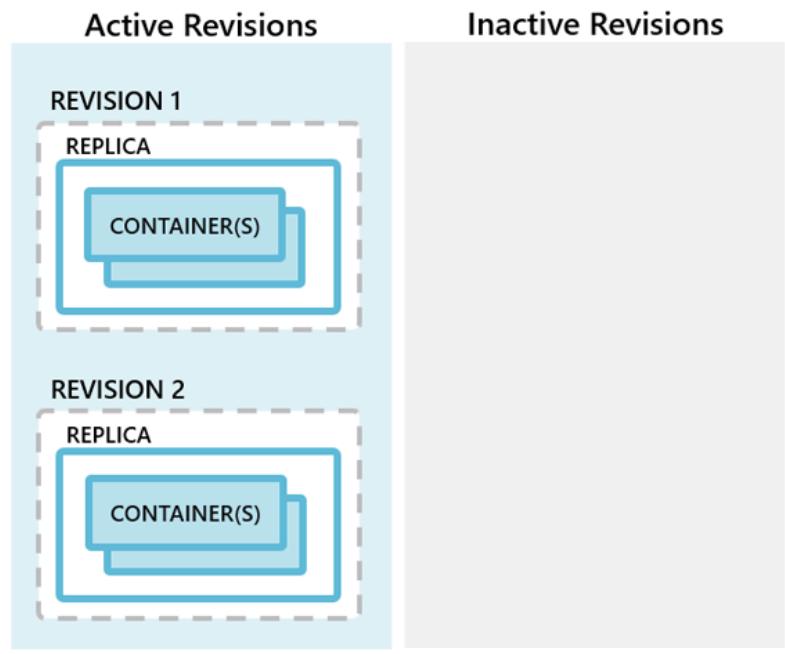


Update

As a container app is updated with a [revision scope-change](#), a new revision is created. You can [choose](#) whether to automatically deactivate old revisions (single revision mode), or allow them to remain available (multiple revision mode).



As the container app is **updated**, a new revision is created.



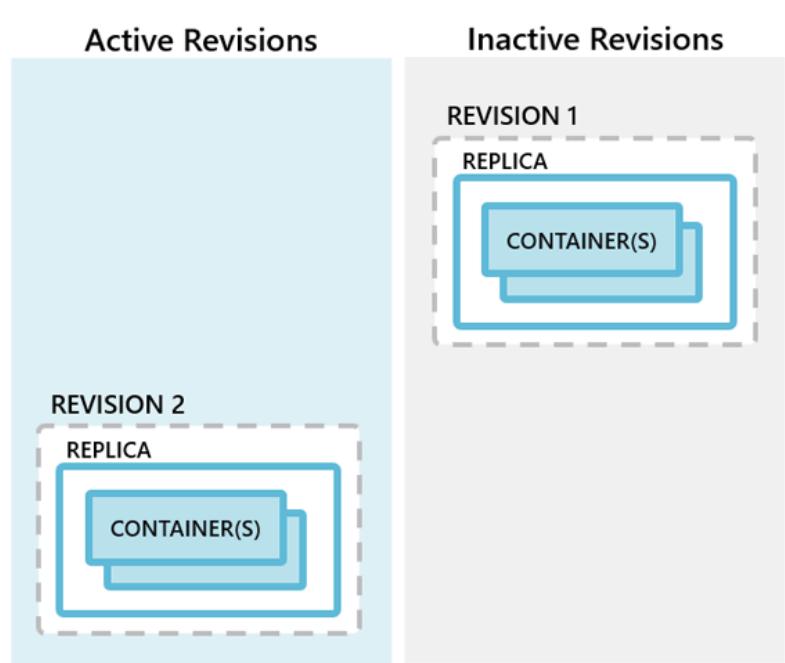
When in single revision mode, Container Apps handles the automatic switch between revisions to support [zero downtime deployment](#).

Deactivate

Once a revision is no longer needed, you can deactivate a revision with the option to reactivate later. During deactivation, containers in the revision are [shut down](#).



Once a revision is no longer needed, you can **deactivate** individual revisions, or choose to automatically deactivate old revisions.



Shutdown

The containers are shut down in the following situations:

- As a container app scales in
- As a container app is being deleted
- As a revision is being deactivated

When a shutdown is initiated, the container host sends a [SIGTERM message](#) to your container. The code implemented in the container can respond to this operating system-level message to handle termination.

If your application doesn't respond within 30 seconds to the `SIGTERM` message, then [SIGKILL](#) terminates your container.

Additionally, make sure your application can gracefully handle shutdowns. Containers restart regularly, so don't expect state to persist inside a container. Instead, use external caches for expensive in-memory cache requirements.

Next steps

[Microservices](#)

Jobs in Azure Container Apps

Article • 04/04/2024

Azure Container Apps jobs enable you to run containerized tasks that execute for a finite duration and exit. You can use jobs to perform tasks such as data processing, machine learning, or any scenario where on-demand processing is required.

Container apps and jobs run in the same [environment](#), allowing them to share capabilities such as networking and logging.

Compare container apps and jobs

There are two types of compute resources in Azure Container Apps: apps and jobs.

Apps are services that run continuously. If a container in an app fails, it's restarted automatically. Examples of apps include HTTP APIs, web apps, and background services that continuously process input.

Jobs are tasks that start, run for a finite duration, and exit when finished. Each execution of a job typically performs a single unit of work. Job executions start manually, on a schedule, or in response to events. Examples of jobs include batch processes that run on demand and scheduled tasks.

Example scenarios

The following table compares common scenarios for apps and jobs:

Expand table

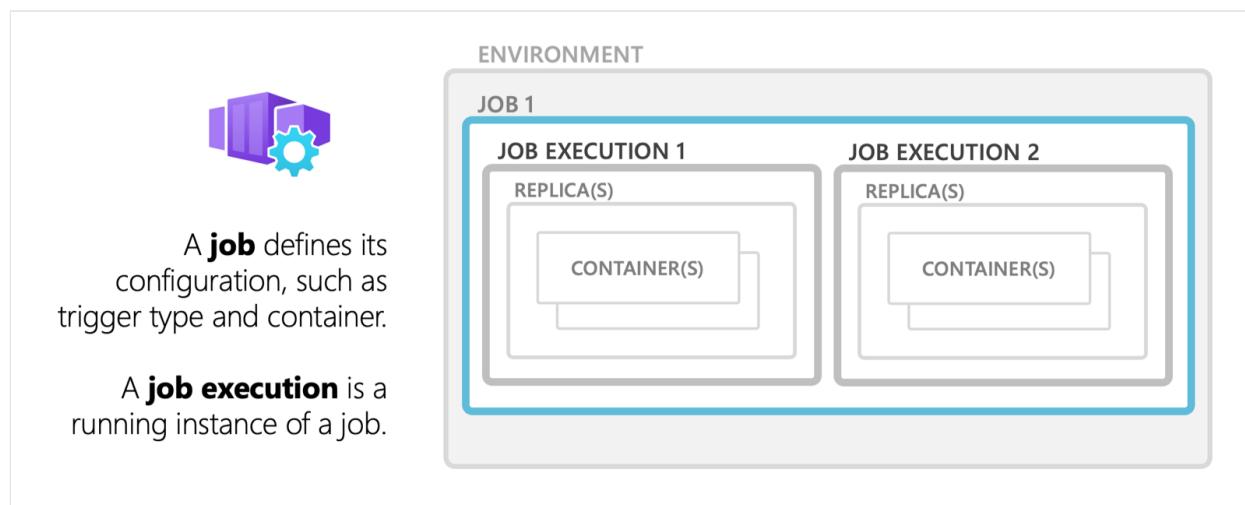
Container	Compute resource	Notes
An HTTP server that serves web content and API requests	App	Configure an HTTP scale rule .
A process that generates financial reports nightly	Job	Use the Schedule job type and configure a cron expression.
A continuously running service that processes messages from an Azure Service Bus queue	App	Configure a custom scale rule .
A job that processes a single message or a small batch of	Job	Use the Event job type and configure a custom scale rule to trigger job executions

Container	Compute resource	Notes
messages from an Azure queue and exits		when there are messages in the queue.
A background task that's triggered on-demand and exits when finished	Job	Use the <i>Manual</i> job type and start executions manually or programmatically using an API.
A self-hosted GitHub Actions runner or Azure Pipelines agent	Job	Use the <i>Event</i> job type and configure a GitHub Actions or Azure Pipelines scale rule.
An Azure Functions app	App	Deploy Azure Functions to Container Apps .
An event-driven app using the Azure WebJobs SDK	App	Configure a scale rule for each event source.

Concepts

A Container Apps environment is a secure boundary around one or more container apps and jobs. Jobs involve a few key concepts:

- **Job:** A job defines the default configuration that is used for each job execution. The configuration includes the container image to use, the resources to allocate, and the command to run.
- **Job execution:** A job execution is a single run of a job that is triggered manually, on a schedule, or in response to an event.
- **Job replica:** A typical job execution runs one replica defined by the job's configuration. In advanced scenarios, a job execution can run multiple replicas.



Job trigger types

A job's trigger type determines how the job is started. The following trigger types are available:

- **Manual**: Manual jobs are triggered on-demand.
- **Schedule**: Scheduled jobs are triggered at specific times and can run repeatedly.
- **Event**: Event-driven jobs are triggered by events such as a message arriving in a queue.

Manual jobs

Manual jobs are triggered on-demand using the Azure CLI, Azure portal, or a request to the Azure Resource Manager API.

Examples of manual jobs include:

- One time processing tasks such as migrating data from one system to another.
- An e-commerce site running as container app starts a job execution to process inventory when an order is placed.

To create a manual job, use the job type `Manual`.

Azure CLI

To create a manual job using the Azure CLI, use the `az containerapp job create` command. The following example creates a manual job named `my-job` in a resource group named `my-resource-group` and a Container Apps environment named `my-environment`:

Azure CLI

```
az containerapp job create \
    --name "my-job" --resource-group "my-resource-group" --environment
"my-environment" \
    --trigger-type "Manual" \
    --replica-timeout 1800 \
    --image "mcr.microsoft.com/k8se/quickstart-jobs:latest" \
    --cpu "0.25" --memory "0.5Gi"
```

The `mcr.microsoft.com/k8se/quickstart-jobs:latest` image is a public sample container image that runs a job that waits a few seconds, prints a message to the console, and then exits. To authenticate and use a private container image, see [Containers](#).

The above command only creates the job. To start a job execution, see [Start a job execution on demand](#).

Scheduled jobs

To create a scheduled job, use the job type `Schedule`.

Container Apps jobs use cron expressions to define schedules. It supports the standard [cron](#) expression format with five fields for minute, hour, day of month, month, and day of week. The following are examples of cron expressions:

[] Expand table

Expression	Description
<code>*/5 * * * *</code>	Runs every 5 minutes.
<code>0 */2 * * *</code>	Runs every two hours.
<code>0 0 * * *</code>	Runs every day at midnight.
<code>0 0 * * 0</code>	Runs every Sunday at midnight.
<code>0 0 1 * *</code>	Runs on the first day of every month at midnight.

Cron expressions in scheduled jobs are evaluated in Coordinated Universal Time (UTC).

Azure CLI

To create a scheduled job using the Azure CLI, use the `az containerapp job create` command. The following example creates a scheduled job named `my-job` in a resource group named `my-resource-group` and a Container Apps environment named `my-environment`:

Azure CLI

```
az containerapp job create \
    --name "my-job" --resource-group "my-resource-group" --environment
"my-environment" \
    --trigger-type "Schedule" \
    --replica-timeout 1800 \
    --image "mcr.microsoft.com/k8se/quickstart-jobs:latest" \
    --cpu "0.25" --memory "0.5Gi" \
    --cron-expression "*/1 * * * *"
```

The `mcr.microsoft.com/k8se/quickstart-jobs:latest` image is a public sample container image that runs a job that waits a few seconds, prints a message to the console, and then exits. To authenticate and use a private container image, see [Containers](#).

The cron expression `*/1 * * * *` runs the job every minute.

Event-driven jobs

Event-driven jobs are triggered by events from supported [custom scalers](#). Examples of event-driven jobs include:

- A job that runs when a new message is added to a queue such as Azure Service Bus, Kafka, or RabbitMQ.
- A self-hosted [GitHub Actions runner](#) or [Azure DevOps agent](#) that runs when a new job is queued in a workflow or pipeline.

Container apps and event-driven jobs use [KEDA](#) scalers. They both evaluate scaling rules on a polling interval to measure the volume of events for an event source, but the way they use the results is different.

In an app, each replica continuously processes events and a scaling rule determines the number of replicas to run to meet demand. In event-driven jobs, each job execution typically processes a single event, and a scaling rule determines the number of job executions to run.

Use jobs when each event requires a new instance of the container with dedicated resources or needs to run for a long time. Event-driven jobs are conceptually similar to [KEDA scaling jobs](#).

To create an event-driven job, use the job type `Event`.

Azure CLI

To create an event-driven job using the Azure CLI, use the `az containerapp job create` command. The following example creates an event-driven job named `my-job` in a resource group named `my-resource-group` and a Container Apps environment named `my-environment`:

Azure CLI

```
az containerapp job create \
    --name "my-job" --resource-group "my-resource-group" --environment
    "my-environment" \
    --trigger-type "Event" \
```

```
--replica-timeout 1800 \
--image "docker.io/myuser/my-event-driven-job:latest" \
--cpu "0.25" --memory "0.5Gi" \
--min-executions "0" \
--max-executions "10" \
--scale-rule-name "queue" \
--scale-rule-type "azure-queue" \
--scale-rule-metadata "accountName=mystorage" "queueName=myqueue"
"queueLength=1" \
--scale-rule-auth "connection=connection-string-secret" \
--secrets "connection-string-secret=<QUEUE_CONNECTION_STRING>"
```

The example configures an Azure Storage queue scale rule.

For a complete tutorial, see [Deploy an event-driven job](#).

Start a job execution on demand

For any job type, you can start a job execution on demand.

Azure CLI

To start a job execution using the Azure CLI, use the `az containerapp job start` command. The following example starts an execution of a job named `my-job` in a resource group named `my-resource-group`:

Azure CLI

```
az containerapp job start --name "my-job" --resource-group "my-resource-group"
```

When you start a job execution, you can choose to override the job's configuration. For example, you can override an environment variable or the startup command to run the same job with different inputs. The overridden configuration is only used for the current execution and doesn't change the job's configuration.

Important

When overriding the configuration, the job's entire template configuration is replaced with the new configuration. Ensure that the new configuration includes all required settings.

Azure CLI

To override the job's configuration while starting an execution, use the `az containerapp job start` command and pass a YAML file containing the template to use for the execution. The following example starts an execution of a job named `my-job` in a resource group named `my-resource-group`.

Retrieve the job's current configuration with the `az containerapp job show` command and save the template to a file named `my-job-template.yaml`:

Azure CLI

```
az containerapp job show --name "my-job" --resource-group "my-resource-group" --query "properties.template" --output yaml > my-job-template.yaml
```

The `--query "properties.template"` option returns only the job's template configuration.

Edit the `my-job-template.yaml` file to override the job's configuration. For example, to override the environment variables, modify the `env` section:

YAML

```
containers:
- name: print-hello
  image: ubuntu
  resources:
    cpu: 1
    memory: 2Gi
  env:
    - name: MY_NAME
      value: Azure Container Apps jobs
  args:
    - /bin/bash
    - -c
    - echo "Hello, $MY_NAME!"
```

Start the job using the template:

Azure CLI

```
az containerapp job start --name "my-job" --resource-group "my-resource-group" \
--yaml my-job-template.yaml
```

Get job execution history

Each Container Apps job maintains a history of recent job executions.

Azure CLI

To get the statuses of job executions using the Azure CLI, use the `az containerapp job execution list` command. The following example returns the status of the most recent execution of a job named `my-job` in a resource group named `my-resource-group`:

Azure CLI

```
az containerapp job execution list --name "my-job" --resource-group "my-resource-group"
```

The execution history for scheduled and event-based jobs is limited to the most recent 100 successful and failed job executions.

To list all executions of a job or to get detailed output from a job, query the logs provider configured for your Container Apps environment.

Advanced job configuration

Container Apps jobs support advanced configuration options such as container settings, retries, timeouts, and parallelism.

Container settings

Container settings define the containers to run in each replica of a job execution. They include environment variables, secrets, and resource limits. For more information, see [Containers](#). Running multiple containers in a single job is an advanced scenario. Most jobs run a single container.

Job settings

The following table includes the job settings that you can configure:

[] Expand table

Setting	Azure Resource Manager property	CLI parameter	Description
Job type	<code>triggerType</code>	<code>--trigger-type</code>	The type of job. (<code>Manual</code> , <code>Schedule</code> , or <code>Event</code>)
Replica timeout	<code>replicaTimeout</code>	<code>--replica-timeout</code>	The maximum time in seconds to wait for a replica to complete.
Polling interval	<code>pollingInterval</code>	<code>--polling-interval</code>	The time in seconds to wait between polling for events. Default is 30 seconds.
Replica retry limit	<code>replicaRetryLimit</code>	<code>--replica-retry-limit</code>	The maximum number of times to retry a failed replica. To fail a replica without retrying, set the value to <code>0</code> .
Parallelism	<code>parallelism</code>	<code>--parallelism</code>	The number of replicas to run per execution. For most jobs, set the value to <code>1</code> .
Replica completion count	<code>replicaCompletionCount</code>	<code>--replica-completion-count</code>	The number of replicas to complete successfully for the execution to succeed. Must be equal or less than the parallelism. For most jobs, set the value to <code>1</code> .

Example

Azure CLI

The following example creates a job with advanced configuration options:

Azure CLI

```
az containerapp job create \
    --name "my-job" --resource-group "my-resource-group" --environment
    "my-environment" \
    --trigger-type "Schedule" \
    --replica-timeout 1800 --replica-retry-limit 3 --replica-completion-
count 5 --parallelism 5 \
    --image "myregistry.azurecr.io/quickstart-jobs:latest" \
    --cpu "0.25" --memory "0.5Gi" \
    --command "/startup.sh" \
    --env-vars "MY_ENV_VAR=my-value" \
    --cron-expression "0 0 * * *" \
    --registry-server "myregistry.azurecr.io" \
```

```
--registry-username "myregistry" \
--registry-password "myregistrypassword"
```

Jobs restrictions

The following features aren't supported:

- Dapr
- Ingress and related features such as custom domains and SSL certificates

Next steps

[Create a job with Azure Container Apps](#)

Azure Container Apps dynamic sessions overview

Article • 06/19/2024

Azure Container Apps dynamic sessions provide fast access to secure sandboxed environments that are ideal for running code or applications that require strong isolation from other workloads.

Sessions have the following attributes:

- **Strong isolation:** Sessions are isolated from each other and from the host environment. Each session runs in its own Hyper-V sandbox, providing enterprise-grade security and isolation. Optionally, you can enable network isolation to further enhance security.
- **Simple access:** Sessions are accessed through a REST API. A unique identifier marks each session. If a session with a given identifier doesn't exist, a new session is automatically allocated.
- **Fully managed:** The platform fully manages a session's lifecycle. Sessions are automatically cleaned up when no longer in use.
- **Fast startup:** A new session is allocated in milliseconds. Rapid start-ups are achieved by automatically maintaining a pool of ready but unallocated sessions.
- **Scalable:** Sessions can run at a high scale. You can run hundreds or thousands of sessions concurrently.

ⓘ Note

Azure Container Apps dynamic sessions is currently in preview.

Session types

Azure Container Apps supports two types of sessions:

[+] Expand table

Type	Description	Billing model
Code interpreter	Fully managed code interpreter	Per session (consumption)

Type	Description	Billing model
Custom container	Bring your own container	Container Apps Dedicated Plan

Code interpreter

Code interpreter sessions allow you to run code in a sandbox that is preinstalled with popular libraries. They're ideal for running untrusted code, such as code provided by users of your application or code generated by a large language model (LLM). Learn more about [code interpreter sessions](#).

Custom container

Custom container sessions allow you to run your own container images in secure, isolated sandboxes. You can use them to run a custom code interpreter for a language that isn't supported out of the box, or to run workloads that require strong isolation. Learn more about [custom container sessions](#).

Concepts

The key concepts in Azure Container Apps dynamic sessions are session pools and sessions.

Session pools

To provide sub-second session allocation times, Azure Container Apps maintains a pool of ready but unallocated sessions. When you submit a request to a new session, the platform allocates a session from the pool to you. As sessions are allocated, the platform automatically replenishes the pool to maintain a constant number of ready sessions.

You can configure pools to set the maximum number of sessions that can be allocated concurrently via the `maxConcurrentSessions` property. You can set the wait duration from the last request before a session is deleted the `cooldownPeriodInSeconds` property. For custom container sessions, you can also specify the container image and settings to use for the sessions in the pool, including the target number of sessions to keep ready in the pool via `readySessionInstances`.

Sessions

A session is a sandboxed environment that runs your code or application. Each session is isolated from other sessions and from the host environment with a [Hyper-V](#) sandbox. Optionally, you can enable network isolation to further enhance security.

Session identifiers

When you interact with sessions in a pool, you must define a session identifier to manage each session. The session identifier is a free-form string, meaning you can define it in any way that suits your application's needs. This identifier is a key element in determining the behavior of the session:

- **Reuse of existing sessions:** This session is reused if there's already a running session that matches the identifier.
- **Allocation of new sessions:** If no running session matches the identifier, a new session is automatically allocated from the pool.

The session identifier is a string that you define that is unique within the session pool. If you're building a web application, you can use the user's ID. If you're building a chatbot, you can use the conversation ID.

The identifier must be a string that is 4 to 128 characters long and can contain only alphanumeric characters and special characters from this list: `|`, `-`, `&`, `^`, `%`, `$`, `#`, `(`, `)`, `{`, `}`, `[`, `]`, `;`, `<`, and `>`.

You pass the session identifier in a query parameter named `identifier` in the URL when you make a request to a session.

For code interpreter sessions, you can also use an integration with an [LLM framework](#). The framework handles the token generation and management for you. Ensure that the application is configured with a managed identity that has the necessary role assignments on the session pool.

Protecting session identifiers

The session identifier is sensitive information which requires a secure process as you create and manage its value. To protect this value, your application must ensure each user or tenant only has access to their own sessions.

The specific strategies that prevent misuse of session identifiers differ depending on the design and architecture of your app. However, your app must always have complete control over the creation and use of session identifiers so that a malicious user can't access another user's session.

Example strategies include:

- **One session per user:** If your app uses one session per user, each user must be securely authenticated, and your app must use a unique session identifier for each logged in user.
- **One session per agent conversation:** If your app uses one session per AI agent conversation, ensure your app uses a unique session identifier for each conversation that can't be modified by the end user.

Important

Failure to secure access to sessions may result in misuse or unauthorized access to data stored in your users' sessions.

Authentication

Authentication is handled using Microsoft Entra (formerly Azure Active Directory) tokens. Valid Microsoft Entra tokens are generated by an identity belonging to the *Azure ContainerApps Session Executor* and *Contributor* roles on the session pool.

To assign the roles to an identity, use the following Azure CLI commands:

Bash

```
az role assignment create \
    --role "Azure ContainerApps Session Executor" \
    --assignee <PRINCIPAL_ID> \
    --scope <SESSION_POOL_RESOURCE_ID>

az role assignment create \
    --role "Contributor" \
    --assignee <PRINCIPAL_ID> \
    --scope <SESSION_POOL_RESOURCE_ID>
```

If you're using an [LLM framework integration](#), the framework handles the token generation and management for you. Ensure that the application is configured with a managed identity with the necessary role assignments on the session pool.

If you're using the pool's management API endpoints directly, you must generate a token and include it in the `Authorization` header of your HTTP requests. In addition to the role assignments previously mentioned, token needs to contain an audience (`aud`) claim with the value `https://dynamicsessions.io`.

To generate a token using the Azure CLI, run the following command:

Bash

```
az account get-access-token --resource https://dynamicsessions.io
```

ⓘ Important

A valid token can be used to create and access any session in the pool. Keep your tokens secure and don't share them with untrusted parties. End users should access sessions through your application, not directly.

Lifecycle

The Container Apps runtime automatically manages the lifecycle for each session in a session pool.

- **Pending:** When a session is starting up, it's in the pending state. The amount of time a session spends in the pending state depends on the container image and settings you specify for the session pool. A pending session isn't added to the pool of ready sessions.
- **Ready:** When a session is done starting up and is ready, it's added to the pool. The session is available at this state for allocation. For custom container sessions, you can specify the target number of ready sessions to keep in the pool. Increase this number if sessions are allocated faster than the pool is being replenished.
- **Allocated:** When you send a request to a non-running session, the pool provides a new session and placed it in an allocated state. Subsequent requests with the same session identifier are routed to the same session.
- **Deleting:** When a session stop receiving requests during the time defined by the `cooldownPeriodInSeconds` setting, the session and its Hyper-V sandbox are completely and securely deleted.

Security

Azure Container Apps dynamic sessions are built to run untrusted code and applications in a secure and isolated environment. While sessions are isolated from one another, anything within a single session, including files and environment variables, is accessible by users of the session. You should only configure or upload sensitive data to a session if you trust the users of the session.

Preview limitations

Azure Container Apps dynamic sessions is currently in preview. The following limitations apply:

- It's only available in the following regions:

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

Region	Code interpreter	Custom container
East Asia	✓	✓
East US	✓	✓
West US 2	✓	✓
North Central US	✓	-
North Europe	✓	-

- Logging isn't supported. Your application can log requests to the session pool management API and its responses.

Next steps

[Code interpreter sessions](#)

Feedback

Was this page helpful?

[↳ Yes](#)

[⟲ No](#)

[Provide product feedback ↗](#)

Serverless code interpreter sessions in Azure Container Apps (preview)

Article • 05/21/2024

Azure Container Apps [dynamic sessions](#) provides fast and scalable access to a code interpreter. Each code interpreter session is fully isolated by a Hyper-V boundary and is designed to run untrusted code.

ⓘ Note

The Azure Container Apps dynamic sessions feature is currently in preview. See [preview limitations](#) for more information.

Uses for code interpreter sessions

Code interpreter sessions are ideal for scenarios where you need to run code that is potentially malicious or could cause harm to the host system or other users, such as:

- Code generated by a large language model (LLM).
- Code submitted by an end user in a web or SaaS application.

For popular LLM frameworks such as LangChain, LlamaIndex, or Semantic Kernel, you can use [tools and plugins](#) to integrate AI apps with code interpreter sessions.

Your applications can also integrate with code interpreter session using a [REST API](#). The API allows you to execute code in a session and retrieve results. You can also upload and download files to and from the session. You can upload and download executable code files, or data files that your code can process.

The built-in code interpreter sessions support the most common code execution scenarios without the need to manage infrastructure or containers. If you need full control over the code execution environment or have a different scenario that requires isolated sandboxes, you can use [custom code interpreter sessions](#).

Code interpreter session pool

To use code interpreter sessions, you need an Azure resource called a session pool that defines the configuration for code interpreter sessions. In the session pool, you can

specify settings such as the maximum number of concurrent sessions and how long a session can be idle before the session is terminated.

You can create a session pool using the Azure portal, Azure CLI, or Azure Resource Manager templates. After you create a session pool, you can use the pool's management API endpoints to manage and execute code inside a session.

Create a session pool with Azure CLI

To create a code interpreter session pool using the Azure CLI, ensure you have the latest versions of the Azure CLI and the Azure Container Apps extension with the following commands:

Bash

```
# Upgrade the Azure CLI
az upgrade

# Install or upgrade the Azure Container Apps extension
az extension add --name containerapp --upgrade --allow-preview true -y
```

Use the `az containerapps sessionpool create` command to create the pool. The following example creates a Python code interpreter session pool named `my-session-pool`. Make sure to replace `<RESOURCE_GROUP>` with your resource group name before you run the command.

Bash

```
az containerapp sessionpool create \
--name my-session-pool \
--resource-group <RESOURCE_GROUP> \
--location westus2 \
--container-type PythonLTS \
--max-sessions 100 \
--cooldown-period 300 \
--network-status EgressDisabled
```

You can define the following settings when you create a session pool:

 [Expand table](#)

Setting	Description
--container-type	The type of code interpreter to use. The only supported value is <code>PythonLTS</code> .

Setting	Description
--max-sessions	The maximum number of allocated sessions allowed concurrently. The maximum value is 600.
--cooldown-period	The number of allowed idle seconds before termination. The idle period is reset each time the session's API is called. The allowed range is between 300 and 3600.
--network-status	Designates whether outbound network traffic is allowed from the session. Valid values are EgressDisabled (default) and EgressEnabled.

ⓘ Important

If you enable egress, code running in the session can access the internet. Use caution when the code is untrusted as it can be used to perform malicious activities such as denial-of-service attacks.

Get the pool management API endpoint with Azure CLI

To use code interpreter sessions with LLM framework integrations or by calling the management API endpoints directly, you need the pool's management API endpoint. The endpoint is in the format

```
https://<REGION>.dynamicsessions.io/subscriptions/<SUBSCRIPTION_ID>/resourceGroups/<RESOURCE_GROUP>/sessionPools/<SESSION_POOL_NAME>.
```

To retrieve the management API endpoint for a session pool, use the `az containerapps sessionpool show` command. Make sure to replace `<RESOURCE_GROUP>` with your resource group name before you run the command.

Bash

```
az containerapp sessionpool show \
--name my-session-pool \
--resource-group <RESOURCE_GROUP> \
--query 'properties.poolManagementEndpoint' -o tsv
```

Code execution in a session

After you create a session pool, your application can interact with sessions in the pool using an integration with an [LLM framework](#) or by using the pool's [management API endpoints](#) directly.

Session identifiers

When you interact with sessions in a pool, you use a session identifier to reference each session. A session identifier is a string that you define that is unique within the session pool. If you're building a web application, you can use the user's ID. If you're building a chatbot, you can use the conversation ID.

If there's a running session with the identifier, the session is reused. If there's no running session with the identifier, a new session is automatically created.

To learn more about session identifiers, see [Sessions overview](#).

Authentication

Authentication is handled using Microsoft Entra (formerly Azure Active Directory) tokens. Valid Microsoft Entra tokens are generated by an identity belonging to the *Azure Container Apps Session Executor* and *Contributor* roles on the session pool.

If you're using an LLM framework integration, the framework handles the token generation and management for you. Ensure that the application is configured with a managed identity with the necessary role assignments on the session pool.

If you're using the pool's management API endpoints directly, you must generate a token and include it in the `Authorization` header of your HTTP requests. In addition to the role assignments previously mentioned, token needs to contain an audience (`aud`) claim with the value `https://dynamicsessions.io`.

To learn more, see [Authentication](#).

LLM framework integrations

Instead of using the session pool management API directly, the following LLM frameworks provide integrations with code interpreter sessions:

[] Expand table

Framework	Package	Tutorial
LangChain	Python: langchain-azure-dynamic-sessions	Tutorial
LlamaIndex	Python: llama-index-tools-azure-code-interpreter	Tutorial
Semantic Kernel	Python: semantic-kernel (version 0.9.8-b1 or later)	Tutorial

Management API endpoints

If you're not using an LLM framework integration, you can interact with the session pool directly using the management API endpoints.

The following endpoints are available for managing sessions in a pool:

 Expand table

Endpoint path	Method	Description
code/execute	POST	Execute code in a session.
files/upload	POST	Upload a file to a session.
files/content/{filename}	GET	Download a file from a session.
files	GET	List the files in a session.

Build the full URL for each endpoint by concatenating the pool's management API endpoint with the endpoint path. The query string must include an `identifier` parameter containing the session identifier, and an `api-version` parameter with the value `2024-02-02-preview`.

For example:

```
https://<REGION>.dynamicsessions.io/subscriptions/<SUBSCRIPTION_ID>/resourceGroups/<RESOURCE_GROUP>/sessionPools/<SESSION_POOL_NAME>/code/execute?api-version=2024-02-02-preview&identifier=<IDENTIFIER>
```

Execute code in a session

To execute code in a session, send a `POST` request to the `code/execute` endpoint with the code to run in the request body. This example prints "Hello, world!" in Python.

Before you send the request, replace the placeholders between the `<>` brackets with the appropriate values for your session pool and session identifier.

```
HTTP  
  
POST  
https://<REGION>.dynamicsessions.io/subscriptions/<SUBSCRIPTION_ID>/resourceGroups/<RESOURCE_GROUP>/sessionPools/<SESSION_POOL_NAME>/code/execute?api-version=2024-02-02-preview&identifier=<SESSION_ID>  
Content-Type: application/json  
Authorization: Bearer <token>
```

```
{  
    "properties": {  
        "codeInputType": "inline",  
        "executionType": "synchronous",  
        "code": "print('Hello, world!')"  
    }  
}
```

To reuse a session, specify the same session identifier in subsequent requests.

Upload a file to a session

To upload a file to a session, send a `POST` request to the `uploadFile` endpoint in a multipart form data request. Include the file data in the request body. The file must include a filename.

Uploaded files are stored in the session's file system under the `/mnt/data` directory.

Before you send the request, replace the placeholders between the `<>` brackets with values specific to your request.

HTTP

```
POST  
https://<REGION>.dynamicsessions.io/subscriptions/<SUBSCRIPTION\_ID>/resourceGroups/<RESOURCE\_GROUP>/sessionPools/<SESSION\_POOL\_NAME>/files/upload?api-version=2024-02-02-preview&identifier=<SESSION\_ID>  
Content-Type: multipart/form-data; boundary=----  
WebKitFormBoundary7MA4YWxkTrZu0gW  
Authorization: Bearer <token>  
  
-----WebKitFormBoundary7MA4YWxkTrZu0gW  
Content-Disposition: form-data; name="file"; filename="  
<FILE_NAME_AND_EXTENSION>"  
Content-Type: application/octet-stream  
  
(data)  
-----WebKitFormBoundary7MA4YWxkTrZu0gW--
```

Download a file from a session

To download a file from a session's `/mnt/data` directory, send a `GET` request to the `file/content/{filename}` endpoint. The response includes the file data.

Before you send the request, replace the placeholders between the `<>` brackets with values specific to your request.

HTTP

```
GET  
https://<REGION>.dynamicsessions.io/subscriptions/<SUBSCRIPTION_ID>/resource  
Groups/<RESOURCE_GROUP>/sessionPools/<SESSION_POOL_NAME>/files/content/<FILE  
_NAME_AND_EXTENSION>?api-version=2024-02-02-preview&identifier=<SESSION_ID>  
Authorization: Bearer <TOKEN>
```

List the files in a session

To list the files in a session's `/mnt/data` directory, send a `GET` request to the `files` endpoint.

Before you send the request, replace the placeholders between the `<>` brackets with values specific to your request.

HTTP

```
GET  
https://<REGION>.dynamicsessions.io/subscriptions/<SUBSCRIPTION_ID>/resource  
Groups/<RESOURCE_GROUP>/sessionPools/<SESSION_POOL_NAME>/files?api-  
version=2024-02-02-preview&identifier=<SESSION_ID>  
Authorization: Bearer <TOKEN>
```

The response contains a list of files in the session.

The following listing shows a sample of the type of response you can expect from requesting session contents.

JSON

```
{  
    "$id": "1",  
    "value": [  
        {  
            "$id": "2",  
            "properties": {  
                "$id": "3",  
                "filename": "test1.txt",  
                "size": 16,  
                "lastModifiedTime": "2024-05-02T07:21:07.9922617Z"  
            }  
        },  
        {  
            "$id": "4",  
            "properties": {  
                "$id": "5",  
                "filename": "test2.txt",  
                "size": 20,  
                "lastModifiedTime": "2024-05-02T07:21:07.9922617Z"  
            }  
        }  
    ]  
}
```

```
        "size": 17,  
        "lastModifiedTime": "2024-05-02T07:21:08.8802793Z"  
    }  
}  
]  
}
```

Preinstalled packages

Python code interpreter sessions include popular Python packages such as NumPy, pandas, and scikit-learn.

To output the list of preinstalled packages, call the `code/execute` endpoint with the following code.

Before you send the request, replace the placeholders between the `<>` brackets with values specific to your request.

HTTP

```
POST  
https://<REGION>.dynamicsessions.io/subscriptions/<SUBSCRIPTION\_ID>/resourceGroups/<RESOURCE\_GROUP>/sessionPools/<SESSION\_POOL\_NAME>/identifier/<SESSION\_ID>/code/execute?api-version=2024-02-02-preview&identifier=<SESSION\_ID>  
Content-Type: application/json  
Authorization: Bearer <TOKEN>  
  
{  
    "properties": {  
        "codeInputType": "inline",  
        "executionType": "synchronous",  
        "code": "import pkg_resources\n[(d.project_name, d.version) for d in  
pkg_resources.working_set]"  
    }  
}
```

Next steps

[Quickstart: use code interpreter sessions with LangChain](#)

Azure Container Apps custom container sessions (preview)

Article • 10/02/2024

In addition to the built-in code interpreter that Azure Container Apps dynamic sessions provide, you can also use custom containers to define your own session sandboxes.

ⓘ Note

Azure Container Apps dynamic sessions is currently in preview. See [preview limitations](#) for more information.

Uses for custom container sessions

Custom containers allow you to build solutions tailored to your needs. They enable you to execute code or run applications in environments that are fast and ephemeral and offer secure, sandboxed spaces with Hyper-V. Additionally, they can be configured with optional network isolation. Some examples include:

- **Code interpreters:** When you need to execute untrusted code in secure sandboxes by a language not supported in the built-in interpreter, or you need full control over the code interpreter environment.
- **Isolated execution:** When you need to run applications in hostile, multitenant scenarios where each tenant or user has their own sandboxed environment. These environments are isolated from each other and from the host application. Some examples include applications that run user-provided code, code that grants end user access to a cloud-based shell, AI agents, and development environments.

Using custom container sessions

To use custom container sessions, you first create a session pool with a custom container image. Azure Container Apps automatically starts containers in their own Hyper-V sandboxes using the provided image. Once the container starts up, it's available to the session pool.

When your application requests a session, an instance is instantly allocated from the pool. The session remains active until it enters an idle state, which is then automatically stopped and destroyed.

Creating a custom container session pool

To create a custom container session pool, you need to provide a container image and pool configuration settings.

You invoke or communicate with each session using HTTP requests. The custom container must expose an HTTP server on a port that you specify to respond to these requests.

Azure CLI

To create a custom container session pool using the Azure CLI, ensure you have the latest versions of the Azure CLI and the Azure Container Apps extension with the following commands:

Bash

```
az upgrade
az extension add --name containerapp --upgrade --allow-preview true -y
```

Custom container session pools require a workload profile enabled Azure Container Apps environment. If you don't have an environment, use the `az containerapp env create -n <ENVIRONMENT_NAME> -g <RESOURCE_GROUP> --location <LOCATION> --enable-workload-profiles` command to create one.

Use the `az containerapp sessionpool create` command to create a custom container session pool.

The following example creates a session pool named `my-session-pool` with a custom container image `myregistry.azurecr.io/my-container-image:1.0`.

Before you send the request, replace the placeholders between the `<>` brackets with the appropriate values for your session pool and session identifier.

Bash

```
az containerapp sessionpool create \
--name my-session-pool \
--resource-group <RESOURCE_GROUP> \
--environment <ENVIRONMENT> \
--registry-server myregistry.azurecr.io \
--registry-username <USER_NAME> \
--registry-password <PASSWORD> \
--container-type CustomContainer \
--image myregistry.azurecr.io/my-container-image:1.0 \
```

```
--cpu 0.25 --memory 0.5Gi \
--target-port 80 \
--cooldown-period 300 \
--network-status EgressDisabled \
--max-sessions 10 \
--ready-sessions 5 \
--env-vars "key1=value1" "key2=value2" \
--location <LOCATION>
```

This command creates a session pool with the following settings:

[Expand table](#)

Parameter	Value	Description
--name	my-session-pool	The name of the session pool.
--resource-group	my-resource-group	The resource group that contains the session pool.
--environment	my-environment	The name or resource ID of the container app's environment.
--container-type	CustomContainer	The container type of the session pool. Must be <code>CustomContainer</code> for custom container sessions.
--image	myregistry.azurecr.io/my-container-image:1.0	The container image to use for the session pool.
--registry-server	myregistry.azurecr.io	The container registry server hostname.
--registry-username	my-username	The username to log in to the container registry.
--registry-password	my-password	The password to log in to the container registry.
--cpu	0.25	The required CPU in cores.
--memory	0.5Gi	The required memory.
--target-port	80	The session port used for ingress traffic.
--cooldown-period	300	The number of seconds that a session can be idle before the session is terminated. The idle period is reset each time the

Parameter	Value	Description
		session's API is called. Value must be between 300 and 3600.
--network-status	Designates whether outbound network traffic is allowed from the session. Valid values are EgressDisabled (default) and EgressEnabled.	
--max-sessions	10	The maximum number of sessions that can be allocated at the same time.
--ready-sessions	5	The target number of sessions that are ready in the session pool all the time. Increase this number if sessions are allocated faster than the pool is being replenished.
--env-vars	"key1=value1" "key2=value2"	The environment variables to set in the container.
--location	"Supported Location"	The location of the session pool.

To update the session pool, use the `az containerapp sessionpool update` command.

ⓘ Important

If the session is used to run untrusted code, don't include information or data that you don't want the untrusted code to access. Assume the code is malicious and has full access to the container, including its environment variables, secrets, and files.

Working with sessions

Your application interacts with a session using the session pool's management API.

A pool management endpoint for custom container sessions follows this format:

`https://<SESSION_POOL>.<ENVIRONMENT_ID>.<REGION>.azurecontainerapps.io`.

To retrieve the session pool's management endpoint, use the `az containerapp sessionpool show` command:

Bash

```
az containerapp sessionpool show \
    --name <SESSION_POOL_NAME> \
    --resource-group <RESOURCE_GROUP> \
    --query "properties.poolManagementEndpoint" \
    --output tsv
```

All requests to the pool management endpoint must include an `Authorization` header with a bearer token. To learn how to authenticate with the pool management API, see [Authentication](#).

Each API request must also include the query string parameter `identifier` with the session ID. This unique session ID enables your application to interact with specific sessions. To learn more about session identifiers, see [Session identifiers](#).

Important

The session identifier is sensitive information which requires a secure process as you create and manage its value. To protect this value, your application must ensure each user or tenant only has access to their own sessions. Failure to secure access to sessions may result in misuse or unauthorized access to data stored in your users' sessions. For more information, see [Session identifiers](#)

Forwarding requests to the session's container:

Anything in the path following the base pool management endpoint is forwarded to the session's container.

For example, if you make a call to `<POOL_MANAGEMENT_ENDPOINT>/api/uploadfile`, the request is routed to the session's container at `0.0.0.0:<TARGET_PORT>/api/uploadfile`.

Continuous session interaction:

You can continue making requests to the same session. If there are no requests to the session for longer than the cooldown period, the session is automatically deleted.

Sample request

The following example shows a request to a custom container session by a user ID.

Before you send the request, replace the placeholders between the `<>` brackets with values specific to your request.

HTTP

```
POST https://<SESSION_POOL_NAME>.<ENVIRONMENT_ID>.  
<REGION>.azurecontainerapps.io/<API_PATH_EXPOSED_BY_CONTAINER>?identifier=<USER_ID>  
Authorization: Bearer <TOKEN>  
{  
    "command": "echo 'Hello, world!'"  
}
```

This request is forwarded to the custom container session with the identifier for the user's ID. If the session isn't already running, Azure Container Apps allocates a session from the pool before forwarding the request.

In the example, the session's container receives the request at `http://0.0.0.0:`

`<INGRESS_PORT>/<API_PATH_EXPOSED_BY_CONTAINER>`.

Billing

Custom container sessions are billed based on the resources consumed by the session pool. For more information, see [Azure Container Apps billing](#).

Next steps

[Azure Container Apps dynamic sessions overview](#)

Feedback

Was this page helpful?

 Yes

 No

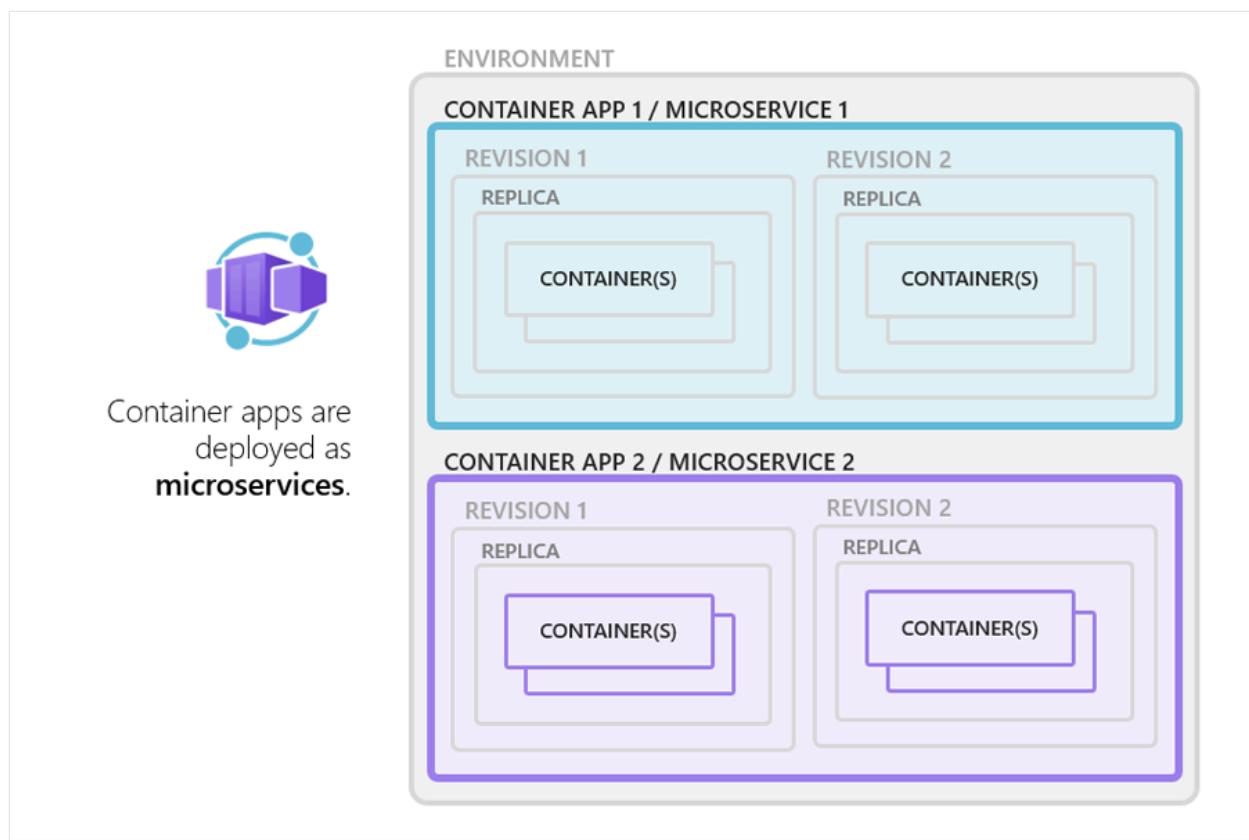
[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

Microservices with Azure Container Apps

Article • 08/02/2024

[Microservice architectures](#) allow you to independently develop, upgrade, version, and scale core areas of functionality in an overall system. Azure Container Apps provides the foundation for deploying microservices featuring:

- Independent [scaling](#), [versioning](#), and [upgrades](#)
- [Service discovery](#)
- [Dapr integration](#)



A Container Apps [environment](#) provides a security boundary around a group of container apps. A single container app typically represents a microservice, which is composed of container apps made up of one or more [containers](#).

You can add [Azure Functions](#) and [Azure Spring Apps](#) to your Azure Container Apps environment.

Dapr integration

When implementing a system composed of microservices, function calls are spread across the network. To support the distributed nature of microservices, you need to

account for failures, retries, and timeouts. While Container Apps features the building blocks for running microservices, use of [Dapr](#) provides an even richer microservices programming model. Dapr includes features like observability, pub/sub, and service-to-service invocation with mutual TLS, retries, and more.

For more information on using Dapr, see [Build microservices with Dapr](#).

Next steps

Scaling

Feedback

Was this page helpful?

 Yes

 No

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

Select the right code-to-cloud path for Azure Container Apps

Article • 10/14/2024

You have several options available as you develop and deploy your apps to Azure Container Apps. As you evaluate your goals and the needs of your team, consider the following questions.

- Are you new to containers?
- Is your focus more on your application or your infrastructure?
- Are you innovating rapidly or in a stable steady state with your application?

Your answers to these questions affect your preferred development and deployment strategies. This article helps you select the most appropriate option for how you develop and deploy your applications to Azure Container Apps.

Depending on your situation, you might want to deploy from a [code editor](#), through the [Azure portal](#), with a hosted [code repository](#), or via [infrastructure as code](#). However, if you're new to containers, you can [learn more](#) about how containers can help your development process.

New to containers

You can simplify the development and deployment of your application by packaging your app into a "container". Containers allow you to wrap up your application and all its dependencies into a single unit that is portable and can be run easily on any container platform.

If you're interested in deploying your application to Azure Container Apps, but don't want to define a container ahead of time, Container Apps can create a container. The Container Apps cloud build feature automatically identifies your application stack and uses [CNCF Buildpacks](#) to generate a container image for you.

Defining containers ahead of time often requires using Docker and publishing your container on a container registry. When you use the Container Apps cloud build, you don't have to worry about special container tooling or registries.

If your application currently doesn't use a container, consider using the Container Apps cloud build to deploy your application.

Resources

- [Build and deploy your app to Azure Container Apps](#)
- [Deploy an artifact file \(JAR\) to Azure Container Apps](#)

Code editor

If you spend most your time editing code and favor rapid iteration of your applications, then you might want to use [Visual Studio](#) or [Visual Studio Code](#). These editors allow you to easily build Docker files and deploy your applications directly to Azure Container Apps.

This approach allows you to experiment with configuration options made in the early stages of an application's life.

Once your application works as expected, then you can formalize the build process through your [code repository](#) to run and deploy your application.

Resources

- [Deploy to Azure Container Apps using Visual Studio](#)
- [Deploy to Azure Container Apps using Visual Studio Code](#)

Azure portal

The Azure portal's focus is on setting up, changing, and experimenting with your Container Apps environment.

While you can't use the portal to deploy your code, it's ideal for making incremental changes to your configuration. The portal's strengths lie in making it easy for you to set up, change, and experiment with your container app.

You can also use the portal with [Azure App Spaces](#) to deploy your applications to Container Apps.

Resources

- [Deploy your first container app using the Azure portal](#)
- [Deploy a web app with Azure App Spaces](#)

Code repository

GitHub and Azure DevOps repositories provide the most structured path to running your code on Azure Container Apps.

As you maintain code in a repository, deployment takes place on the server rather than your local workstation. Remote execution engages safeguards to ensure your application is only updated through trusted channels.

Resources

- [Deploy to Azure Container Apps with GitHub Actions](#)
- [Deploy to Azure Container Apps from Azure Pipelines](#)

Infrastructure as code

Infrastructure as Code (IaC) allows you to maintain your infrastructure setup and configuration in code. Once in your codebase, you can ensure every deployed container environment is consistent, reproducible, and version-controlled.

In Azure Container Apps, you can use the [Azure CLI](#) or the [Azure Developer CLI](#) to configure your applications.

[] [Expand table](#)

CLI	Description	Best used with
Azure CLI	The Azure CLI allows you to deploy directly from your local workstation in the form of local code or container image. You can use PowerShell or Bash to automate application and infrastructure deployment.	Individuals or small teams during initial iteration phases.
Azure Developer CLI (AZD)	AZD is a hybrid solution for handling both the development and operation of your application. When you use AZD, you need to maintain both your application code and infrastructure code in the same repository. The application code requires a Dockerfile for packaging, and the infrastructure code is defined in Bicep.	Applications managed by a single team.

Resources

- [Azure CLI](#)
 - [Build and deploy your container app from a repository](#)
 - [Deploy your first container app using the command line](#)
 - [Set up GitHub Actions with Azure CLI](#)

- Build and deploy your container app from a repository
- **Azure Developer CLI (AZD)**
 - [Get started using Azure Developer CLI](#)

Next steps

- Deploy to Azure Container Apps using Visual Studio
 - Deploy to Azure Container Apps using Visual Studio Code
-

Feedback

Was this page helpful?

 Yes

 No

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

Deploy Azure Container Apps with the az containerapp up command

Article • 07/24/2024

The `az containerapp up` (or `up`) command is the fastest way to deploy an app in Azure Container Apps from an existing image, local source code, or a GitHub repo. With this single command, you can have your container app up and running in minutes.

The `az containerapp up` command is a streamlined way to create and deploy container apps that primarily use default settings. However, you need to run other CLI commands to configure more advanced settings:

- Dapr: [az containerapp dapr enable](#)
- Secrets: [az containerapp secret set](#)
- Transport protocols: [az containerapp ingress update](#)

To customize your container app's resource or scaling settings, you can use the `up` command and then the `az containerapp update` command to change these settings.

The `az containerapp up` command isn't an abbreviation of the `az containerapp update` command.

The `up` command can create or use existing resources including:

- Resource group
- Azure Container Registry
- Container Apps environment and Log Analytics workspace
- Your container app

The command can build and push a container image to an Azure Container Registry (ACR) when you provide local source code or a GitHub repo. When you're working from a GitHub repo, it creates a GitHub Actions workflow that automatically builds and pushes a new container image when you commit changes to your GitHub repo.

If you need to customize the Container Apps environment, first create the environment using the `az containerapp env create` command. If you don't provide an existing environment, the `up` command looks for one in your resource group and, if found, uses that environment. If not found, it creates an environment with a Log Analytics workspace.

To learn more about the `az containerapp up` command and its options, see [az containerapp up](#).

Prerequisites

 Expand table

Requirement	Instructions
Azure account	If you don't have one, create an account for free . You need the <i>Contributor</i> or <i>Owner</i> permission on the Azure subscription to proceed. Refer to Assign Azure roles using the Azure portal for details.
GitHub Account	If you use a GitHub repo, sign up for free .
Azure CLI	Install the Azure CLI .
Local source code	You need to have a local source code directory if you use local source code.
Existing Image	If you use an existing image, you need your registry server, image name, and tag. If you're using a private registry, you need your credentials.

Set up

1. Sign in to Azure with the Azure CLI.

```
Azure CLI
```

```
az login
```

2. Next, install the Azure Container Apps extension for the CLI.

```
Azure CLI
```

```
az extension add --name containerapp --upgrade
```

3. Now that the current extension or module is installed, register the `Microsoft.App` namespace.

```
Azure CLI
```

```
az provider register --namespace Microsoft.App
```

4. Register the `Microsoft.OperationalInsights` provider for the Azure Monitor Log Analytics workspace.

Azure CLI

```
az provider register --namespace Microsoft.OperationalInsights
```

Deploy from an existing image

You can deploy a container app that uses an existing image in a public or private container registry. If you're deploying from a private registry, you need to provide your credentials using the `--registry-server`, `--registry-username`, and `--registry-password` options.

In this example, the `az containerapp up` command performs the following actions:

1. Creates a resource group.
2. Creates an environment and Log Analytics workspace.
3. Creates and deploys a container app that pulls the image from a public registry.
4. Sets the container app's ingress to external with a target port set to the specified value.

Run the following command to deploy a container app from an existing image. Replace the <PLACEHOLDERS> with your values.

Azure CLI

```
az containerapp up \
--name <CONTAINER_APP_NAME> \
--image <REGISTRY_SERVER>/<IMAGE_NAME>:<TAG> \
--ingress external \
--target-port <PORT_NUMBER>
```

You can use the `up` command to redeploy a container app. If you want to redeploy with a new image, use the `--image` option to specify a new image. Ensure that the `--resource-group` and `environment` options are set to the same values as the original deployment.

Azure CLI

```
az containerapp up \
--name <CONTAINER_APP_NAME> \
--image <REGISTRY_SERVER>/<IMAGE_NAME>:<TAG> \
--resource-group <RESOURCE_GROUP_NAME> \
--environment <ENVIRONMENT_NAME> \
```

```
--ingress external \
--target-port <PORT_NUMBER>
```

Deploy from local source code

When you use the `up` command to deploy from a local source, it builds the container image, pushes it to a registry, and deploys the container app. It creates the registry in Azure Container Registry if you don't provide one.

The command can build the image with or without a Dockerfile. If building without a Dockerfile the following languages are supported:

- .NET
- Node.js
- PHP
- Python

The following example shows how to deploy a container app from local source code.

In the example, the `az containerapp up` command performs the following actions:

1. Creates a resource group.
2. Creates an environment and Log Analytics workspace.
3. Creates a registry in Azure Container Registry.
4. Builds the container image (using the Dockerfile if it exists).
5. Pushes the image to the registry.
6. Creates and deploys the container app.

Run the following command to deploy a container app from local source code:

Azure CLI

```
az containerapp up \
--name <CONTAINER_APP_NAME> \
--source <SOURCE_DIRECTORY> \
--ingress external
```

When the Dockerfile includes the EXPOSE instruction, the `up` command configures the container app's ingress and target port using the information in the Dockerfile.

If you configure ingress through your Dockerfile or your app doesn't require ingress, you can omit the `ingress` option.

The output of the command includes the URL for the container app.

If there's a failure, you can run the command again with the `--debug` option to get more information about the failure. If the build fails without a Dockerfile, you can try adding a Dockerfile and running the command again.

To use the `az containerapp up` command to redeploy your container app with an updated image, include the `--resource-group` and `--environment` arguments. The following example shows how to redeploy a container app from local source code.

1. Make changes to the source code.
2. Run the following command:

```
Azure CLI

az containerapp up \
--name <CONTAINER_APP_NAME> \
--source <SOURCE_DIRECTORY> \
--resource-group <RESOURCE_GROUP_NAME> \
--environment <ENVIRONMENT_NAME>
```

Deploy from a GitHub repository

When you use the `az containerapp up` command to deploy from a GitHub repository, it generates a GitHub Actions workflow that builds the container image, pushes it to a registry, and deploys the container app. The command creates the registry in Azure Container Registry if you don't provide one.

A Dockerfile is required to build the image. When the Dockerfile includes the EXPOSE instruction, the command configures the container app's ingress and target port using the information in the Dockerfile.

The following example shows how to deploy a container app from a GitHub repository.

In the example, the `az containerapp up` command performs the following actions:

1. Creates a resource group.
2. Creates an environment and Log Analytics workspace.
3. Creates a registry in Azure Container Registry.
4. Builds the container image using the Dockerfile.
5. Pushes the image to the registry.
6. Creates and deploys the container app.
7. Creates a GitHub Actions workflow to build the container image and deploy the container app when future changes are pushed to the GitHub repository.

To deploy an app from a GitHub repository, run the following command:

Azure CLI

```
az containerapp up \
--name <CONTAINER_APP_NAME> \
--repo <GitHub repository URL> \
--ingress external
```

If you configure ingress through your Dockerfile or your app doesn't require ingress, you can omit the `ingress` option.

Because the `up` command creates a GitHub Actions workflow, rerunning it to deploy changes to your app's image has the unwanted effect of creating multiple workflows. Instead, push changes to your GitHub repository, and the GitHub workflow automatically builds and deploys your app. To change the workflow, edit the workflow file in GitHub.

Next steps

[Quickstart: Deploy your code to Azure Container Apps](#)

Feedback

Was this page helpful?

 Yes

 No

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

Tutorial: Deploy your first container app

Article • 08/04/2024

The Azure Container Apps service enables you to run microservices and containerized applications on a serverless platform. With Container Apps, you enjoy the benefits of running containers while you leave behind the concerns of manually configuring cloud infrastructure and complex container orchestrators.

In this tutorial, you create a secure Container Apps environment and deploy your first container app.

ⓘ Note

You can also deploy this app using the `az containerapp up` by following the instructions in the [Quickstart: Deploy your first container app with containerapp up](#) article. The `az containerapp up` command is a fast and convenient way to build and deploy your app to Azure Container Apps using a single command. However, it doesn't provide the same level of customization for your container app.

Prerequisites

- An Azure account with an active subscription.
 - If you don't have one, you [can create one for free ↗](#).
- Install the [Azure CLI](#).

Setup

To sign in to Azure from the CLI, run the following command and follow the prompts to complete the authentication process.

```
Bash
Azure CLI
az login
```

To ensure you're running the latest version of the CLI, run the upgrade command.

Bash

Azure CLI

```
az upgrade
```

Next, install or update the Azure Container Apps extension for the CLI.

If you receive errors about missing parameters when you run `az containerapp` commands in Azure CLI or cmdlets from the `Az.App` module in Azure PowerShell, be sure you have the latest version of the Azure Container Apps extension installed.

Bash

Azure CLI

```
az extension add --name containerapp --upgrade
```

⚠ Note

Starting in May 2024, Azure CLI extensions no longer enable preview features by default. To access Container Apps [preview features](#), install the Container Apps extension with `--allow-preview true`.

Azure CLI

```
az extension add --name containerapp --upgrade --allow-preview true
```

Now that the current extension or module is installed, register the `Microsoft.App` and `Microsoft.OperationalInsights` namespaces.

Bash

Azure CLI

```
az provider register --namespace Microsoft.App
```

Azure CLI

```
az provider register --namespace Microsoft.OperationalInsights
```

Set environment variables

Set the following environment variables. Replace <PLACEHOLDERS> with your values:

Bash

Azure CLI

```
RESOURCE_GROUP="<>RESOURCE_GROUP<>"  
LOCATION="<>LOCATION<>"  
CONTAINERAPPS_ENVIRONMENT="<>CONTAINERAPPS_ENVIRONMENT<>"
```

Create an Azure resource group

Create a resource group to organize the services related to your container app deployment.

Bash

Azure CLI

```
az group create \  
--name $RESOURCE_GROUP \  
--location "$LOCATION"
```

Create an environment

An environment in Azure Container Apps creates a secure boundary around a group of container apps. Container Apps deployed to the same environment are deployed in the same virtual network and write logs to the same Log Analytics workspace.

To create the environment, run the following command:

Bash

Azure CLI

```
az containerapp env create \
--name $CONTAINERAPPS_ENVIRONMENT \
--resource-group $RESOURCE_GROUP \
--location "$LOCATION"
```

Create a container app

Now that you have an environment created, you can deploy your first container app. With the `containerapp create` command, deploy a container image to Azure Container Apps.

Bash

Azure CLI

```
az containerapp create \
--name my-container-app \
--resource-group $RESOURCE_GROUP \
--environment $CONTAINERAPPS_ENVIRONMENT \
--image mcr.microsoft.com/k8se/quickstart:latest \
--target-port 80 \
--ingress external \
--query properties.configuration.ingress.fqdn
```

⚠ Note

Make sure the value for the `--image` parameter is in lower case.

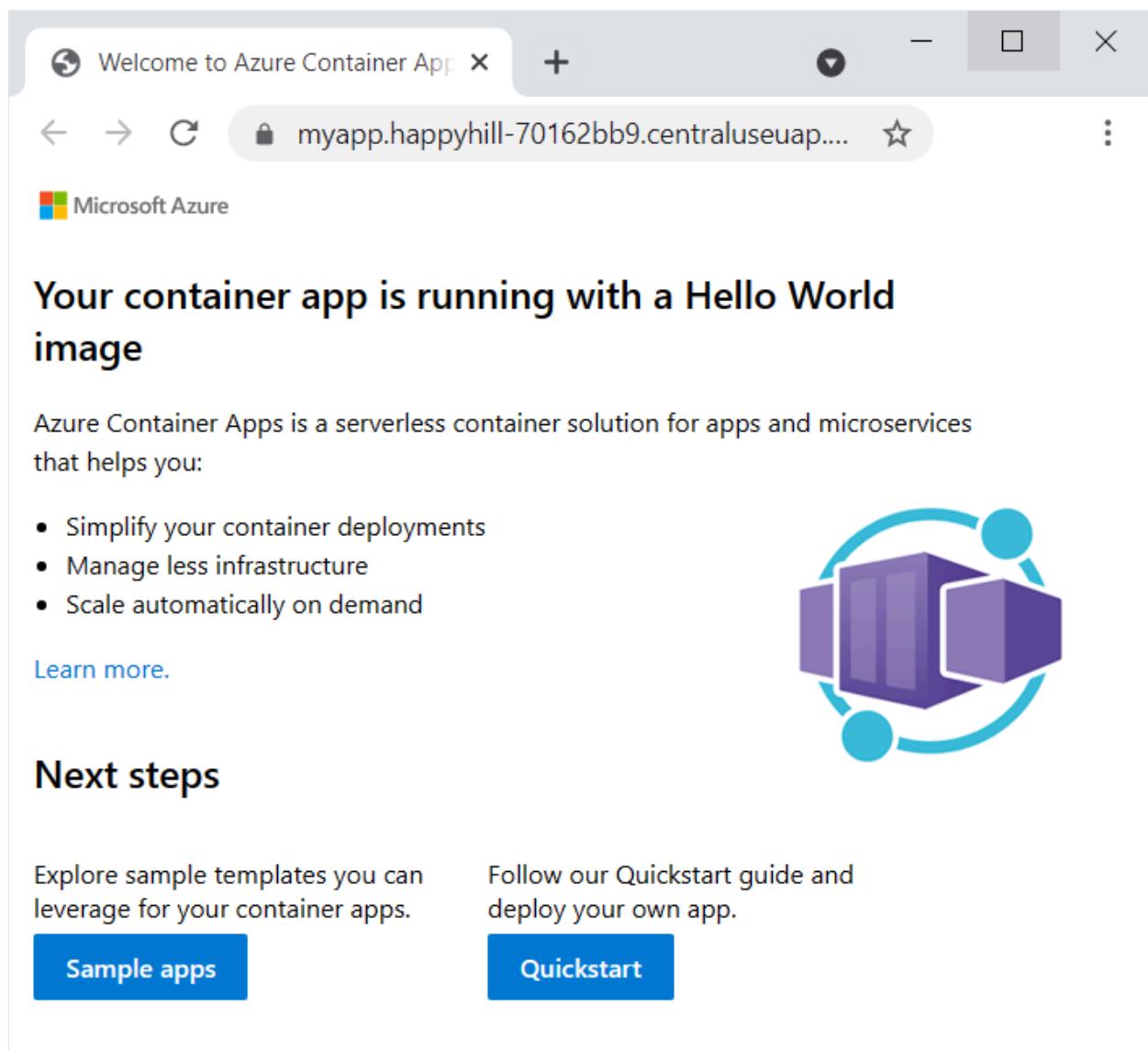
By setting `--ingress` to `external`, you make the container app available to public requests.

Verify deployment

Bash

The `create` command returns the fully qualified domain name for the container app. Copy this location to a web browser.

The following message is displayed when the container app is deployed:



A screenshot of a web browser window titled "Welcome to Azure Container App". The address bar shows the URL "myapp.happyhill-70162bb9.centraluseuap....". The Microsoft Azure logo is visible at the top left. The main content area displays the message: "Your container app is running with a Hello World image". Below this, a description states: "Azure Container Apps is a serverless container solution for apps and microservices that helps you:". A bulleted list follows: • Simplify your container deployments • Manage less infrastructure • Scale automatically on demand. A "Learn more." link is present. To the right is a purple 3D cube icon with a blue circular arrow around it, representing a container or microservice. At the bottom, there are two buttons: "Sample apps" and "Quickstart".

Clean up resources

If you're not going to continue to use this application, run the following command to delete the resource group along with all the resources created in this tutorial.

⊗ Caution

The following command deletes the specified resource group and all resources contained within it. If resources outside the scope of this tutorial exist in the specified resource group, they will also be deleted.

Bash

Azure CLI

```
az group delete --name $RESOURCE_GROUP
```

💡 Tip

Having issues? Let us know on GitHub by opening an issue in the [Azure Container Apps repo](#).

Next steps

[Communication between microservices](#)

Feedback

Was this page helpful?

 Yes

 No

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

Connect to services in Azure Container Apps (preview)

Article • 02/15/2024

As you develop applications in Azure Container Apps, you often need to connect to different services. Rather than creating services ahead of time and manually connecting them to your container app, you can quickly create instances of development-grade services that are designed for nonproduction environments known as add-ons.

Add-ons allow you to use OSS services without the burden of manual downloads, creation, and configuration.

Once you're ready for your app to use a production level service, you can connect your application to an Azure managed service.

Services available as an add-on include:

[+] Expand table

Title	Service name
Kafka ↗	kafka
MariaDB ↗	mariadb
Milvus ↗	milvus
PostgreSQL ↗ (open source)	postgres
Qdrant ↗	qdrant
Redis ↗ (open source)	redis
Weaviate ↗	weaviate

You can get most recent list of add-on services by running the following command:

Azure CLI

```
az containerapp service --help
```

See the section on how to [manage a service](#) for usage instructions.

Features

Add-ons come with the following features:

- **Scope:** The add-on runs in the same environment as the connected container app.
- **Scaling:** The add-on can scale in to zero when there's no demand for the service.
- **Pricing:** Add-on billing falls under consumption-based pricing. Billing only happens when instances of the add-on are running.
- **Storage:** The add-on uses persistent storage to ensure there's no data loss as the add-on scales in to zero.
- **Revisions:** Anytime you change an add-on, a new revision of your container app is created.

See the service-specific features for managed services.

Binding

Both add-ons and managed services connect to a container via a binding.

The Container Apps runtime binds a container app to a service by:

- Discovering the service
- Extracting networking and connection configuration values
- Injecting configuration and connection information into container app environment variables

Once a binding is established, the container app can read these configuration and connection values from environment variables.

Development vs production

As you move from development to production, you can move from an add-on to a managed service.

The following table shows you which service to use in development, and which service to use in production.

[] Expand table

Functionality	Add on	Production managed service
Cache	Open-source Redis	Azure Cache for Redis

Functionality	Add on	Production managed service
Database	N/A	Azure Cosmos DB
Database	Open-source PostgreSQL	Azure Database for PostgreSQL Flexible Server

You're responsible for data continuity between development and production environments.

Manage a service

To connect a service to an application, you first need to create the service.

Use the `containerapp service <SERVICE_TYPE> create` command with the service type and name to create a new service.

CLI

```
az containerapp service redis create \
--name myredis \
--environment myenv
```

This command creates a new Redis service called `myredis` in a Container Apps environment called `myenv`.

To bind a service to an application, use the `--bind` argument for `containerapp create`.

CLI

```
az containerapp create \
--name myapp \
--image myimage \
--bind myredis \
--environment myenv
```

This command features the typical Container App `create` with the `--bind` argument. The bind argument tells the Container Apps runtime to connect a service to the application.

The `--bind` argument is available to the `create` or `update` commands.

To disconnect a service from an application, use the `--unbind` argument on the `update` command

The following example shows you how to unbind a service.

```
az containerapp update --name myapp --unbind myredis
```

For a full tutorial on connecting to services, see [Connect services in Azure Container Apps](#).

For more information on the service commands and arguments, see the [az containerapp](#) reference.

Limitations

- Add-ons are in public preview.
- Any container app created before May 23, 2023 isn't eligible to use add-ons.
- Add-ons come with minimal guarantees. For instance, they're automatically restarted if they crash, however there's no formal quality of service or high-availability guarantees associated with them. For production workloads, use Azure-managed services.
- If you use your own VNET, you must use a workload profiles environment. The Add-ons feature is not supported in consumption only environments that use custom VNETs.

Next steps

[Connect services to a container app](#)

Tutorial: Create and use an Apache Kafka service for development

Article • 06/22/2023

Azure Container Apps allows you to connect to development and production-grade services to provide a wide variety of functionality to your applications.

In this tutorial, you learn to create and use a development Apache Kafka service.

Azure CLI commands and Bicep template fragments are featured in this tutorial. If you use Bicep, you can add all the fragments to a single Bicep file and [deploy the template all at once](#).

- ✓ Create a Container Apps environment to deploy your service and container app
- ✓ Create an Apache Kafka service
- ✓ Set up a command line app to use the dev Apache Kafka service
- ✓ Deploy a *kafka-ui* app to view application data
- ✓ Compile a final bicep template to deploy all resources using a consistent and predictable template deployment
- ✓ Use an `azd` template for a one command deployment of all resources

Prerequisites

- Install the [Azure CLI](#).
- Optional: [Azure Developer CLI](#) for following AZD instructions.

ⓘ Note

For a one command deployment, skip to the last `azd template step`.

Setup

1. Define variables for common values.



```
RESOURCE_GROUP="kafka-dev"
LOCATION="northcentralus"
ENVIRONMENT="aca-env"
KAFKA_SVC="kafka01"
KAFKA_CLI_APP="kafka-cli-app"
KAFKA_UI_APP="kafka-ui-app"
```

2. Log in to Azure.

```
Bash
```

```
az login
```

3. Upgrade the CLI to the latest version.

```
Bash
```

```
az upgrade
```

4. Upgrade Bicep to the latest version.

```
Bash
```

```
az bicep upgrade
```

5. Add the containerapp extension.

```
Bash
```

```
az extension add --name containerapp --upgrade
```

6. Register required namespaces.

```
Bash
```

```
az provider register --namespace Microsoft.App
```

```
Bash
```

```
az provider register --namespace Microsoft.OperationalInsights
```

Create a Container Apps environment

1. Create a resource group.

```
Bash
Bash

az group create \
--name "$RESOURCE_GROUP" \
--location "$LOCATION"
```

2. Create a Container Apps environment.

```
Bash
Bash

az containerapp env create \
--name "$ENVIRONMENT" \
--resource-group "$RESOURCE_GROUP" \
--location "$LOCATION"
```

Create an Apache Kafka service

1. Create an Apache Kafka service.

```
Bash
Bash

ENVIRONMENT_ID=$(az containerapp env show \
--name "$ENVIRONMENT" \
--resource-group "$RESOURCE_GROUP" \
--output tsv \
--query id)
```

2. Deploy the template.

```
Bash
```

Bash

```
az rest \
--method PUT \
--url "/subscriptions/${(az account show --output tsv --query id)/resourceGroups/$RESOURCE_GROUP/providers/Microsoft.App/containers/$KAFKA_SVC?api-version=2023-04-01-preview" \
--body "{\"location\": \"$LOCATION\", \"properties\": {\"environmentId\": \"$ENVIRONMENT_ID\", \"configuration\": {\"service\": {\"type\": \"kafka\"}}}}
```

3. View log output from the Kafka instance

Bash

Use the `logs` command to view log messages.

Bash

```
az containerapp logs show \
--name $KAFKA_SVC \
--resource-group $RESOURCE_GROUP \
--follow --tail 30
```

```
{"TimeStamp": "2023-06-07T02:08:05.5473813+00:00", "Log": "= 604800000"}  
{"TimeStamp": "2023-06-07T02:08:05.5473851+00:00", "Log": "= false"}  
{"TimeStamp": "2023-06-07T02:08:05.5473894+00:00", "Log": "= null"}  
{"TimeStamp": "2023-06-07T02:08:05.5473934+00:00", "Log": "= null"}  
{"TimeStamp": "2023-06-07T02:08:05.5473963+00:00", "Log": "= null"}  
{"TimeStamp": "2023-06-07T02:08:05.5474017+00:00", "Log": "= 10"}  
{"TimeStamp": "2023-06-07T02:08:05.5474049+00:00", "Log": "= false"}  
{"TimeStamp": "2023-06-07T02:08:05.5474084+00:00", "Log": "= 18000"}  
{"TimeStamp": "2023-06-07T02:08:05.5474111+00:00", "Log": "= false"}  
{"TimeStamp": "2023-06-07T02:08:05.5474154+00:00", "Log": "= null"}  
{"TimeStamp": "2023-06-07T02:08:05.5474194+00:00", "Log": "= false"}  
{"TimeStamp": "2023-06-07T02:08:05.5474226+00:00", "Log": "= false"}  
{"TimeStamp": "2023-06-07T02:08:05.5474261+00:00", "Log": "= null"}  
{"TimeStamp": "2023-06-07T02:08:05.5474293+00:00", "Log": "= HTTPS"}  
{"TimeStamp": "2023-06-07T02:08:05.5474329+00:00", "Log": "= null"}  
{"TimeStamp": "2023-06-07T02:08:05.5474364+00:00", "Log": "= null"}  
{"TimeStamp": "2023-06-07T02:08:05.5474392+00:00", "Log": "= null"}  
{"TimeStamp": "2023-06-07T02:08:05.5474425+00:00", "Log": "= false"}  
{"TimeStamp": "2023-06-07T02:08:05.5474464+00:00", "Log": "= TLSv1.2"}  
{"TimeStamp": "2023-06-07T02:08:05.54745+00:00", "Log": "= null"}  
{"TimeStamp": "2023-06-07T02:08:05.5474545+00:00", "Log": "= null"}  
{"TimeStamp": "2023-06-07T02:08:05.5474578+00:00", "Log": "= null"}  
{"TimeStamp": "2023-06-07T02:08:05.5474624+00:00", "Log": "= [kafka.server.KafkaConfig]}"}  
{"TimeStamp": "2023-06-07T02:08:05.6102429+00:00", "Log": "= 02:08:05,610] INFO [SocketServer listenerType=BROKER, nodeId=1] Enabling request processing. (kafka.network.SocketServer)"}  
{"TimeStamp": "2023-06-07T02:08:05.6466283+00:00", "Log": "= 02:08:05,646] INFO [BrokerLifecycleManager id=1] The broker has been unfenced. Transitioning from RECOVERY to RUNNING. (kafka.server.BrokerLifecycleManager)"}  
{"TimeStamp": "2023-06-07T02:08:05.6468583+00:00", "Log": "= 02:08:05,646] INFO [BrokerServer id=1] Transition from STARTING to STARTED (kafka.server.BrokerServer)"}  
{"TimeStamp": "2023-06-07T02:08:05.6473756+00:00", "Log": "= 02:08:05,647] INFO Kafka version: 3.4.0 (org.apache.kafka.common.utils.AppInfoParser)"}  
{"TimeStamp": "2023-06-07T02:08:05.6474318+00:00", "Log": "= 02:08:05,647] INFO Kafka commitId: 2e1947d240607d53 (org.apache.kafka.common.utils.AppInfoParser)"}  
{"TimeStamp": "2023-06-07T02:08:05.6474952+00:00", "Log": "= 02:08:05,647] INFO Kafka startTimeMs: 1686103685646 (org.apache.kafka.common.utils.AppInfoParser)"}  
{"TimeStamp": "2023-06-07T02:08:05.6484361+00:00", "Log": "= 02:08:05,648] INFO [KafkaRaftServer nodeId=1] Kafka Server started (kafka.server.KafkaRaftServer)"}
```

Create an app to test the service

When you create the app, you'll set it up to use `./kafka-topics.sh`, `./kafka-console-producer.sh`, and `kafka-console-consumer.sh` to connect to the Kafka instance.

1. Create a `kafka-cli-app` app that binds to the PostgreSQL service.

```
Bash

Bash

az containerapp create \
    --name "$KAFKA_CLI_APP" \
    --image mcr.microsoft.com/k8se/services/kafka:3.4 \
    --environment "$ENVIRONMENT" \
    --resource-group "$RESOURCE_GROUP" \
    --min-replicas 1 \
    --max-replicas 1 \
    --command "/bin/sleep" "infinity"

az rest \
    --method PATCH \
    --headers "Content-Type=application/json" \
    --url "/subscriptions/$(az account show --output tsv --query id)/resourceGroups/$RESOURCE_GROUP/providers/Microsoft.App/containerApps/$KAFKA_CLI_APP?api-version=2023-04-01-preview" \
    --body "{\"properties\": {\"template\": {\"serviceBinds\": [{\"serviceId\": \"/subscriptions/$(az account show --output tsv --query id)/resourceGroups/$RESOURCE_GROUP/providers/Microsoft.App/containerApps/$KAFKA_SVC\"}]}}}"
```

2. Run the CLI `exec` command to connect to the test app.

```
Bash

Bash

az containerapp exec \
    --name $KAFKA_CLI_APP \
    --resource-group $RESOURCE_GROUP \
    --command /bin/bash
```

When you use `--bind` or `serviceBinds` on the test app, the connection information is injected into the application environment. Once you connect to the test container, you can inspect the values using the `env` command.

Bash

```
env | grep "^KAFKA_"

KAFKA_SECURITYPROTOCOL=SASL_PLAINTEXT
KAFKA_BOOTSTRAPSERVER=kafka01:9092
KAFKA_HOME=/opt/kafka
KAFKA_PROPERTIES_SASL_JAAS_CONFIG=org.apache.kafka.common.security.plain.PlainLoginModule required username="kafka-user" password="7dw..."
user_kafka-user="7dw..." ;
KAFKA_BOOTSTRAP_SERVERS=kafka01:9092
KAFKA_SASLUSERNAME=kafka-user
KAFKA_SASL_USER=kafka-user
KAFKA_VERSION=3.4.0
KAFKA_SECURITY_PROTOCOL=SASL_PLAINTEXT
KAFKA_SASL_PASSWORD=7dw...
KAFKA_SASLPASSWORD=7dw...
KAFKA_SASL_MECHANISM=PLAIN
KAFKA_SASLMECHANISM=PLAIN
```

3. Use `kafka-topics.sh` to create an event topic.

Create a `kafka.props` file.

Bash

```
echo "security.protocol=$KAFKA_SECURITY_PROTOCOL" >> kafka.props && \
echo "sasl.mechanism=$KAFKA_SASL_MECHANISM" >> kafka.props && \
echo "sasl.jaas.config=$KAFKA_PROPERTIES_SASL_JAAS_CONFIG" >>
kafka.props
```

Create a `quickstart-events` event topic.

Bash

```
/opt/kafka/bin/kafka-topics.sh \
--create --topic quickstart-events \
--bootstrap-server $KAFKA_BOOTSTRAP_SERVERS \
--command-config kafka.props
# Created topic quickstart-events.

/opt/kafka/bin/kafka-topics.sh \
--describe --topic quickstart-events \
--bootstrap-server $KAFKA_BOOTSTRAP_SERVERS \
--command-config kafka.props
# Topic: quickstart-events  TopicId: lCKTKmvZSgSUCHzhhvz1Q
PartitionCount: 1  ReplicationFactor: 1   Configs:
segment.bytes=1073741824
# Topic: quickstart-events  Partition: 0      Leader: 1    Replicas: 1
Isr: 1
```

4. Use `kafka-console-producer.sh` to write events to the topic.

Bash

```
/opt/kafka/bin/kafka-console-producer.sh \
  --topic quickstart-events \
  --bootstrap-server $KAFKA_BOOTSTRAP_SERVERS \
  --producer.config kafka.props

> this is my first event
> this is my second event
> this is my third event
> CTRL-C
```

① Note

The `./kafka-console-producer.sh` command prompts you to write events with `>`. Write some events as shown, then hit `CTRL-C` to exit.

5. Use `kafka-console-consumer.sh` to read events from the topic.

Bash

```
/opt/kafka/bin/kafka-console-consumer.sh \
  --topic quickstart-events \
  --bootstrap-server $KAFKA_BOOTSTRAP_SERVERS \
  --from-beginning \
  --consumer.config kafka.props

# this is my first event
# this is my second event
# this is my third event
```

```

root@kafka-cli-app--keu370t-7cfb95dc5d-nwpfm:/# /opt/kafka/bin/kafka-topics.sh \
  --create --topic quickstart-events \
  --bootstrap-server $KAFKA_BOOTSTRAP_SERVERS \
  --command-config kafka.props
Created topic quickstart-events.
root@kafka-cli-app--keu370t-7cfb95dc5d-nwpfm:/# /opt/kafka/bin/kafka-topics.sh \
  --describe --topic quickstart-events \
  --bootstrap-server $KAFKA_BOOTSTRAP_SERVERS \
  --command-config kafka.props
Topic: quickstart-events          TopicId: lD6GKnEAQ626f2kvK5u4JA PartitionCount: 1      ReplicationFactor: 1   C
onfigs: segment.bytes=1073741824
  Topic: quickstart-events      Partition: 0      Leader: 1      Replicas: 1      Isr: 1
root@kafka-cli-app--keu370t-7cfb95dc5d-nwpfm:/# /opt/kafka/bin/kafka-console-producer.sh \
  --topic quickstart-events \
  --bootstrap-server $KAFKA_BOOTSTRAP_SERVERS \
  --producer.config kafka.props
>this is my first event
>this is my second event
>this is my third event
>"Croot@kafka-cli-app--keu370t-7cfb95dc5d-nwpfm:/#
root@kafka-cli-app--keu370t-7cfb95dc5d-nwpfm:/# /opt/kafka/bin/kafka-console-consumer.sh \
  --topic quickstart-events \
  --bootstrap-server $KAFKA_BOOTSTRAP_SERVERS \
  --from-beginning \
  --consumer.config kafka.props
this is my first event
this is my second event
this is my third event
"CProcessed a total of 3 messages
root@kafka-cli-app--keu370t-7cfb95dc5d-nwpfm:/# ■

```

Using a dev service with an existing app

If you already have an app that uses Apache Kafka, you can change how connection information is loaded.

First, create the following environment variables.

Bash

```

KAFKA_HOME=/opt/kafka
KAFKA_PROPERTIES_SASL_JAAS_CONFIG=org.apache.kafka.common.security.plain.Pla
inLoginModule required username="kafka-user" password="7dw..." user_kafka-
user="7dw..." ;
KAFKA_BOOTSTRAP_SERVERS=kafka01:9092
KAFKA_SASL_USER=kafka-user
KAFKA_VERSION=3.4.0
KAFKA_SECURITY_PROTOCOL=SASL_PLAINTEXT
KAFKA_SASL_PASSWORD=7dw...
KAFKA_SASL_MECHANISM=PLAIN

```

Using the CLI (or Bicep) you can update the app to add `--bind $KAFKA_SVC` to use the dev service.

Binding to the dev service

Deploy [kafka-ui](#) to view and manage the Kafka instance.

Bash

See Bicep or azd example.

DONE 81 ms 67 Bytes 3 messages consumed

Offset	Partition	Timestamp	Key	Preview	Value
0	0	6/6/2023, 19:40:34			this is my first event
1	0	6/6/2023, 19:40:38			this is my second event
2	0	6/6/2023, 19:40:46			this is my third event

Deploy all resources

Use the following examples to if you want to deploy all resources at once.

Bicep

The following Bicep template contains all the resources in this tutorial.

You can create a `postgres-dev.bicep` file with this content.

```
Bicep

targetScope = 'resourceGroup'
param location string = resourceGroup().location
param appEnvironmentName string = 'aca-env'
param kafkaSvcName string = 'kafka01'
param kafkaCliAppName string = 'kafka-cli-app'
param kafkaUiAppName string = 'kafka-ui'

resource logAnalytics 'Microsoft.OperationalInsights/workspaces@2022-10-01'
= {
  name: '${appEnvironmentName}-log-analytics'
  location: location
  properties: {
    sku: {
      name: 'PerGB2018'
    }
  }
}
```

```

    }

}

resource appEnvironment 'Microsoft.App/managedEnvironments@2023-04-01-preview' = {
    name: appEnvironmentName
    location: location
    properties: {
        appLogsConfiguration: {
            destination: 'log-analytics'
            logAnalyticsConfiguration: {
                customerId: logAnalytics.properties.customerId
                sharedKey: logAnalytics.listKeys().primarySharedKey
            }
        }
    }
}

resource kafka 'Microsoft.App/containerApps@2023-04-01-preview' = {
    name: kafkaSvcName
    location: location
    properties: {
        environmentId: appEnvironment.id
        configuration: {
            service: {
                type: 'kafka'
            }
        }
    }
}

resource kafkaCli 'Microsoft.App/containerApps@2023-04-01-preview' = {
    name: kafkaCliAppName
    location: location
    properties: {
        environmentId: appEnvironment.id
        template: {
            serviceBinds: [
                {
                    serviceId: kafka.id
                }
            ]
            containers: [
                {
                    name: 'kafka-cli'
                    image: 'mcr.microsoft.com/k8se/services/kafka:3.4'
                    command: [ '/bin/sleep', 'infinity' ]
                }
            ]
            scale: {
                minReplicas: 1
                maxReplicas: 1
            }
        }
    }
}

```

```

}

resource kafkaUi 'Microsoft.App/containerApps@2023-04-01-preview' = {
  name: kafkaUiAppName
  location: location
  properties: {
    environmentId: appEnvironment.id
    configuration: {
      ingress: {
        external: true
        targetPort: 8080
      }
    }
    template: {
      serviceBinds: [
        {
          serviceId: kafka.id
          name: 'kafka'
        }
      ]
      containers: [
        {
          name: 'kafka-ui'
          image: 'docker.io/provectuslabs/kafka-ui:latest'
          command: [
            '/bin/sh'
          ]
          args: [
            '-c'
            '''export
KAFKA_CLUSTERS_0_BOOTSTRAPSERVERS="$KAFKA_BOOTSTRAP_SERVERS" && \
            export
KAFKA_CLUSTERS_0_PROPERTIES_SASL_JAAS_CONFIG="$KAFKA_PROPERTIES_SASL_JAAS_CO
NFIG" && \
            export
KAFKA_CLUSTERS_0_PROPERTIES_SASL_MECHANISM="$KAFKA_SASL_MECHANISM" && \
            export
KAFKA_CLUSTERS_0_PROPERTIES_SECURITY_PROTOCOL="$KAFKA_SECURITY_PROTOCOL" &&
\
            java $JAVA_OPTS -jar kafka-ui-api.jar'''"
          ]
          resources: {
            cpu: json('1.0')
            memory: '2.0Gi'
          }
        }
      ]
    }
  }
}

output kafkaUiUrl string =
'https://${kafkaUi.properties.configuration.ingress.fqdn}'

output kafkaCliExec string = 'az containerapp exec -n ${kafkaCli.name} -g

```

```
`${resourceGroup().name} --command /bin/bash'

output kafkaLogs string = 'az containerapp logs show -n ${kafka.name} -g
${resourceGroup().name} --follow --tail 30'
```

Use the Azure CLI to deploy it the template.

Bash

```
RESOURCE_GROUP="kafka-dev"
LOCATION="northcentralus"

az group create \
    --name "$RESOURCE_GROUP" \
    --location "$LOCATION"

az deployment group create -g $RESOURCE_GROUP \
    --query 'properties.outputs.*.value' \
    --template-file kafka-dev.bicep
```

Azure Developer CLI

A [final template](#) is available on GitHub.

Use `azd up` to deploy the template.

Bash

```
git clone https://github.com/Azure-Samples/aca-dev-service-kafka-azd
cd aca-dev-service-kafka-azd
azd up
```

Clean up resources

Once you're done, run the following command to delete the resource group that contains your Container Apps resources.

⊗ Caution

The following command deletes the specified resource group and all resources contained within it. If resources outside the scope of this tutorial exist in the specified resource group, they will also be deleted.

```
az group delete \  
  --resource-group $RESOURCE_GROUP
```

Tutorial: Use a PostgreSQL service for development

Article • 06/22/2023

Azure Container Apps allows you to connect to development and production-grade services to provide a wide variety of functionality to your applications.

In this tutorial, you learn to use a development PostgreSQL service with Container Apps.

Azure CLI commands and Bicep template fragments are featured in this tutorial. If you use Bicep, you can add all the fragments to a single Bicep file and [deploy the template all at once](#).

- ✓ Create a Container Apps environment to deploy your service and container apps
- ✓ Create a PostgreSQL service
- ✓ Create and use a command line app to use the dev PostgreSQL service
- ✓ Create a *pgweb* app
- ✓ Write data to the PostgreSQL database

Prerequisites

- Install the [Azure CLI](#).
- Optional: [Azure Developer CLI](#) for following AZD instructions.

ⓘ Note

For a one command deployment, skip to the last `azd template step`.

Setup

1. Define variables for common values.

```
Bash
Bash
RESOURCE_GROUP="postgres-dev"
LOCATION="northcentralus"
ENVIRONMENT="aca-env"
```

```
PG_SVC="postgres01"
PSQL_CLI_APP="psql-cloud-cli-app"
```

2. Log in to Azure.

```
Bash
```

```
az login
```

3. Upgrade the CLI to the latest version.

```
Bash
```

```
az upgrade
```

4. Upgrade Bicep to the latest version.

```
Bash
```

```
az bicep upgrade
```

5. Add the `containerapp` extension.

```
Bash
```

```
az extension add --name containerapp --upgrade
```

6. Register required namespaces.

```
Bash
```

```
az provider register --namespace Microsoft.App
```

```
Bash
```

```
az provider register --namespace Microsoft.OperationalInsights
```

Create a Container Apps environment

1. Create a resource group.



Bash

Bash

```
az group create \
--name "$RESOURCE_GROUP" \
--location "$LOCATION"
```

2. Create a Container Apps environment.

Bash

Bash

```
az containerapp env create \
--name "$ENVIRONMENT" \
--resource-group "$RESOURCE_GROUP" \
--location "$LOCATION"
```

Create a PostgreSQL service

1. Create a PostgreSQL service.

Bash

Bash

```
az containerapp service postgres create \
--name "$PG_SVC" \
--resource-group "$RESOURCE_GROUP" \
--environment "$ENVIRONMENT"
```

2. View log output from the Postgres instance

Bash

Use the `logs` command to view log messages.

Bash

```
az containerapp logs show \
    --name $PG_SVC \
    --resource-group $RESOURCE_GROUP \
    --follow --tail 30
```

```
+ az containerapp logs show -n postgres-01 -g postgres-dev --revision postgres-01--plus9v8 --follow --tail 30
{"TimeStamp": "2023-06-06T20:05:45.44013", "Log": "Connecting to the container 'postgres' ..."}
{"TimeStamp": "2023-06-06T20:05:45.45931", "Log": "Successfully Connected to container: 'postgres' [Revision: 'postgres-01--plus9v8', Replica: 'postgres-01--plus9v8-74d9fd878b-x6rhw']"}
{"TimeStamp": "2023-06-06T20:04:15.8704564+00:00", "Log": ""}
{"TimeStamp": "2023-06-06T20:04:15.8704813+00:00", "Log": "You can now start the database server using:"}
{"TimeStamp": "2023-06-06T20:04:15.8704858+00:00", "Log": ""}
{"TimeStamp": "2023-06-06T20:04:15.8704892+00:00", "Log": "pg_ctl -D /mnt/data/pgdata -l logfile start"}
 {"TimeStamp": "2023-06-06T20:04:15.8704911+00:00", "Log": ""}
 {"TimeStamp": "2023-06-06T20:04:15.8704936+00:00", "Log": "For server to start... 2023-06-06 20:04:15.996 UTC [49] LOG: starting PostgreSQL 14.8 (Debian 14.8-1.pgdg110+1) on x86_64-pc-linux-gnu, compiled by gcc (Debian 10.2.1-20210110, 64-bit)"}
 {"TimeStamp": "2023-06-06T20:04:15.9972697+00:00", "Log": "20:04:15.997 UTC [49] LOG: listening on Unix socket "/var/run/postgresql/.s.PGSQL.5432""}
 {"TimeStamp": "2023-06-06T20:04:16.1091987+00:00", "Log": "20:04:16.109 UTC [50] LOG: database system was shut down at 2023-06-06 20:04:05 UTC"}
 {"TimeStamp": "2023-06-06T20:04:16.2238047+00:00", "Log": "20:04:16.223 UTC [49] LOG: database system is ready to accept connections"}
 {"TimeStamp": "2023-06-06T20:04:16.3259918+00:00", "Log": "done"}
 {"TimeStamp": "2023-06-06T20:04:16.3260101+00:00", "Log": "started"}
 {"TimeStamp": "2023-06-06T20:04:16.5341094+00:00", "Log": "done"}
 {"TimeStamp": "2023-06-06T20:04:16.5341344+00:00", "Log": "ignoring /docker-entrypoint-initdb.d/*"}
 {"TimeStamp": "2023-06-06T20:04:16.5540668+00:00", "Log": ""}
 {"TimeStamp": "2023-06-06T20:04:16.600639+00:00", "Log": "for server to shut down ... 2023-06-06 20:04:16.600 UTC [49] LOG: aborting any active transactions"}
 {"TimeStamp": "2023-06-06T20:04:16.645991+00:00", "Log": "20:04:16.645 UTC [49] LOG: background worker \"logical replication launcher\" (PID 56) exited with exit code 1"}
 {"TimeStamp": "2023-06-06T20:04:16.77022+00:00", "Log": "20:04:16.770 UTC [51] LOG: shutting down"}
 {"TimeStamp": "2023-06-06T20:04:17.0714615+00:00", "Log": "20:04:17.071 UTC [49] LOG: database system is shut down"}
 {"TimeStamp": "2023-06-06T20:04:17.1287324+00:00", "Log": "done"}
 {"TimeStamp": "2023-06-06T20:04:17.1287626+00:00", "Log": "stopped"}
 {"TimeStamp": "2023-06-06T20:04:17.1297693+00:00", "Log": ""}
 {"TimeStamp": "2023-06-06T20:04:17.1297933+00:00", "Log": "init process complete; ready for start up."}
 {"TimeStamp": "2023-06-06T20:04:17.1992697+00:00", "Log": "20:04:17.199 UTC [1] LOG: starting PostgreSQL 14.8 (Debian 14.8-1.pgdg110+1) on x86_64-pc-linux-gnu, compiled by gcc (Debian 10.2.1-20210110, 64-bit)"}
 {"TimeStamp": "2023-06-06T20:04:17.1993774+00:00", "Log": "20:04:17.199 UTC [1] LOG: listening on IPv4 address \"0.0.0.0\", port 5432"}
 {"TimeStamp": "2023-06-06T20:04:17.1993986+00:00", "Log": "20:04:17.199 UTC [1] LOG: listening on IPv6 address ::, port 5432"}
 {"TimeStamp": "2023-06-06T20:04:17.2679896+00:00", "Log": "20:04:17.266 UTC [62] LOG: database system was shut down at 2023-06-06 20:04:16 UTC"}
 {"TimeStamp": "2023-06-06T20:04:17.3182285+00:00", "Log": "20:04:17.317 UTC [1] LOG: database system is ready to accept connections"}]
```

Create an app to test the service

When you create the app, you begin by creating a debug app to use the `psql` CLI to connect to the PostgreSQL instance.

1. Create a `psql` app that binds to the PostgreSQL service.

```
Bash
az containerapp create \
    --name "$PSQL_CLI_APP" \
    --image mcr.microsoft.com/k8se/services/postgres:14 \
    --bind "$PG_SVC" \
    --environment "$ENVIRONMENT" \
    --resource-group "$RESOURCE_GROUP" \
    --min-replicas 1 \
    --max-replicas 1 \
    --command "/bin/sleep" "infinity"
```

2. Run the CLI `exec` command to connect to the test app.

Bash

Bash

```
az containerapp exec \
--name $PSQL_CLI_APP \
--resource-group $RESOURCE_GROUP \
--command /bin/bash
```

When you use `--bind` or `serviceBinds` on the test app, the connection information is injected into the application environment. Once you connect to the test container, you can inspect the values using the `env` command.

Bash

```
env | grep "^\$POSTGRES_"

POSTGRES_HOST=postgres01
POSTGRES_PASSWORD=AiSf...
POSTGRES_SSL=disable
POSTGRES_URL=postgres://postgres:AiSf...@postgres01:5432/postgres?
sslmode=disable
POSTGRES_DATABASE=postgres
POSTGRES_PORT=5432
POSTGRES_USERNAME=postgres
POSTGRES_CONNECTION_STRING=host=postgres01 database=postgres
user=postgres password=AiSf...
```

3. Use `psql` to connect to the service

Bash

```
psql $POSTGRES_URL
```

```
[+ ~ az containerapp exec -n $PSQL_CLI_APP -g $RESOURCE_GROUP --command /bin/bash           29.07s
INFO: Connecting to the container 'pgsql-cloud-cli-app' ...
Use ctrl + D to exit.
INFO: Successfully connected to container: 'pgsql-cloud-cli-app'. [ Revision: 'pgsql-cloud-cli-app--km806fv', Replica: 'pgsql-cloud-cli-app--km806fv--6597cc7696-fwd9w' ]
root@pgsql-cloud-cli-app--km806fv-6597cc7696-fwd9w:/# psql $POSTGRES_URL
psql (14.8 (Debian 14.8-1.pgdg110+1))
Type "help" for help.

postgres=# \list
              List of databases
   Name    |  Owner   | Encoding | Collate   | Ctype    | Access privileges
   postgres | postgres | UTF8     | en_US.utf8 | en_US.utf8 | =c/postgres
template0 | postgres | UTF8     | en_US.utf8 | en_US.utf8 | =c/postgres=CTc/postgres
template1 | postgres | UTF8     | en_US.utf8 | en_US.utf8 | =c/postgres=CTc/postgres
(3 rows)

postgres=#
```

4. Create a table named `accounts` and insert data.

SQL

```
postgres=# CREATE TABLE accounts (
    user_id serial PRIMARY KEY,
    username VARCHAR ( 50 ) UNIQUE NOT NULL,
    email VARCHAR ( 255 ) UNIQUE NOT NULL,
    created_on TIMESTAMP NOT NULL,
    last_login TIMESTAMP
);

postgres=# INSERT INTO accounts (username, email, created_on)
VALUES
('user1', 'user1@example.com', current_timestamp),
('user2', 'user2@example.com', current_timestamp),
('user3', 'user3@example.com', current_timestamp);

postgres=# SELECT * FROM accounts;
```

```
+ - az containerapp exec -n $PGSQL_CLI_APP -g $RESOURCE_GROUP --command /bin/bash
INFO: Connecting to the container 'pgsql-cloud-cli-app' ...
Use ctrl + D to exit.
INFO: Successfully connected to container: 'pgsql-cloud-cli-app'. [ Revision: 'pgsql-cloud-cli-app--km806fv', Replica: 'pgsql-cloud-cli-app--km806fv-6597cc7696-fwd9w' ]
root@pgsql-cloud-cli-app--km806fv-6597cc7696-fwd9w:/# psql $POSTGRES_URL
psql (14.8 (Debian 14.8-1.pgdg110+1))
Type "help" for help.

postgres=# CREATE TABLE accounts (
    user_id serial PRIMARY KEY,
    username VARCHAR ( 50 ) UNIQUE NOT NULL,
    email VARCHAR ( 255 ) UNIQUE NOT NULL,
    created_on TIMESTAMP NOT NULL,
    last_login TIMESTAMP
);
CREATE TABLE
postgres=# INSERT INTO accounts (username, email, created_on)
VALUES
('user1', 'user1@example.com', current_timestamp),
('user2', 'user2@example.com', current_timestamp),
('user3', 'user3@example.com', current_timestamp);
INSERT 0 3
postgres=# SELECT * FROM accounts;
user_id | username |      email       |      created_on      | last_login
-----+-----+-----+-----+-----+
  1 | user1  | user1@example.com | 2023-06-06 21:28:53.309114 |
  2 | user2  | user2@example.com | 2023-06-06 21:28:53.309114 |
  3 | user3  | user3@example.com | 2023-06-06 21:28:53.309114 |
(3 rows)

postgres=#
2m20s
```

Using a dev service with an existing app

If you already have an app that uses PostgreSQL, you can change how connection information is loaded.

First, create the following environment variables.

Bash

```
POSTGRES_HOST=postgres01
POSTGRES_PASSWORD=AiSf...
POSTGRES_SSL=disable
POSTGRES_URL=postgres://postgres:AiSf...@postgres01:5432/postgres?
sslmode=disable
POSTGRES_DATABASE=postgres
POSTGRES_PORT=5432
POSTGRES_USERNAME=postgres
```

```
POSTGRES_CONNECTION_STRING=host=postgres01 database=postgres user=postgres  
password=AiSf...
```

Using the CLI (or Bicep) you can update the app to add `--bind $PG_SVC` to use the dev service.

Binding to the dev service

Deploy [pgweb](#) to view and manage the PostgreSQL instance.

Bash

See Bicep or `azd` example.

The screenshot shows the pgweb interface for a PostgreSQL database named 'postgres'. The left sidebar shows the schema structure with 'public' and 'accounts' tables under 'Tables'. The main area displays the 'accounts' table with the following data:

	user_id	username	email	created_on	last_login
1	1	user1	user1@example.com	2023-06-06T21:28:53.309114Z	NULL
	2	user2	user2@example.com	2023-06-06T21:28:53.309114Z	NULL
	3	user3	user3@example.com	2023-06-06T21:28:53.309114Z	NULL

Below the table, there is a 'TABLE INFORMATION' section with the following details:

- Size: 56 kB
- Data size: 8192 bytes
- Index size: 48 kB
- Estimated rows: -1

Deploy all resources

Use the following examples to if you want to deploy all resources at once.

Bicep

The following Bicep template contains all the resources in this tutorial.

You can create a `postgres-dev.bicep` file with this content.

Bicep

```

targetScope = 'resourceGroup'
param location string = resourceGroup().location
param appEnvironmentName string = 'aca-env'
param pgSvcName string = 'postgres01'
param pgsqlCliAppName string = 'pgsql-cloud-cli-app'

resource logAnalytics 'Microsoft.OperationalInsights/workspaces@2022-10-01'
= {
  name: '${appEnvironmentName}-log-analytics'
  location: location
  properties: {
    sku: {
      name: 'PerGB2018'
    }
  }
}

resource appEnvironment 'Microsoft.App/managedEnvironments@2023-04-01-preview' = {
  name: appEnvironmentName
  location: location
  properties: {
    appLogsConfiguration: {
      destination: 'log-analytics'
      logAnalyticsConfiguration: {
        customerId: logAnalytics.properties.customerId
        sharedKey: logAnalytics.listKeys().primarySharedKey
      }
    }
  }
}

resource postgres 'Microsoft.App/containerApps@2023-04-01-preview' = {
  name: pgSvcName
  location: location
  properties: {
    environmentId: appEnvironment.id
    configuration: {
      service: {
        type: 'postgres'
      }
    }
  }
}

resource pgsqlCli 'Microsoft.App/containerApps@2023-04-01-preview' = {
  name: pgsqlCliAppName
  location: location
  properties: {
    environmentId: appEnvironment.id
    template: {
      serviceBinds: [
        {
          serviceId: postgres.id
        }
      ]
    }
  }
}

```

```

        }
    ]
  containers: [
    {
      name: 'pgsql'
      image: 'mcr.microsoft.com/k8se/services/postgres:14'
      command: [ '/bin/sleep', 'infinity' ]
    }
  ]
  scale: {
    minReplicas: 1
    maxReplicas: 1
  }
}
}

resource pgweb 'Microsoft.App/containerApps@2023-04-01-preview' = {
  name: 'pgweb'
  location: location
  properties: {
    environmentId: appEnvironment.id
    configuration: {
      ingress: {
        external: true
        targetPort: 8081
      }
    }
    template: {
      serviceBinds: [
        {
          serviceId: postgres.id
          name: 'postgres'
        }
      ]
      containers: [
        {
          name: 'pgweb'
          image: 'docker.io/sosedoff/pgweb:latest'
          command: [
            '/bin/sh'
          ]
          args: [
            '-c'
            'PGWEB_DATABASE_URL=$POSTGRES_URL /usr/bin/pgweb --bind=0.0.0.0
--listen=8081'
          ]
        }
      ]
    }
  }
}

output pgsqlCliExec string = 'az containerapp exec -n ${pgsqlCli.name} -g
${resourceGroup().name} --revision ${pgsqlCli.properties.latestRevisionName}'

```

```
--command /bin/bash'

output postgresLogs string = 'az containerapp logs show -n ${postgres.name}
-g ${resourceGroup().name} --follow --tail 30'

output pgwebUrl string =
'https://${pgweb.properties.configuration.ingress fqdn}'
```

Use the Azure CLI to deploy it the template.

Bash

```
RESOURCE_GROUP="postgres-dev"
LOCATION="northcentralus"

az group create \
--name "$RESOURCE_GROUP" \
--location "$LOCATION"

az deployment group create -g $RESOURCE_GROUP \
--query 'properties.outputs.*.value' \
--template-file postgres-dev.bicep
```

Azure Developer CLI

A [final template](#) is available on GitHub.

Use `azd up` to deploy the template.

Bash

```
git clone https://github.com/Azure-Samples/aca-dev-service-postgres-azd
cd aca-dev-service-postgres-azd
azd up
```

Clean up resources

Once you're done, run the following command to delete the resource group that contains your Container Apps resources.

⊗ Caution

The following command deletes the specified resource group and all resources contained within it. If resources outside the scope of this tutorial exist in the specified resource group, they will also be deleted.

Azure CLI

```
az group delete \
--resource-group $RESOURCE_GROUP
```

Tutorial: Connect to a Qdrant vector database in Azure Container Apps (preview)

Article • 11/15/2023

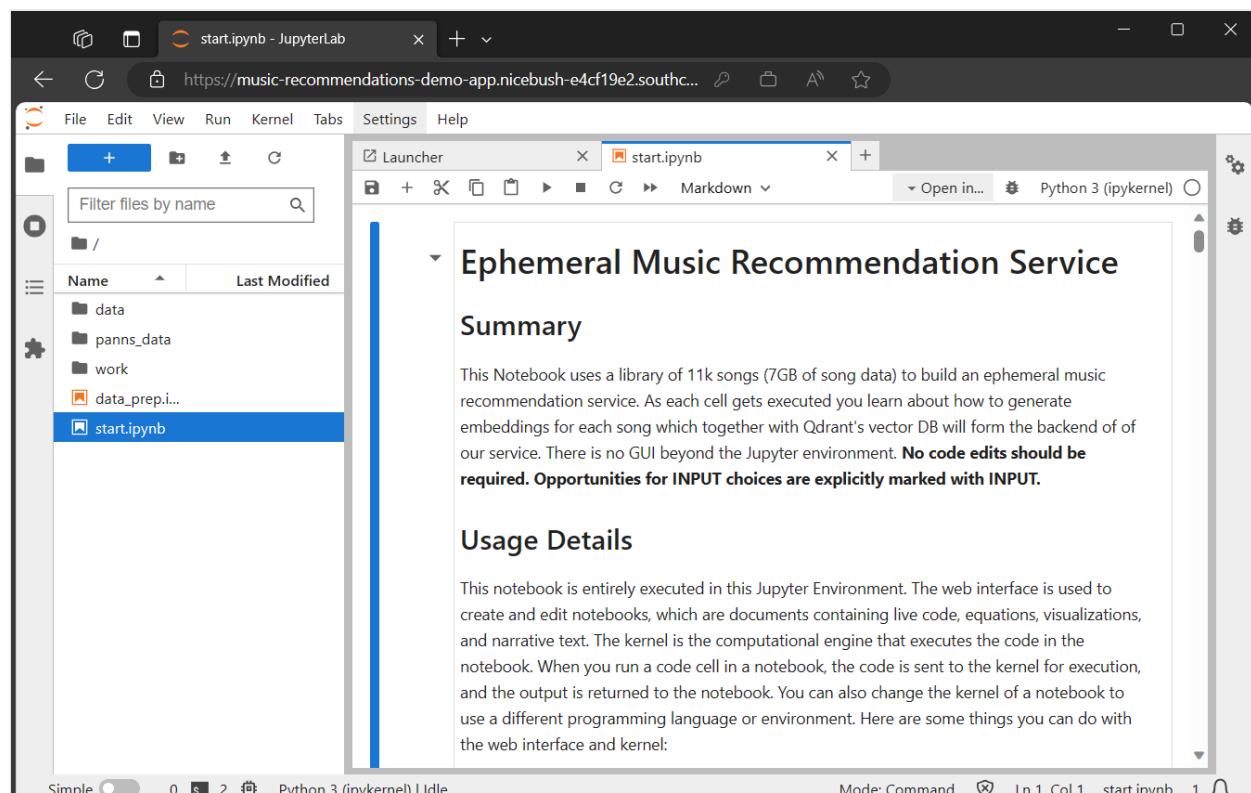
Azure Container Apps uses [add-ons](#) to make it easy to connect to various development-grade cloud services. Rather than creating instances of services ahead of time to establish connections with complex configuration settings, you can use an add-on to connect your container app to a database like Qdrant.

For a full list of supported services, see [Connect to services in Azure Container Apps](#).

The sample application deployed in this tutorial allows you to interface with a music recommendation engine based on the Qdrant vector database. The container image hosts a Jupyter Notebook that contains the code that you can run against the database to:

- Interface with song data
- Generate embeddings for each song
- View song recommendations

Once deployed, you have the opportunity to run code in the Jupyter Notebook to interface with song data in the database.



In this tutorial you:

- ✓ Create a container app
- ✓ Use a Container Apps add-on to connect to a Qdrant database
- ✓ Interact with a Jupyter Notebook to explore the data

ⓘ Important

This tutorial uses services that can affect your Azure bill. If you decide to follow along step-by-step, make sure you deactivate or delete the resources featured in this article to avoid unexpected billing.

Prerequisites

To complete this project, you need the following items:

Requirement	Instructions
Azure account	If you don't have one, create an account for free ↗ . You need the <i>Contributor</i> or <i>Owner</i> permission on the Azure subscription to proceed. Refer to Assign Azure roles using the Azure portal for details.
Azure CLI	Install the Azure CLI .

Setup

Before you begin to work with the Qdrant database, you first need to create your container app and the required resources.

Execute the following commands to create your resource group, container apps environment, and workload profile.

1. Set up application name and resource group variables. You can change these values to your preference.

Bash

```
export APP_NAME=music-recommendations-demo-app
export RESOURCE_GROUP=playground
```

2. Create variables to support your application configuration. These values are provided for you for the purposes of this lesson. Don't change these values.

Bash

```
export SERVICE_NAME=qdrantdb
export LOCATION=southcentralus
export ENVIRONMENT=music-recommendations-demo-environment
export WORKLOAD_PROFILE_TYPE=D32
export CPU_SIZE=8.0
export MEMORY_SIZE=16.0Gi
export IMAGE=simonj.azurecr.io/aca-ephemeral-music-recommendation-image
```

Variable	Description
SERVICE_NAME	The name of the add-on service created for your container app. In this case, you create a development-grade instance of a Qdrant database.
LOCATION	The Azure region location where you create your container app and add-on.
ENVIRONMENT	The Azure Container Apps environment name for your demo application.
WORKLOAD_PROFILE_TYPE	The workload profile type used for your container app. This example uses a general purpose workload profile with 32 cores 128 GiB of memory.
CPU_SIZE	The allocated size of the CPU.
MEMORY_SIZE	The allocated amount of memory.
IMAGE	The container image used in this tutorial. This container image includes the Jupyter Notebook that allows you to interact with data in the Qdrant database.

3. Login to Azure with the Azure CLI.

Azure CLI

```
az login
```

4. Create a resource group.

Azure CLI

```
az group create --name $RESOURCE_GROUP --location $LOCATION
```

5. Create your container apps environment.

Azure CLI

```
az containerapp env create \
--name $ENVIRONMENT \
--resource-group $RESOURCE_GROUP \
--location $LOCATION \
--enable-workload-profiles
```

6. Create a dedicated workload profile with enough resources to work with a vector database.

Azure CLI

```
az containerapp env workload-profile add \
--name $ENVIRONMENT \
--resource-group $RESOURCE_GROUP \
--workload-profile-type $WORKLOAD_PROFILE_TYPE \
--workload-profile-name bigProfile \
--min-nodes 0 \
--max-nodes 2
```

Use the Qdrant add-on

Now that you have an existing environment and workload profile, you can create your container app and bind it to an add-on instance of Qdrant.

1. Create the Qdrant add-on service.

Azure CLI

```
az containerapp service qdrant create \
--environment $ENVIRONMENT \
--resource-group $RESOURCE_GROUP \
--name $SERVICE_NAME
```

2. Create the container app.

Azure CLI

```
az containerapp create \
--name $APP_NAME \
--resource-group $RESOURCE_GROUP \
--environment $ENVIRONMENT \
--workload-profile-name bigProfile \
--cpu $CPU_SIZE \
```

```
--memory $MEMORY_SIZE \
--image $IMAGE \
--min-replicas 1 \
--max-replicas 1 \
--env-vars RESTARTABLE=yes \
--ingress external \
--target-port 8888 \
--transport auto \
--query properties.outputs.fqdn
```

This command returns the fully qualified domain name (FQDN) of your container app. Copy this location to a text editor as you need it in an upcoming step.

An upcoming step instructs you to request an access token to log into the application hosted by the container app. Make sure to wait three to five minutes before you attempt to execute the request for the access token after creating the container app to give enough time to set up all required resources.

3. Bind the Qdrant add-on service to the container app.

Azure CLI

```
az containerapp update \
--name $APP_NAME \
--resource-group $RESOURCE_GROUP \
--bind qdrantdb
```

Configure the container app

Now that your container app is running and connected to Qdrant, you can configure your container app to accept incoming requests.

1. Configure CORS settings on the container app.

Azure CLI

```
az containerapp ingress cors enable \
--name $APP_NAME \
--resource-group $RESOURCE_GROUP \
--allowed-origins "*" \
--allow-credentials true
```

2. Once you wait three to five minutes for the app to complete the setup operations, request an access token for the hosted Jupyter Notebook.

Bash

```
echo Your access token is: `az containerapp logs show -g  
$RESOURCE_GROUP --name $APP_NAME --tail 300 | \  
grep token | cut -d= -f 2 | cut -d\" -f 1 | uniq`
```

When you run this command, your token is printed to the terminal. The message should look like the following example.

text

```
Your access token is: 348c8aed080b44f3aab646287624c70aed080b44f
```

Copy your token value to your text editor to use to sign-in to the Jupyter Notebook.

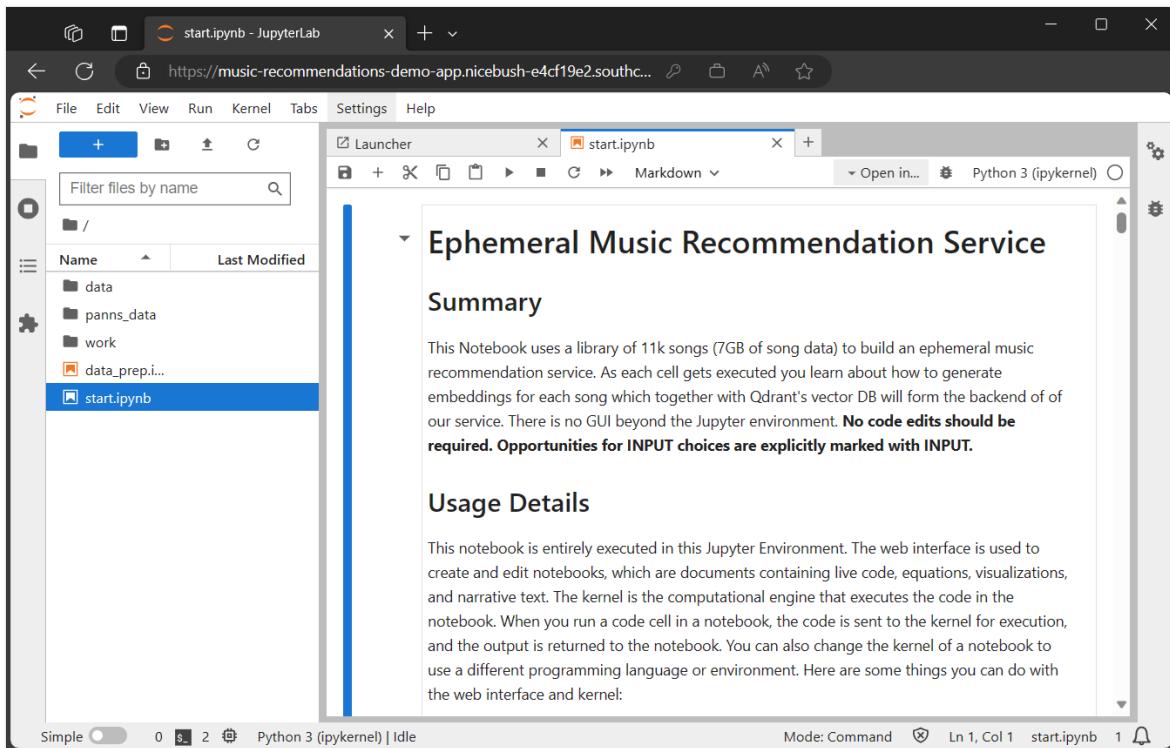
Use the Jupyter Notebook

1. Open a web browser and paste in the URL for your container app you set aside in a text editor.

When the page loads, you're presented with an input box to enter your access token.

2. Next to the *Password to token* label, enter your token in the input box and select **Login**.

Once you authenticate, you are able to interact with the code and data in the Jupyter Notebook.



With the notebook launched, follow the instructions to interact with the code and data.

Clean up resources

The resources created in this tutorial have an effect on your Azure bill. If you aren't going to use these services long-term, run the following command to remove everything created in this tutorial.

```
Azure CLI

az group delete \
--resource-group $RESOURCE_GROUP
```

Next steps

[Learn about other add-on services](#)

Tutorial: Connect services in Azure Container Apps (preview)

Article • 06/13/2023

Azure Container Apps allows you to connect to services that support your app that run in the same environment as your container app.

When in development, your application can quickly create and connect to [dev services](#). These services are easy to create and are development-grade services designed for nonproduction environments.

As you move to production, your application can connect production-grade managed services.

This tutorial shows you how to connect both dev and production grade services to your container app.

In this tutorial, you learn to:

- ✓ Create a new Redis development service
- ✓ Connect a container app to the Redis dev service
- ✓ Disconnect the service from the application
- ✓ Inspect the service running an in-memory cache

Prerequisites

Resource	Description
Azure account	An active subscription is required. If you don't have one, you can create one for free .
Azure CLI	Install the Azure CLI if you don't have it on your machine.
Azure resource group	Create a resource group named my-services-resource-group in the East US region.
Azure Cache for Redis	Create an instance of Azure Cache for Redis in the my-services-resource-group .

Set up

1. Sign in to the Azure CLI.

Azure CLI

```
az login
```

2. Upgrade the Container Apps CLI extension.

Azure CLI

```
az extension add --name containerapp --upgrade
```

3. Register the `Microsoft.App` namespace.

Azure CLI

```
az provider register --namespace Microsoft.App
```

4. Register the `Microsoft.ServiceLinker` namespace.

Azure CLI

```
az provider register --namespace Microsoft.ServiceLinker
```

5. Set up the resource group variable.

Azure CLI

```
RESOURCE_GROUP="my-services-resource-group"
```

6. Create a variable for the Azure Cache for Redis DNS name.

To display a list of the Azure Cache for Redis instances, run the following command.

Azure CLI

```
az redis list --resource-group "$RESOURCE_GROUP" --query "[].name" -o table
```

7. Create a variable to hold your environment name.

Replace `<MY_ENVIRONMENT_NAME>` with the name of your container apps environment.

Azure CLI

```
ENVIRONMENT=<MY_ENVIRONMENT_NAME>
```

8. Set up the location variable.

Azure CLI

```
LOCATION="eastus"
```

9. Create a new environment.

Azure CLI

```
az containerapp env create \
--location "$LOCATION" \
--resource-group "$RESOURCE_GROUP" \
--name "$ENVIRONMENT"
```

With the CLI configured and an environment created, you can now create an application and dev service.

Create a dev service

The sample application manages a set of strings, either in-memory, or in Redis cache.

Create the Redis dev service and name it `myredis`.

Azure CLI

```
az containerapp service redis create \
--name myredis \
--resource-group "$RESOURCE_GROUP" \
--environment "$ENVIRONMENT"
```

Create a container app

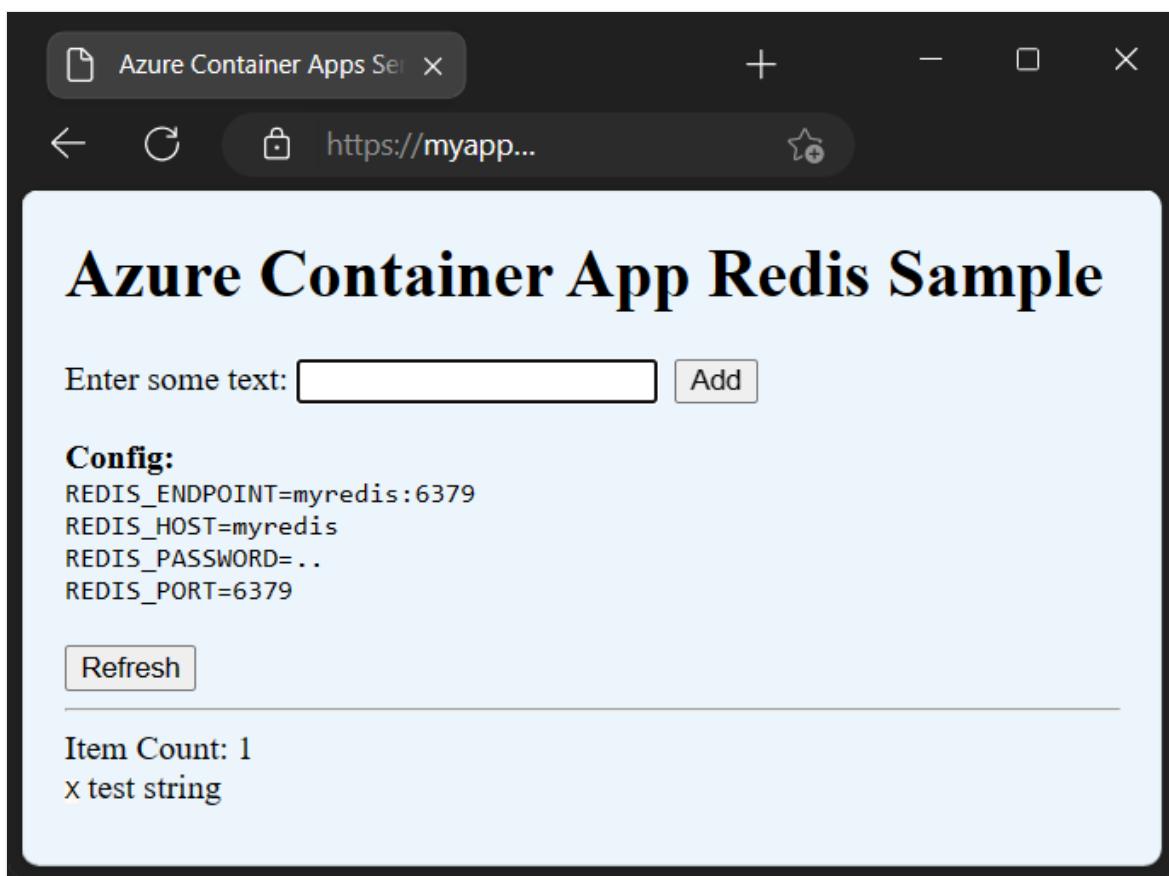
Next, create your internet-accessible container app.

1. Create a new container app and bind it to the Redis service.

Azure CLI

```
az containerapp create \
--name myapp \
--image mcr.microsoft.com/k8se/samples/sample-service-redis:latest \
--ingress external \
--target-port 8080 \
--bind myredis \
--environment "$ENVIRONMENT" \
--resource-group "$RESOURCE_GROUP" \
--query properties.configuration.ingress.fqdn
```

This command returns the fully qualified domain name (FQDN). Paste this location into a web browser so you can inspect the application's behavior throughout this tutorial.



The `containerapp create` command uses the `--bind` option to create a link between the container app and the Redis dev service.

The bind request gathers connection information, including credentials and connection strings, and injects it into the application as environment variables. These values are now available to the application code to use in order to create a connection to the service.

In this case, the following environment variables are available to the application:

Bash

```
REDIS_ENDPOINT=myredis:6379  
REDIS_HOST=myredis  
REDIS_PASSWORD=...  
REDIS_PORT=6379
```

If you access the application via a browser, you can add and remove strings from the Redis database. The Redis cache is responsible for storing application data, so data is available even after the application is restarted after scaling to zero.

You can also remove a binding from your application.

2. Unbind the Redis dev service.

To remove a binding from a container app, use the `--unbind` option.

Azure CLI

```
az containerapp update \  
--name myapp \  
--unbind myredis \  
--resource-group "$RESOURCE_GROUP"
```

The application is written so that if the environment variables aren't defined, then the text strings are stored in memory.

In this state, if the application scales to zero, then data is lost.

You can verify this change by returning to your web browser and refreshing the web application. You can now see the configuration information displayed indicates data is stored in-memory.

Now you can rebind the application to the Redis service, to see your previously stored data.

3. Rebind the Redis dev service.

Azure CLI

```
az containerapp update \  
--name myapp \  
--bind myredis \  
--resource-group "$RESOURCE_GROUP"
```

With the service reconnected, you can refresh the web application to see data stored in Redis.

Connecting to a managed service

When your application is ready to move to production, you can bind your application to a managed service instead of a dev service.

The following steps bind your application to an existing instance of Azure Cache for Redis.

1. Create a variable for your DNS name.

Make sure to replace `<YOUR_DNS_NAME>` with the DNS name of your instance of Azure Cache for Redis.

```
Azure CLI
```

```
AZURE_REDISHOST=<YOUR_DNS_NAME>
```

2. Bind to Azure Cache for Redis.

```
Azure CLI
```

```
az containerapp update \
--name myapp \
--unbind myredis \
--bind "$AZURE_REDISHOST" \
--resource-group "$RESOURCE_GROUP"
```

This command simultaneously removes the development binding and establishes the binding to the production-grade managed service.

3. Return to your browser and refresh the page.

The console prints up values like the following example.

```
Bash
```

```
AZURE_REDISHOST=azureRedis.redis.cache.windows.net
AZURE_REDISHOSTNAME=azureRedis.redis.cache.windows.net
AZURE_REDISHOSTPORT=6380
AZURE_REDISHOSTSSL=true
```

 Note

Environment variable names used for dev mode services and managed service vary slightly.

If you'd like to see the sample code used for this tutorial please see <https://github.com/Azure-Samples/sample-service-redis>.

Now when you add new strings, the values are stored in an instance Azure Cache for Redis instead of the dev service.

Clean up resources

If you don't plan to continue to use the resources created in this tutorial, you can delete the application and the Redis service.

The application and the service are independent. This independence means the service can be connected to any number of applications in the environment and exists until explicitly deleted, even if all applications are disconnect from it.

Run the following commands to delete your container app and the dev service.

Azure CLI

```
az containerapp delete --name myapp  
az containerapp service redis delete --name myredis
```

Alternatively you can delete the resource group to remove the container app and all services.

Azure CLI

```
az group delete \  
--resource-group "$RESOURCE_GROUP"
```

Next steps

[Application lifecycle management](#)

Connect applications in Azure Container Apps

Article • 07/23/2024

Azure Container Apps exposes each container app through a domain name if [ingress](#) is enabled. You can expose ingress endpoints either publicly to the world or to the other container apps in the same environment. Alternatively, you can limit ingress to only other container apps in the same [environment](#).

Application code can call other container apps in the same environment using one of the following methods:

- default fully qualified domain name (FQDN)
- a custom domain name
- the container app name, for instance `http://<APP_NAME>` for internal requests
- a Dapr URL

ⓘ Note

When you call another container in the same environment using the FQDN or app name, the network traffic never leaves the environment.

A sample solution showing how you can call between containers using both the FQDN Location or Dapr can be found on [Azure Samples](#)

Location

A container app's location is composed of values associated with its environment, name, and region. Available through the `azurecontainerapps.io` top-level domain, the fully qualified domain name (FQDN) uses:

- the container app name
- the environment unique identifier
- region name

The following diagram shows how these values are used to compose a container app's fully qualified domain name.



Get fully qualified domain name

The `az containerapp show` command returns the fully qualified domain name of a container app.

Bash

```
az containerapp show \
--resource-group <RESOURCE_GROUP_NAME> \
--name <CONTAINER_APP_NAME> \
--query properties.configuration.ingress.fqdn
```

In this example, replace the placeholders surrounded by `<>` with your values.

The value returned from this command resembles a domain name like the following example:

Console

```
myapp.happyhill-70162bb9.canadacentral.azurecontainerapps.io
```

Dapr location

Developing microservices often requires you to implement patterns common to distributed architecture. Dapr allows you to secure microservices with mutual Transport Layer Security (TLS) (client certificates), trigger retries when errors occur, and take advantage of distributed tracing when Azure Application Insights is enabled.

A microservice that uses Dapr is available through the following URL pattern:



CONTAINER APP ADDRESS WITH DAPR

`http://localhost:3500/v1.0/invoke/myapp`

1

1 Container app name

Call a container app by name

You can call a container app by doing so by sending a request to

`http://<CONTAINER_APP_NAME>` from another app in the environment.

Next steps

[Get started](#)

Feedback

Was this page helpful?

Yes

No

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

Connect a container app to a cloud service with Service Connector

Article • 07/24/2024

Azure Container Apps allows you to use Service Connector to connect to cloud services in just a few steps. Service Connector manages the configuration of the network settings and connection information between different services. To view all supported services, [learn more about Service Connector](#).

In this article, you learn to connect a container app to Azure Blob Storage.

ⓘ Important

This feature in Container Apps is currently in preview. See the [Supplemental Terms of Use for Microsoft Azure Previews](#) for legal terms that apply to Azure features that are in beta, preview, or otherwise not yet released into general availability.

Prerequisites

- An Azure account with an active subscription. [Create an account for free](#).
- An application deployed to Container Apps in a [region supported by Service Connector](#). If you don't have one yet, [create and deploy a container to Container Apps](#)
- An Azure Blob Storage account

Sign in to Azure

First, sign in to Azure.

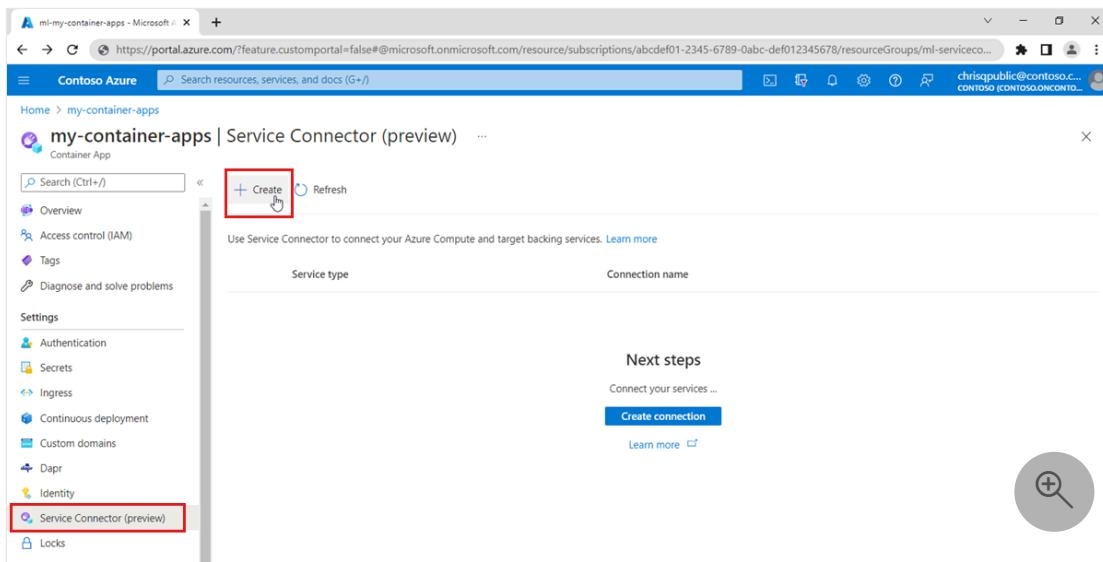
Portal

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

Create a new service connection

Use Service Connector to create a new service connection in Container Apps using the Azure portal or the CLI.

1. Navigate to the Azure portal.
2. Select **All resources** on the left of the Azure portal.
3. Enter **Container Apps** in the filter and select the name of the container app you want to use in the list.
4. Select **Service Connector** from the left table of contents.
5. Select **Create**.



6. Select or enter the following settings.

[] Expand table

Setting	Suggested value	Description
Container	Your container name	Select your Container Apps.
Service type	Blob Storage	This is the target service type. If you don't have a Storage Blob container, you can create one or use another service type.
Subscription	One of your subscriptions	The subscription containing your target service. The default value is the subscription for your container app.
Connection name	Generated unique name	The connection name that identifies the connection between your container app and target

Setting	Suggested value	Description
		service.
Storage account	Your storage account name	The target storage account to which you want to connect. If you choose a different service type, select the corresponding target service instance.
Client type	The app stack in your selected container	Your application stack that works with the target service you selected. The default value is none , which generates a list of configurations. If you know about the app stack or the client SDK in the container you selected, select the same app stack for the client type.

7. Select **Next: Authentication** to select the authentication type. Then select **Connection string** to use access key to connect your Blob Storage account.
8. Select **Next: Network** to select the network configuration. Then select **Enable firewall settings** to update firewall allowlist in Blob Storage so that your container apps can reach the Blob Storage.
9. Then select **Next: Review + Create** to review the provided information. Running the final validation takes a few seconds. Then select **Create** to create the service connection. It might take a minute or so to complete the operation.

View service connections in Container Apps

View your existing service connections using the Azure portal or the CLI.

Portal

1. In **Service Connector**, select **Refresh** and you see a Container Apps connection displayed.
2. Select **>** to expand the list. You can see the environment variables required by your application code.
3. Select **...** and then **Validate**. You can see the connection validation details in the pop-up panel on the right.

Use Service Connector to connect your Azure Compute and target backing services. [Learn more](#)

Service type	Connection name
Blob Storage	storageblob_iscmx
AZURE_STORAGEBLOB_CONNECTION	Hidden value. Click to show value

⋮

Sample code

Validate

Edit

Next steps

[Environments in Azure Container Apps](#)

Feedback

Was this page helpful?

[Yes](#)

[No](#)

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

Tutorial: Connect to PostgreSQL Database from a Java Quarkus Container App without secrets using a managed identity

Article • 10/17/2024

Azure Container Apps 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 Container Apps make your app more secure by eliminating secrets from your app, such as credentials in the environment variables.

This tutorial walks you through the process of building, configuring, deploying, and scaling Java container apps on Azure. At the end of this tutorial, you'll have a [Quarkus](#) application storing data in a [PostgreSQL](#) database with a managed identity running on [Container Apps](#).

What you will learn:

- ✓ Configure a Quarkus app to authenticate using Microsoft Entra ID with a PostgreSQL Database.
- ✓ Create an Azure container registry and push a Java app image to it.
- ✓ Create a Container App in Azure.
- ✓ Create a PostgreSQL database in Azure.
- ✓ Connect to a PostgreSQL Database with managed identity using Service Connector.

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

1. Prerequisites

- [Azure CLI](#) version 2.45.0 or higher.
- [Git](#)
- [Java JDK](#)
- [Maven](#)
- [Docker](#)

2. Create a container registry

Create a resource group with the [az group create](#) command. An Azure resource group is a logical container into which Azure resources are deployed and managed.

The following example creates a resource group named `myResourceGroup` in the East US Azure region.

Azure CLI

```
RESOURCE_GROUP="myResourceGroup"
LOCATION="eastus"

az group create --name $RESOURCE_GROUP --location $LOCATION
```

Create an Azure container registry instance using the [az acr create](#) command and retrieve its login server using the [az acr show](#) command. The registry name must be unique within Azure and contain 5-50 alphanumeric characters. All letters must be specified in lower case. In the following example, `mycontainerregistry007` is used. Update this to a unique value.

Azure CLI

```
REGISTRY_NAME=mycontainerregistry007
az acr create \
    --resource-group $RESOURCE_GROUP \
    --name $REGISTRY_NAME \
    --sku Basic

REGISTRY_SERVER=$(az acr show \
    --name $REGISTRY_NAME \
    --query 'loginServer' \
    --output tsv | tr -d '\r')
```

3. Clone the sample app and prepare the container image

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.

git

```
git clone https://github.com/quarkusio/quarkus-quickstarts  
cd quarkus-quickstarts/hibernate-orm-panache-quickstart
```

Modify your project

1. Add the required dependencies to your project's BOM file.

XML

```
<dependency>  
  <groupId>com.azure</groupId>  
  <artifactId>azure-identity-extensions</artifactId>  
  <version>1.1.20</version>  
</dependency>
```

2. Configure the Quarkus app properties.

The 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 the database for dev, test, and production modes:

properties

```
quarkus.hibernate-orm.database.generation=drop-and-create  
quarkus.datasource.db-kind=postgresql  
quarkus.datasource.jdbc.max-size=8  
quarkus.datasource.jdbc.min-size=2  
quarkus.hibernate-orm.log.sql=true  
quarkus.hibernate-orm.sql-load-script=import.sql  
quarkus.datasource.jdbc.acquisition-timeout = 10  
  
%dev.quarkus.datasource.username=${CURRENT_USERNAME}  
%dev.quarkus.datasource.jdbc.url=jdbc:postgresql://${AZURE_POSTGRESQL_HOST}: ${AZURE_POSTGRESQL_PORT}/ ${AZURE_POSTGRESQL_DATABASE}?\\  
authenticationPluginClassName=com.azure.identity.extensions.jdbc.postgresql.AzurePostgresqlAuthenticationPlugin\\  
&sslmode=require  
  
%prod.quarkus.datasource.username=${AZURE_POSTGRESQL_USERNAME}
```

```
%prod.quarkus.datasource.jdbc.url=jdbc:postgresql://${AZURE_POSTGRESQL_HOST}:${AZURE_POSTGRESQL_PORT}/${AZURE_POSTGRESQL_DATABASE}?\nauthenticationPluginClassName=com.azure.identity.extensions.jdbc.postgresql.AzurePostgresqlAuthenticationPlugin\n&sslmode=require\n\n%dev.quarkus.class-loading.parent-first-artifacts=com.azure:azure-core::jar,\ncom.azure:azure-core-http-netty::jar,\nio.projectreactor.netty:reactor-netty-core::jar,\nio.projectreactor.netty:reactor-netty-http::jar,\nio.netty:netty-resolver-dns::jar,\nio.netty:netty-codec::jar,\nio.netty:netty-codec-http::jar,\nio.netty:netty-codec-http2::jar,\nio.netty:netty-handler::jar,\nio.netty:netty-resolver::jar,\nio.netty:netty-common::jar,\nio.netty:netty-transport::jar,\nio.netty:netty-buffer::jar,\ncom.azure:azure-identity::jar,\ncom.azure:azure-identity-extensions::jar,\ncom.fasterxml.jackson.core:jackson-core::jar,\ncom.fasterxml.jackson.core:jackson-annotations::jar,\ncom.fasterxml.jackson.core:jackson-databind::jar,\ncom.fasterxml.jackson.dataformat:jackson-dataformat-xml::jar,\ncom.fasterxml.jackson.datatype:jackson-datatype-jsr310::jar,\norg.reactivestreams:reactive-streams::jar,\nio.projectreactor:reactor-core::jar,\ncom.microsoft.azure:msal4j::jar,\ncom.microsoft.azure:msal4j-persistence-extension::jar,\norg.codehaus.woodstox:stax2-api::jar,\ncom.fasterxml.woodstox:woodstox-core::jar,\ncom.nimbusds:oauth2-oidc-sdk::jar,\ncom.nimbusds:content-type::jar,\ncom.nimbusds:nimbus-jose-jwt::jar,\nnet.minidev:json-smart::jar,\nnet.minidev:accessors-smart::jar,\nio.netty:netty-transport-native-unix-common::jar,\nnet.java.dev.jna:jna::jar
```

Build and push a Docker image to the container registry

1. Build the container image.

Run the following command to build the Quarkus app image. You must tag it with the fully qualified name of your registry login server.

```
Bash
```

```
CONTAINER_IMAGE=${REGISTRY_SERVER}/quarkus-postgres-passwordless-app:v1

mvn quarkus:add-extension -Dextensions="container-image-jib"
mvn clean package -Dquarkus.container-image.build=true -
Dquarkus.container-image.image=${CONTAINER_IMAGE}
```

2. Log in to the registry.

Before pushing container images, you must log in to the registry. To do so, use the [az acr login][az-acr-login] command.

Azure CLI

```
az acr login --name $REGISTRY_NAME
```

The command returns a `Login Succeeded` message once completed.

3. Push the image to the registry.

Use [docker push][docker-push] to push the image to the registry instance. This example creates the `quarkus-postgres-passwordless-app` repository, containing the `quarkus-postgres-passwordless-app:v1` image.

Bash

```
docker push $CONTAINER_IMAGE
```

4. Create a Container App on Azure

1. Create a Container Apps instance by running the following command. Make sure you replace the value of the environment variables with the actual name and location you want to use.

Azure CLI

```
CONTAINERAPPS_ENVIRONMENT="my-environment"

az containerapp env create \
--resource-group $RESOURCE_GROUP \
--name $CONTAINERAPPS_ENVIRONMENT \
--location $LOCATION
```

2. Create a container app with your app image by running the following command:

Azure CLI

```
APP_NAME=my-container-app
az containerapp create \
    --resource-group $RESOURCE_GROUP \
    --name $APP_NAME \
    --image $CONTAINER_IMAGE \
    --environment $CONTAINERAPPS_ENVIRONMENT \
    --registry-server $REGISTRY_SERVER \
    --registry-identity system \
    --ingress 'external' \
    --target-port 8080 \
    --min-replicas 1
```

ⓘ Note

The options `--registry-username` and `--registry-password` are still supported but aren't recommended because using the identity system is more secure.

5. Create and connect a PostgreSQL database with identity connectivity

Next, create a PostgreSQL Database and configure your container app to connect to a PostgreSQL Database with a system-assigned managed identity. The Quarkus 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'

az postgres flexible-server create \
    --resource-group $RESOURCE_GROUP \
    --name $DB_SERVER_NAME \
    --location $LOCATION \
    --public-access None \
    --sku-name Standard_B1ms \
    --tier Burstable \
    --active-directory-auth Enabled
```

ⓘ Note

The options `--admin-user` and `--admin-password` are still supported but aren't recommended because using the identity system is more secure.

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. Change to a different location if it doesn't work.
- *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, `Standard_B1ms`. For more information, see [Azure Database for PostgreSQL pricing ↗](#).
- *tier* → The compute tier of the server. For more information, see [Azure Database for PostgreSQL pricing ↗](#).
- *active-directory-auth* → `Enabled` to enable Microsoft Entra authentication.

2. Create a database named `fruits` within the PostgreSQL service with this command:

Azure CLI

```
DB_NAME=fruits
az postgres flexible-server db create \
--resource-group $RESOURCE_GROUP \
--server-name $DB_SERVER_NAME \
--database-name $DB_NAME
```

3. Install the [Service Connector](#) passwordless extension for the Azure CLI:

Azure CLI

```
az extension add --name serviceconnector-passwordless --upgrade --
allow-preview true
```

4. Connect the database to the container app with a system-assigned managed identity, using the connection command.

Azure CLI

```
az containerapp connection create postgres-flexible \
--resource-group $RESOURCE_GROUP \
--name $APP_NAME \
--target-resource-group $RESOURCE_GROUP \
--server $DB_SERVER_NAME \
--database $DB_NAME \
--system-identity \
--container $APP_NAME
```

6. Review your changes

You can find the application URL(FQDN) by using the following command:

Azure CLI

```
echo https://$(az containerapp show \
--name $APP_NAME \
--resource-group $RESOURCE_GROUP \
--query properties.configuration.ingress.fqdn \
--output tsv)
```

When the new webpage shows your list of fruits, your app is connecting to the database using the managed identity. You should now be able to edit fruit 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

Learn more about running Java apps on Azure in the developer guide.

Azure for Java Developers

Feedback

Was this page helpful?

 Yes

 No

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

Tutorial: Build and deploy your app to Azure Container Apps

Article • 08/28/2024

This article demonstrates how to build and deploy a microservice to Azure Container Apps from a source repository using your preferred programming language.

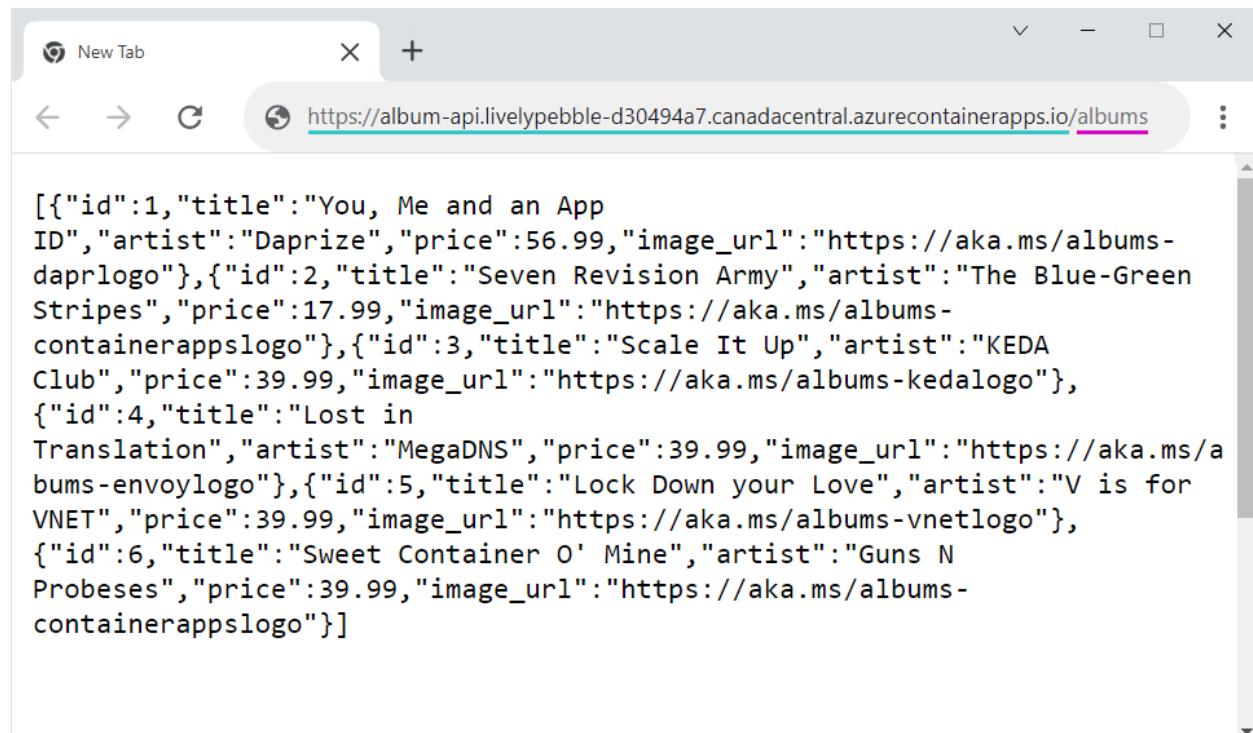
This is the first tutorial in the series of articles that walk you through how to use core capabilities within Azure Container Apps. The first step is to create a back end web API service that returns a static collection of music albums.

ⓘ Note

You can also build and deploy this app using the `az containerapp up` by following the instructions in the [Quickstart: Build and deploy an app to Azure Container Apps from a repository](#) article. The `az containerapp up` command is a fast and convenient way to build and deploy your app to Azure Container Apps using a single command. However, it doesn't provide the same level of customization for your container app.

The next tutorial in the series will build and deploy the front end web application to Azure Container Apps.

The following screenshot shows the output from the album API deployed in this tutorial.



Prerequisites

To complete this project, you need the following items:

[] [Expand table](#)

Requirement	Instructions
Azure account	If you don't have one, create an account for free . You need the <i>User Access Administrator</i> or <i>Owner</i> permission on the Azure subscription to proceed. Make sure to use the most restrictive role for your context. See Assign Azure roles using the Azure portal and Azure roles, Microsoft Entra roles, and classic subscription administrator roles for details.
GitHub Account	Sign up for free .
git	Install git
Azure CLI	Install the Azure CLI .

Setup

To sign in to Azure from the CLI, run the following command and follow the prompts to complete the authentication process.

```
Bash
az login
```

To ensure you're running the latest version of the CLI, run the upgrade command.

```
Bash
az upgrade
```

Next, install or update the Azure Container Apps extension for the CLI.

If you receive errors about missing parameters when you run `az containerapp` commands in Azure CLI or cmdlets from the `Az.App` module in Azure PowerShell, be sure you have the latest version of the Azure Container Apps extension installed.

Bash

Azure CLI

```
az extension add --name containerapp --upgrade
```

 **Note**

Starting in May 2024, Azure CLI extensions no longer enable preview features by default. To access Container Apps [preview features](#), install the Container Apps extension with `--allow-preview true`.

Azure CLI

```
az extension add --name containerapp --upgrade --allow-preview true
```

Now that the current extension or module is installed, register the `Microsoft.App` and `Microsoft.OperationalInsights` namespaces.

Bash

Azure CLI

```
az provider register --namespace Microsoft.App
```

Azure CLI

```
az provider register --namespace Microsoft.OperationalInsights
```

Create environment variables

Now that your Azure CLI setup is complete, you can define the environment variables that are used throughout this article.

Bash

Define the following variables in your bash shell.

Azure CLI

```
RESOURCE_GROUP="album-containerapps"
LOCATION="canadacentral"
ENVIRONMENT="env-album-containerapps"
API_NAME="album-api"
FRONTEND_NAME="album-ui"
GITHUB_USERNAME=<YOUR_GITHUB_USERNAME>"
```

Before you run this command, make sure to replace `<YOUR_GITHUB_USERNAME>` with your GitHub username.

Next, define a container registry name unique to you.

Azure CLI

```
ACR_NAME="acaalbums"$GITHUB_USERNAME
```

Prepare the GitHub repository

Navigate to the repository for your preferred language and fork the repository.

C#

Select the **Fork** button at the top of the [album API repo](#) to fork the repo to your account.

Now you can clone your fork of the sample repository.

Use the following git command to clone your forked repo into the *code-to-cloud* folder:

git

```
git clone https://github.com/$GITHUB_USERNAME/containerapps-albumapi-csharp.git code-to-cloud
```

Next, change the directory into the root of the cloned repo.

Console

```
cd code-to-cloud/src
```

Create an Azure resource group

Create a resource group to organize the services related to your container app deployment.

Bash

Azure CLI

```
az group create \
--name $RESOURCE_GROUP \
--location "$LOCATION"
```

Create an Azure Container Registry

After the album API container image is built, create an Azure Container Registry (ACR) instance in your resource group to store it.

Bash

Azure CLI

```
az acr create \
--resource-group $RESOURCE_GROUP \
--name $ACR_NAME \
--sku Basic \
--admin-enabled true
```

Build your application

With [ACR tasks](#), you can build and push the docker image for the album API without installing Docker locally.

Build the container with ACR

Run the following command to initiate the image build and push process using ACR. The `.` at the end of the command represents the docker build context, meaning this command should be run within the `src` folder where the Dockerfile is located.

Bash

Azure CLI

```
az acr build --registry $ACR_NAME --image $API_NAME .
```

Output from the `az acr build` command shows the upload progress of the source code to Azure and the details of the `docker build` and `docker push` operations.

Create a Container Apps environment

The Azure Container Apps environment acts as a secure boundary around a group of container apps.

Create the Container Apps environment using the following command.

Bash

Azure CLI

```
az containerapp env create \
--name $ENVIRONMENT \
--resource-group $RESOURCE_GROUP \
--location "$LOCATION"
```

Deploy your image to a container app

Now that you have an environment created, you can create and deploy your container app with the `az containerapp create` command.

Create and deploy your container app with the following command.

Bash

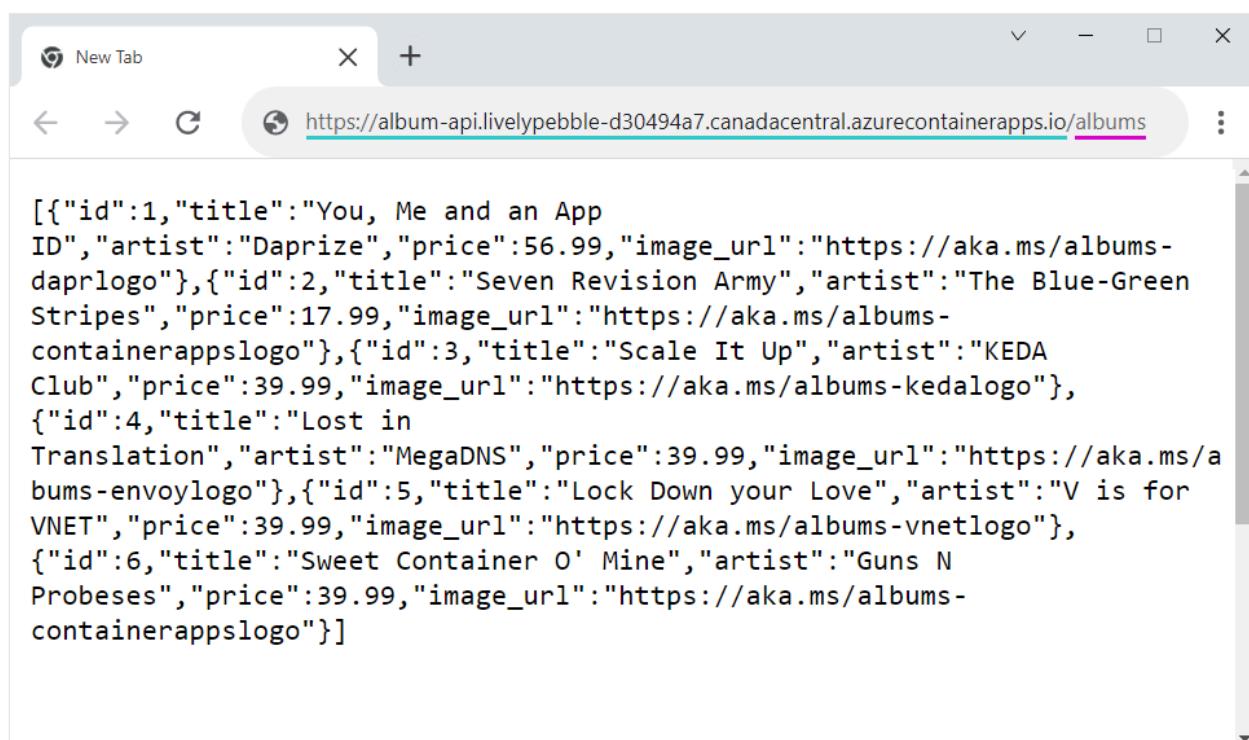
Azure CLI

```
az containerapp create \
--name $API_NAME \
--resource-group $RESOURCE_GROUP \
--environment $ENVIRONMENT \
--image $ACR_NAME.azurecr.io/$API_NAME \
--target-port 8080 \
--ingress external \
--registry-server $ACR_NAME.azurecr.io \
--query properties.configuration.ingress.fqdn
```

- By setting `--ingress` to `external`, your container app is accessible from the public internet.
- The `target-port` is set to `8080` to match the port that the container is listening to for requests.
- Without a `query` property, the call to `az containerapp create` returns a JSON response that includes a rich set of details about the application. Adding a query parameter filters the output to just the app's fully qualified domain name (FQDN).

Verify deployment

Copy the FQDN to a web browser. From your web browser, navigate to the `/albums` endpoint of the FQDN.



Clean up resources

If you're not going to continue on to the [Communication between microservices](#) tutorial, you can remove the Azure resources created during this quickstart. Run the following command to delete the resource group along with all the resources created in this quickstart.

Bash

Azure CLI

```
az group delete --name $RESOURCE_GROUP
```



Tip

Having issues? Let us know on GitHub by opening an issue in the [Azure Container Apps repo](#).

Next steps

This quickstart is the entrypoint for a set of progressive tutorials that showcase the various features within Azure Container Apps. Continue on to learn how to enable communication from a web front end that calls the API you deployed in this article.

[Tutorial: Communication between microservices](#)

Feedback

Was this page helpful?

Yes

No

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

Tutorial: Communication between microservices in Azure Container Apps

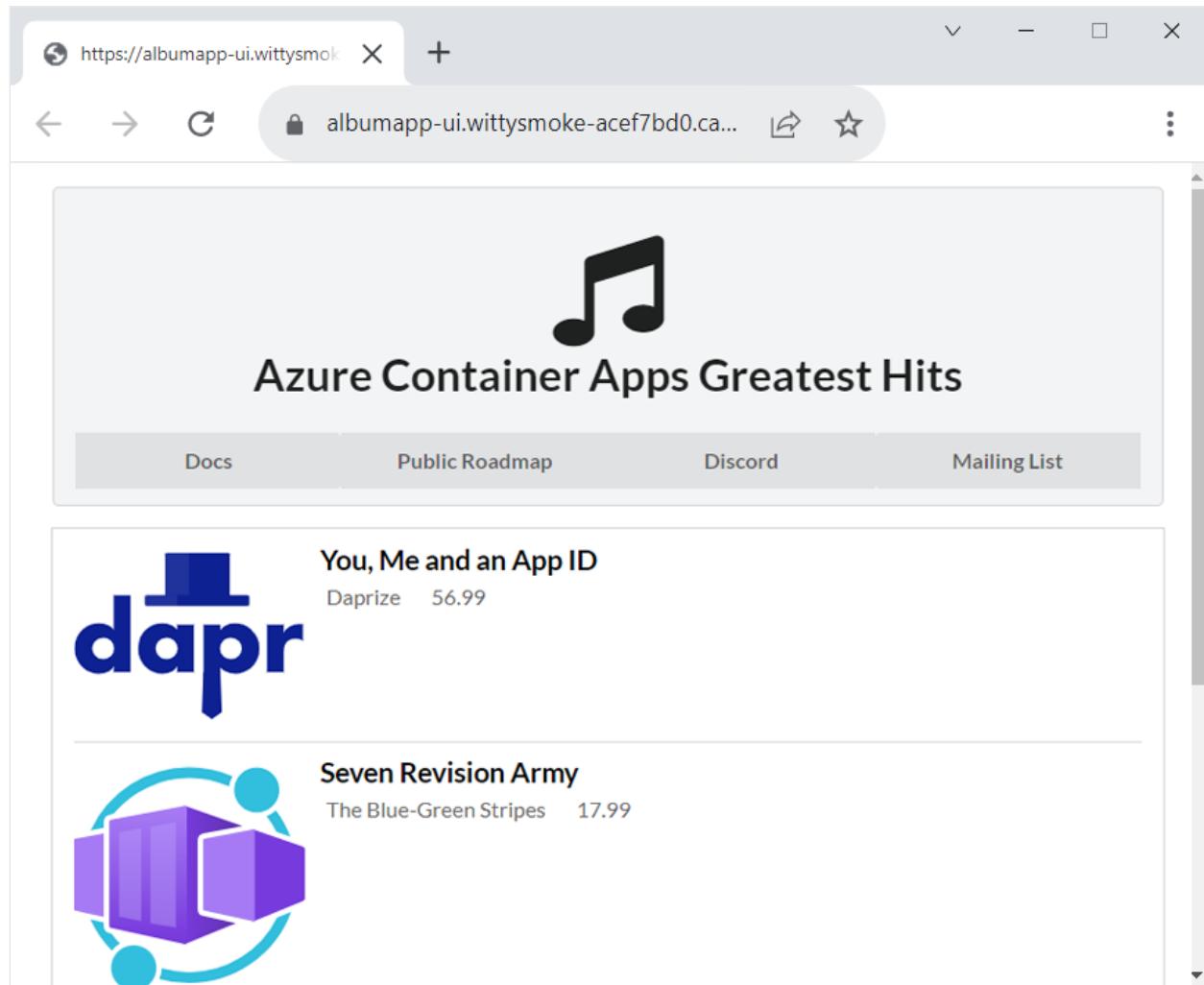
Article • 03/08/2023

Azure Container Apps exposes each container app through a domain name if [ingress](#) is enabled. Ingress endpoints for container apps within an external environment can be either publicly accessible or only available to other container apps in the same [environment](#).

Once you know the fully qualified domain name for a given container app, you can make direct calls to the service from other container apps within the shared environment.

In this tutorial, you deploy a second container app that makes a direct service call to the API deployed in the [Deploy your code to Azure Container Apps](#) quickstart.

The following screenshot shows the UI microservice deploys to container apps at the end of this article.



In this tutorial, you learn to:

- ✓ Deploy a front end application to Azure Container Apps
- ✓ Link the front end app to the API endpoint deployed in the previous quickstart
- ✓ Verify the frontend app can communicate with the back end API

Prerequisites

In the [code to cloud quickstart](#), a back end web API is deployed to return a list of music albums. If you haven't deployed the album API microservice, return to [Quickstart: Deploy your code to Azure Container Apps](#) to continue.

Setup

If you're still authenticated to Azure and still have the environment variables defined from the quickstart, you can skip the following steps and go directly to the [Prepare the GitHub repository](#) section.

Bash

Define the following variables in your bash shell.

```
Azure CLI  
  
RESOURCE_GROUP="album-containerapps"  
LOCATION="canadacentral"  
ENVIRONMENT="env-album-containerapps"  
API_NAME="album-api"  
FRONTEND_NAME="album-ui"  
GITHUB_USERNAME="<YOUR_GITHUB_USERNAME>"
```

Before you run this command, make sure to replace `<YOUR_GITHUB_USERNAME>` with your GitHub username.

Next, define a container registry name unique to you.

```
Azure CLI  
  
ACR_NAME="acaalbums"$GITHUB_USERNAME
```

Sign in to the Azure CLI.

Bash

Azure CLI

```
az login
```

Prepare the GitHub repository

1. In a new browser tab, navigate to the [repository for the UI application](#) and select the **Fork** button at the top of the page to fork the repo to your account.

Follow the prompts from GitHub to fork the repository and return here once the operation is complete.

2. Navigate to the parent of the *code-to-cloud* folder. If you're still in the *code-to-cloud/src* directory, you can use the below command to return to the parent folder.

Console

```
cd ../../
```

3. Use the following git command to clone your forked repo into the *code-to-cloud-ui* folder:

git

```
git clone https://github.com/$GITHUB_USERNAME/containerapps-albumui.git  
code-to-cloud-ui
```

⚠ Note

If the `clone` command fails, check that you have successfully forked the repository.

4. Next, change the directory into the *src* folder of the cloned repo.

Console

```
cd code-to-cloud-ui/src
```

Build the front end application

Bash

Azure CLI

```
az acr build --registry $ACR_NAME --image albumapp-ui .
```

Output from the `az acr build` command shows the upload progress of the source code to Azure and the details of the `docker build` operation.

Communicate between container apps

In the previous quickstart, the album API was deployed by creating a container app and enabling external ingress. Setting the container app's ingress to *external* made its HTTP endpoint URL publicly available.

Now you can configure the front end application to call the API endpoint by going through the following steps:

- Query the API application for its fully qualified domain name (FQDN).
- Pass the API FQDN to `az containerapp create` as an environment variable so the UI app can set the base URL for the album API call within the code.

The [UI application](#) uses the endpoint provided to invoke the album API. The following code is an excerpt from the code used in the *routes > index.js* file.

JavaScript

```
const api = axios.create({
  baseURL: process.env.API_BASE_URL,
  params: {},
  timeout: process.env.TIMEOUT || 5000,
});
```

Notice how the `baseURL` property gets its value from the `API_BASE_URL` environment variable.

Run the following command to query for the API endpoint address.

Bash

Azure CLI

```
API_BASE_URL=$(az containerapp show --resource-group $RESOURCE_GROUP --name $API_NAME --query properties.configuration.ingress.fqdn -o tsv)
```

Now that you have set the `API_BASE_URL` variable with the FQDN of the album API, you can provide it as an environment variable to the frontend container app.

Deploy front end application

Create and deploy your container app with the following command.

Bash

Azure CLI

```
az containerapp create \
--name $FRONTEND_NAME \
--resource-group $RESOURCE_GROUP \
--environment $ENVIRONMENT \
--image $ACR_NAME.azurecr.io/albumapp-ui \
--target-port 3000 \
--env-vars API_BASE_URL=https://$API_BASE_URL \
--ingress 'external' \
--registry-server $ACR_NAME.azurecr.io \
--query properties.configuration.ingress.fqdn
```

By adding the argument `--env-vars "API_BASE_URL=https://$API_ENDPOINT"` to `az containerapp create`, you define an environment variable for your front end application. With this syntax, the environment variable named `API_BASE_URL` is set to the API's FQDN.

The output from the `az containerapp create` command shows the URL of the front end application.

View website

Use the container app's FQDN to view the website. The page will resemble the following screenshot.

Docs Public Roadmap Discord Mailing List

You, Me and an App ID
Daprize 56.99

Seven Revision Army
The Blue-Green Stripes 17.99

Clean up resources

If you're not going to continue to use this application, run the following command to delete the resource group along with all the resources created in this quickstart.

⊗ Caution

This command deletes the specified resource group and all resources contained within it. If resources outside the scope of this tutorial exist in the specified resource group, they will also be deleted.

Bash

Azure CLI

```
az group delete --name $RESOURCE_GROUP
```



Tip

Having issues? Let us know on GitHub by opening an issue in the [Azure Container Apps repo](#).

Next steps

[Environments in Azure Container Apps](#)

Deploy to Azure Container Apps from Azure Pipelines

Article • 09/01/2023

Azure Container Apps allows you to use Azure Pipelines to publish [revisions](#) to your container app. As commits are pushed to your [Azure DevOps repository](#), a pipeline is triggered which updates the container image in the container registry. Azure Container Apps creates a new revision based on the updated container image.

The pipeline is triggered by commits to a specific branch in your repository. When creating the pipeline, you decide which branch is the trigger.

Container Apps Azure Pipelines task

The task supports the following scenarios:

- Build from a Dockerfile and deploy to Container Apps
- Build from source code without a Dockerfile and deploy to Container Apps.
Supported languages include .NET, Node.js, PHP, Python, and Ruby
- Deploy an existing container image to Container Apps

With the production release this task comes with Azure DevOps and no longer requires explicit installation. For the complete documentation please see [AzureContainerApps@1 - Azure Container Apps Deploy v1 task](#).

Usage examples

Here are some common scenarios for using the task. For more information, see the [task's documentation](#).

Build and deploy to Container Apps

The following snippet shows how to build a container image from source code and deploy it to Container Apps.

YAML

```
steps:  
- task: AzureContainerApps@1  
  inputs:  
    appSourcePath: '$(Build.SourcesDirectory)/src'
```

```
azureSubscription: 'my-subscription-service-connection'
acrName: 'myregistry'
containerAppName: 'my-container-app'
resourceGroup: 'my-container-app-rg'
```

The task uses the Dockerfile in `appSourcePath` to build the container image. If no Dockerfile is found, the task attempts to build the container image from source code in `appSourcePath`.

Deploy an existing container image to Container Apps

The following snippet shows how to deploy an existing container image to Container Apps. The task authenticates with the registry using the service connection. If the service connection's identity isn't assigned the `AcrPush` role for the registry, supply the registry's admin credentials using the `acrUsername` and `acrPassword` input parameters.

YAML

```
steps:
- task: AzureContainerApps@1
  inputs:
    azureSubscription: 'my-subscription-service-connection'
    containerAppName: 'my-container-app'
    resourceGroup: 'my-container-app-rg'
    imageToDeploy: 'myregistry.azurecr.io/my-container-
app:${Build.BuildId}'
```

ⓘ Important

If you're building a container image in a separate step, make sure you use a unique tag such as the build ID instead of a stable tag like `latest`. For more information, see [Image tag best practices](#).

Authenticate with Azure Container Registry

The Azure Container Apps task needs to authenticate with your Azure Container Registry to push the container image. The container app also needs to authenticate with your Azure Container Registry to pull the container image.

To push images, the task automatically authenticates with the container registry specified in `acrName` using the service connection provided in `azureSubscription`. If the

service connection's identity isn't assigned the `AcrPush` role for the registry, supply the registry's admin credentials using `acrUsername` and `acrPassword`.

To pull images, Azure Container Apps uses either managed identity (recommended) or admin credentials to authenticate with the Azure Container Registry. To use managed identity, the target container app for the task must be [configured to use managed identity](#). To authenticate with the registry's admin credentials, set the task's `acrUsername` and `acrPassword` inputs.

Configuration

Take the following steps to configure an Azure DevOps pipeline to deploy to Azure Container Apps.

- ✓ Create an Azure DevOps repository for your app
- ✓ Create a container app with managed identity enabled
- ✓ Assign the `AcrPull` role for the Azure Container Registry to the container app's managed identity
- ✓ Install the Azure Container Apps task from the Azure DevOps Marketplace
- ✓ Configure an Azure DevOps service connection for your Azure subscription
- ✓ Create an Azure DevOps pipeline

Prerequisites

Requirement	Instructions
Azure account	If you don't have one, create an account for free . You need the <i>Contributor</i> or <i>Owner</i> permission on the Azure subscription to proceed. Refer to Assign Azure roles using the Azure portal for details.
Azure Devops project	Go to Azure DevOps and select <i>Start free</i> . Then create a new project.
Azure CLI	Install the Azure CLI .

Create an Azure DevOps repository and clone the source code

Before creating a pipeline, the source code for your app must be in a repository.

1. Log in to [Azure DevOps](#) and navigate to your project.

2. Open the **Repos** page.
3. In the top navigation bar, select the repositories dropdown and select **Import repository**.
4. Enter the following information and select **Import**:

Field	Value
Repository type	Git
Clone URL	<code>https://github.com/Azure-Samples/containerapps-albumapi-csharp.git</code>
Name	<code>my-container-app</code>

5. Select **Clone** to view the repository URL and copy it.
6. Open a terminal and run the following command to clone the repository:

```
Bash  
git clone <REPOSITORY_URL> my-container-app
```

Replace `<REPOSITORY_URL>` with the URL you copied.

Create a container app and configure managed identity

Create your container app using the `az containerapp up` command with the following steps. This command creates Azure resources, builds the container image, stores the image in a registry, and deploys to a container app.

After your app is created, you can add a managed identity to your app and assign the identity the `AcrPull` role to allow the identity to pull images from the registry.

1. Change into the `src` folder of the cloned repository.

```
Bash  
cd my-container-app  
cd src
```

2. Create Azure resources and deploy a container app with the [az containerapp up command](#).

```
Azure CLI
```

```
az containerapp up \
--name my-container-app \
--source . \
--ingress external
```

3. In the command output, note the name of the Azure Container Registry.

4. Get the full resource ID of the container registry.

Azure CLI

```
az acr show --name <ACR_NAME> --query id --output tsv
```

Replace `<ACR_NAME>` with the name of your registry.

5. Enable managed identity for the container app.

Azure CLI

```
az containerapp identity assign \
--name my-container-app \
--resource-group my-container-app-rg \
--system-assigned \
--output tsv
```

Note the principal ID of the managed identity in the command output.

6. Assign the `AcrPull` role for the Azure Container Registry to the container app's managed identity.

Azure CLI

```
az role assignment create \
--assignee <MANAGED_IDENTITY_PRINCIPAL_ID> \
--role AcrPull \
--scope <ACR_RESOURCE_ID>
```

Replace `<MANAGED_IDENTITY_PRINCIPAL_ID>` with the principal ID of the managed identity and `<ACR_RESOURCE_ID>` with the resource ID of the Azure Container Registry.

7. Configure the container app to use the managed identity to pull images from the Azure Container Registry.

Azure CLI

```
az containerapp registry set \
--name my-container-app \
--resource-group my-container-app-rg \
--server <ACR_NAME>.azurecr.io \
--identity system
```

Replace `<ACR_NAME>` with the name of your Azure Container Registry.

Create an Azure DevOps service connection

To deploy to Azure Container Apps, you need to create an Azure DevOps service connection for your Azure subscription.

1. In Azure DevOps, select **Project settings**.
2. Select **Service connections**.
3. Select **New service connection**.
4. Select **Azure Resource Manager**.
5. Select **Service principal (automatic)** and select **Next**.
6. Enter the following information and select **Save**:

Field	Value
Subscription	Select your Azure subscription.
Resource group	Select the resource group (<code>my-container-app-rg</code>) that contains your container app and container registry.
Service connection name	<code>my-subscription-service-connection</code>

To learn more about service connections, see [Connect to Microsoft Azure](#).

Create an Azure DevOps YAML pipeline

1. In your Azure DevOps project, select **Pipelines**.
2. Select **New pipeline**.
3. Select **Azure Repos Git**.
4. Select the repo that contains your source code (`my-container-app`).

5. Select **Starter pipeline**.

6. In the editor, replace the contents of the file with the following YAML:

```
YAML

trigger:
  branches:
    include:
      - main

pool:
  vmImage: ubuntu-latest

steps:
  - task: AzureContainerApps@1
    inputs:
      appSourcePath: '$(Build.SourcesDirectory)/src'
      azureSubscription: '<AZURE_SUBSCRIPTION_SERVICE_CONNECTION>'
      acrName: '<ACR_NAME>'
      containerAppName: 'my-container-app'
      resourceGroup: 'my-container-app-rg'
```

Replace `<AZURE_SUBSCRIPTION_SERVICE_CONNECTION>` with the name of the Azure DevOps service connection (`my-subscription-service-connection`) you created in the previous step and `<ACR_NAME>` with the name of your Azure Container Registry.

7. Select **Save and run**.

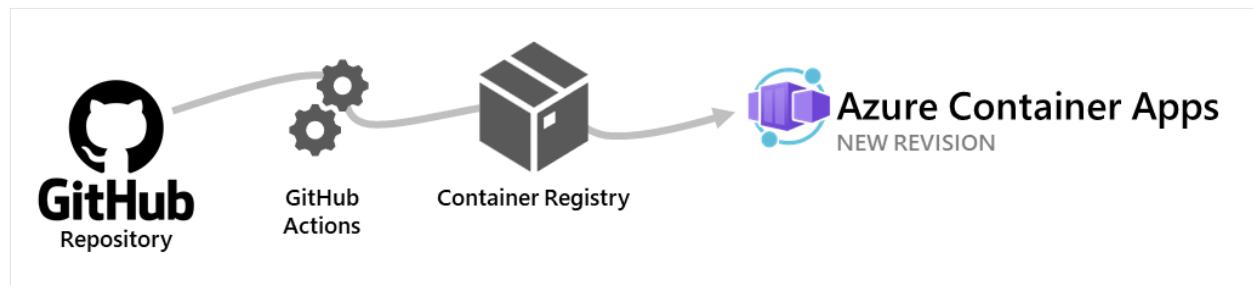
An Azure Pipelines run starts to build and deploy your container app. To check its progress, navigate to *Pipelines* and select the run. During the first pipeline run, you may be prompted to authorize the pipeline to use your service connection.

To deploy a new revision of your app, push a new commit to the *main* branch.

Deploy to Azure Container Apps with GitHub Actions

Article • 03/08/2023

Azure Container Apps allows you to use GitHub Actions to publish [revisions](#) to your container app. As commits are pushed to your GitHub repository, a workflow is triggered which updates the container image in the container registry. Azure Container Apps creates a new revision based on the updated container image.



The GitHub Actions workflow is triggered by commits to a specific branch in your repository. When creating the workflow, you decide which branch triggers the workflow.

This article shows you how to create a fully customizable workflow. To generate a starter GitHub Actions workflow with Azure CLI, see [Generate GitHub Actions workflow with Azure CLI](#).

Azure Container Apps GitHub action

To build and deploy your container app, you add the [azure/container-apps-deploy-action](#) action to your GitHub Actions workflow.

The action supports the following scenarios:

- Build from a Dockerfile and deploy to Container Apps
- Build from source code without a Dockerfile and deploy to Container Apps.
Supported languages include .NET, Node.js, PHP, Python, and Ruby
- Deploy an existing container image to Container Apps

Usage examples

Here are some common scenarios for using the action. For more information, see the action's [GitHub Marketplace page](#).

Build and deploy to Container Apps

The following snippet shows how to build a container image from source code and deploy it to Container Apps.

YAML

```
steps:  
  
  - name: Log in to Azure  
    uses: azure/login@v1  
    with:  
      creds: ${{ secrets.AZURE_CREDENTIALS }}  
  
  - name: Build and deploy Container App  
    uses: azure/container-apps-deploy-action@v1  
    with:  
      appSourcePath: ${{ github.workspace }}/src  
      acrName: myregistry  
      containerAppName: my-container-app  
      resourceGroup: my-rg
```

The action uses the Dockerfile in `appSourcePath` to build the container image. If no Dockerfile is found, the action attempts to build the container image from source code in `appSourcePath`.

Deploy an existing container image to Container Apps

The following snippet shows how to deploy an existing container image to Container Apps.

YAML

```
steps:  
  
  - name: Log in to Azure  
    uses: azure/login@v1  
    with:  
      creds: ${{ secrets.AZURE_CREDENTIALS }}  
  
  - name: Build and deploy Container App  
    uses: azure/container-apps-deploy-action@v1  
    with:  
      acrName: myregistry  
      containerAppName: my-container-app  
      resourceGroup: my-rg  
      imageToDeploy: myregistry.azurecr.io/app:${{ github.sha }}
```

ⓘ Important

If you're building a container image in a separate step, make sure you use a unique tag such as the commit SHA instead of a stable tag like `latest`. For more information, see [Image tag best practices](#).

Authenticate with Azure Container Registry

The Azure Container Apps action needs to authenticate with your Azure Container Registry to push the container image. The container app also needs to authenticate with your Azure Container Registry to pull the container image.

To push images, the action automatically authenticates with the container registry specified in `acrName` using the credentials provided to the `azure/login` action.

To pull images, Azure Container Apps uses either managed identity (recommended) or admin credentials to authenticate with the Azure Container Registry. To use managed identity, the container app the action is deploying must be [configured to use managed identity](#). To authenticate with the registry's admin credentials, set the action's `acrUsername` and `acrPassword` inputs.

Configuration

You take the following steps to configure a GitHub Actions workflow to deploy to Azure Container Apps.

- ✓ Create a GitHub repository for your app
- ✓ Create a container app with managed identity enabled
- ✓ Assign the `AcrPull` role for the Azure Container Registry to the container app's managed identity
- ✓ Configure secrets in your GitHub repository
- ✓ Create a GitHub Actions workflow

Prerequisites

Requirement	Instructions
Azure account	If you don't have one, create an account for free . You need the <i>Contributor</i> or <i>Owner</i> permission on the Azure subscription to proceed. Refer to Assign Azure roles using the Azure portal for details.

Requirement	Instructions
GitHub Account	Sign up for free ↗ .
Azure CLI	Install the Azure CLI .

Create a GitHub repository and clone source code

Before creating a workflow, the source code for your app must be in a GitHub repository.

1. Log in to Azure with the Azure CLI.

```
Azure CLI
```

```
az login
```

2. Next, install the latest Azure Container Apps extension for the CLI.

```
Azure CLI
```

```
az extension add --name containerapp --upgrade
```

3. If you do not have your own GitHub repository, create one from a sample.

- a. Navigate to the following location to create a new repository:

- [https://github.com/Azure-Samples/containerapps-albumapi-csharp/generate ↗](https://github.com/Azure-Samples/containerapps-albumapi-csharp/generate)

- b. Name your repository `my-container-app`.

4. Clone the repository to your local machine.

```
git
```

```
git clone https://github.com/<YOUR_GITHUB_ACCOUNT_NAME>/my-container-app.git
```

Create a container app with managed identity enabled

Create your container app using the `az containerapp up` command in the following steps. This command will create Azure resources, build the container image, store the

image in a registry, and deploy to a container app.

After you create your app, you can add a managed identity to the app and assign the identity the `AcrPull` role to allow the identity to pull images from the registry.

1. Change into the `src` folder of the cloned repository.

```
Bash
```

```
cd my-container-app  
cd src
```

2. Create Azure resources and deploy a container app with the [az containerapp up command](#).

```
Azure CLI
```

```
az containerapp up \  
--name my-container-app \  
--source . \  
--ingress external
```

3. In the command output, note the name of the Azure Container Registry.

4. Get the full resource ID of the container registry.

```
Azure CLI
```

```
az acr show --name <ACR_NAME> --query id --output tsv
```

Replace `<ACR_NAME>` with the name of your registry.

5. Enable managed identity for the container app.

```
Azure CLI
```

```
az containerapp identity assign \  
--name my-container-app \  
--resource-group my-container-app-rg \  
--system-assigned \  
--output tsv
```

Note the principal ID of the managed identity in the command output.

6. Assign the `AcrPull` role for the Azure Container Registry to the container app's managed identity.

Azure CLI

```
az role assignment create \
--assignee <MANAGED_IDENTITY_PRINCIPAL_ID> \
--role AcrPull \
--scope <ACR_RESOURCE_ID>
```

Replace `<MANAGED_IDENTITY_PRINCIPAL_ID>` with the principal ID of the managed identity and `<ACR_RESOURCE_ID>` with the resource ID of the Azure Container Registry.

7. Configure the container app to use the managed identity to pull images from the Azure Container Registry.

Azure CLI

```
az containerapp registry set \
--name my-container-app \
--resource-group my-container-app-rg \
--server <ACR_NAME>.azurecr.io \
--identity system
```

Replace `<ACR_NAME>` with the name of your Azure Container Registry.

Configure secrets in your GitHub repository

The GitHub workflow requires a secret named `AZURE_CREDENTIALS` to authenticate with Azure. The secret contains the credentials for a service principal with the *Contributor* role on the resource group containing the container app and container registry.

1. Create a service principal with the *Contributor* role on the resource group that contains the container app and container registry.

Azure CLI

```
az ad sp create-for-rbac \
--name my-app-credentials \
--role contributor \
--scopes /subscriptions/<SUBSCRIPTION_ID>/resourceGroups/my-
container-app-rg \
--sdk-auth \
--output json
```

Replace `<SUBSCRIPTION_ID>` with the ID of your Azure subscription. If your container registry is in a different resource group, specify both resource groups in

the `--scopes` parameter.

2. Copy the JSON output from the command.
3. In the GitHub repository, navigate to *Settings > Secrets > Actions* and select **New repository secret**.
4. Enter `AZURE_CREDENTIALS` as the name and paste the contents of the JSON output as the value.
5. Select **Add secret**.

Create a GitHub Actions workflow

1. In the GitHub repository, navigate to *Actions* and select **New workflow**.
2. Select **Set up a workflow yourself**.
3. Paste the following YAML into the editor.

```
YAML

name: Azure Container Apps Deploy

on:
  push:
    branches:
      - main

jobs:
  build:
    runs-on: ubuntu-latest

  steps:
    - uses: actions/checkout@v3

    - name: Log in to Azure
      uses: azure/login@v1
      with:
        creds: ${{ secrets.AZURE_CREDENTIALS }}

    - name: Build and deploy Container App
      uses: azure/container-apps-deploy-action@v1
      with:
        appSourcePath: ${{ github.workspace }}/src
        acrName: <ACR_NAME>
        containerAppName: my-container-app
        resourceGroup: my-container-app-rg
```

Replace `<ACR_NAME>` with the name of your Azure Container Registry. Confirm that the branch name under `branches` and values for `appSourcePath`, `containerAppName`, and `resourceGroup` match the values for your repository and Azure resources.

4. Commit the changes to the *main* branch.

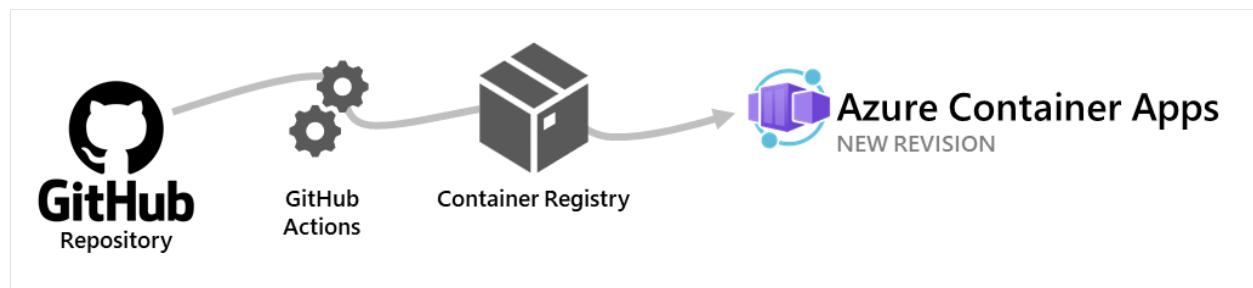
A GitHub Actions workflow run should start to build and deploy your container app. To check its progress, navigate to *Actions*.

To deploy a new revision of your app, push a new commit to the *main* branch.

Set up GitHub Actions with Azure CLI in Azure Container Apps

Article • 04/09/2023

Azure Container Apps allows you to use GitHub Actions to publish [revisions](#) to your container app. As commits are pushed to your GitHub repository, a GitHub Actions workflow is triggered which updates the [container](#) image in the container registry. Once the container is updated in the registry, Azure Container Apps creates a new revision based on the updated container image.



The GitHub Actions workflow is triggered by commits to a specific branch in your repository. When creating the workflow, you decide which branch triggers the action.

This article shows you how to generate a starter GitHub Actions workflow with Azure CLI. To create your own workflow that you can fully customize, see [Deploy to Azure Container Apps with GitHub Actions](#).

Authentication

When adding or removing a GitHub Actions integration, you can authenticate by either passing in a GitHub [personal access token](#), or using the interactive GitHub login experience. The interactive experience opens a form in your web browser and gives you the opportunity to log in to GitHub. Once successfully authenticated, then a token is passed back to the CLI that is used by GitHub for the rest of the current session.

- To pass a personal access token, use the `--token` parameter and provide a token value.
- If you choose to use interactive login, use the `--login-with-github` parameter with no value.

! Note

Your GitHub personal access token needs to have the `workflow` scope selected.

Add

The `containerapp github-action add` command creates a GitHub Actions integration with your container app.

ⓘ Note

Before you proceed with the example below, you must have your first container app already deployed.

The first time you attach GitHub Actions to your container app, you need to provide a service principal context. The following command shows you how to create a service principal.

Bash

Azure CLI

```
az ad sp create-for-rbac \
--name <SERVICE_PRINCIPAL_NAME> \
--role "contributor" \
--scopes
/subscriptions/<SUBSCRIPTION_ID>/resourceGroups/<RESOURCE_GROUP_NAME>
```

As you interact with this example, replace the placeholders surrounded by `<>` with your values.

The return values from this command include the service principal's `appId`, `password`, and `tenant`. You need to pass these values to the `az containerapp github-action add` command.

The following example shows you how to add an integration while using a personal access token.

Bash

Azure CLI

```
az containerapp github-action add \
--repo-url "https://github.com/<OWNER>/<REPOSITORY_NAME>" \
--context-path "./dockerfile" \
```

```
--branch <BRANCH_NAME> \
--name <CONTAINER_APP_NAME> \
--resource-group <RESOURCE_GROUP> \
--registry-url <URL_TO_CONTAINER_REGISTRY> \
--registry-username <REGISTRY_USER_NAME> \
--registry-password <REGISTRY_PASSWORD> \
--service-principal-client-id <appId> \
--service-principal-client-secret <password> \
--service-principal-tenant-id <tenant> \
--token <YOUR_GITHUB_PERSONAL_ACCESS_TOKEN>
```

As you interact with this example, replace the placeholders surrounded by `<>` with your values.

Show

The `containerapp github-action show` command returns the GitHub Actions configuration settings for a container app.

This example shows how to add an integration while using the personal access token.

Bash

Azure CLI

```
az containerapp github-action show \
--resource-group <RESOURCE_GROUP_NAME> \
--name <CONTAINER_APP_NAME>
```

As you interact with this example, replace the placeholders surrounded by `<>` with your values.

This command returns a JSON payload with the GitHub Actions integration configuration settings.

Delete

The `containerapp github-action delete` command removes the GitHub Actions from the container app.

Bash

Azure CLI

```
az containerapp github-action delete \
--resource-group <RESOURCE_GROUP_NAME> \
--name <CONTAINER_APP_NAME> \
--token <YOUR_GITHUB_PERSONAL_ACCESS_TOKEN>
```

As you interact with this example, replace the placeholders surrounded by `<>` with your values.

Tutorial: Deploy self-hosted CI/CD runners and agents with Azure Container Apps jobs

Article • 10/16/2024

GitHub Actions and Azure Pipelines allow you to run CI/CD workflows with self-hosted runners and agents. You can run self-hosted runners and agents using event-driven Azure Container Apps [jobs](#).

Self-hosted runners are useful when you need to run workflows that require access to local resources or tools that aren't available to a cloud-hosted runner. For example, a self-hosted runner in a Container Apps job allows your workflow to access resources inside the job's virtual network that isn't accessible to a cloud-hosted runner.

Running self-hosted runners as event-driven jobs allows you to take advantage of the serverless nature of Azure Container Apps. Jobs execute automatically when a workflow is triggered and exit when the job completes.

You only pay for the time that the job is running.

In this tutorial, you learn how to run Azure Pipelines agents as an [event-driven Container Apps job](#).

- ✓ Create a Container Apps environment to deploy your self-hosted agent
- ✓ Create an Azure DevOps organization and project
- ✓ Build a container image that runs an Azure Pipelines agent
- ✓ Use a manual job to create a placeholder agent in the Container Apps environment
- ✓ Deploy the agent as a job to the Container Apps environment
- ✓ Create a pipeline that uses the self-hosted agent and verify that it runs

Important

Self-hosted agents are only recommended for *private* projects. Using them with public projects can allow dangerous code to execute on your self-hosted agent. For more information, see [Self-hosted agent security](#).

Note

Container apps and jobs don't support running Docker in containers. Any steps in your workflows that use Docker commands will fail when run on a self-hosted runner or agent in a Container Apps job.

Prerequisites

- **Azure account:** If you don't have one, you [can create one for free ↗](#).
- **Azure CLI:** Install the [Azure CLI](#).
- **Azure DevOps organization:** If you don't have a DevOps organization with an active subscription, you [can create one for free ↗](#).

Refer to [jobs restrictions](#) for a list of limitations.

Setup

To sign in to Azure from the CLI, run the following command and follow the prompts to complete the authentication process.

```
Bash
az login
```

To ensure you're running the latest version of the CLI, run the upgrade command.

```
Bash
az upgrade
```

Next, install or update the Azure Container Apps extension for the CLI.

If you receive errors about missing parameters when you run `az containerapp` commands in Azure CLI or cmdlets from the `Az.App` module in Azure PowerShell, be sure you have the latest version of the Azure Container Apps extension installed.

Bash

Azure CLI

```
az extension add --name containerapp --upgrade
```

① Note

Starting in May 2024, Azure CLI extensions no longer enable preview features by default. To access Container Apps [preview features](#), install the Container Apps extension with `--allow-preview true`.

Azure CLI

```
az extension add --name containerapp --upgrade --allow-preview true
```

Now that the current extension or module is installed, register the `Microsoft.App` and `Microsoft.OperationalInsights` namespaces.

Bash

Azure CLI

```
az provider register --namespace Microsoft.App
```

Azure CLI

```
az provider register --namespace Microsoft.OperationalInsights
```

Create environment variables

Now that your Azure CLI setup is complete, you can define the environment variables that are used throughout this article.

Bash

Bash

```
RESOURCE_GROUP="jobs-sample"
LOCATION="northcentralus"
ENVIRONMENT="env-jobs-sample"
JOB_NAME="azure-pipelines-agent-job"
PLACEHOLDER_JOB_NAME="placeholder-agent-job"
```

Create a Container Apps environment

The Azure Container Apps environment acts as a secure boundary around container apps and jobs so they can share the same network and communicate with each other.

ⓘ Note

To create a Container Apps environment that's integrated with an existing virtual network, see [Provide a virtual network to an internal Azure Container Apps environment](#).

1. Create a resource group using the following command.

```
Bash
```

```
Bash
```

```
az group create \
--name "$RESOURCE_GROUP" \
--location "$LOCATION"
```

2. Create the Container Apps environment using the following command.

```
Bash
```

```
Bash
```

```
az containerapp env create \
--name "$ENVIRONMENT" \
--resource-group "$RESOURCE_GROUP" \
--location "$LOCATION"
```

Create an Azure DevOps project and repository

To execute a pipeline, you need an Azure DevOps project and repository.

1. Navigate to [Azure DevOps](#) and sign in to your account.
2. Select an existing organization or create a new one.
3. In the organization overview page, select **New project** and enter the following values.

[] [Expand table](#)

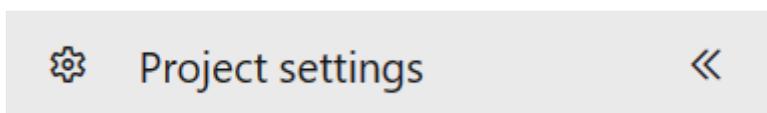
Setting	Value
<i>Project name</i>	Enter a name for your project.
<i>Visibility</i>	Select Private .

4. Select **Create**.
5. From the side navigation, select **Repos**.
6. Under *Initialize main branch with a README or .gitignore*, select **Add a README**.
7. Leave the rest of the values as defaults and select **Initialize**.

Create a new agent pool

Create a new agent pool to run the self-hosted runner.

1. In your Azure DevOps project, expand the left navigation bar and select **Project settings**.



2. Under the *Pipelines* section in the *Project settings* navigation menu, select **Agent pools**.

Pipelines

 Agent pools

 Parallel jobs

 Settings

 Test management

 Release retention

 Service connections

 XAML build services

3. Select **Add pool** and enter the following values.

 Expand table

Setting	Value
<i>Pool to link</i>	Select New .
<i>Pool type</i>	Select Self-hosted .
<i>Name</i>	Enter container-apps .
<i>Grant access permission to all pipelines</i>	Select this checkbox.

4. Select **Create**.

Get an Azure DevOps personal access token

To run a self-hosted runner, you need to create a personal access token (PAT) in Azure DevOps. The PAT is used to authenticate the runner with Azure DevOps. It's also used by the scale rule to determine the number of pending pipeline runs and trigger new job executions.

[!NOTE]

Personal Access Tokens (PATs) have an expiration date. Regularly rotate your tokens to ensure they remain valid (not expired) to maintain uninterrupted service.

1. In Azure DevOps, select *User settings* next to your profile picture in the upper-right corner.
2. Select **Personal access tokens**.
3. In the *Personal access tokens* page, select **New Token** and enter the following values.

[] [Expand table](#)

Setting	Value
<i>Name</i>	Enter a name for your token.
<i>Organization</i>	Select the organization you chose or created earlier.
<i>Scopes</i>	Select Custom defined .
<i>Show all scopes</i>	Select Show all scopes .
<i>Agent Pools (Read & manage)</i>	Select Agent Pools (Read & manage) .

Leave all other scopes unselected.

4. Select **Create**.
5. Copy the token value to a secure location.
You can't retrieve the token after you leave the page.
6. Define variables that are used to configure the Container Apps jobs later.

```
Bash
Bash
AZP_TOKEN=<AZP_TOKEN>
ORGANIZATION_URL=<ORGANIZATION_URL>
AZP_POOL="container-apps"
REGISTRATION_TOKEN_API_URL=<YOUR_REGISTRATION_TOKEN_API_URL>
```

Replace the placeholders with the following values:

[] [Expand table](#)

Placeholder	Value	Comments
<AZP_TOKEN>	The Azure DevOps PAT you generated.	
<ORGANIZATION_URL>	The URL of your Azure DevOps organization. Make sure no trailing / is present at the end of the URL.	For example, https://dev.azure.com/myorg or https://myorg.visualstudio.com .
<YOUR_REGISTRATION_TOKEN_API_URL>	The registration token API URL in the <i>entrypoint.sh</i> file.	For example, ' https://myapi.example.com/get-token '

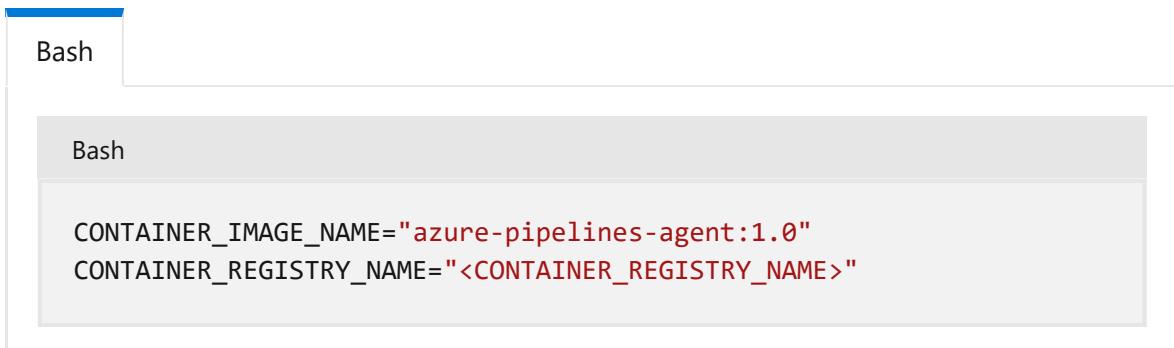
Build the Azure Pipelines agent container image

To create a self-hosted agent, you need to build a container image that runs the agent. In this section, you build the container image and push it to a container registry.

ⓘ Note

The image you build in this tutorial contains a basic self-hosted agent that's suitable for running as a Container Apps job. You can customize it to include additional tools or dependencies that your pipelines require.

1. Back in your terminal, define a name for your container image and registry.



```
Bash
CONTAINER_IMAGE_NAME="azure-pipelines-agent:1.0"
CONTAINER_REGISTRY_NAME="<CONTAINER_REGISTRY_NAME>"
```

Replace `<CONTAINER_REGISTRY_NAME>` with a unique name for creating a container registry.

Container registry names must be *unique within Azure* and be from 5 to 50 characters in length containing numbers and lowercase letters only.

2. Create a container registry.

The screenshot shows a terminal window with a blue header bar and a white body. The title bar says "Bash". Inside the terminal, there is another "Bash" prompt. Below it, a command is being typed:

```
az acr create \
--name "$CONTAINER_REGISTRY_NAME" \
--resource-group "$RESOURCE_GROUP" \
--location "$LOCATION" \
--sku Basic \
--admin-enabled true
```

3. The Dockerfile for creating the runner image is available on [GitHub](#). Run the following command to clone the repository and build the container image in the cloud using the `az acr build` command.

The screenshot shows a terminal window with a blue header bar and a white body. The title bar says "Bash". Inside the terminal, there is another "Bash" prompt. Below it, a command is being typed:

```
az acr build \
--registry "$CONTAINER_REGISTRY_NAME" \
--image "$CONTAINER_IMAGE_NAME" \
--file "Dockerfile.azure-pipelines" \
"https://github.com/Azure-Samples/container-apps-ci-cd-runner-tutorial.git"
```

The image is now available in the container registry.

Create a placeholder self-hosted agent

Before you can run a self-hosted agent in your new agent pool, you need to create a placeholder agent. The placeholder agent ensures the agent pool is available. Pipelines that use the agent pool fail when there's no placeholder agent.

You can run a manual job to register an offline placeholder agent. The job runs once and can be deleted. The placeholder agent doesn't consume any resources in Azure Container Apps or Azure DevOps.

1. Create a manual job in the Container Apps environment that creates the placeholder agent.

```
az containerapp job create -n "$PLACEHOLDER_JOB_NAME" -g "$RESOURCE_GROUP" --environment "$ENVIRONMENT" \
    --trigger-type Manual \
    --replica-timeout 300 \
    --replica-retry-limit 0 \
    --replica-completion-count 1 \
    --parallelism 1 \
    --image
"$CONTAINER_REGISTRY_NAME.azurecr.io/$CONTAINER_IMAGE_NAME" \
    --cpu "2.0" \
    --memory "4Gi" \
    --secrets "personal-access-token=$AZP_TOKEN" "organization-
url=$ORGANIZATION_URL" \
    --env-vars "AZP_TOKEN=secretref:personal-access-token"
"AZP_URL=secretref:organization-url" "AZP_POOL=$AZP_POOL"
"AZP_PLACEHOLDER=1" "AZP_AGENT_NAME=placeholder-agent" \
    --registry-server "$CONTAINER_REGISTRY_NAME.azurecr.io"
```

The following table describes the key parameters used in the command.

[\[\]](#) Expand table

Parameter	Description
--replica-timeout	The maximum duration a replica can execute.
--replica-retry-limit	The number of times to retry a failed replica.
--replica-completion-count	The number of replicas to complete successfully before a job execution is considered successful.
--parallelism	The number of replicas to start per job execution.
--secrets	The secrets to use for the job.
--env-vars	The environment variables to use for the job.

Parameter	Description
--registry-server	The container registry server to use for the job. For an Azure Container Registry, the command automatically configures authentication.

Setting the `AZP_PLACEHOLDER` environment variable configures the agent container to register as an offline placeholder agent without running a job.

2. Execute the manual job to create the placeholder agent.

```
Bash
az containerapp job start -n "$PLACEHOLDER_JOB_NAME" -g "$RESOURCE_GROUP"
```

3. List the executions of the job to confirm a job execution was created and completed successfully.

```
Bash
az containerapp job execution list \
--name "$PLACEHOLDER_JOB_NAME" \
--resource-group "$RESOURCE_GROUP" \
--output table \
--query '[].{Status: properties.status, Name: name, StartTime: properties.startTime}'
```

4. Verify the placeholder agent was created in Azure DevOps.
 - In Azure DevOps, navigate to your project.
 - Select **Project settings > Agent pools > container-apps > Agents**.
 - Confirm that a placeholder agent named `placeholder-agent` is listed and its status is offline.
5. The job isn't needed again. You can delete it.

Bash

```
az containerapp job delete -n "$PLACEHOLDER_JOB_NAME" -g  
"$RESOURCE_GROUP"
```

Create a self-hosted agent as an event-driven job

Now that you have a placeholder agent, you can create a self-hosted agent. In this section, you create an event-driven job that runs a self-hosted agent when a pipeline is triggered.

Bash

Bash

```
az containerapp job create -n "$JOB_NAME" -g "$RESOURCE_GROUP" --  
environment "$ENVIRONMENT" \  
    --trigger-type Event \  
    --replica-timeout 1800 \  
    --replica-retry-limit 0 \  
    --replica-completion-count 1 \  
    --parallelism 1 \  
    --image "$CONTAINER_REGISTRY_NAME.azurecr.io/$CONTAINER_IMAGE_NAME"  
    \  
    --min-executions 0 \  
    --max-executions 10 \  
    --polling-interval 30 \  
    --scale-rule-name "azure-pipelines" \  
    --scale-rule-type "azure-pipelines" \  
    --scale-rule-metadata "poolName=$AZP_POOL"  
    "targetPipelinesQueueLength=1" \  
    --scale-rule-auth "personalAccessToken=personal-access-token"  
    "organizationURL=organization-url" \  
    --cpu "2.0" \  
    --memory "4Gi" \  
    --secrets "personal-access-token=$AZP_TOKEN" "organization-  
url=$ORGANIZATION_URL" \  
    --env-vars "AZP_TOKEN=secretref:personal-access-token"  
    "AZP_URL=secretref:organization-url" "AZP_POOL=$AZP_POOL" \  
    --registry-server "$CONTAINER_REGISTRY_NAME.azurecr.io"
```

The following table describes the scale rule parameters used in the command.

Parameter	Description
--min-executions	The minimum number of job executions to run per polling interval.
--max-executions	The maximum number of job executions to run per polling interval.
--polling-interval	The polling interval at which to evaluate the scale rule.
--scale-rule-name	The name of the scale rule.
--scale-rule-type	The type of scale rule to use. To learn more about the Azure Pipelines scaler, see the KEDA documentation .
--scale-rule-metadata	The metadata for the scale rule.
--scale-rule-auth	The authentication for the scale rule.

The scale rule configuration defines the event source to monitor. It's evaluated on each polling interval and determines how many job executions to trigger. To learn more, see [Set scaling rules](#).

The event-driven job is now created in the Container Apps environment.

Run a pipeline and verify the job

Now that you've configured a self-hosted agent job, you can run a pipeline and verify it's working correctly.

1. In the left-hand navigation of your Azure DevOps project, navigate to **Pipelines**.
2. Select **Create pipeline**.
3. Select **Azure Repos Git** as the location of your code.
4. Select the repository you created earlier.
5. Select **Starter pipeline**.
6. In the pipeline YAML, change the `pool` from `vmImage: ubuntu-latest` to `name: container-apps`.

YAML

```
pool:  
  name: container-apps
```

7. Select Save and run.

The pipeline runs and uses the self-hosted agent job you created in the Container Apps environment.

8. List the executions of the job to confirm a job execution was created and completed successfully.

Bash

Bash

```
az containerapp job execution list \  
  --name "$JOB_NAME" \  
  --resource-group "$RESOURCE_GROUP" \  
  --output table \  
  --query '[].{Status: properties.status, Name: name, StartTime: properties.startTime}'
```

💡 Tip

Having issues? Let us know on GitHub by opening an issue in the [Azure Container Apps repo](#).

Clean up resources

Once you're done, run the following command to delete the resource group that contains your Container Apps resources.

⊗ Caution

The following command deletes the specified resource group and all resources contained within it. If resources outside the scope of this tutorial exist in the specified resource group, they will also be deleted.

Bash

Bash

```
az group delete \
--resource-group $RESOURCE_GROUP
```

To delete your GitHub repository, see [Deleting a repository ↗](#).

Next steps

[Container Apps jobs](#)

Feedback

Was this page helpful?

 Yes

 No

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

Manage revisions in Azure Container Apps

Article • 04/21/2024

Azure Container Apps allows your container app to support multiple revisions. With this feature, you can activate and deactivate revisions, and control the amount of [traffic sent to each revision](#). To learn more about revisions, see [Revisions in Azure Container Apps](#).

A revision is created when you first deploy your application. New revisions are created when you [update](#) your application with [revision-scope changes](#). You can also update your container app based on a specific revision.

This article describes the commands to manage your container app's revisions. For more information about Container Apps commands, see [az containerapp](#). For more information about commands to manage revisions, see [az containerapp revision](#).

Updating your container app

To update a container app, use the [az containerapp update](#) command. With this command you can modify environment variables, compute resources, scale parameters, and deploy a different image. If your container app update includes [revision-scope changes](#), a new revision is generated.

Bash

This example updates the container image. Replace the <PLACEHOLDERS> with your values.

Azure CLI

```
az containerapp update \
--name <APPLICATION_NAME> \
--resource-group <RESOURCE_GROUP_NAME> \
--image <IMAGE_NAME>
```

You can also update your container app with the [Revision copy](#) command.

Revision list

List all revisions associated with your container app with `az containerapp revision list`. For more information about this command, see [az containerapp revision list](#)

Bash

Replace the <PLACEHOLDERS> with your values.

Azure CLI

```
az containerapp revision list \
--name <APPLICATION_NAME> \
--resource-group <RESOURCE_GROUP_NAME> \
-o table
```

Revision show

Show details about a specific revision by using the [az containerapp revision show](#) command.

Bash

Replace the <PLACEHOLDERS> with your values.

Azure CLI

```
az containerapp revision show \
--name <APPLICATION_NAME> \
--revision <REVISION_NAME> \
--resource-group <RESOURCE_GROUP_NAME>
```

Revision copy

To create a new revision based on an existing revision, use the `az containerapp revision copy`. Container Apps uses the configuration of the existing revision, which you can then modify.

With this command, you can modify environment variables, compute resources, scale parameters, and deploy a different image. You can also use a YAML file to define these and other configuration options and parameters. For more information regarding this command, see [az containerapp revision copy](#).

This example copies the latest revision and sets the compute resource parameters.
(Replace the <PLACEHOLDERS> with your values.)

Bash

Azure CLI

```
az containerapp revision copy \
--name <APPLICATION_NAME> \
--resource-group <RESOURCE_GROUP_NAME> \
--cpu 0.75 \
--memory 1.5Gi
```

Revision activate

Activate a revision by using the [az containerapp revision activate](#) command.

Bash

Example: (Replace the <PLACEHOLDERS> with your values.)

Azure CLI

```
az containerapp revision activate \
--revision <REVISION_NAME> \
--resource-group <RESOURCE_GROUP_NAME>
```

Revision deactivate

Deactivate revisions that are no longer in use with the [az containerapp revision deactivate](#) command. Deactivation stops all running replicas of a revision.

Bash

Example: (Replace the <PLACEHOLDERS> with your values.)

Azure CLI

```
az containerapp revision deactivate \  
  --revision <REVISION_NAME> \  
  --resource-group <RESOURCE_GROUP_NAME>
```

Revision restart

The [az containerapp revision restart](#) command restarts a revision.

When you modify secrets in your container app, you need to restart the active revisions so they can access the secrets.

Bash

Example: (Replace the <PLACEHOLDERS> with your values.)

Azure CLI

```
az containerapp revision restart \  
  --revision <REVISION_NAME> \  
  --resource-group <RESOURCE_GROUP_NAME>
```

Revision set mode

The revision mode controls whether only a single revision or multiple revisions of your container app can be simultaneously active. To set your container app to support [single revision mode](#) or [multiple revision mode](#), use the [az containerapp revision set-mode](#) command.

The default setting is *single revision mode*. For more information about this command, see [az containerapp revision set-mode](#).

The mode values are `single` or `multiple`. Changing the revision mode doesn't create a new revision.

Example: (Replace the <PLACEHOLDERS> with your values.)

Bash

Example: (Replace the <PLACEHOLDERS> with your values.)

Azure CLI

```
az containerapp revision set-mode \
--name <APPLICATION_NAME> \
--resource-group <RESOURCE_GROUP_NAME> \
--mode <REVISION_MODE>
```

Revision labels

Labels provide a unique URL that you can use to direct traffic to a revision. You can move a label between revisions to reroute traffic directed to the label's URL to a different revision. For more information about revision labels, see [Revision Labels](#).

You can add and remove a label from a revision. For more information about the label commands, see [az containerapp revision label](#)

Revision label add

To add a label to a revision, use the [az containerapp revision label add](#) command.

You can only assign a label to one revision at a time, and a revision can only be assigned one label. If the revision you specify has a label, the add command replaces the existing label.

This example adds a label to a revision: (Replace the <PLACEHOLDERS> with your values.)

Bash

Azure CLI

```
az containerapp revision label add \
--revision <REVISION_NAME> \
--resource-group <RESOURCE_GROUP_NAME> \
--label <LABEL_NAME>
```

Revision label remove

To remove a label from a revision, use the [az containerapp revision label remove](#) command.

This example removes a label to a revision: (Replace the <PLACEHOLDERS> with your values.)

Bash

Azure CLI

```
az containerapp revision label remove \
--revision <REVISION_NAME> \
--resource-group <RESOURCE_GROUP_NAME> \
--label <LABEL_NAME>
```

Traffic splitting

Applied by assigning percentage values, you can decide how to balance traffic among different revisions. Traffic splitting rules are assigned by setting weights to different revisions by their name or [label](#). For more information, see, [Traffic Splitting](#).

Next steps

- [Revisions in Azure Container Apps](#)
- [Application lifecycle management in Azure Container Apps](#)

Manage environment variables on Azure Container Apps

Article • 09/24/2024

In Azure Container Apps, you're able to set runtime environment variables. These variables can be set as manually entries or as references to [secrets](#). These environment variables are loaded onto your Container App during runtime.

Configure environment variables

You can configure the Environment Variables upon the creation of the Container App or later by creating a new revision.

 Note

To avoid confusion, it is not recommended to duplicate environment variables. When multiple environment variables have the same name, the last one in the list takes effect.

Azure portal

If you're creating a new Container App through the [Azure portal](#), you can set up the environment variables on the Container section:

Create Container App

Basics Container Bindings Ingress Tags Review + create

Select a quickstart image for your container, or deselect quickstart image to use an existing container.

Use quickstart image

Container details

Name *

Image source

Azure Container Registry

Docker Hub or other registries

Registry *

No registries found

Image *

Select a registry first

Image tag *

Select a image first

Command override (i)

Example: /bin/bash, -c, echo hello; sleep 100000

Container resource allocation

Choose the workload profile for this app. You can adjust the CPU and memory allocation for this app up to the workload profile limit. [Learn more](#)

Workload profile *

Consumption - Up to 4 vCPUs, 8 Gib memory

CPU and Memory *

0.5 CPU cores, 1 Gi memory

Environment variables

Name	Value	Delete
<input type="text" value="Enter name"/>	<input type="text" value="Enter value"/>	<input type="button" value="Delete"/>

Add environment variables on existing container apps

After the Container App is created, the only way to update the Container App environment variables is by creating a new revision with the needed changes.

Azure portal

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

Container Apps

All Services (46) Marketplace (3)

Services

- Container Apps Environments
- Container Apps**
- App Services
- Container App Jobs

2. In the app's left menu, select Revisions and replicas > Create new revision

Container App

Search Create new revision

Overview Activity log Access control (IAM) Tags Diagnose and solve problems

Revision Mode: Single

Active revisions Inactive revisions

Name ↑

Name
vapp--kmckl9h
vapp--fosv9b0

3. Then you have to edit the current existing container image:

Container image

A revision needs one main app container. Add sidecar containers that support the main app container before app containers are started.

Edit Delete Add

Name	Source	Type	Tag
simple-hello-w...	Azure Container Registry	App container	latest

4. In the Environment variables section, you can Add new Environment variables by clicking on Add.

5. Then set the Name of your Environment variable and the Source (it can be a reference to a secret).

Environment variables				
Name	Source	Value	Delete	
envVarFromSecret	Reference a se...	Select a secret		

- a. If you select the Source as manual, you can manually input the Environment variable value.

Environment variables				
Name	Source	Value	Delete	
envVarFromSecret	Reference a se...	Select a secret		
envVar	Manual entry	envVarValue		

Built-in environment variables

Azure Container Apps automatically adds environment variables that your apps and jobs can use to obtain platform metadata at run-time.

Apps

The following variables are available to container apps:

Expand table

Variable name	Description	Example value
CONTAINER_APP_NAME	The name of the container app.	my-containerapp
CONTAINER_APP_REVISION	The name of the container app revision.	my-containerapp--20mh1s9
CONTAINER_APP_HOSTNAME	The revision-specific hostname of the container app.	my-containerapp--20mh1s9. <DEFAULT_HOSTNAME>. <REGION>.azurecontainerapps.io
CONTAINER_APP_ENV_DNS_SUFFIX	The DNS suffix for the Container Apps environment. To obtain the fully qualified domain name (FQDN) of the app, append the app name to the DNS suffix in the format \$CONTAINER_APP_NAME.\$CONTAINER_APP_ENV_DNS_SUFFIX.	<DEFAULT_HOSTNAME>. <REGION>.azurecontainerapps.io
CONTAINER_APP_PORT	The target port of the container app.	8080

Variable name	Description	Example value
CONTAINER_APP_REPLICA_NAME	The name of the container app replica.	my-containerapp--20mh1s9-86c8c4b497-zx9bq

Jobs

The following variables are available to Container Apps jobs:

[Expand table](#)

Variable name	Description	Example value
CONTAINER_APP_JOB_NAME	The name of the job.	my-job
CONTAINER_APP_JOB_EXECUTION_NAME	The name of the job execution.	my-job-iwpi4il

Next steps

- [Revision management](#)

Feedback

Was this page helpful?

[!\[\]\(b192f47813074671dca41aa92c26b069_img.jpg\) Yes](#)

[!\[\]\(9bd704001d526f9130bb075b2acd795b_img.jpg\) No](#)

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

Health probes in Azure Container Apps

Article • 08/08/2024

Azure Container Apps health probes allow the Container Apps runtime to regularly inspect the status of your container apps.

You can set up probes using either TCP or HTTP(S) exclusively.

Container Apps supports the following probes:

[+] Expand table

Probe	Description
Startup	Checks if your application has successfully started. This check is separate from the liveness probe and executes during the initial startup phase of your application.
Liveness	Checks if your application is still running and responsive.
Readiness	Checks to see if a replica is ready to handle incoming requests.

For a full list of the probe specification supported in Azure Container Apps, refer to [Azure REST API specs ↗](#).

HTTP probes

HTTP probes allow you to implement custom logic to check the status of application dependencies before reporting a healthy status.

Configure your health probe endpoints to respond with an HTTP status code greater than or equal to `200` and less than `400` to indicate success. Any other response code outside this range indicates a failure.

The following example demonstrates how to implement a liveness endpoint in JavaScript.

JavaScript

```
const express = require('express');
const app = express();

app.get('/liveness', (req, res) => {
  let isSystemStable = false;

  // check for database availability
```

```
// check filesystem structure
// etc.

// set isSystemStable to true if all checks pass

if (isSystemStable) {
    res.status(200); // Success
} else {
    res.status(503); // Service unavailable
}
})
```

TCP probes

TCP probes wait to establish a connection with the server to indicate success. The probe fails if it can't establish a connection to your application.

Restrictions

- You can only add one of each probe type per container.
- `exec` probes aren't supported.
- Port values must be integers; named ports aren't supported.
- gRPC isn't supported.

Examples

The following code listing shows how you can define health probes for your containers.

The `...` placeholders denote omitted code. Refer to [Container Apps ARM template API specification](#) for full ARM template details.

ARM template

JSON

```
{
  ...
  "containers": [
    {
      "image": "nginx",
      "name": "web",
      "probes": [
        {
          "type": "liveness",
          "httpGet": {
            "port": 80
          }
        }
      ]
    }
  ]
}
```

```

    "path": "/health",
    "port": 8080,
    "httpHeaders": [
        {
            "name": "Custom-Header",
            "value": "liveness probe"
        }
    ],
    "initialDelaySeconds": 7,
    "periodSeconds": 3
},
{
    "type": "readiness",
    "tcpSocket": {
        "port": 8081
    },
    "initialDelaySeconds": 10,
    "periodSeconds": 3
},
{
    "type": "startup",
    "httpGet": {
        "path": "/startup",
        "port": 8080,
        "httpHeaders": [
            {
                "name": "Custom-Header",
                "value": "startup probe"
            }
        ],
        "initialDelaySeconds": 3,
        "periodSeconds": 3
    }
}
...
}

```

The optional `failureThreshold` setting defines the number of attempts Container Apps tries to execute the probe if execution fails. Attempts that exceed the `failureThreshold` amount cause different results for each probe type.

Default configuration

If ingress is enabled, the following default probes are automatically added to the main app container if none is defined for each type.

[\[\] Expand table](#)

Probe type	Default values
Startup	Protocol: TCP Port: ingress target port Timeout: 3 seconds Period: 1 second Initial delay: 1 second Success threshold: 1 Failure threshold: 240
Readiness	Protocol: TCP Port: ingress target port Timeout: 5 seconds Period: 5 seconds Initial delay: 3 seconds Success threshold: 1 Failure threshold: 48
Liveness	Protocol: TCP Port: ingress target port

If your app takes an extended amount of time to start (which is common in Java) you often need to customize the probes so your container doesn't crash.

The following example demonstrates how to configure the liveness and readiness probes in order to extend the startup times.

JSON

```

"probes": [
  {
    "type": "liveness",
    "failureThreshold": 3,
    "periodSeconds": 10,
    "successThreshold": 1,
    "tcpSocket": {
      "port": 80
    },
    "timeoutSeconds": 1
  },
  {
    "type": "readiness",
    "failureThreshold": 48,
    "initialDelaySeconds": 3,
    "periodSeconds": 5,
    "successThreshold": 1,
    "tcpSocket": {
      "port": 80
    },
  }
]

```

```
        "timeoutSeconds": 5  
    }]
```

Next steps

[Application logging](#)

Feedback

Was this page helpful?

 Yes

 No

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

Use storage mounts in Azure Container Apps

Article • 09/19/2024

A container app has access to different types of storage. A single app can take advantage of more than one type of storage if necessary.

[+] Expand table

Storage type	Description	Persistence	Usage example
Container-scoped storage	Ephemeral storage available to a running container	Data is available until container shuts down	Writing a local app cache.
Replica-scoped storage	Ephemeral storage for sharing files between containers in the same replica	Data is available until replica shuts down	The main app container writing log files that a sidecar container processes.
Azure Files	Permanent storage	Data is persisted to Azure Files	Writing files to a file share to make data accessible by other systems.

Ephemeral storage

A container app can read and write temporary data to ephemeral storage. Ephemeral storage can be scoped to a container or a replica. The total amount of container-scoped and replica-scoped storage available to each replica depends on the total number of vCPUs allocated to the replica.

[+] Expand table

vCPUs	Total ephemeral storage
0.25 or lower	1 GiB
0.5 or lower	2 GiB
1 or lower	4 GiB
Over 1	8 GiB

Container-scoped storage

A container can write to its own file system.

Container file system storage has the following characteristics:

- The storage is temporary and disappears when the container is shut down or restarted.
- Files written to this storage are only visible to processes running in the current container.

Replica-scoped storage

You can mount an ephemeral, temporary volume that is equivalent to [EmptyDir](#) (empty directory) in Kubernetes. This storage is scoped to a single replica. Use an [EmptyDir](#) volume to share data between containers in the same replica.

Replica-scoped storage has the following characteristics:

- Files are persisted for the lifetime of the replica.
 - If a container in a replica restarts, the files in the volume remain.
- Any init or app containers in the replica can mount the same volume.
- A container can mount multiple [EmptyDir](#) volumes.

To configure replica-scoped storage, first define an [EmptyDir](#) volume in the revision.

Then define a volume mount in one or more containers in the revision.

Prerequisites

[+] [Expand table](#)

Requirement	Instructions
Azure account	If you don't have one, create an account for free .
Azure Container Apps environment	Create a container apps environment .

Configuration

When configuring replica-scoped storage using the Azure CLI, you must use a YAML definition to create or update your container app.

1. To update an existing container app to use replica-scoped storage, export your app's specification to a YAML file named *app.yaml*.

```
azure-cli

az containerapp show -n <APP_NAME> -g <RESOURCE_GROUP_NAME> -o yaml >
app.yaml
```

2. Make the following changes to your container app specification.

- Add a `volumes` array to the `template` section of your container app definition and define a volume. If you already have a `volumes` array, add a new volume to the array.
 - The `name` is an identifier for the volume.
 - Use `EmptyDir` as the `storageType`.
- For each container in the template that you want to mount the volume, define a volume mount in the `volumeMounts` array of the container definition.
 - The `volumeName` is the name defined in the `volumes` array.
 - The `mountPath` is the path in the container to mount the volume.

```
YAML

properties:
  managedEnvironmentId:
    /subscriptions/<SUBSCRIPTION_ID>/resourceGroups/<RESOURCE_GROUP_NAME>/providers/Microsoft.App/managedEnvironments/<ENVIRONMENT_NAME>
  configuration:
    activeRevisionsMode: Single
  template:
    containers:
      - image: <IMAGE_NAME1>
        name: my-container-1
        volumeMounts:
          - mountPath: /myempty
            volumeName: myempty
      - image: <IMAGE_NAME_2>
        name: my-container-2
        volumeMounts:
          - mountPath: /myempty
            volumeName: myempty
    volumes:
      - name: myempty
        storageType: EmptyDir
```

3. Update your container app using the YAML file.

```
azure-cli
```

```
az containerapp update --name <APP_NAME> --resource-group  
<RESOURCE_GROUP_NAME> \  
--yaml app.yaml
```

See the [YAML specification](#) for a full example.

Azure Files volume

You can mount a file share from [Azure Files](#) as a volume in a container.

Azure Files storage has the following characteristics:

- Files written under the mount location are persisted to the file share.
- Files in the share are available via the mount location.
- Multiple containers can mount the same file share, including ones that are in another replica, revision, or container app.
- All containers that mount the share can access files written by any other container or method.
- More than one Azure Files volume can be mounted in a single container.

Azure Files supports both SMB (Server Message Block) and NFS (Network File System) protocols. You can mount an Azure Files share using either protocol. The file share you define in the environment must be configured with the same protocol used by the file share in the storage account.

Note

Support for mounting NFS shares in Azure Container Apps is in preview.

To enable Azure Files storage in your container, you need to set up your environment and container app as follows:

- Create a storage definition in the Container Apps environment.
- If you're using NFS, your environment must be configured with a custom VNet and the storage account must be configured to allow access from the VNet. For more information, see [NFS file shares in Azure Files](#).
- If your environment is configured with a custom VNet, you must allow ports 445 and 2049 in the network security group (NSG) associated with the subnet.
- Define a volume of type `AzureFile` (SMB) or `NfsAzureFile` (NFS) in a revision.
- Define a volume mount in one or more containers in the revision.

- The Azure Files storage account used must be accessible from your container app's virtual network. For more information, see [Grant access from a virtual network](#).
 - If you're using NFS, you must also disable secure transfer. For more information, see [NFS file shares in Azure Files](#) and the *Create an NFS Azure file share* section in [this tutorial](#).

Prerequisites

 Expand table

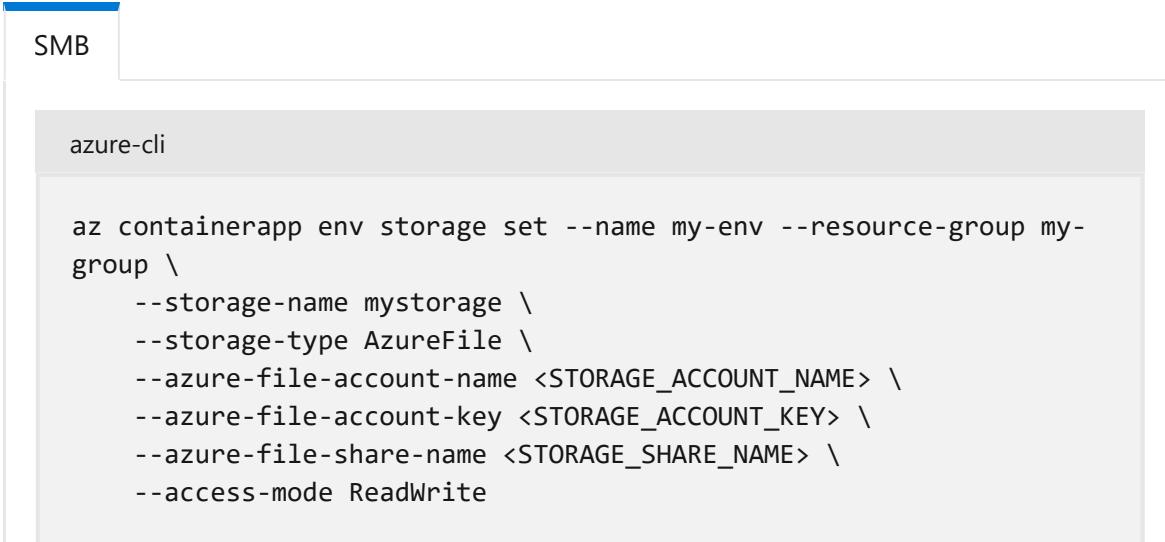
Requirement	Instructions
Azure account	If you don't have one, create an account for free .
Azure Storage account	Create a storage account .
Azure Container Apps environment	Create a container apps environment .

Configuration

When configuring a container app to mount an Azure Files volume using the Azure CLI, you must use a YAML definition to create or update your container app.

For a step-by-step tutorial on mounting an SMB file share, refer to [Create an Azure Files storage mount in Azure Container Apps](#).

- Add a storage definition to your Container Apps environment.



```

SMB

azurite
az containerapp env storage set --name my-env --resource-group my-group \
    --storage-name mystorage \
    --storage-type AzureFile \
    --azure-file-account-name <STORAGE_ACCOUNT_NAME> \
    --azure-file-account-key <STORAGE_ACCOUNT_KEY> \
    --azure-file-share-name <STORAGE_SHARE_NAME> \
    --access-mode ReadWrite
  
```

Replace `<STORAGE_ACCOUNT_NAME>` and `<STORAGE_ACCOUNT_KEY>` with the name and key of your storage account. Replace `<STORAGE_SHARE_NAME>` with the name of the file share in the storage account.

Valid values for `--access-mode` are `ReadWrite` and `ReadOnly`.

2. To update an existing container app to mount a file share, export your app's specification to a YAML file named `app.yaml`.

azure-cli

```
az containerapp show -n <APP_NAME> -g <RESOURCE_GROUP_NAME> -o yaml > app.yaml
```

3. Make the following changes to your container app specification.

- Add a `volumes` array to the `template` section of your container app definition and define a volume. If you already have a `volumes` array, add a new volume to the array.
 - The `name` is an identifier for the volume.
 - For `storageType`, use `AzureFile` for SMB, or `NfsAzureFile` for NFS. This value must match the storage type you defined in the environment.
 - For `storageName`, use the name of the storage you defined in the environment.
- For each container in the template that you want to mount Azure Files storage, define a volume mount in the `volumeMounts` array of the container definition.
 - The `volumeName` is the name defined in the `volumes` array.
 - The `mountPath` is the path in the container to mount the volume.

SMB

YAML

```
properties:  
  managedEnvironmentId:  
    /subscriptions/<SUBSCRIPTION_ID>/resourceGroups/<RESOURCE_GROUP_NAM  
E>/providers/Microsoft.App/managedEnvironments/<ENVIRONMENT_NAME>  
  configuration:  
  template:  
    containers:  
      - image: <IMAGE_NAME>  
        name: my-container  
        volumeMounts:  
          - volumeName: azure-files-volume  
            mountPath: /my-files  
    volumes:  
      - name: azure-files-volume
```

```
storageType: AzureFile  
storageName: mystorage
```

4. Update your container app using the YAML file.

```
azure-cli
```

```
az containerapp update --name <APP_NAME> --resource-group  
<RESOURCE_GROUP_NAME> \  
--yaml app.yaml
```

See the [YAML specification](#) for a full example.

Feedback

Was this page helpful?

 Yes

 No

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

Tutorial: Create an Azure Files volume mount in Azure Container Apps

Article • 04/15/2024

Learn to write to permanent storage in a container app using an Azure Files storage mount. For more information about storage mounts, see [Use storage mounts in Azure Container Apps](#).

In this tutorial, you learn how to:

- ✓ Create a Container Apps environment
- ✓ Create an Azure Storage account
- ✓ Define a file share in the storage account
- ✓ Link the environment to the storage file share
- ✓ Mount the storage share in an individual container
- ✓ Verify the storage mount by viewing the website access log

ⓘ Note

Azure Container Apps supports mounting file shares using SMB and NFS protocols. This tutorial demonstrates mounting an Azure Files share using the SMB protocol. To learn more about mounting NFS shares, see [Use storage mounts in Azure Container Apps](#).

Prerequisites

- Install the latest version of the [Azure CLI](#).

Set up the environment

The following commands help you define variables and ensure your Container Apps extension is up to date.

1. Sign in to the Azure CLI.



```
az login
```

2. Set up environment variables used in various commands to follow.

Bash

Azure CLI

```
RESOURCE_GROUP="my-container-apps-group"
ENVIRONMENT_NAME="my-storage-environment"
LOCATION="canadacentral"
```

3. Ensure you have the latest version of the Container Apps Azure CLI extension.

Bash

Azure CLI

```
az extension add -n containerapp --upgrade
```

4. Register the `Microsoft.App` namespace.

Bash

Azure CLI

```
az provider register --namespace Microsoft.App
```

5. Register the `Microsoft.OperationalInsights` provider for the Azure Monitor Log Analytics workspace if you haven't used it before.

Bash

Azure CLI

```
az provider register --namespace Microsoft.OperationalInsights
```

Create an environment

The following steps create a resource group and a Container Apps environment.

1. Create a resource group.

A screenshot of a terminal window titled "Bash". The "Azure CLI" tab is selected. Inside the terminal, the command `az group create \ --name $RESOURCE_GROUP \ --location $LOCATION \ --query "properties.provisioningState"` is displayed. The command uses backslashes to escape the space characters in the location and query parameters.

Once created, the command returns a "Succeeded" message.

At the end of this tutorial, you can delete the resource group to remove all the services created during this article.

2. Create a Container Apps environment.

A screenshot of a terminal window titled "Bash". The "Azure CLI" tab is selected. Inside the terminal, the command `az containerapp env create \ --name $ENVIRONMENT_NAME \ --resource-group $RESOURCE_GROUP \ --location \"$LOCATION\" \ --query "properties.provisioningState"` is displayed. The command uses backslashes to escape the space characters in the location and query parameters.

Once created, the command returns a "Succeeded" message.

Storage mounts are associated with a Container Apps environment and configured within individual container apps.

Set up a storage account

Next, create a storage account and establish a file share to mount to the container app.

1. Define a storage account name.

This command generates a random suffix to the storage account name to ensure uniqueness.

Bash

Azure CLI

```
STORAGE_ACCOUNT_NAME="myacastorageaccount$RANDOM"
```

2. Create an Azure Storage account.

Bash

Azure CLI

```
az storage account create \
--resource-group $RESOURCE_GROUP \
--name $STORAGE_ACCOUNT_NAME \
--location "$LOCATION" \
--kind StorageV2 \
--sku Standard_LRS \
--enable-large-file-share \
--query provisioningState
```

Once created, the command returns a "Succeeded" message.

3. Define a file share name.

Bash

Bash

```
STORAGE_SHARE_NAME="myfileshare"
```

4. Create the Azure Storage file share.

Bash

Azure CLI

```
az storage share-rm create \
--resource-group $RESOURCE_GROUP \
```

```
--storage-account $STORAGE_ACCOUNT_NAME \
--name $STORAGE_SHARE_NAME \
--quota 1024 \
--enabled-protocols SMB \
--output table
```

5. Get the storage account key.

Bash

Bash

```
STORAGE_ACCOUNT_KEY=`az storage account keys list -n
$STORAGE_ACCOUNT_NAME --query "[0].value" -o tsv`
```

The storage account key is required to create the storage link in your Container Apps environment.

6. Define the storage mount name.

Bash

Bash

```
STORAGE_MOUNT_NAME="mystoragemount"
```

This value is the name used to define the storage mount link from your Container Apps environment to your Azure Storage account.

Create the storage mount

Now you can update the container app configuration to support the storage mount.

1. Create the storage link in the environment.

Bash

Azure CLI

```
az containerapp env storage set \
--access-mode ReadWrite \
--azure-file-account-name $STORAGE_ACCOUNT_NAME \
```

```
--azure-file-account-key $STORAGE_ACCOUNT_KEY \
--azure-file-share-name $STORAGE_SHARE_NAME \
--storage-name $STORAGE_MOUNT_NAME \
--name $ENVIRONMENT_NAME \
--resource-group $RESOURCE_GROUP \
--output table
```

This command creates a link between container app environment and the file share created with the `az storage share rm` command.

Now that the storage account and environment are linked, you can create a container app that uses the storage mount.

2. Define the container app name.

The screenshot shows a terminal window with a blue header bar. The main area is titled "Bash". Below it, another "Bash" title is visible above a code block. The code block contains the command: `CONTAINER_APP_NAME="my-container-app"`.

3. Create the container app.

The screenshot shows a terminal window with a blue header bar. The main area is titled "Azure CLI". Below it, another "Azure CLI" title is visible above a code block. The code block contains the command: `az containerapp create \ --name $CONTAINER_APP_NAME \ --resource-group $RESOURCE_GROUP \ --environment $ENVIRONMENT_NAME \ --image nginx \ --min-replicas 1 \ --max-replicas 1 \ --target-port 80 \ --ingress external \ --query properties.configuration.ingress.fqdn`.

This command displays the URL of your new container app.

4. Copy the URL and paste into your web browser to navigate to the website.

Once the page loads, you'll see the "Welcome to nginx!" message. Keep this browser tab open. You'll return to the website during the storage mount

verification steps.

Now that you've confirmed the container app is configured, you can update the app to with a storage mount definition.

5. Export the container app's configuration.

```
Bash
```

```
Azure CLI
```

```
az containerapp show \
--name $CONTAINER_APP_NAME \
--resource-group $RESOURCE_GROUP \
--output yaml > app.yaml
```

⚠ Note

While this application doesn't have secrets, many apps do feature secrets. By default, when you export an app's configuration, the values for secrets aren't included in the generated YAML.

If you don't need to change secret values, then you can remove the `secrets` section and your secrets remain unaltered. Alternatively, if you need to change a secret's value, make sure to provide both the `name` and `value` for all secrets in the file before attempting to update the app. Omitting a secret from the `secrets` section deletes the secret.

6. Open *app.yaml* in a code editor.

7. Replace the `volumes: null` definition in the `template` section with a `volumes:` definition referencing the storage volume. The template section should look like the following:

```
yml
```

```
template:
  volumes:
    - name: my-azure-file-volume
      storageName: mystoragemount
      storageType: AzureFile
  containers:
    - image: nginx
      name: my-container-app
```

```

volumeMounts:
- volumeName: my-azure-file-volume
  mountPath: /var/log/nginx
resources:
  cpu: 0.5
  ephemeralStorage: 3Gi
  memory: 1Gi
initContainers: null
revisionSuffix: ''
scale:
  maxReplicas: 1
  minReplicas: 1
  rules: null

```

The new `template.volumes` section includes the following properties.

[] [Expand table](#)

Property	Description
<code>name</code>	This value matches the volume created by calling the <code>az containerapp env storage set</code> command.
<code>storageName</code>	This value defines the name used by containers in the environment to access the storage volume.
<code>storageType</code>	This value determines the type of storage volume defined for the environment. In this case, an Azure Files mount is declared.

The `volumes` section defines volumes at the app level that your application container or sidecar containers can reference via a `volumeMounts` section associated with a container.

8. Add a `volumeMounts` section to the `nginx` container in the `containers` section.

```

yml

containers:
- image: nginx
  name: my-container-app
  volumeMounts:
    - volumeName: my-azure-file-volume
      mountPath: /var/log/nginx

```

The new `volumeMounts` section includes the following properties:

[] [Expand table](#)

Property	Description
volumeName	This value must match the name defined in the <code>volumes</code> definition.
mountPath	This value defines the path in your container where the storage is mounted.

9. Update the container app with the new storage mount configuration.

Bash


```
Azure CLI

az containerapp update \
--name $CONTAINER_APP_NAME \
--resource-group $RESOURCE_GROUP \
--yaml app.yaml \
--output table
```

Verify the storage mount

Now that the storage mount is established, you can manipulate files in Azure Storage from your container. Use the following commands to observe the storage mount at work.

1. Open an interactive shell inside the container app to execute commands inside the running container.

Bash


```
Azure CLI

az containerapp exec \
--name $CONTAINER_APP_NAME \
--resource-group $RESOURCE_GROUP
```

This command may take a moment to open the remote shell. Once the shell is ready, you can interact with the storage mount via file system commands.

2. Change into the nginx `/var/log/nginx` folder.

Bash

```
Bash
```

```
cd /var/log/nginx
```

3. Return to the browser and navigate to the website and refresh the page a few times.

The requests made to the website create a series of log stream entries.

4. Return to your terminal and list the values of the `/var/log/nginx` folder.

```
Bash
```

```
Bash
```

```
ls
```

Note how the `access.log` and `error.log` files appear in this folder. These files are written to the Azure Files mount in your Azure Storage share created in the previous steps.

5. View the contents of the `access.log` file.

```
Bash
```

```
Bash
```

```
cat access.log
```

6. Exit out of the container's interactive shell to return to your local terminal session.

```
Bash
```

```
Bash
```

```
exit
```

7. Now, you can view the files in the Azure portal to verify they exist in your Azure Storage account. Print the name of your randomly generated storage account.

Bash

Bash

```
echo $STORAGE_ACCOUNT_NAME
```

8. Navigate to the Azure portal and open up the storage account created in this procedure.

9. Under Data Storage select File shares.

10. Select myshare to view the *access.log* and *error.log* files.

Clean up resources

If you're not going to continue to use this application, run the following command to delete the resource group along with all the resources created in this article.

Bash

Azure CLI

```
az group delete \  
--name $RESOURCE_GROUP
```

Next steps

[Connect container apps together](#)

Tutorial: Use code interpreter sessions in AutoGen with Azure Container Apps

Article • 05/21/2024

AutoGen [↗](#) is a framework for developing large language model (LLM) applications using multiple agents that converse with each other to solve tasks. Agents built with AutoGen can operate in various modes that employ combinations of LLMs, human inputs, and tools. One important type of tool for AutoGen agents is code executors. They enable agents to perform complex tasks by writing and executing code. By integrating Azure Container Apps dynamic sessions with AutoGen, you give the agent a [code interpreter](#) to use to perform useful computations and take actions.

In this tutorial, you learn how to run an AI agent authored in AutoGen in a web API. The API accepts user input and returns a response generated by the AI agent. The agent uses a code interpreter in dynamic sessions to perform calculations.

ⓘ Note

Azure Container Apps dynamic sessions is currently in preview. See [preview limitations](#) for more information.

Prerequisites

- An Azure account with an active subscription.
 - If you don't have one, you [can create one for free ↗](#).
- Install the [Azure CLI](#).
- Git.
- Python 3.10 or later.

Create Azure resources

The sample app in this quickstart uses an LLM from Azure OpenAI. It also uses Azure Container Apps sessions to run code generated by the LLM.

1. Update the Azure CLI to the latest version.

Bash

```
az upgrade
```

2. Remove the Azure Container Apps extension if it's already installed and install a preview version the Azure Container Apps extension containing commands for sessions:

Bash

```
az extension remove --name containerapp
az extension add \
    --name containerapp \
    --allow-preview true -y
```

3. Sign in to Azure:

Bash

```
az login
```

4. Set the variables used in this quickstart:

Bash

```
RESOURCE_GROUP_NAME=aca-sessions-tutorial
AZURE_OPENAI_LOCATION=swedencentral
AZURE_OPENAI_NAME=<UNIQUE_OPEN_AI_NAME>
SESSION_POOL_LOCATION=eastasia
SESSION_POOL_NAME=code-interpreter-pool
```

Replace `<UNIQUE_OPEN_AI_NAME>` with a unique name to create your Azure OpenAI account.

5. Create a resource group:

Bash

```
az group create --name $RESOURCE_GROUP_NAME --location
$SESSION_POOL_LOCATION
```

6. Create an Azure OpenAI account:

Bash

```
az cognitiveservices account create \
    --name $AZURE_OPENAI_NAME \
```

```
--resource-group $RESOURCE_GROUP_NAME \
--location $AZURE_OPENAI_LOCATION \
--kind OpenAI \
--sku s0 \
--custom-domain $AZURE_OPENAI_NAME
```

7. Create a GPT 3.5 Turbo model deployment named `gpt-35-turbo` in the Azure OpenAI account:

Bash

```
az cognitiveservices account deployment create \
--resource-group $RESOURCE_GROUP_NAME \
--name $AZURE_OPENAI_NAME \
--deployment-name gpt-35-turbo \
--model-name gpt-35-turbo \
--model-version "1106" \
--model-format OpenAI \
--sku-capacity "100" \
--sku-name "Standard"
```

8. Create a code interpreter session pool:

Bash

```
az containerapp sessionpool create \
--name $SESSION_POOL_NAME \
--resource-group $RESOURCE_GROUP_NAME \
--location $SESSION_POOL_LOCATION \
--max-sessions 100 \
--container-type PythonLTS \
--cooldown-period 300
```

Run the sample app locally

Before you deploy the app to Azure Container Apps, you can run it locally to test it.

Clone the app

1. Clone the [Azure Container Apps sessions samples repository](#).

Bash

```
git clone https://github.com/Azure-Samples/container-apps-dynamic-
sessions-samples.git
```

2. Change to the directory that contains the sample app:

```
Bash
```

```
cd container-apps-dynamic-sessions-samples/autogen-python-webapi
```

Configure the app

1. Create a Python virtual environment and activate it:

```
Bash
```

```
python3.11 -m venv .venv  
source .venv/bin/activate
```

Change the Python version in the command if you're using a different version. It's recommended to use Python 3.10 or later.

 **Note**

If you're using Windows, replace `.venv/bin/activate` with `.venv\Scripts\activate`.

2. Install the required Python packages:

```
Bash
```

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

3. To run the app, you need to configure environment variables.

a. Retrieve the Azure OpenAI account endpoint:

```
Bash
```

```
az cognitiveservices account show \  
  --name $AZURE_OPENAI_NAME \  
  --resource-group $RESOURCE_GROUP_NAME \  
  --query properties.endpoint \  
  --output tsv
```

b. Retrieve the Azure OpenAI API key:

Bash

```
az cognitiveservices account keys list \
--name $AZURE_OPENAI_NAME \
--resource-group $RESOURCE_GROUP_NAME \
--query key1 \
--output tsv
```

- c. Retrieve the Azure Container Apps session pool management endpoint:

Bash

```
az containerapp sessionpool show \
--name $SESSION_POOL_NAME \
--resource-group $RESOURCE_GROUP_NAME \
--query properties.poolManagementEndpoint \
--output tsv
```

- d. Create a `.env` file in the root of the sample app directory (same location as `main.py`). Add the following content to the file:

text

```
OAI_CONFIG_LIST=[{"model": "gpt-4", "api_key": "<AZURE_OPENAI_KEY>", \
"api_type": "azure", "base_url": "<AZURE_OPENAI_ENDPOINT>", \
"api_version": "2023-12-01-preview"}] \
POOL_MANAGEMENT_ENDPOINT=<SESSION_POOL_MANAGEMENT_ENDPOINT>
```

Replace `<AZURE_OPENAI_ENDPOINT>` with the Azure OpenAI account endpoint, `<AZURE_OPENAI_KEY>` with the Azure OpenAI API key, and `<SESSION_POOL_MANAGEMENT_ENDPOINT>` with the session pool management endpoint.

4. The app uses `DefaultAzureCredential` to authenticate with Azure services. On your local machine, it uses your current Azure CLI login credentials. You must give yourself the *Azure ContainerApps Session Executor* role on the session pool for the app to access the session pool.

- a. Retrieve your Azure CLI user name:

Bash

```
az account show --query user.name --output tsv
```

- b. Run the following commands to retrieve the session pool resource ID:

Bash

```
az containerapp sessionpool show --name $SESSION_POOL_NAME --  
resource-group $RESOURCE_GROUP_NAME --query id --output tsv
```

- c. Assign the *Azure ContainerApps Session Executor* role to your Azure CLI user on the session pool:

Bash

```
az role assignment create \  
--role "Azure ContainerApps Session Executor" \  
--assignee <CLI_USERNAME> \  
--scope <SESSION_POOL_RESOURCE_ID>
```

Replace `<CLI_USERNAME>` with your Azure CLI user name and `<SESSION_POOL_RESOURCE_ID>` with the session pool resource ID.

Run the app

Before running the sample app, open [main.py](#) in an editor and review the code. The app uses FastAPI to create a web API that accepts a user message in the query string.

The following lines of code instantiate a *ACASessionsExecutor* and provides it to the autogen agent:

Python

```
aca_sessions_executor = ACASessionsExecutor(aca_pool_management_endpoint)  
code_executor_agent = ConversableAgent(  
    name="CodeExecutor",  
    llm_config=False,  
    code_execution_config={"executor": aca_sessions_executor},  
    human_input_mode="NEVER",  
    is_termination_msg=lambda msg: "TERMINATE" in msg.get("content",  
    "").strip().upper()  
)
```

When it needs to perform calculations and tasks, the agent uses the code interpreter in *ACASessionsExecutor* to run the code. The code is executed in a session in the session pool. By default, a random session identifier is generated when you instantiate the tool. If the agent uses the same tool to run multiple Python code snippets, it uses the same session. To ensure each end user has a unique session, use a separate agent and tool for each user.

ACASessionsExecutor is implemented in [aca_sessions_executor.py](#).

1. Run the sample app:

```
Bash  
fastapi dev main.py
```

2. Open a browser and navigate to <http://localhost:8000/docs>. You see the Swagger UI for the sample app.

3. Expand the `/chat` endpoint and select **Try it out**.

4. Enter `What time is it right now?` in the `message` field and select **Execute**.

The agent responds with the current time. In the terminal, you see the logs showing the agent generated Python code to get the current time and ran it in a code interpreter session.

5. To stop the app, enter `Ctrl+C` in the terminal.

Optional: Deploy the sample app to Azure Container Apps

To deploy the FastAPI app to Azure Container Apps, you need to create a container image and push it to a container registry. Then you can deploy the image to Azure Container Apps. The `az containerapp up` command combines these steps into a single command.

You then need to configure managed identity for the app and assign it the proper roles to access Azure OpenAI and the session pool.

1. Set the variables for the Container Apps environment and app name:

```
Bash  
ENVIRONMENT_NAME=aca-sessions-tutorial-env  
CONTAINER_APP_NAME=chat-api
```

2. Build and deploy the app to Azure Container Apps:

```
Bash
```

```
az containerapp up \
    --name $CONTAINER_APP_NAME \
    --resource-group $RESOURCE_GROUP_NAME \
    --location $SESSION_POOL_LOCATION \
    --environment $ENVIRONMENT_NAME \
    --env-vars 'OAI_CONFIG_LIST=[{"model": "gpt-4", "api_key": "  
<AZURE_OPENAI_KEY>", "api_type": "azure", "base_url": "  
<AZURE_OPENAI_ENDPOINT>", "api_version": "2023-12-01-preview"}]' \
    'POOL_MANAGEMENT_ENDPOINT=<SESSION_POOL_MANAGMENT_ENDPOINT>' \
    --source .
```

Replace `<AZURE_OPENAI_ENDPOINT>` with the Azure OpenAI account endpoint, `<AZURE_OPENAI_KEY>` with the Azure OpenAI key, and `<SESSION_POOL_MANAGMENT_ENDPOINT>` with the session pool management endpoint.

3. Enable the system-assigned managed identity for the app:

Bash

```
az containerapp identity assign \
    --name $CONTAINER_APP_NAME \
    --resource-group $RESOURCE_GROUP_NAME \
    --system-assigned
```

4. For the app to access the session pool, you need to assign the managed identity the proper roles.

a. Retrieve the managed identity's principal ID:

Bash

```
az containerapp show \
    --name $CONTAINER_APP_NAME \
    --resource-group $RESOURCE_GROUP_NAME \
    --query identity.principalId \
    --output tsv
```

b. Retrieve the session pool resource ID:

Bash

```
az containerapp sessionpool show \
    --name $SESSION_POOL_NAME \
    --resource-group $RESOURCE_GROUP_NAME \
    --query id \
    --output tsv
```

- c. Assign the managed identity the `Azure ContainerApps Session Executor` and `Contributor` roles on the session pool:

Before you run the following command, replace `<PRINCIPAL_ID>` and `<SESSION_POOL_RESOURCE_ID>` with the values you retrieved in the previous steps.

Bash

```
az role assignment create \
    --role "Azure ContainerApps Session Executor" \
    --assignee <PRINCIPAL_ID> \
    --scope <SESSION_POOL_RESOURCE_ID>

az role assignment create \
    --role "Contributor" \
    --assignee <PRINCIPAL_ID> \
    --scope <SESSION_POOL_RESOURCE_ID>
```

5. Retrieve the app's fully qualified domain name (FQDN):

Bash

```
az containerapp show \
    --name $CONTAINER_APP_NAME \
    --resource-group $RESOURCE_GROUP_NAME \
    --query properties.configuration.ingress.fqdn \
    --output tsv
```

6. Open the browser to `https://<FQDN>/docs` to test the deployed app.

Clean up resources

When you're done with the resources, you can delete them to avoid incurring charges:

Bash

```
az group delete --name $RESOURCE_GROUP_NAME --yes --no-wait
```

Next steps

[Code interpreter sessions](#)

Tutorial: Use code interpreter sessions in LangChain with Azure Container Apps

Article • 05/21/2024

[LangChain](#) is a framework designed to simplify the creation of applications using large language models (LLMs). When you build an AI agent with LangChain, an LLM interprets user input and generates a response. The AI agent often struggles when it needs to perform mathematical and symbolic reasoning to produce a response. By integrating Azure Container Apps dynamic sessions with LangChain, you give the agent a [code interpreter](#) to use to perform specialized tasks.

In this tutorial, you learn how to run a LangChain AI agent in a web API. The API accepts user input and returns a response generated by the AI agent. The agent uses a code interpreter in dynamic sessions to perform calculations.

ⓘ Note

Azure Container Apps dynamic sessions is currently in preview. See [preview limitations](#) for more information.

Prerequisites

- An Azure account with an active subscription.
 - If you don't have one, you [can create one for free](#).
- Install the [Azure CLI](#).
- Git.
- Python 3.10 or later.

Create Azure resources

The sample app in this quickstart uses an LLM from Azure OpenAI. It also uses Azure Container Apps sessions to run code generated by the LLM.

1. Update the Azure CLI to the latest version.

```
Bash
```

```
az upgrade
```

2. Remove the Azure Container Apps extension if it's already installed and install a preview version the Azure Container Apps extension containing commands for sessions:

```
Bash

az extension remove --name containerapp
az extension add \
    --name containerapp \
    --allow-preview true -y
```

3. Sign in to Azure:

```
Bash

az login
```

4. Set the variables used in this quickstart:

```
Bash

RESOURCE_GROUP_NAME=aca-sessions-tutorial
AZURE_OPENAI_LOCATION=swedencentral
AZURE_OPENAI_NAME=<UNIQUE_OPEN_AI_NAME>
SESSION_POOL_LOCATION=eastasia
SESSION_POOL_NAME=code-interpreter-pool
```

Replace `<UNIQUE_OPEN_AI_NAME>` with a unique name to create your Azure OpenAI account.

5. Create a resource group:

```
Bash

az group create --name $RESOURCE_GROUP_NAME --location
$SESSION_POOL_LOCATION
```

6. Create an Azure OpenAI account:

```
Bash

az cognitiveservices account create \
    --name $AZURE_OPENAI_NAME \
    --resource-group $RESOURCE_GROUP_NAME \
    --location $AZURE_OPENAI_LOCATION \
    --kind OpenAI \
```

```
--sku s0 \
--custom-domain $AZURE_OPENAI_NAME
```

7. Create a GPT 3.5 Turbo model deployment named `gpt-35-turbo` in the Azure OpenAI account:

Bash

```
az cognitiveservices account deployment create \
--resource-group $RESOURCE_GROUP_NAME \
--name $AZURE_OPENAI_NAME \
--deployment-name gpt-35-turbo \
--model-name gpt-35-turbo \
--model-version "1106" \
--model-format OpenAI \
--sku-capacity "100" \
--sku-name "Standard"
```

8. Create a code interpreter session pool:

Bash

```
az containerapp sessionpool create \
--name $SESSION_POOL_NAME \
--resource-group $RESOURCE_GROUP_NAME \
--location $SESSION_POOL_LOCATION \
--max-sessions 100 \
--container-type PythonLTS \
--cooldown-period 300
```

Run the sample app locally

Before you deploy the app to Azure Container Apps, you can run it locally to test it.

Clone the app

1. Clone the [Azure Container Apps sessions samples repository](#).

Bash

```
git clone https://github.com/Azure-Samples/container-apps-dynamic-
sessions-samples.git
```

2. Change to the directory that contains the sample app:

Bash

```
cd container-apps-dynamic-sessions-samples/langchain-python-webapi
```

Configure the app

1. Create a Python virtual environment and activate it:

Bash

```
python3.11 -m venv .venv  
source .venv/bin/activate
```

Change the Python version in the command if you're using a different version. It's recommended to use Python 3.10 or later.

 **Note**

If you're using Windows, replace `.venv/bin/activate` with `.venv\Scripts\activate`.

2. Install the required Python packages:

Bash

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

3. To run the app, you need to configure environment variables.

- a. Retrieve the Azure OpenAI account endpoint:

Bash

```
az cognitiveservices account show \  
--name $AZURE_OPENAI_NAME \  
--resource-group $RESOURCE_GROUP_NAME \  
--query properties.endpoint \  
--output tsv
```

- b. Retrieve the Azure Container Apps session pool management endpoint:

Bash

```
az containerapp sessionpool show \
    --name $SESSION_POOL_NAME \
    --resource-group $RESOURCE_GROUP_NAME \
    --query properties.poolManagementEndpoint \
    --output tsv
```

- c. Create a `.env` file in the root of the sample app directory (same location as `main.py`). Add the following content to the file:

text

```
AZURE_OPENAI_ENDPOINT=<AZURE_OPENAI_ENDPOINT>
POOL_MANAGEMENT_ENDPOINT=<SESSION_POOL_MANAGEMENT_ENDPOINT>
```

Replace `<AZURE_OPENAI_ENDPOINT>` with the Azure OpenAI account endpoint and `<SESSION_POOL_MANAGEMENT_ENDPOINT>` with the session pool management endpoint.

4. The app uses `DefaultAzureCredential` to authenticate with Azure services. On your local machine, it uses your current Azure CLI login credentials. You must give yourself the *Cognitive Services OpenAI User* role on the Azure OpenAI account for the app to access the model endpoints, and the *Azure ContainerApps Session Executor* role on the session pool for the app to access the session pool.

- a. Retrieve your Azure CLI user name:

Bash

```
az account show --query user.name --output tsv
```

- b. Run the following commands to retrieve the Azure OpenAI account resource ID:

Bash

```
az cognitiveservices account show --name $AZURE_OPENAI_NAME --
    resource-group $RESOURCE_GROUP_NAME --query id --output tsv
```

- c. Assign the *Cognitive Services OpenAI User* role to your Azure CLI user on the Azure OpenAI account:

Bash

```
az role assignment create --role "Cognitive Services OpenAI User" --
```

```
assignee <CLI_USERNAME> --scope <AZURE_OPENAI_RESOURCE_ID>
```

Replace `<CLI_USERNAME>` with your Azure CLI user name and `<AZURE_OPENAI_RESOURCE_ID>` with the Azure OpenAI account resource ID.

- d. Run the following commands to retrieve the session pool resource ID:

Bash

```
az containerapp sessionpool show --name $SESSION_POOL_NAME --resource-group $RESOURCE_GROUP_NAME --query id --output tsv
```

- e. Assign the *Azure ContainerApps Session Executor* role using its ID to your Azure CLI user on the session pool:

Bash

```
az role assignment create \
    --role "Azure ContainerApps Session Executor" \
    --assignee <CLI_USERNAME> \
    --scope <SESSION_POOL_RESOURCE_ID>
```

Replace `<CLI_USERNAME>` with your Azure CLI user name and `<SESSION_POOL_RESOURCE_ID>` with the session pool resource ID.

Run the app

Before running the sample app, open [main.py](#) in an editor and review the code. The app uses FastAPI to create a web API that accepts a user message in the query string.

The following lines of code instantiate a *SessionPythonREPLTool* and provide it to the LangChain agent:

Python

```
repl =
SessionsPythonREPLTool(pool_management_endpoint=pool_management_endpoint)

tools = [repl]
prompt = hub.pull("hwchase17/openai-functions-agent")
agent = agents.create_tool_calling_agent(l1m, tools, prompt)
```

When it needs to perform calculations, the agent uses the *SessionPythonREPLTool* to run the code. The code is executed in a session in the session pool. By default, a random

session identifier is generated when you instantiate the tool. If the agent uses the tool to run multiple Python code snippets, it uses the same session. To ensure each end user has a unique session, use a separate agent and tool for each user.

SessionPythonREPLTool is available in the [langchain-azure-dynamic-sessions](#) package.

1. Run the sample app:

```
Bash
```

```
fastapi dev main.py
```

2. Open a browser and navigate to `http://localhost:8000/docs`. You see the Swagger UI for the sample app.

3. Expand the `/chat` endpoint and select **Try it out**.

4. Enter `What time is it right now?` in the `message` field and select **Execute**.

The agent responds with the current time. In the terminal, you see the logs showing the agent generated Python code to get the current time and ran it in a code interpreter session.

5. To stop the app, enter `Ctrl+C` in the terminal.

Optional: Deploy the sample app to Azure Container Apps

To deploy the FastAPI app to Azure Container Apps, you need to create a container image and push it to a container registry. Then you can deploy the image to Azure Container Apps. The `az containerapp up` command combines these steps into a single command.

You then need to configure managed identity for the app and assign it the proper roles to access Azure OpenAI and the session pool.

1. Set the variables for the Container Apps environment and app name:

```
Bash
```

```
ENVIRONMENT_NAME=aca-sessions-tutorial-env  
CONTAINER_APP_NAME=chat-api
```

2. Build and deploy the app to Azure Container Apps:

Bash

```
az containerapp up \
  --name $CONTAINER_APP_NAME \
  --resource-group $RESOURCE_GROUP_NAME \
  --location $SESSION_POOL_LOCATION \
  --environment $ENVIRONMENT_NAME \
  --env-vars "AZURE_OPENAI_ENDPOINT=<OPEN_AI_ENDPOINT>" \
"POOL_MANAGEMENT_ENDPOINT=<SESSION_POOL_MANAGMENT_ENDPOINT>" \
  --source .
```

Replace `<OPEN_AI_ENDPOINT>` with the Azure OpenAI account endpoint and `<SESSION_POOL_MANAGMENT_ENDPOINT>` with the session pool management endpoint.

3. Enable the system-assigned managed identity for the app:

Bash

```
az containerapp identity assign \
  --name $CONTAINER_APP_NAME \
  --resource-group $RESOURCE_GROUP_NAME \
  --system-assigned
```

4. For the app to access Azure OpenAI and the session pool, you need to assign the managed identity the proper roles.

a. Retrieve the managed identity's principal ID:

Bash

```
az containerapp show \
  --name $CONTAINER_APP_NAME \
  --resource-group $RESOURCE_GROUP_NAME \
  --query identity.principalId \
  --output tsv
```

b. Retrieve the session pool resource ID:

Bash

```
az containerapp sessionpool show \
  --name $SESSION_POOL_NAME \
  --resource-group $RESOURCE_GROUP_NAME \
  --query id \
  --output tsv
```

- c. Assign the managed identity the `Azure ContainerApps Session Executor` and `Contributor` roles on the session pool:

Before you run the following command, replace `<PRINCIPAL_ID>` and `<SESSION_POOL_RESOURCE_ID>` with the values you retrieved in the previous steps.

Bash

```
az role assignment create \
    --role "Azure ContainerApps Session Executor" \
    --assignee <PRINCIPAL_ID> \
    --scope <SESSION_POOL_RESOURCE_ID>

az role assignment create \
    --role "Contributor" \
    --assignee <PRINCIPAL_ID> \
    --scope <SESSION_POOL_RESOURCE_ID>
```

- d. Retrieve the Azure OpenAI account resource ID:

Bash

```
az cognitiveservices account show \
    --name $AZURE_OPENAI_NAME \
    --resource-group $RESOURCE_GROUP_NAME \
    --query id \
    --output tsv
```

- e. Assign the managed identity the `Cognitive Services OpenAI User` role on the Azure OpenAI account:

Before you run the following command, replace `<PRINCIPAL_ID>` and `<AZURE_OPENAI_RESOURCE_ID>` with the values you retrieved in the previous steps.

Bash

```
az role assignment create \
    --role "Cognitive Services OpenAI User" \
    --assignee <PRINCIPAL_ID> \
    --scope <AZURE_OPENAI_RESOURCE_ID>
```

5. Retrieve the app's fully qualified domain name (FQDN):

Bash

```
az containerapp show \
    --name $CONTAINER_APP_NAME \
```

```
--resource-group $RESOURCE_GROUP_NAME \
--query properties.configuration.ingress.fqdn \
--output tsv
```

6. Open the browser to `https://<FQDN>/docs` to test the deployed app.

Clean up resources

When you're done with the resources, you can delete them to avoid incurring charges:

Bash

```
az group delete --name $RESOURCE_GROUP_NAME --yes --no-wait
```

Next steps

[Code interpreter sessions](#)

Tutorial: Use code interpreter sessions in LlamalIndex with Azure Container Apps

Article • 05/21/2024

LlamalIndex [↗](#) is a powerful framework for building context-augmented language model (LLM) applications. When you build an AI agent with LlamalIndex, an LLM interprets user input and generates a response. The AI agent often struggles when it needs to perform mathematical and symbolic reasoning to produce a response. By integrating Azure Container Apps dynamic sessions with LlamalIndex, you give the agent a [code interpreter](#) to use to perform specialized tasks.

In this tutorial, you learn how to run a LlamalIndex AI agent in a web API. The API accepts user input and returns a response generated by the AI agent. The agent uses a code interpreter in dynamic sessions to perform calculations.

ⓘ Note

Azure Container Apps dynamic sessions is currently in preview. See [preview limitations](#) for more information.

Prerequisites

- An Azure account with an active subscription.
 - If you don't have one, you [can create one for free ↗](#).
- Install the [Azure CLI](#).
- Git.
- Python 3.10 or later.

Create Azure resources

The sample app in this quickstart uses an LLM from Azure OpenAI. It also uses Azure Container Apps sessions to run code generated by the LLM.

1. Update the Azure CLI to the latest version.

```
Bash
```

```
az upgrade
```

2. Remove the Azure Container Apps extension if it's already installed and install a preview version the Azure Container Apps extension containing commands for sessions:

```
Bash

az extension remove --name containerapp
az extension add \
    --name containerapp \
    --allow-preview true -y
```

3. Sign in to Azure:

```
Bash

az login
```

4. Set the variables used in this quickstart:

```
Bash

RESOURCE_GROUP_NAME=aca-sessions-tutorial
AZURE_OPENAI_LOCATION=swedencentral
AZURE_OPENAI_NAME=<UNIQUE_OPEN_AI_NAME>
SESSION_POOL_LOCATION=eastasia
SESSION_POOL_NAME=code-interpreter-pool
```

Replace `<UNIQUE_OPEN_AI_NAME>` with a unique name to create your Azure OpenAI account.

5. Create a resource group:

```
Bash

az group create --name $RESOURCE_GROUP_NAME --location
$SESSION_POOL_LOCATION
```

6. Create an Azure OpenAI account:

```
Bash

az cognitiveservices account create \
    --name $AZURE_OPENAI_NAME \
    --resource-group $RESOURCE_GROUP_NAME \
    --location $AZURE_OPENAI_LOCATION \
    --kind OpenAI \
```

```
--sku s0 \
--custom-domain $AZURE_OPENAI_NAME
```

7. Create a GPT 3.5 Turbo model deployment named `gpt-35-turbo` in the Azure OpenAI account:

Bash

```
az cognitiveservices account deployment create \
--resource-group $RESOURCE_GROUP_NAME \
--name $AZURE_OPENAI_NAME \
--deployment-name gpt-35-turbo \
--model-name gpt-35-turbo \
--model-version "1106" \
--model-format OpenAI \
--sku-capacity "100" \
--sku-name "Standard"
```

8. Create a code interpreter session pool:

Bash

```
az containerapp sessionpool create \
--name $SESSION_POOL_NAME \
--resource-group $RESOURCE_GROUP_NAME \
--location $SESSION_POOL_LOCATION \
--max-sessions 100 \
--container-type PythonLTS \
--cooldown-period 300
```

Run the sample app locally

Before you deploy the app to Azure Container Apps, you can run it locally to test it.

Clone the app

1. Clone the [Azure Container Apps sessions samples repository](#).

Bash

```
git clone https://github.com/Azure-Samples/container-apps-dynamic-
sessions-samples.git
```

2. Change to the directory that contains the sample app:

Bash

```
cd container-apps-dynamic-sessions-samples/llamaindex-python-webapi
```

Configure the app

1. Create a Python virtual environment and activate it:

Bash

```
python3.11 -m venv .venv  
source .venv/bin/activate
```

Change the Python version in the command if you're using a different version. It's recommended to use Python 3.10 or later.

 **Note**

If you're using Windows, replace `.venv/bin/activate` with `.venv\Scripts\activate`.

2. Install the required Python packages:

Bash

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

3. To run the app, you need to configure environment variables.

- a. Retrieve the Azure OpenAI account endpoint:

Bash

```
az cognitiveservices account show \  
--name $AZURE_OPENAI_NAME \  
--resource-group $RESOURCE_GROUP_NAME \  
--query properties.endpoint \  
--output tsv
```

- b. Retrieve the Azure Container Apps session pool management endpoint:

Bash

```
az containerapp sessionpool show \
    --name $SESSION_POOL_NAME \
    --resource-group $RESOURCE_GROUP_NAME \
    --query properties.poolManagementEndpoint \
    --output tsv
```

- c. Create a `.env` file in the root of the sample app directory (same location as `main.py`). Add the following content to the file:

text

```
AZURE_OPENAI_ENDPOINT=<AZURE_OPENAI_ENDPOINT>
POOL_MANAGEMENT_ENDPOINT=<SESSION_POOL_MANAGEMENT_ENDPOINT>
```

Replace `<AZURE_OPENAI_ENDPOINT>` with the Azure OpenAI account endpoint and `<SESSION_POOL_MANAGEMENT_ENDPOINT>` with the session pool management endpoint.

4. The app uses `DefaultAzureCredential` to authenticate with Azure services. On your local machine, it uses your current Azure CLI login credentials. You must give yourself the *Cognitive Services OpenAI User* role on the Azure OpenAI account for the app to access the model endpoints, and the *Azure ContainerApps Session Executor* role on the session pool for the app to access the session pool.

- a. Retrieve your Azure CLI user name:

Bash

```
az account show --query user.name --output tsv
```

- b. Run the following commands to retrieve the Azure OpenAI account resource ID:

Bash

```
az cognitiveservices account show --name $AZURE_OPENAI_NAME --
    resource-group $RESOURCE_GROUP_NAME --query id --output tsv
```

- c. Assign the *Cognitive Services OpenAI User* role to your Azure CLI user on the Azure OpenAI account:

Bash

```
az role assignment create --role "Cognitive Services OpenAI User" --
```

```
assignee <CLI_USERNAME> --scope <AZURE_OPENAI_RESOURCE_ID>
```

Replace `<CLI_USERNAME>` with your Azure CLI user name and `<AZURE_OPENAI_RESOURCE_ID>` with the Azure OpenAI account resource ID.

- d. Run the following commands to retrieve the session pool resource ID:

Bash

```
az containerapp sessionpool show --name $SESSION_POOL_NAME --resource-group $RESOURCE_GROUP_NAME --query id --output tsv
```

- e. Assign the *Azure ContainerApps Session Executor* role using its ID to your Azure CLI user on the session pool:

Bash

```
az role assignment create \
    --role "Azure ContainerApps Session Executor" \
    --assignee <CLI_USERNAME> \
    --scope <SESSION_POOL_RESOURCE_ID>
```

Replace `<CLI_USERNAME>` with your Azure CLI user name and `<SESSION_POOL_RESOURCE_ID>` with the session pool resource ID.

Run the app

Before running the sample app, open [main.py](#) in an editor and review the code. The app uses FastAPI to create a web API that accepts a user message in the query string.

The following lines of code instantiate a *AzureCodeInterpreterToolSpec* and provide it to the LlamalIndex agent:

Python

```
code_interpreter_tool = AzureCodeInterpreterToolSpec(
    pool_management_endpoint=pool_management_endpoint,
)
agent = ReActAgent.from_tools(code_interpreter_tool.to_tool_list(), llm=llm,
verbose=True)
```

When it needs to perform calculations, the agent uses the code interpreter in *AzureCodeInterpreterToolSpec* to run the code. The code is executed in a session in the session pool. By default, a random session identifier is generated when you instantiate

the tool. If the agent uses the same tool to run multiple Python code snippets, it uses the same session. To ensure each end user has a unique session, use a separate agent and tool for each user.

`AzureCodeInterpreterToolSpec` is available in the [llama-index-tools-azure-code-interpreter](#) package.

1. Run the sample app:

```
Bash  
fastapi dev main.py
```

2. Open a browser and navigate to `http://localhost:8000/docs`. You see the Swagger UI for the sample app.
3. Expand the `/chat` endpoint and select **Try it out**.
4. Enter `What time is it right now?` in the `message` field and select **Execute**.
The agent responds with the current time. In the terminal, you see the logs showing the agent generated Python code to get the current time and ran it in a code interpreter session.
5. To stop the app, enter `Ctrl+C` in the terminal.

Optional: Deploy the sample app to Azure Container Apps

To deploy the FastAPI app to Azure Container Apps, you need to create a container image and push it to a container registry. Then you can deploy the image to Azure Container Apps. The `az containerapp up` command combines these steps into a single command.

You then need to configure managed identity for the app and assign it the proper roles to access Azure OpenAI and the session pool.

1. Set the variables for the Container Apps environment and app name:

```
Bash  
ENVIRONMENT_NAME=aca-sessions-tutorial-env  
CONTAINER_APP_NAME=chat-api
```

2. Build and deploy the app to Azure Container Apps:

Bash

```
az containerapp up \
  --name $CONTAINER_APP_NAME \
  --resource-group $RESOURCE_GROUP_NAME \
  --location $SESSION_POOL_LOCATION \
  --environment $ENVIRONMENT_NAME \
  --env-vars "AZURE_OPENAI_ENDPOINT=<OPEN_AI_ENDPOINT>" \
"POOL_MANAGEMENT_ENDPOINT=<SESSION_POOL_MANAGMENT_ENDPOINT>" \
  --source .
```

Replace `<OPEN_AI_ENDPOINT>` with the Azure OpenAI account endpoint and `<SESSION_POOL_MANAGMENT_ENDPOINT>` with the session pool management endpoint.

3. Enable the system-assigned managed identity for the app:

Bash

```
az containerapp identity assign \
  --name $CONTAINER_APP_NAME \
  --resource-group $RESOURCE_GROUP_NAME \
  --system-assigned
```

4. For the app to access Azure OpenAI and the session pool, you need to assign the managed identity the proper roles.

a. Retrieve the managed identity's principal ID:

Bash

```
az containerapp show \
  --name $CONTAINER_APP_NAME \
  --resource-group $RESOURCE_GROUP_NAME \
  --query identity.principalId \
  --output tsv
```

b. Retrieve the session pool resource ID:

Bash

```
az containerapp sessionpool show \
  --name $SESSION_POOL_NAME \
  --resource-group $RESOURCE_GROUP_NAME \
  --query id \
  --output tsv
```

c. Assign the managed identity the `Azure ContainerApps Session Executor` and `Contributor` roles on the session pool:

Before you run the following command, replace `<PRINCIPAL_ID>` and `<SESSION_POOL_RESOURCE_ID>` with the values you retrieved in the previous steps.

Bash

```
az role assignment create \
    --role "Azure ContainerApps Session Executor" \
    --assignee <PRINCIPAL_ID> \
    --scope <SESSION_POOL_RESOURCE_ID>

az role assignment create \
    --role "Contributor" \
    --assignee <PRINCIPAL_ID> \
    --scope <SESSION_POOL_RESOURCE_ID>
```

d. Retrieve the Azure OpenAI account resource ID:

Bash

```
az cognitiveservices account show \
    --name $AZURE_OPENAI_NAME \
    --resource-group $RESOURCE_GROUP_NAME \
    --query id \
    --output tsv
```

e. Assign the managed identity the `Cognitive Services OpenAI User` role on the Azure OpenAI account:

Before you run the following command, replace `<PRINCIPAL_ID>` and `<AZURE_OPENAI_RESOURCE_ID>` with the values you retrieved in the previous steps.

Bash

```
az role assignment create \
    --role "Cognitive Services OpenAI User" \
    --assignee <PRINCIPAL_ID> \
    --scope <AZURE_OPENAI_RESOURCE_ID>
```

5. Retrieve the app's fully qualified domain name (FQDN):

Bash

```
az containerapp show \
    --name $CONTAINER_APP_NAME \
```

```
--resource-group $RESOURCE_GROUP_NAME \
--query properties.configuration.ingress.fqdn \
--output tsv
```

6. Open the browser to `https://<FQDN>/docs` to test the deployed app.

Clean up resources

When you're done with the resources, you can delete them to avoid incurring charges:

Bash

```
az group delete --name $RESOURCE_GROUP_NAME --yes --no-wait
```

Next steps

[Code interpreter sessions](#)

Tutorial: Use code interpreter sessions in Semantic Kernel with Azure Container Apps

Article • 05/21/2024

Semantic Kernel is an open-source AI framework created by Microsoft for .NET, Python, and Java developers working with Large Language Models (LLMs). When you build an AI agent with Semantic Kernel, an LLM interprets user input and generates a response. The AI agent often struggles when it needs to perform mathematical and symbolic reasoning to produce a response. By integrating Azure Container Apps dynamic sessions with Semantic Kernel, you give the agent a [code interpreter](#) to use to perform specialized tasks.

In this tutorial, you learn how to run a Semantic Kernel AI agent in a web API. The API accepts user input and returns a response generated by the AI agent. The agent uses a code interpreter in dynamic sessions to perform calculations.

ⓘ Note

Azure Container Apps dynamic sessions is currently in preview. See [preview limitations](#) for more information.

Prerequisites

- An Azure account with an active subscription.
 - If you don't have one, you [can create one for free ↗](#).
- Install the [Azure CLI](#).
- Git.
- Python 3.10 or later.

Create Azure resources

The sample app in this quickstart uses an LLM from Azure OpenAI. It also uses Azure Container Apps sessions to run code generated by the LLM.

1. Update the Azure CLI to the latest version.

Bash

```
az upgrade
```

2. Remove the Azure Container Apps extension if it's already installed and install a preview version the Azure Container Apps extension containing commands for sessions:

Bash

```
az extension remove --name containerapp
az extension add \
    --name containerapp \
    --allow-preview true -y
```

3. Sign in to Azure:

Bash

```
az login
```

4. Set the variables used in this quickstart:

Bash

```
RESOURCE_GROUP_NAME=aca-sessions-tutorial
AZURE_OPENAI_LOCATION=swedencentral
AZURE_OPENAI_NAME=<UNIQUE_OPEN_AI_NAME>
SESSION_POOL_LOCATION=eastasia
SESSION_POOL_NAME=code-interpreter-pool
```

Replace `<UNIQUE_OPEN_AI_NAME>` with a unique name to create your Azure OpenAI account.

5. Create a resource group:

Bash

```
az group create --name $RESOURCE_GROUP_NAME --location
$SESSION_POOL_LOCATION
```

6. Create an Azure OpenAI account:

Bash

```
az cognitiveservices account create \
    --name $AZURE_OPENAI_NAME \
```

```
--resource-group $RESOURCE_GROUP_NAME \
--location $AZURE_OPENAI_LOCATION \
--kind OpenAI \
--sku s0 \
--custom-domain $AZURE_OPENAI_NAME
```

7. Create a GPT 3.5 Turbo model deployment named `gpt-35-turbo` in the Azure OpenAI account:

Bash

```
az cognitiveservices account deployment create \
--resource-group $RESOURCE_GROUP_NAME \
--name $AZURE_OPENAI_NAME \
--deployment-name gpt-35-turbo \
--model-name gpt-35-turbo \
--model-version "1106" \
--model-format OpenAI \
--sku-capacity "100" \
--sku-name "Standard"
```

8. Create a code interpreter session pool:

Bash

```
az containerapp sessionpool create \
--name $SESSION_POOL_NAME \
--resource-group $RESOURCE_GROUP_NAME \
--location $SESSION_POOL_LOCATION \
--max-sessions 100 \
--container-type PythonLTS \
--cooldown-period 300
```

Run the sample app locally

Before you deploy the app to Azure Container Apps, you can run it locally to test it.

Clone the app

1. Clone the [Azure Container Apps sessions samples repository](#).

Bash

```
git clone https://github.com/Azure-Samples/container-apps-dynamic-
sessions-samples.git
```

2. Change to the directory that contains the sample app:

Bash

```
cd container-apps-dynamic-sessions-samples/semantic-kernel-python-webapi
```

Configure the app

1. Create a Python virtual environment and activate it:

Bash

```
python3.11 -m venv .venv  
source .venv/bin/activate
```

Change the Python version in the command if you're using a different version. It's recommended to use Python 3.10 or later.

ⓘ Note

If you're using Windows, replace `.venv/bin/activate` with `.venv\Scripts\activate`.

2. Install the required Python packages:

Bash

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

3. To run the app, you need to configure environment variables.

a. Retrieve the Azure OpenAI account endpoint:

Bash

```
az cognitiveservices account show \  
--name $AZURE_OPENAI_NAME \  
--resource-group $RESOURCE_GROUP_NAME \  
--query properties.endpoint \  
--output tsv
```

b. Retrieve the Azure Container Apps session pool management endpoint:

Bash

```
az containerapp sessionpool show \
--name $SESSION_POOL_NAME \
--resource-group $RESOURCE_GROUP_NAME \
--query properties.poolManagementEndpoint \
--output tsv
```

- c. Create a `.env` file in the root of the sample app directory (same location as `main.py`). Add the following content to the file:

text

```
AZURE_OPENAI_ENDPOINT=<AZURE_OPENAI_ENDPOINT>
POOL_MANAGEMENT_ENDPOINT=<SESSION_POOL_MANAGEMENT_ENDPOINT>
```

Replace `<AZURE_OPENAI_ENDPOINT>` with the Azure OpenAI account endpoint and `<SESSION_POOL_MANAGEMENT_ENDPOINT>` with the session pool management endpoint.

4. The app uses `DefaultAzureCredential` to authenticate with Azure services. On your local machine, it uses your current Azure CLI login credentials. You must give yourself the *Cognitive Services OpenAI User* role on the Azure OpenAI account for the app to access the model endpoints, and the *Azure ContainerApps Session Executor* role on the session pool for the app to access the session pool.

- a. Retrieve your Azure CLI user name:

Bash

```
az account show --query user.name --output tsv
```

- b. Run the following commands to retrieve the Azure OpenAI account resource ID:

Bash

```
az cognitiveservices account show --name $AZURE_OPENAI_NAME --
resource-group $RESOURCE_GROUP_NAME --query id --output tsv
```

- c. Assign the *Cognitive Services OpenAI User* role to your Azure CLI user on the Azure OpenAI account:

Bash

```
az role assignment create --role "Cognitive Services OpenAI User" --assignee <CLI_USERNAME> --scope <AZURE_OPENAI_RESOURCE_ID>
```

Replace `<CLI_USERNAME>` with your Azure CLI user name and `<AZURE_OPENAI_RESOURCE_ID>` with the Azure OpenAI account resource ID.

d. Run the following commands to retrieve the session pool resource ID:

Bash

```
az containerapp sessionpool show --name $SESSION_POOL_NAME --resource-group $RESOURCE_GROUP_NAME --query id --output tsv
```

e. Assign the *Azure ContainerApps Session Executor* role using its ID to your Azure CLI user on the session pool:

Bash

```
az role assignment create \  
    --role "Azure ContainerApps Session Executor" \  
    --assignee <CLI_USERNAME> \  
    --scope <SESSION_POOL_RESOURCE_ID>
```

Replace `<CLI_USERNAME>` with your Azure CLI user name and `<SESSION_POOL_RESOURCE_ID>` with the session pool resource ID.

Run the app

Before running the sample app, open [main.py](#) in an editor and review the code. The app uses FastAPI to create a web API that accepts a user message in the query string.

The following lines of code instantiate a *SessionsPythonTool* and provide it to the Semantic Kernel agent:

Python

```
sessions_tool = SessionsPythonTool(  
    pool_management_endpoint,  
  
    auth_callback=auth_callback_factory("https://dynamicsessions.io/.default"),  
)  
kernel.add_plugin(sessions_tool, "SessionsTool")
```

When it needs to perform calculations, the kernel uses the code interpreter in *SessionsPythonTool* to run the code. The code is executed in a session in the session pool. By default, a random session identifier is generated when you instantiate the tool. If the kernel uses the tool to run multiple Python code snippets, it uses the same session. To ensure each end user has a unique session, use a separate kernel and tool for each user.

SessionsPythonTool is available in version `0.9.8b1` or later of the [semantic-kernel](#) package.

1. Run the sample app:

```
Bash  
fastapi dev main.py
```

2. Open a browser and navigate to `http://localhost:8000/docs`. You see the Swagger UI for the sample app.

3. Expand the `/chat` endpoint and select **Try it out**.

4. Enter `What time is it right now?` in the `message` field and select **Execute**.

The agent responds with the current time. In the terminal, you see the logs showing the agent generated Python code to get the current time and ran it in a code interpreter session.

5. To stop the app, enter `Ctrl+C` in the terminal.

Optional: Deploy the sample app to Azure Container Apps

To deploy the FastAPI app to Azure Container Apps, you need to create a container image and push it to a container registry. Then you can deploy the image to Azure Container Apps. The `az containerapp up` command combines these steps into a single command.

You then need to configure managed identity for the app and assign it the proper roles to access Azure OpenAI and the session pool.

1. Set the variables for the Container Apps environment and app name:

```
Bash
```

```
ENVIRONMENT_NAME=aca-sessions-tutorial-env  
CONTAINER_APP_NAME=chat-api
```

2. Build and deploy the app to Azure Container Apps:

Bash

```
az containerapp up \  
  --name $CONTAINER_APP_NAME \  
  --resource-group $RESOURCE_GROUP_NAME \  
  --location $SESSION_POOL_LOCATION \  
  --environment $ENVIRONMENT_NAME \  
  --env-vars "AZURE_OPENAI_ENDPOINT=<OPEN_AI_ENDPOINT>"  
  "POOL_MANAGEMENT_ENDPOINT=<SESSION_POOL_MANAGMENT_ENDPOINT>" \  
  --source .
```

Replace `<OPEN_AI_ENDPOINT>` with the Azure OpenAI account endpoint and `<SESSION_POOL_MANAGMENT_ENDPOINT>` with the session pool management endpoint.

3. Enable the system-assigned managed identity for the app:

Bash

```
az containerapp identity assign \  
  --name $CONTAINER_APP_NAME \  
  --resource-group $RESOURCE_GROUP_NAME \  
  --system-assigned
```

4. For the app to access Azure OpenAI and the session pool, you need to assign the managed identity the proper roles.

a. Retrieve the managed identity's principal ID:

Bash

```
az containerapp show \  
  --name $CONTAINER_APP_NAME \  
  --resource-group $RESOURCE_GROUP_NAME \  
  --query identity.principalId \  
  --output tsv
```

b. Retrieve the session pool resource ID:

Bash

```
az containerapp sessionpool show \
    --name $SESSION_POOL_NAME \
    --resource-group $RESOURCE_GROUP_NAME \
    --query id \
    --output tsv
```

- c. Assign the managed identity the `Azure ContainerApps Session Executor` and `Contributor` roles on the session pool:

Before you run the following command, replace `<PRINCIPAL_ID>` and `<SESSION_POOL_RESOURCE_ID>` with the values you retrieved in the previous steps.

Bash

```
az role assignment create \
    --role "Azure ContainerApps Session Executor" \
    --assignee <PRINCIPAL_ID> \
    --scope <SESSION_POOL_RESOURCE_ID>

az role assignment create \
    --role "Contributor" \
    --assignee <PRINCIPAL_ID> \
    --scope <SESSION_POOL_RESOURCE_ID>
```

- d. Retrieve the Azure OpenAI account resource ID:

Bash

```
az cognitiveservices account show \
    --name $AZURE_OPENAI_NAME \
    --resource-group $RESOURCE_GROUP_NAME \
    --query id \
    --output tsv
```

- e. Assign the managed identity the `Cognitive Services OpenAI User` role on the Azure OpenAI account:

Before you run the following command, replace `<PRINCIPAL_ID>` and `<AZURE_OPENAI_RESOURCE_ID>` with the values you retrieved in the previous steps.

Bash

```
az role assignment create \
    --role "Cognitive Services OpenAI User" \
    --assignee <PRINCIPAL_ID> \
    --scope <AZURE_OPENAI_RESOURCE_ID>
```

5. Retrieve the app's fully qualified domain name (FQDN):

Bash

```
az containerapp show \  
    --name $CONTAINER_APP_NAME \  
    --resource-group $RESOURCE_GROUP_NAME \  
    --query properties.configuration.ingress.fqdn \  
    --output tsv
```

6. Open the browser to <https://<FQDN>/docs> to test the deployed app.

Clean up resources

When you're done with the resources, you can delete them to avoid incurring charges:

Bash

```
az group delete --name $RESOURCE_GROUP_NAME --yes --no-wait
```

Next steps

[Code interpreter sessions](#)

Tutorial: Deploy an event-driven job with Azure Container Apps

Article • 04/04/2024

Azure Container Apps [jobs](#) allow you to run containerized tasks that execute for a finite duration and exit. You can trigger a job execution manually, on a schedule, or based on events. Jobs are best suited to for tasks such as data processing, machine learning, or any scenario that requires serverless ephemeral compute resources.

In this tutorial, you learn how to work with [event-driven jobs](#).

- ✓ Create a Container Apps environment to deploy your container apps
- ✓ Create an Azure Storage Queue to send messages to the container app
- ✓ Build a container image that runs a job
- ✓ Deploy the job to the Container Apps environment
- ✓ Verify that the queue messages are processed by the container app

The job you create starts an execution for each message that is sent to an Azure Storage queue. Each job execution runs a container that performs the following steps:

1. Gets one message from the queue.
2. Logs the message to the job execution logs.
3. Deletes the message from the queue.
4. Exits.

Important

The scaler monitors the queue's length to determine how many jobs to start. For accurate scaling, don't delete a message from the queue until the job execution has finished processing it.

The source code for the job you run in this tutorial is available in an Azure Samples [GitHub repository](#).

Prerequisites

- An Azure account with an active subscription.
 - If you don't have one, you [can create one for free](#).
- Install the [Azure CLI](#).
- Refer to [jobs restrictions](#) for a list of limitations.

Setup

1. To sign in to Azure from the CLI, run the following command and follow the prompts to complete the authentication process.

```
Azure CLI
```

```
az login
```

2. Ensure you're running the latest version of the CLI via the upgrade command.

```
Azure CLI
```

```
az upgrade
```

3. Install the latest version of the Azure Container Apps CLI extension.

```
Azure CLI
```

```
az extension add --name containerapp --upgrade
```

4. Register the `Microsoft.App` and `Microsoft.OperationalInsights` namespaces if you haven't already registered them in your Azure subscription.

```
Azure CLI
```

```
az provider register --namespace Microsoft.App  
az provider register --namespace Microsoft.OperationalInsights
```

5. Now that your Azure CLI setup is complete, you can define the environment variables that are used throughout this article.

```
Azure CLI
```

```
RESOURCE_GROUP="jobs-quickstart"  
LOCATION="northcentralus"  
ENVIRONMENT="env-jobs-quickstart"  
JOB_NAME="my-job"
```

Create a Container Apps environment

The Azure Container Apps environment acts as a secure boundary around container apps and jobs so they can share the same network and communicate with each other.

1. Create a resource group using the following command.

```
Azure CLI
```

```
az group create \
--name "$RESOURCE_GROUP" \
--location "$LOCATION"
```

2. Create the Container Apps environment using the following command.

```
Azure CLI
```

```
az containerapp env create \
--name "$ENVIRONMENT" \
--resource-group "$RESOURCE_GROUP" \
--location "$LOCATION"
```

Set up a storage queue

The job uses an Azure Storage queue to receive messages. In this section, you create a storage account and a queue.

1. Define a name for your storage account.

```
Bash
```

```
STORAGE_ACCOUNT_NAME=<STORAGE_ACCOUNT_NAME>
QUEUE_NAME="myqueue"
```

Replace `<STORAGE_ACCOUNT_NAME>` with a unique name for your storage account.

Storage account names must be *unique within Azure* and be from 3 to 24 characters in length containing numbers and lowercase letters only.

2. Create an Azure Storage account.

```
Azure CLI
```

```
az storage account create \
--name "$STORAGE_ACCOUNT_NAME" \
--resource-group "$RESOURCE_GROUP" \
--location "$LOCATION" \
--sku Standard_LRS \
--kind StorageV2
```

3. Save the queue's connection string into a variable.

Bash

```
QUEUE_CONNECTION_STRING=`az storage account show-connection-string -g $RESOURCE_GROUP --name $STORAGE_ACCOUNT_NAME --query connectionString --output tsv`
```

4. Create the message queue.

Azure CLI

```
az storage queue create \
--name "$QUEUE_NAME" \
--account-name "$STORAGE_ACCOUNT_NAME" \
--connection-string "$QUEUE_CONNECTION_STRING"
```

Build and deploy the job

To deploy the job, you must first build a container image for the job and push it to a registry. Then, you can deploy the job to the Container Apps environment.

1. Define a name for your container image and registry.

Bash

```
CONTAINER_IMAGE_NAME="queue-reader-job:1.0"
CONTAINER_REGISTRY_NAME=<CONTAINER_REGISTRY_NAME>
```

Replace `<CONTAINER_REGISTRY_NAME>` with a unique name for your container registry. Container registry names must be *unique within Azure* and be from 5 to 50 characters in length containing numbers and lowercase letters only.

2. Create a container registry.

Azure CLI

```
az acr create \
--name "$CONTAINER_REGISTRY_NAME" \
--resource-group "$RESOURCE_GROUP" \
--location "$LOCATION" \
--sku Basic \
--admin-enabled true
```

3. The source code for the job is available on [GitHub](#). Run the following command to clone the repository and build the container image in the cloud using the `az`

`acr build` command.

Azure CLI

```
az acr build \
    --registry "$CONTAINER_REGISTRY_NAME" \
    --image "$CONTAINER_IMAGE_NAME" \
    "https://github.com/Azure-Samples/container-apps-event-driven-jobs-tutorial.git"
```

The image is now available in the container registry.

4. Create a job in the Container Apps environment.

Azure CLI

```
az containerapp job create \
    --name "$JOB_NAME" \
    --resource-group "$RESOURCE_GROUP" \
    --environment "$ENVIRONMENT" \
    --trigger-type "Event" \
    --replica-timeout "1800" \
    --min-executions "0" \
    --max-executions "10" \
    --polling-interval "60" \
    --scale-rule-name "queue" \
    --scale-rule-type "azure-queue" \
    --scale-rule-metadata "accountName=$STORAGE_ACCOUNT_NAME"
"queueName=$QUEUE_NAME" "queueLength=1" \
    --scale-rule-auth "connection=connection-string-secret" \
    --image "$CONTAINER_REGISTRY_NAME.azurecr.io/$CONTAINER_IMAGE_NAME"
\
    --cpu "0.5" \
    --memory "1Gi" \
    --secrets "connection-string-secret=$QUEUE_CONNECTION_STRING" \
    --registry-server "$CONTAINER_REGISTRY_NAME.azurecr.io" \
    --env-vars "AZURE_STORAGE_QUEUE_NAME=$QUEUE_NAME"
"AZURE_STORAGE_CONNECTION_STRING=secretref:connection-string-secret"
```

The following table describes the key parameters used in the command.

[\[\] Expand table](#)

Parameter	Description
<code>--replica-timeout</code>	The maximum duration a replica can execute.
<code>--min-</code>	The minimum number of job executions to run per polling interval.

Parameter	Description
<code>executions</code>	
<code>--max-executions</code>	The maximum number of job executions to run per polling interval.
<code>--polling-interval</code>	The polling interval at which to evaluate the scale rule.
<code>--scale-rule-name</code>	The name of the scale rule.
<code>--scale-rule-type</code>	The type of scale rule to use.
<code>--scale-rule-metadata</code>	The metadata for the scale rule.
<code>--scale-rule-auth</code>	The authentication for the scale rule.
<code>--secrets</code>	The secrets to use for the job.
<code>--registry-server</code>	The container registry server to use for the job. For an Azure Container Registry, the command automatically configures authentication.
<code>--env-vars</code>	The environment variables to use for the job.

The scale rule configuration defines the event source to monitor. It is evaluated on each polling interval and determines how many job executions to trigger. To learn more, see [Set scaling rules](#).

The event-driven job is now created in the Container Apps environment.

Verify the deployment

The job is configured to evaluate the scale rule every 60 seconds, which checks the number of messages in the queue. For each evaluation period, it starts a new job execution for each message in the queue, up to a maximum of 10 executions.

To verify the job was configured correctly, you can send some messages to the queue, confirm that job executions are started, and the messages are logged to the job execution logs.

1. Send a message to the queue.

```
az storage message put \  
    --content "Hello Queue Reader Job" \  
    --queue-name "$QUEUE_NAME" \  
    --connection-string "$QUEUE_CONNECTION_STRING"
```

2. List the executions of a job.

Azure CLI

```
az containerapp job execution list \  
    --name "$JOB_NAME" \  
    --resource-group "$RESOURCE_GROUP" \  
    --output json
```

Since the job is configured to evaluate the scale rule every 60 seconds, it may take up to a full minute for the job execution to start. Repeat the command until you see the job execution and its status is `Succeeded`.

3. Run the following commands to see logged messages. These commands require the Log analytics extension, so accept the prompt to install extension when requested.

Azure CLI

```
LOG_ANALYTICS_WORKSPACE_ID=`az containerapp env show --name  
$ENVIRONMENT --resource-group $RESOURCE_GROUP --query  
properties.appLogsConfiguration.logAnalyticsConfiguration.customerId --  
out tsv`  
  
az monitor log-analytics query \  
    --workspace "$LOG_ANALYTICS_WORKSPACE_ID" \  
    --analytics-query "ContainerAppConsoleLogs_CL | where  
ContainerJobName_s == '$JOB_NAME' | order by _timestamp_d asc"
```

Until the `ContainerAppConsoleLogs_CL` table is ready, the command returns an error: `BadArgumentError: The request had some invalid properties`. Wait a few minutes and try again.

Tip

Having issues? Let us know on GitHub by opening an issue in the [Azure Container Apps repo](#).

Clean up resources

Once you're done, run the following command to delete the resource group that contains your Container Apps resources.

⊗ Caution

The following command deletes the specified resource group and all resources contained within it. If resources outside the scope of this tutorial exist in the specified resource group, they will also be deleted.

Azure CLI

```
az group delete \
--resource-group $RESOURCE_GROUP
```

Next steps

[Container Apps jobs](#)

Troubleshoot a container app

Article • 06/06/2024

Reviewing Azure Container Apps logs and configuration settings can reveal underlying issues if your container app isn't behaving correctly. Use the following guide to help you locate and view details about your container app.

Scenarios

The following table lists issues you might encounter while using Azure Container Apps, and the actions you can take to resolve them.

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

Scenario	Description	Actions
All scenarios		View logs Use Diagnose and solve problems
Error deploying new revision	You receive an error message when you try to deploy a new revision.	Verify Container Apps can pull your container image
Provisioning takes too long	After you deploy a new revision, the new revision has a <i>Provision status</i> of <i>Provisioning</i> and a <i>Running status</i> of <i>Processing</i> indefinitely.	Verify health probes are configured correctly
Revision is degraded	A new revision takes more than 10 minutes to provision. It finally has a <i>Provision status</i> of <i>Provisioned</i> , but a <i>Running status</i> of <i>Degraded</i> . The <i>Running status</i> tooltip reads <code>Details: Deployment Progress Deadline Exceeded. 0/1 replicas ready.</code>	Verify health probes are configured correctly
Requests to endpoints fail	The container app endpoint doesn't respond to requests.	Review ingress configuration
Requests return status 403	The container app endpoint responds to requests with HTTP error 403 (access denied).	Verify networking configuration is correct
Responses not as expected	The container app endpoint responds to requests, but the responses aren't as expected.	Verify traffic is routed to the correct revision Verify you're using

Scenario	Description	Actions
		unique tags when deploying images to the container registry
Missing parameters error	You receive error messages about missing parameters when you run <code>az containerapp</code> commands in the Azure CLI, or run cmdlets from the <code>Az.App</code> module in Azure PowerShell.	Verify latest version of Azure Container Apps extension is installed
Preview features not available	Preview features are not available when you run <code>az containerapp</code> commands in the Azure CLI.	Verify Azure Container Apps extension allows preview features

View logs

One of the first steps to take as you look for issues with your container app is to view log messages. You can view the output of both console and system logs. Your container app's console log captures the app's `stdout` and `stderr` streams. Container Apps generates [system logs](#) for service level events.

1. Sign in to the [Azure portal](#).
2. In the **Search** bar, enter your container app's name.
3. Under **Resources** section, select your container app's name.
4. In the navigation bar, expand **Monitoring** and select **Log stream** (not **Logs**).
5. If the *Log stream* page says *This revision is scaled to zero.*, select the **Go to Revision Management** button. Deploy a new revision scaled to a minimum replica count of 1. For more information, see [Scaling in Azure Container Apps](#).
6. In the *Log stream* page, set *Logs* to either **Console** or **System**.

Use the diagnose and solve problems tool

You can use the *diagnose and solve problems* tool to find issues with your container app's health, configuration, and performance.

1. Sign in to the [Azure portal](#).
2. In the **Search** bar, enter your container app's name.
3. Under **Resources** section, select your container app's name.
4. In the navigation bar, select **Diagnose and solve problems**.
5. In the *Diagnose and solve problems* page, select one of the *Troubleshooting categories*.

6. Select one of the categories in the navigation bar to find ways to fix problems with your container app.

Verify accessibility of container image

If you receive an error message when you try to deploy a new revision, verify that Container Apps is able to pull your container image.

- Ensure your container environment firewall isn't blocking access to the container registry. For more information, see [Control outbound traffic with user defined routes](#).
- If your existing VNet uses a custom DNS server instead of the default Azure-provided DNS server, verify your DNS server is configured correctly and that DNS lookup of the container registry doesn't fail. For more information, see [DNS](#).
- If you used the Container Apps cloud build feature to generate a container image for you (see [Code-to-cloud path for Azure Container Apps](#), your image isn't publicly accessible, so this section doesn't apply.

For a Docker container that can run as a console application, verify that your image is publicly accessible by running the following command in an elevated command prompt. Before you run this command, replace placeholders surrounded by <> with your values.

```
docker run --rm <YOUR_CONTAINER_IMAGE>
```

Verify that Docker runs your image without reporting any errors. If you're running [Docker on Windows](#), make sure you have the Docker Engine running.

If your image is not publicly accessible, you might receive the following error.

```
docker: Error response from daemon: pull access denied for
<YOUR_CONTAINER_IMAGE>, repository does not exist or may require 'docker
login': denied: requested access to the resource is denied. See 'docker run
--help'.
```

For more information, see [Networking in Azure Container Apps environment](#).

Review ingress configuration

Your container app's ingress settings are enforced through a set of rules that control the routing of external and internal traffic to your container app. If you're unable to connect to your container app, review these ingress settings to make sure your ingress settings aren't blocking requests.

1. Sign in to the [Azure portal](#).
2. In the *Search* bar, enter your container app's name.
3. Under *Resources*, select your container app's name.
4. In the navigation bar, expand *Settings* and select **Ingress**.

[] [Expand table](#)

Issue	Action
Is ingress enabled?	Verify the Enabled checkbox is checked.
Do you want to allow external ingress?	Verify that Ingress Traffic is set to Accepting traffic from anywhere . If your container app doesn't listen for HTTP traffic, set Ingress Traffic to Limited to Container Apps Environment .
Does your client use HTTP or TCP to access your container app?	Verify Ingress type is set to the correct protocol (HTTP or TCP).
Does your client support mTLS?	Verify Client certificate mode is set to Require only if your client supports mTLS. For more information, see Environment level network encryption .
Does your client use HTTP/1 or HTTP/2?	Verify Transport is set to the correct HTTP version (HTTP/1 or HTTP/2).
Is the target port set correctly?	Verify Target port is set to the same port your container app is listening on, or the same port exposed by your container app's Dockerfile.
Is your client IP address denied?	If IP Security Restrictions Mode isn't set to Allow all traffic , verify your client doesn't have an IP address that is denied.

For more information, see [Ingress in Azure Container Apps](#).

Verify networking configuration

Azure recursive resolvers uses the IP address `168.63.129.16` to resolve requests.

1. If your VNet uses a custom DNS server instead of the default Azure-provided DNS server, configure your DNS server to forward unresolved DNS queries to `168.63.129.16`.

2. When configuring your NSG or firewall, don't block the 168.63.129.16 address.

For more information, see [Networking in Azure Container Apps environment](#).

Verify health probes configuration

For all health probe types (liveness, readiness, and startup) that use TCP as their transport, verify their port numbers match the ingress target port you configured for your container app.

1. Sign in to the [Azure portal](#).
2. In the **Search** bar, enter your container app's name.
3. Under *Resources*, select your container app's name.
4. In the navigation bar, expand *Application* and select **Containers**.
5. In the *Containers* page, select **Health probes**.
6. Expand **Liveness probes**, **Readiness probes**, and **Startup probes**.
7. For each probe, verify the **Port** value is correct.

Update **Port** values as follows:

1. Select **Edit and deploy** to create a new revision.
2. In the *Create and deploy new revision* page, select the checkbox next to your container image and select **Edit**.
3. In the *Edit a container* window, select **Health probes**.
4. Expand **Liveness probes**, **Readiness probes**, and **Startup probes**.
5. For each probe, edit the **Port** value.
6. Select the **Save** button.
7. In the *Create and deploy new revision* page, select the **Create** button.

Configure health probes for extended startup time

If ingress is enabled, the following default probes are automatically added to the main app container if none is defined for each type.

Here are the default values for each probe type.

[+] [Expand table](#)

Property	Startup	Readiness	Liveness
Protocol	TCP	TCP	TCP
Port	Ingress target port	Ingress target port	Ingress target port

Property	Startup	Readiness	Liveness
Timeout	3 seconds	5 seconds	n/a
Period	1 second	5 seconds	n/a
Initial delay	1 second	3 seconds	n/a
Success threshold	1	1	n/a
Failure threshold	240	48	n/a

If your container app takes an extended amount of time to start (which is common in Java) you might need to customize your liveness and readiness probe *Initial delay seconds* property accordingly. You can [view the logs](#) to see the typical startup time for your container app.

1. Sign in to the [Azure portal](#).
2. In the **Search** bar, enter your container app's name.
3. Under *Resources*, select your container app's name.
4. In the navigation bar, expand *Application* and select **Containers**.
5. In the *Containers* page, select **Health probes**.
6. Select **Edit and deploy** to create a new revision.
7. In the *Create and deploy new revision* page, select the checkbox next to your container image and select **Edit**.
8. In the *Edit a container* window, select **Health probes**.
9. Expand **Liveness probes**.
10. If **Enable liveness probes** is selected, increase the value for **Initial delay seconds**.
11. Expand **Readiness probes**.
12. If **Enable readiness probes** is selected, increase the value for **Initial delay seconds**.
13. Select **Save**.
14. In the *Create and deploy new revision* page, select the **Create** button.

You can then [view the logs](#) to see if your container app starts successfully.

For more information, see [Use Health Probes](#).

Verify traffic is routed to the correct revision

If your container app doesn't behave as expected, the issue might be that requests are being routed to an outdated revision.

1. Sign in to the [Azure portal](#).
2. In the **Search** bar, enter your container app's name.

3. Under *Resources*, select your container app's name.
4. In the navigation bar, expand *Application* and select **Revisions**.

If **Revision Mode** is set to `Single`, all traffic is routed to your latest revision by default.

The *Active revisions* tab should list only one revision, with a *Traffic* value of `100%`.

If **Revision Mode** is set to `Multiple`, verify you're not routing traffic to outdated revisions.

For more information about configuring traffic splitting, see [Traffic splitting in Azure Container Apps](#).

Verify latest version of Azure Container Apps extension is installed

If you receive errors about missing parameters when you run `az containerapp` commands in Azure CLI or cmdlets from the `Az.App` module in Azure PowerShell, be sure you have the latest version of the Azure Container Apps extension installed.

```
Bash
```

```
Azure CLI
```

```
az extension add --name containerapp --upgrade
```

Verify Azure Container Apps extension allows preview features

If [preview features](#) are not available when you run `az containerapp` commands in the Azure CLI, enable preview features on the Azure Container Apps extension.

```
Azure CLI
```

```
az extension add --name containerapp --upgrade --allow-preview true
```

Next steps

[Reliability in Azure Container Apps](#)

Reliability in Azure Container Apps

Article • 11/21/2023

This article describes reliability support in [Azure Container Apps](#), and covers both regional resiliency with availability zones and cross-region resiliency with disaster recovery. For a more detailed overview of reliability in Azure, see [Azure reliability](#).

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 Container Apps uses [availability zones](#) in regions where they're available to provide high-availability protection for your applications and data from data center failures.

By enabling Container Apps' zone redundancy feature, replicas are automatically distributed across the zones in the region. Traffic is load balanced among the replicas. If a zone outage occurs, traffic is automatically routed to the replicas in the remaining zones.

Note

There is no extra charge for enabling zone redundancy, but it only provides benefits when you have 2 or more replicas, with 3 or more being ideal since most regions that support zone redundancy have 3 zones.

Prerequisites

Azure Container Apps offers the same reliability support regardless of your plan type.

Azure Container Apps uses [availability zones](#) in regions where they're available. For a list of regions that support availability zones, see [Availability zone service and regional support](#).

SLA improvements

There are no increased SLAs for Azure Container Apps. For more information on the Azure Container Apps SLAs, see [Service Level Agreement for Azure Container Apps](#).

Create a resource with availability zone enabled

Set up zone redundancy in your Container Apps environment

To take advantage of availability zones, you must enable zone redundancy when you create a Container Apps environment. The environment must include a virtual network with an available subnet. To ensure proper distribution of replicas, set your app's minimum replica count to three.

Enable zone redundancy via the Azure portal

To create a container app in an environment with zone redundancy enabled using the Azure portal:

1. Navigate to the Azure portal.
2. Search for **Container Apps** in the top search box.
3. Select **Container Apps**.
4. Select **Create New** in the *Container Apps Environment* field to open the *Create Container Apps Environment* panel.
5. Enter the environment name.
6. Select **Enabled** for the *Zone redundancy* field.

Zone redundancy requires a virtual network with an infrastructure subnet. You can choose an existing virtual network or create a new one. When creating a new virtual network, you can accept the values provided for you or customize the settings.

1. Select the **Networking** tab.
2. To assign a custom virtual network name, select **Create New** in the *Virtual Network* field.
3. To assign a custom infrastructure subnet name, select **Create New** in the *Infrastructure subnet* field.
4. You can select **Internal** or **External** for the *Virtual IP*.
5. Select **Create**.

Create Container Apps Environment

Basics Monitoring **Networking**

Selecting your own virtual network allows you to connect your application to other Azure resources or on-premises systems through the same network. [Learn more](#)

Virtual network

Use your own virtual network No Yes

Virtual network *

Infrastructure subnet *

Virtual IP

Internal: The endpoint is an internal load balancer
 External: Exposes the hosted apps on an internet-accessible IP address

Create **Cancel**

Enable zone redundancy with the Azure CLI

Create a virtual network and infrastructure subnet to include with the Container Apps environment.

When using these commands, replace the <PLACEHOLDERS> with your values.

ⓘ Note

The Consumption only environment requires a dedicated subnet with a CIDR range of /23 or larger. The workload profiles environment requires a dedicated subnet with a CIDR range of /27 or larger. To learn more about subnet sizing, see the [networking architecture overview](#).

Azure CLI

```
az network vnet create \
--resource-group <RESOURCE_GROUP_NAME> \
--name <VNET_NAME> \
--location <LOCATION> \
--address-prefix 10.0.0.0/16
```

Azure CLI

```
az network vnet subnet create \
--resource-group <RESOURCE_GROUP_NAME> \
--vnet-name <VNET_NAME> \
--name infrastructure \
--address-prefixes 10.0.0.0/21
```

Next, query for the infrastructure subnet ID.

Azure CLI

```
INFRASTRUCTURE_SUBNET=`az network vnet subnet show --resource-group
<RESOURCE_GROUP_NAME> --vnet-name <VNET_NAME> --name infrastructure --
query "id" -o tsv | tr -d '[:space:]'`
```

Finally, create the environment with the `--zone-redundant` parameter. The location must be the same location used when creating the virtual network.

Azure CLI

Azure CLI

```
az containerapp env create \
--name <CONTAINER_APP_ENV_NAME> \
--resource-group <RESOURCE_GROUP_NAME> \
--location "<LOCATION>" \
--infrastructure-subnet-resource-id $INFRASTRUCTURE_SUBNET \
--zone-redundant
```

Verify zone redundancy with the Azure CLI

ⓘ Note

The Azure Portal does not show whether zone redundancy is enabled.

Use the [az container app env show](#) command to verify zone redundancy is enabled for your Container Apps environment.



Azure CLI

```
az containerapp env show \
--name <CONTAINER_APP_ENV_NAME> \
--resource-group <RESOURCE_GROUP_NAME> \
--subscription <SUBSCRIPTION_ID>
```

The command returns a JSON response. Verify the response contains "zoneRedundant": true.

Safe deployment techniques

When you set up [zone redundancy in your container app](#), replicas are distributed automatically across the zones in the region. After the replicas are distributed, traffic is load balanced among them. If a zone outage occurs, traffic automatically routes to the replicas in the remaining zone.

You should still use safe deployment techniques such as [blue-green deployment](#). Azure Container Apps doesn't provide one-zone-at-a-time deployment or upgrades.

If you have enabled [session affinity](#), and a zone goes down, clients for that zone are routed to new replicas because the previous replicas are no longer available. Any state

associated with the previous replicas is lost.

Availability zone redeployment and migration

To take advantage of availability zones, enable zone redundancy as you create the Container Apps environment. The environment must include a virtual network with an available subnet. You can't migrate an existing Container Apps environment from nonavailability zone support to availability zone support.

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. Before you begin to think about creating your disaster recovery plan, see [Recommendations for designing a disaster recovery strategy](#).

When it comes to DR, Microsoft uses the [shared responsibility model](#). In a shared responsibility model, Microsoft ensures that the baseline infrastructure and platform services are available. At the same time, many Azure services don't automatically replicate data or fall back from a failed region to cross-replicate to another enabled region. For those services, you are responsible for setting up a disaster recovery plan that works for your workload. Most services that run on Azure platform as a service (PaaS) offerings provide features and guidance to support DR and you can use [service-specific features to support fast recovery](#) to help develop your DR plan.

In the unlikely event of a full region outage, you have the option of using one of two strategies:

- **Manual recovery:** Manually deploy to a new region, or wait for the region to recover, and then manually redeploy all environments and apps.
- **Resilient recovery:** First, deploy your container apps in advance to multiple regions. Next, use Azure Front Door or Azure Traffic Manager to handle incoming requests, pointing traffic to your primary region. Then, should an outage occur, you can redirect traffic away from the affected region. For more information, see [Cross-region replication in Azure](#).

Note

Regardless of which strategy you choose, make sure your deployment configuration files are in source control so you can easily redeploy if necessary.

More guidance

The following resources can help you create your own disaster recovery plan:

- [Failure and disaster recovery for Azure applications](#)
- [Azure resiliency technical guidance](#)

Next steps

[Reliability in Azure](#)

Observability in Azure Container Apps

Article • 05/02/2024

Azure Container Apps provides several built-in observability features that together give you a holistic view of your container app's health throughout its application lifecycle. These features help you monitor and diagnose the state of your app to improve performance and respond to trends and critical problems.

These features include:

[+] Expand table

Feature	Description
Log streaming	View streaming system and console logs from a container in near real-time.
Container console	Connect to the Linux console in your containers to debug your application from inside the container.
Azure Monitor metrics	View and analyze your application's compute and network usage through metric data.
Application logging	Monitor, analyze, and debug your app using log data.
Azure Monitor Log Analytics	Run queries to view and analyze your app's system and application logs.
Azure Monitor alerts	Create and manage alerts to notify you of events and conditions based on metric and log data.

ⓘ Note

While not a built-in feature, [Azure Monitor Application Insights](#) is a powerful tool to monitor your web and background applications. Although Container Apps doesn't support the Application Insights auto-instrumentation agent, you can instrument your application code using Application Insights SDKs.

Application lifecycle observability

With Container Apps observability features, you can monitor your app throughout the development-to-production lifecycle. The following sections describe the most effective monitoring features for each phase.

Development and test

During the development and test phase, real-time access to your containers' application logs and console is critical for debugging issues. Container Apps provides:

- [Log streaming](#): View real-time log streams from your containers.
- [Container console](#): Access the container console to debug your application.

Deployment

Once you deploy your container app, continuous monitoring helps you quickly identify problems that occur around error rates, performance, and resource consumption.

Azure Monitor gives you the ability to track your app with the following features:

- [Azure Monitor metrics](#): Monitor and analyze key metrics.
- [Azure Monitor alerts](#): Receive alerts for critical conditions.
- [Azure Monitor Log Analytics](#): View and analyze application logs.

Maintenance

Container Apps manages updates to your container app by creating [revisions](#). You can run multiple revisions concurrently in blue green deployments or to perform A/B testing. These observability features help you monitor your app across revisions:

- [Azure Monitor metrics](#): Monitor and compare key metrics for multiple revisions.
- [Azure Monitor alerts](#): Receive individual alerts per revision.
- [Azure Monitor Log Analytics](#): View, analyze, and compare log data for multiple revisions.

Next steps

[Health probes in Azure Container Apps](#)

Application Logging in Azure Container Apps

Article • 05/28/2024

Azure Container Apps provides two types of application logging categories:

- [Container console logs](#) stream from your container console.
- [System logs](#) are generated by the Azure Container Apps service.

You can view the [log streams](#) in near real-time in the Azure portal or CLI. For more options to store and monitor your logs, see [Logging options](#).

Container console Logs

Console logs originate from the `stderr` and `stdout` messages from the containers in your container app and Dapr sidecars. When you implement logging in your application, you can troubleshoot problems and monitor the health of your app.

Tip

Instrumenting your code with well-defined log messages can help you to understand how your code is performing and to debug issues. To learn more about best practices refer to [Design for operations](#).

System logs

Azure Container Apps generates system logs to inform you about the status of service-level events. The log messages include the following information:

- Successfully created dapr component
- Successfully updated dapr component
- Error creating dapr component
- Successfully mounted volume
- Error mounting volume
- Successfully bound Domain
- Auth enabled on app
- Creating authentication config
- Auth config created successfully
- Setting a traffic weight

- Creating a new revision:
- Successfully provisioned revision
- Deactivating Old revisions
- Error provisioning revision

System logs emit the following messages:

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

Source	Type	Message
Dapr	Info	Successfully created dapr component <component-name> with scope <dapr-component-scope>
Dapr	Info	Successfully updated dapr component <component-name> with scope <component-type>
Dapr	Error	Error creating dapr component <component-name>
Volume Mounts	Info	Successfully mounted volume <volume-name> for revision <revision-scope>
Volume Mounts	Error	Error mounting volume <volume-name>
Domain Binding	Info	Successfully bound Domain <domain> to the container app <container app name>
Authentication	Info	Auth enabled on app. Creating authentication config
Authentication	Info	Auth config created successfully
Traffic weight	Info	Setting a traffic weight of <percentage>% for revision <revision-name\>
Revision Provisioning	Info	Creating a new revision: <revision-name>
Revision Provisioning	Info	Successfully provisioned revision <name>
Revision Provisioning	Info	Deactivating Old revisions since 'ActiveRevisionsMode=Single'
Revision Provisioning	Error	Error provisioning revision <revision-name>. ErrorCode: <[ErrImagePull] [Timeout] [ContainerCrashing]>

Next steps

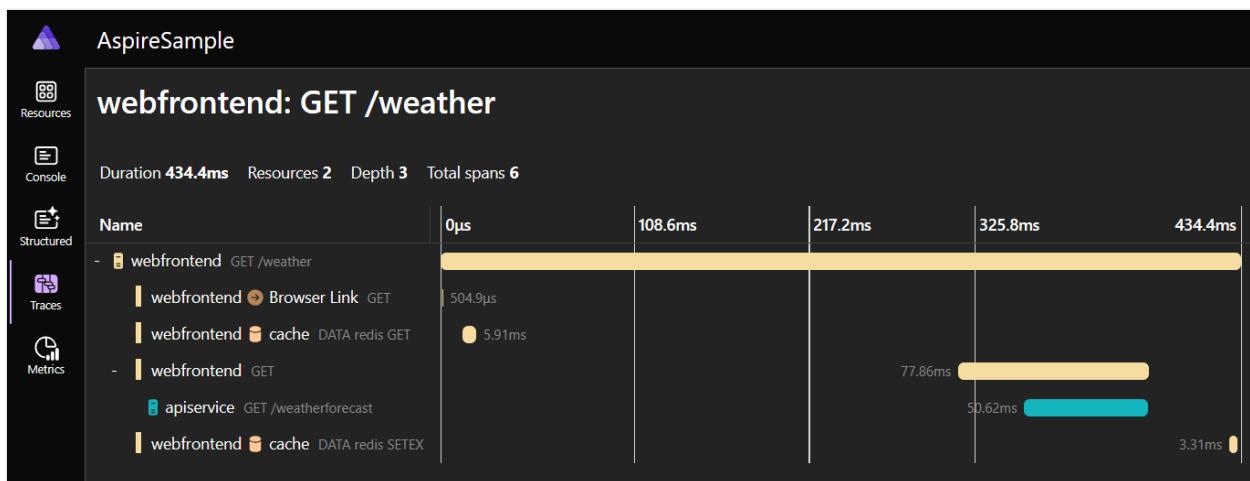
Logging options

Read real time app data with .NET Aspire Dashboard in Azure Container Apps (preview)

Article • 09/19/2024

The [.NET Aspire Dashboard](#) displays live data about how applications and other resources are running within an environment.

The following image is a screenshot of a trace visualization generated by the .NET Aspire Dashboard.



The information displayed on the dashboard comes from two sources:

- OpenTelemetry (OTel), an open-source library for tracking **traces**, **metrics**, and **logs** for your applications. [This documentation](#) provides more information on how the Aspire dashboard integrates with OTel.
 - **Traces** track the lifecycle of requests - how a request is received and processed as it moves between different parts of the application. This information is useful for identifying bottlenecks and other issues.
 - **Metrics** are real-time measurements of the general health and performance of the infrastructure - for example, how many CPU resources are consumed and how many transactions that the application handles per second. This information is useful for understanding the responsiveness of your app or identifying early warning signs of performance issues.
 - **Logs** record all events and errors that take place during the running of the application. This information is useful for finding when a problem occurred and correlated events.

- The Kubernetes API provides information about the underlying Kubernetes pods on which your application is running on and their logs.

The dashboard is secured against unauthorized access and modification. To use the dashboard, a user must have 'Write' permissions or higher - in other words, they must be a Contributor or Owner on the environment.

Enable the dashboard

You can enable the .NET Aspire Dashboard on any existing container app using the following steps.

Azure CLI

```
dotnet new aspire-starter
azd init --location westus2
azd config set aspire.dashboard on
azd up
```

The `up` command returns the dashboard URL that you can open in a browser.

Troubleshooting

Refer to the following items if you have issues enabling your dashboard:

- The portal can take up to two minutes for the dashboard to activate. If you try to go to the dashboard before it's ready, the server returns a `404` or `421` error.
- If you encounter a `421` "Misdirected Request" error, close the browser window, wait a few minutes, and try again.
- You might receive an authentication error when accessing the dashboard that reads, "Could not authenticate user with requested resource."

To solve this problem, make sure you grant the `Microsoft.App/managedEnvironments/write`, `Contributor`, or `Owner` roles on your Container Apps environment.

Related content

[.NET Aspire dashboard overview](#)

Feedback

Was this page helpful?

 Yes

 No

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

Log storage and monitoring options in Azure Container Apps

Article • 03/26/2023

Azure Container Apps gives you options for storing and viewing your application logs. Logging options are configured in your Container Apps environment where you select the log destination.

Container Apps application logs consist of two different categories:

- Container console output (`stdout/stderr`) messages.
- System logs generated by Azure Container Apps.
- Spring App console logs.

You can choose between these logs destinations:

- **Log Analytics:** Azure Monitor Log Analytics is the default storage and viewing option. Your logs are stored in a Log Analytics workspace where they can be viewed and analyzed using Log Analytics queries. To learn more about Log Analytics, see [Azure Monitor Log Analytics](#).
- **Azure Monitor:** Azure Monitor routes logs to one or more destinations:
 - Log Analytics workspace for viewing and analysis.
 - Azure storage account to archive.
 - Azure event hub for data ingestion and analytic services. For more information, see [Azure Event Hubs](#).
 - An Azure partner monitoring solution such as, Datadog, Elastic, Logz.io and others. For more information, see [Partner solutions](#).
- **None:** You can disable the storage of log data. When disabled, you can still view real-time container logs via the **Logs stream** feature in your container app. For more information, see [Log streaming](#).

ⓘ Note

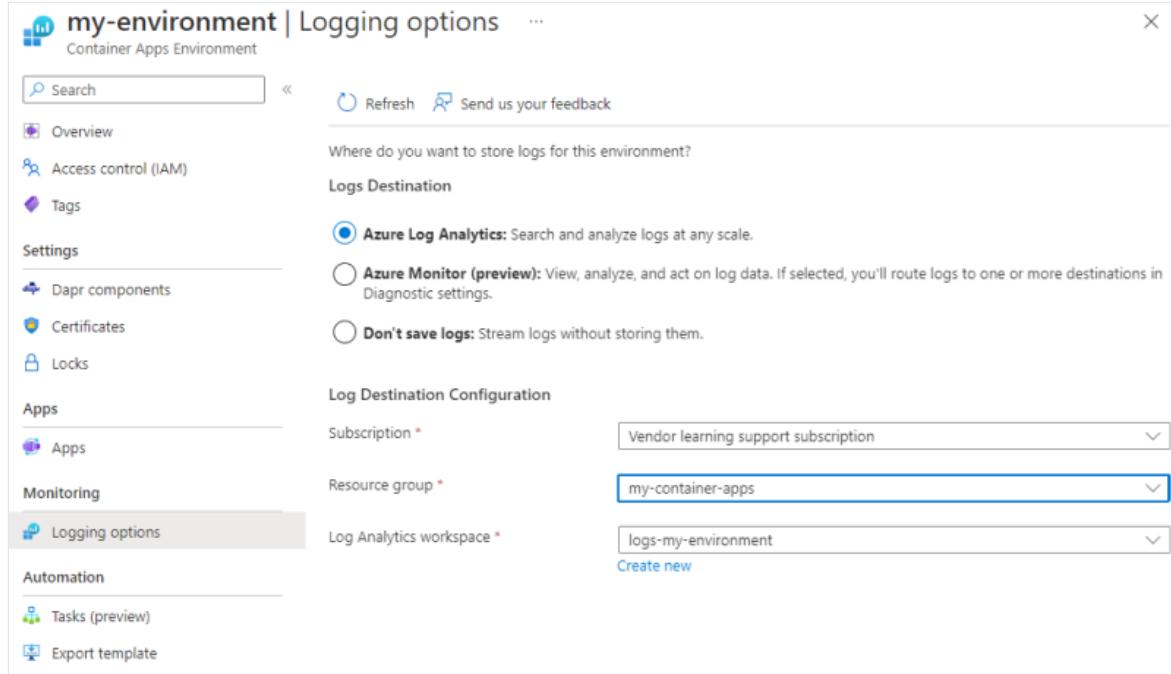
Azure Monitor is not currently supported in the Consumption + Dedicated plan structure.

When *None* or the *Azure Monitor* destination is selected, the **Logs** menu item providing the Log Analytics query editor in the Azure portal is disabled.

Configure options via the Azure portal

Use these steps to configure the logging options for your Container Apps environment in the Azure portal:

1. Go to the **Logging Options** on your Container Apps environment window in the portal.

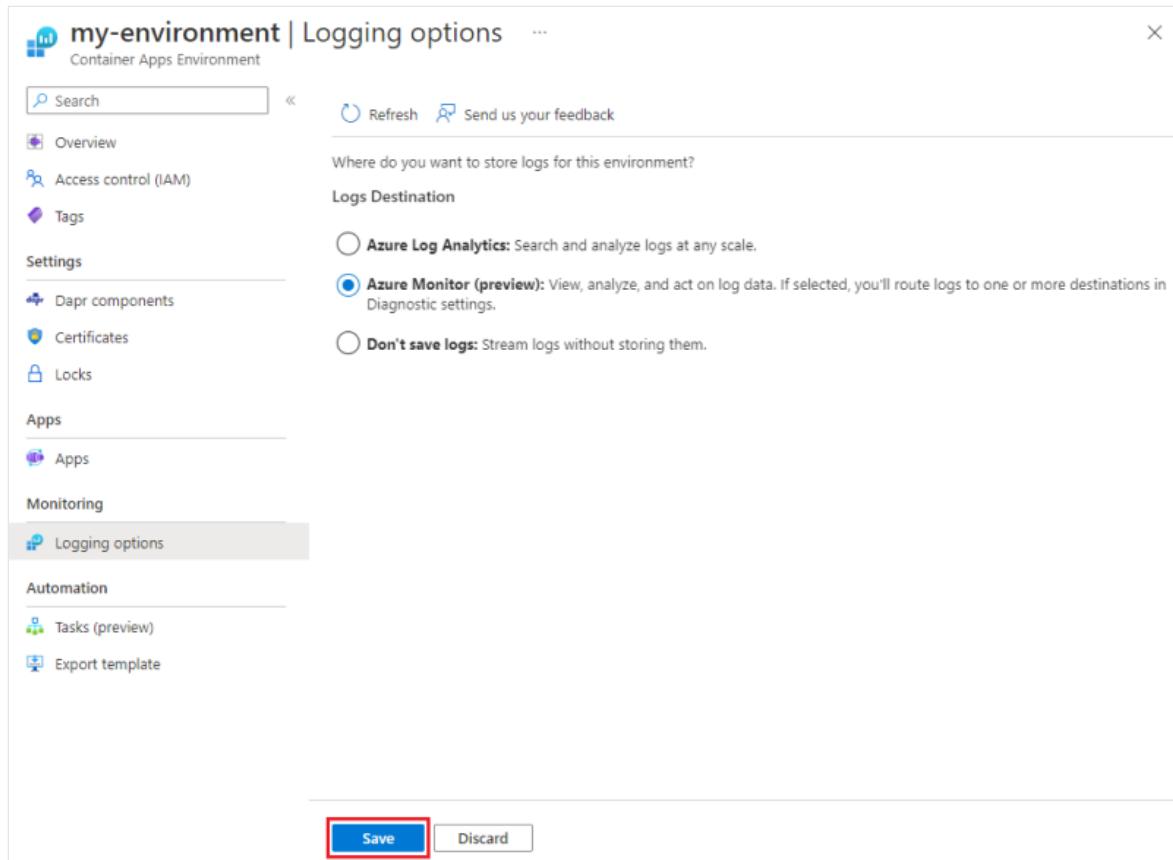


The screenshot shows the 'Logging options' section of the Azure Container Apps portal. On the left, there's a sidebar with links like Overview, Access control (IAM), Tags, Settings (selected), Dapr components, Certificates, Locks, Apps (selected), Monitoring, Logging options (selected), Automation, Tasks (preview), and Export template. The main area has a heading 'Logs Destination' with three options: 'Azure Log Analytics' (selected), 'Azure Monitor (preview)', and 'Don't save logs'. Below this is a 'Log Destination Configuration' section with fields for Subscription (Vendor learning support subscription), Resource group (my-container-apps), and Log Analytics workspace (logs-my-environment). A 'Create new' button is also present.

2. You can choose from the following **Logs Destination** options:

- **Log Analytics:** With this option, you select a Log Analytics workspace to store your log data. Your logs can be viewed through Log Analytics queries. To learn more about Log Analytics, see [Azure Monitor Log Analytics](#).
- **Azure Monitor:** Azure Monitor routes your logs to a destination. When you select this option, you must select **Diagnostic settings** to complete the configuration after you select **Save** on this page.
- **None:** This option disables the storage of log data.

3. Select Save.



4. If you have selected **Azure Monitor** as your logs destination, you must configure **Diagnostic settings**. The **Diagnostic settings** item appears below the **Logging options** menu item.

Diagnostic settings

When you select **Azure Monitor** as your logs destination, you must configure the destination details. Select **Diagnostic settings** from the left side menu of the Container Apps Environment window in the portal.

my-environment | Logging options

Container Apps Environment

Search Refresh Send us your feedback

Open Diagnostic Settings and select one or more log destinations →

Where do you want to store logs for this environment?

Logs Destination

Azure Log Analytics: Search and analyze logs at any scale.

Azure Monitor (preview): View, analyze, and act on log data. If selected, you'll route logs to one or more destinations in Diagnostic settings.

Don't save logs: Stream logs without storing them.

Overview Access control (IAM) Tags

Settings

Dapr components Certificates Locks

Apps

Apps

Monitoring

Logging options **Diagnostic settings**

Automation

Tasks (preview)

Export template

Destination details are saved as *diagnostic settings*. You can create up to five diagnostic settings for your container app environment. You can configure different log categories for each diagnostic setting. For example, create one diagnostic setting to send the system logs category to one destination, and another to send the container console logs category to another destination.

To create a new *diagnostic setting*:

1. Select Add diagnostic setting.

my-environment | Diagnostic settings

Container Apps Environment

Search Refresh Feedback

Diagnostic settings are used to configure streaming export of platform logs and metrics for a resource to the destination of your choice. You may create up to five different diagnostic settings to send different logs and metrics to independent destinations. [Learn more about diagnostic settings](#)

Name	Storage account	Event hub	Log Analytics works...	Partner solution	Edit setting
No diagnostic settings defined					

+ Add diagnostic setting

Click 'Add Diagnostic setting' above to configure the collection of the following data:

- Container App console logs
- Container App system logs

Overview Access control (IAM) Tags

Settings

Dapr components Certificates Locks

Apps

Apps

Monitoring

Logging options **Diagnostic settings**

Automation

Tasks (preview)

Export template

2. Enter a name for your diagnostic setting.

The screenshot shows the 'Diagnostic setting' configuration page. At the top, there are buttons for 'Save', 'Discard', 'Delete', and 'Feedback'. Below that, a descriptive text about diagnostic settings is shown. A red box highlights the 'Diagnostic setting name *' input field, which contains a placeholder. Another red box highlights the 'Logs' section, specifically the 'Category groups' and 'Categories' sections. The 'Category groups' section has two options: 'audit' and 'allLogs'. The 'Categories' section has two options: 'Container App console logs' and 'Container App system logs'. To the right, under 'Destination details', there are four checkboxes: 'Send to Log Analytics workspace', 'Archive to a storage account', 'Stream to an event hub', and 'Send to partner solution'. The 'Send to Log Analytics workspace' checkbox is unselected.

3. Select the log **Category groups** or **Categories** you want to send to this destination. You can select one or more categories.

4. Select one or more **Destination details**:

- **Send to Log Analytics workspace:** Select from existing Log Analytics workspaces.

The screenshot shows the same 'Diagnostic setting' configuration page as before, but with different selections. The 'Diagnostic setting name *' field now contains 'console-logs-log-analytics'. In the 'Logs' section, the 'Container App console logs' checkbox is selected. In the 'Destination details' section, the 'Send to Log Analytics workspace' checkbox is selected, and its dropdown shows 'Subscription: Demo-Subscription' and 'Log Analytics workspace: workspace-mycontainerappsworKsHpe (canadacentral)'. Other destination options like 'Archive to a storage account' and 'Stream to an event hub' are unselected.

- **Archive to a storage account:** You can choose from existing storage accounts. When the individual log categories are selected, you can set the

Retention (days) for each category.

The screenshot shows the 'Diagnostic setting' configuration page. Under 'Logs', 'Container App console logs' and 'Container App system logs' are selected, with retention set to 365 and 90 days respectively. In the 'Destination details' section, 'Archive to a storage account' is selected, and a storage account named 'mystorageaccount' is chosen. Other options like 'Stream to an event hub' and 'Send to partner solution' are available but not selected.

- Stream to an event hub: Select from Azure event hubs.

The screenshot shows the 'Diagnostic setting' configuration page with 'all-logs-to-event-hub' as the name. Under 'Logs', 'audit' is unselected and 'allLogs' is selected. In the 'Destination details' section, 'Stream to an event hub' is selected, and configuration fields for 'Subscription' (Demo-Subscription), 'Event hub namespace' (my-event-hub-name-space), 'Event hub name (optional)' (my-event-hub), and 'Event hub policy name' (my-event-hub-policy) are filled out. Other options like 'Send to Log Analytics workspace' and 'Send to partner solution' are available but not selected.

- Send to a partner solution: Select from Azure partner solutions.

5. Select Save.

For more information about Diagnostic settings, see [Diagnostic settings in Azure Monitor](#).

Configure options using the Azure CLI

Configure logs destination for your Container Apps environment using the Azure CLI `az containerapp create` and `az containerapp update` commands with the `--logs-destination` argument.

The destination values are: `log-analytics`, `azure-monitor`, and `none`.

For example, to create a Container Apps environment using an existing Log Analytics workspace as the logs destination, you must provide the `--logs-destination` argument with the value `log-analytics` and the `--logs-destination-id` argument with the value of the Log Analytics workspace resource ID. You can get the resource ID from the Log Analytics workspace page in the Azure portal or from the `az monitor log-analytics workspace show` command.

Replace <PLACEHOLDERS> with your values:

Azure CLI

```
az containerapp env create \
--name <ENVIRONMENT_NAME> \
--resource-group <RESOURCE_GROUP> \
--logs-destination log-analytics \
--logs-workspace-id <WORKSPACE_ID>
```

To update an existing Container Apps environment to use Azure Monitor as the logs destination:

Replace <PLACEHOLDERS> with your values:

Azure CLI

```
az containerapp env update \
--name <ENVIRONMENT_NAME> \
--resource-group <RESOURCE_GROUP> \
--logs-destination azure-monitor
```

When `--logs-destination` is set to `azure-monitor`, create diagnostic settings to configure the destination details for the log categories with the `az monitor diagnostics-settings` command.

For more information about Azure Monitor diagnostic settings commands, see [az monitor diagnostic-settings](#). Container Apps log categories are `ContainerAppConsoleLogs` and `ContainerAppSystemLogs`.

Next steps

[Monitor logs with Log Analytics](#)

View log streams in Azure Container Apps

Article • 03/26/2023

While developing and troubleshooting your container app, it's essential to see the logs for your container app in real time. Azure Container Apps lets you stream:

- system logs from the Container Apps environment and your container app.
- container console logs from your container app.

Log streams are accessible through the Azure portal or the Azure CLI.

View log streams via the Azure portal

You can view system logs and console logs in the Azure portal. System logs are generated by the container app's runtime. Console logs are generated by your container app.

Environment system log stream

To troubleshoot issues in your container app environment, you can view the system log stream from your environment page. The log stream displays the system logs for the Container Apps service and the apps actively running in the environment:

1. Go to your environment in the Azure portal.
2. Select Log stream under the *Monitoring* section on the sidebar menu.

```
Connecting...
{"TimeStamp": "2023-02-13T21:40:09Z", "Type": "Normal", "ContainerAppName": null, "RevisionName": null, "ReplicaName": null, "Msg": "Connecting... to the events collector...", "Reason": "StartingGettingEvents", "EventSource": "ContainerAppController", "Count": 1}
("TimeStamp": "2023-02-13T21:40:10Z", "Type": "Normal", "ContainerAppName": null, "RevisionName": null, "ReplicaName": null, "Msg": "Successfully connected to events server", "Reason": "ConnectedToEventsServer", "EventSource": "ContainerAppController", "Count": 1)
("TimeStamp": null, "Type": "Normal", "ContainerAppName": null, "RevisionName": null, "ReplicaName": null, "Msg": "Metrics server is starting to listen", "Reason": null, "EventSource": null, "Count": 0)
("TimeStamp": null, "Type": "Normal", "ContainerAppName": null, "RevisionName": null, "ReplicaName": null, "Msg": "Starting server", "Reason": null, "EventSource": null, "Count": 0)
("TimeStamp": null, "Type": "Normal", "ContainerAppName": null, "RevisionName": null, "ReplicaName": null, "Msg": "Starting EventSource", "Reason": null, "EventSource": null, "Count": 0)
("TimeStamp": null, "Type": "Normal", "ContainerAppName": null, "RevisionName": null, "ReplicaName": null, "Msg": "Starting Controller", "Reason": null, "EventSource": null, "Count": 0)
("TimeStamp": null, "Type": "Normal", "ContainerAppName": null, "RevisionName": null, "ReplicaName": null, "Msg": "Starting workers", "Reason": null, "EventSource": null, "Count": 0)
("TimeStamp": "2023-02-13 09:05:01 \u00d70280000 UTC", "Type": "Normal", "ContainerAppName": "album-api", "RevisionName": "album-api-74e9b5", "ReplicaName": "", "Msg": "Started scalers watch", "Reason": "KEDAScalarsStarted", "EventSource": "KEDA", "Count": 1}
("TimeStamp": "2023-02-13T21:41:10Z", "Type": "Normal", "ContainerAppName": null, "RevisionName": null, "ReplicaName": null, "Msg": "No events since last 60 seconds", "Reason": "NoNewEvents", "EventSource": "ContainerAppController", "Count": 1)
("TimeStamp": "2023-02-13T21:42:10Z", "Type": "Normal", "ContainerAppName": null, "RevisionName": null, "ReplicaName": null, "Msg": "No events since last 60 seconds", "Reason": "NoNewEvents", "EventSource": "ContainerAppController", "Count": 1)
("TimeStamp": "2023-02-13T21:43:11Z", "Type": "Normal", "ContainerAppName": null, "RevisionName": null, "ReplicaName": null, "Msg": "No events since last 60 seconds", "Reason": "NoNewEvents", "EventSource": "ContainerAppController", "Count": 1)
("TimeStamp": "2023-02-13T21:44:11Z", "Type": "Normal", "ContainerAppName": null, "RevisionName": null, "ReplicaName": null, "Msg": "No events since last 60 seconds", "Reason": "NoNewEvents", "EventSource": "ContainerAppController", "Count": 1)
```

Container app log stream

You can view a log stream of your container app's system or console logs from your container app page.

1. Go to your container app in the Azure portal.
2. Select **Log stream** under the *Monitoring* section on the sidebar menu.
3. To view the console log stream, select **Console**.
 - a. If you have multiple revisions, replicas, or containers, you can select from the drop-down menus to choose a container. If your app has only one container, you can skip this step.

```
Connecting...
2023-02-13T21:56:00.96290 Connecting to the container 'album-api'...
2023-02-13T21:56:00.98787 Successfully Connected to container: 'album-api' [Revision: 'album-api--vgirtv5', Replica: 'album-api--vgirtv5-56f4bb96db-5lh8t']
2023-02-13T21:54:57.1817613372 [Info]: Microsoft.Hosting.Lifetime[14]
2023-02-13T21:54:57.1818013372 Now listening on: http://[::]:3500
2023-02-13T21:54:57.1829765552 [Info]: Microsoft.Hosting.Lifetime[0]
2023-02-13T21:54:57.1829923562 Application started. Press Ctrl+C to shut down.
2023-02-13T21:54:57.1836716662 [Info]: Microsoft.Hosting.Lifetime[0]
2023-02-13T21:54:57.1836794662 Hosting environment: Production
2023-02-13T21:54:57.1836820662 [Info]: Microsoft.Hosting.Lifetime[0]
2023-02-13T21:54:57.1836847672 Content root path: /app/
2023-02-13T21:57:01.39368 No logs since last 60 seconds
2023-02-13T21:58:01.85126 No logs since last 60 seconds
2023-02-13T21:59:02.16494 No logs since last 60 seconds
2023-02-13T22:00:02.50087 No logs since last 60 seconds
```

4. To view the system log stream, select **System**. The system log stream displays the system logs for all running containers in your container app.

```
Connecting...
("TimeStamp": "2023-02-13T22:06:40Z", "Type": "Normal", "ContainerAppName": null, "RevisionName": null, "ReplicaName": null, "Msg": "Connecting to the events collector...", "Reason": "StartingGettingEvents", "EventSource": "ContainerAppController", "Count": 1}
("TimeStamp": "2023-02-13T22:06:40Z", "Type": "Normal", "ContainerAppName": null, "RevisionName": null, "ReplicaName": null, "Msg": "Successfully connected to events server", "Reason": "ConnectedToEventsServer", "EventSource": "ContainerAppController", "Count": 1)
("TimeStamp": "2023-02-13 21:54:56 \u00d702B0000 UTC", "Type": "Normal", "ContainerAppName": "album-api", "RevisionName": "album-api--vgirtv5", "ReplicaName": "album-api--vgirtv5-56f4bb96db-k84sn", "Msg": "Successfully pulled image \u00d702ca704c3e9d38acr.azurecr.io:/album-api:20230207182936040366\u0022 in 3.271844754s", "Reason": "ImagePulled", "EventSource": "ContainerAppController", "Count": 1}
("TimeStamp": "2023-02-13 21:54:56 \u00d702B0000 UTC", "Type": "Normal", "ContainerAppName": "album-api", "RevisionName": "album-api--vgirtv5", "ReplicaName": "album-api--vgirtv5-56f4bb96db-k84sn", "Msg": "Started container album-api", "Reason": "ContainerStarted", "EventSource": "ContainerAppController", "Count": 1}
("TimeStamp": "2023-02-13 21:54:56 \u00d702B0000 UTC", "Type": "Normal", "ContainerAppName": "album-api", "RevisionName": "album-api--vgirtv5", "ReplicaName": "album-api--vgirtv5-56f4bb96db-k84sn", "Msg": "Created container album-api", "Reason": "ContainerCreated", "EventSource": "ContainerAppController", "Count": 1}
("TimeStamp": "2023-02-13 21:54:56 \u00d702B0000 UTC", "Type": "Normal", "ContainerAppName": "album-api", "RevisionName": "album-api--vgirtv5", "ReplicaName": "album-api--vgirtv5-56f4bb96db-5lh8t", "Msg": "Started container album-api", "Reason": "ContainerStarted", "EventSource": "ContainerAppController", "Count": 1}
("TimeStamp": "2023-02-13 21:54:56 \u00d702B0000 UTC", "Type": "Normal", "ContainerAppName": "album-api", "RevisionName": "album-api--vgirtv5", "ReplicaName": "album-api--vgirtv5-56f4bb96db-5lh8t", "Msg": "Started container album-api", "Reason": "ContainerStarted", "EventSource": "ContainerAppController", "Count": 1}
("TimeStamp": "2023-02-13 21:55:25 \u00d702B0000 UTC", "Type": "Normal", "ContainerAppName": "album-api", "RevisionName": "album-api--m7v4e9b", "ReplicaName": "", "Msg": "Stopped scalers watch", "Reason": "KEDAScalarsStopped", "EventSource": "KEDA", "Count": 1}
("TimeStamp": "2023-02-13 21:55:25 \u00d702B0000 UTC", "Type": "Normal", "ContainerAppName": "album-api", "RevisionName": "album-api--m7v4e9b", "ReplicaName": "", "Msg": "ScaledObject was deleted", "Reason": "ScaledObjectDeleted", "EventSource": "KEDA", "Count": 1)
```

View log streams via the Azure CLI

You can view your container app's log streams from the Azure CLI with the `az containerapp logs show` command or your container app's environment system log stream with the `az containerapp env logs show` command.

Control the log stream with the following arguments:

- `--tail` (Default) View the last n log messages. Values are 0-300 messages. The default is 20.
- `--follow` View a continuous live stream of the log messages.

Stream Container app logs

You can stream the system or console logs for your container app. To stream the container app system logs, use the `--type` argument with the value `system`. To stream the container console logs, use the `--type` argument with the value `console`. The default is `console`.

View container app system log stream

This example uses the `--tail` argument to display the last 50 system log messages from the container app. Replace the <placeholders> with your container app's values.

Bash

Azure CLI

```
az containerapp logs show \
--name <ContainerAppName> \
--resource-group <ResourceGroup> \
--type system \
--tail 50
```

This example displays a continuous live stream of system log messages from the container app using the `--follow` argument. Replace the <placeholders> with your container app's values.

Bash

Azure CLI

```
az containerapp logs show \
--name <ContainerAppName> \
--resource-group <ResourceGroup> \
--type system \
--follow
```

Use `Ctrl-C` or `Cmd-C` to stop the live stream.

View container console log stream

To connect to a container's console log stream in a container app with multiple revisions, replicas, and containers, include the following parameters in the `az containerapp logs show` command.

Argument	Description
<code>--revision</code>	The revision name.
<code>--replica</code>	The replica name in the revision.
<code>--container</code>	The container name to connect to.

You can get the revision names with the `az containerapp revision list` command. Replace the `<placeholders>` with your container app's values.

Bash

Azure CLI

```
az containerapp revision list \
--name <ContainerAppName> \
--resource-group <ResourceGroup> \
--query "[].name"
```

Use the `az containerapp replica list` command to get the replica and container names. Replace the `<placeholders>` with your container app's values.

Bash

Azure CLI

```
az containerapp replica list \
--name <ContainerAppName> \
--resource-group <ResourceGroup> \
--revision <RevisionName> \
--query "[].{Containers:properties.containers[].name, Name:name}"
```

Live stream the container console using the `az container app show` command with the `--follow` argument. Replace the <placeholders> with your container app's values.

Bash

Azure CLI

```
az containerapp logs show \
--name <ContainerAppName> \
--resource-group <ResourceGroup> \
--revision <RevisionName> \
--replica <ReplicaName> \
--container <ContainerName> \
--type console \
--follow
```

Use `Ctrl-C` or `Cmd-C` to stop the live stream.

View the last 50 console log messages using the `az containerapp logs show` command with the `--tail` argument. Replace the <placeholders> with your container app's values.

Bash

Azure CLI

```
az containerapp logs show \
--name <ContainerAppName> \
--resource-group <ResourceGroup> \
--revision <RevisionName> \
--replica <ReplicaName> \
--container <ContainerName> \
--type console \
--tail 50
```

View environment system log stream

Use the following command with the `--follow` argument to view the live system log stream from the Container Apps environment. Replace the <placeholders> with your environment values.

Bash

```
Azure CLI
```

```
az containerapp env logs show \
--name <ContainerAppEnvironmentName> \
--resource-group <ResourceGroup> \
--follow
```

Use `Ctrl-C` or `Cmd-C` to stop the live stream.

This example uses the `--tail` argument to display the last 50 environment system log messages. Replace the <placeholders> with your environment values.

Bash

```
Azure CLI
```

```
az containerapp env logs show \
--name <ContainerAppName> \
--resource-group <ResourceGroup> \
--tail 50
```

[Log storage and monitoring options in Azure Container Apps](#)

Connect to a container console in Azure Container Apps

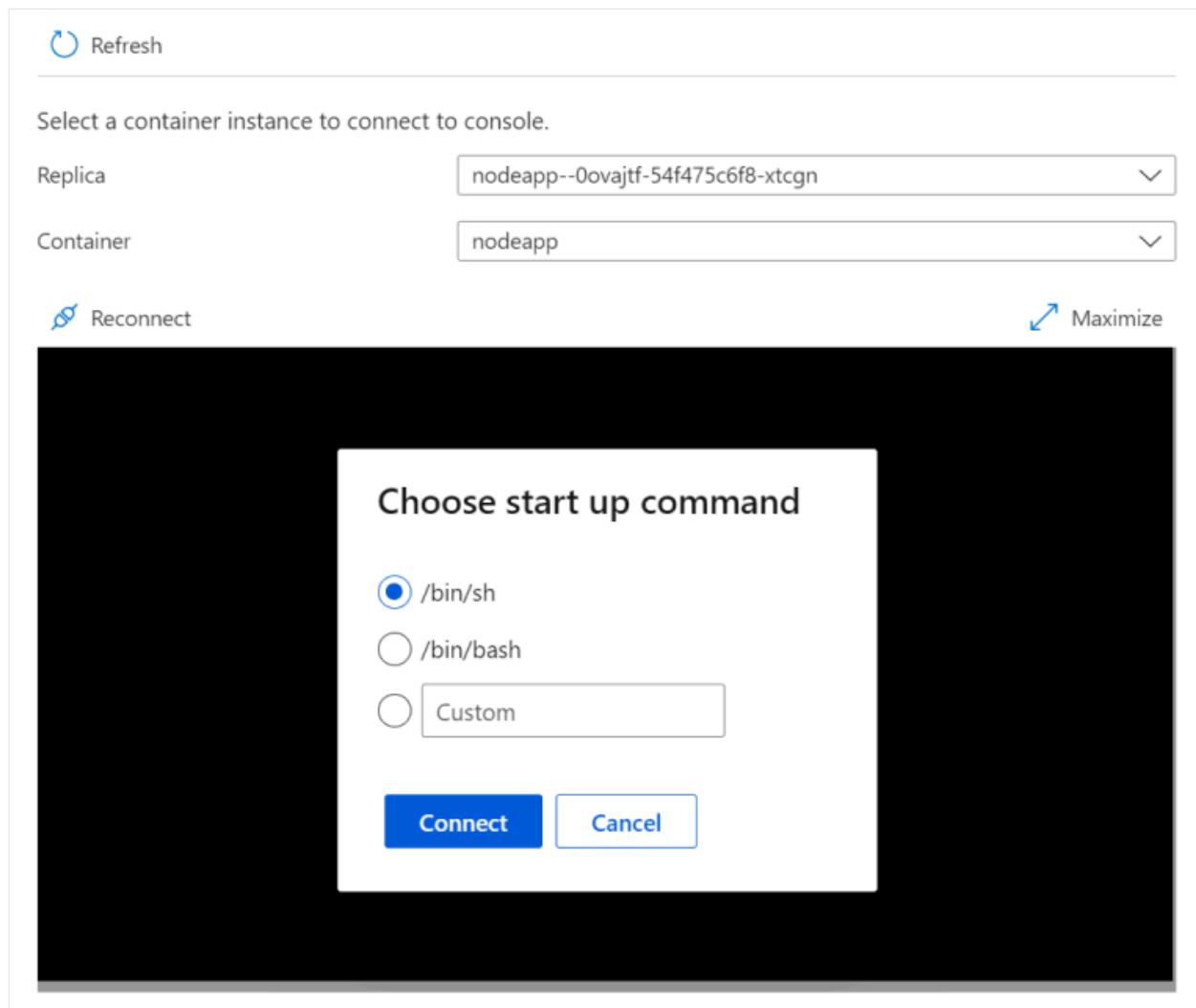
Article • 07/15/2024

Connecting to a container's console is useful when you want to troubleshoot your application inside a container. Azure Container Apps allows you to connect to a container's console using the Azure portal or Azure CLI.

Azure portal

To connect to a container's console in the Azure portal, follow these steps.

1. In the Azure portal, select **Console** in the **Monitoring** menu group from your container app page.
2. Select the revision, replica, and container you want to connect to.
3. Choose to access your console via bash, sh, or a custom executable. If you choose a custom executable, it must be available in the container.



Azure CLI

To connect to a container console, Use the `az containerapp exec` command. To exit the console, select **Ctrl-D**.

For example, connect to a container console in a container app with a single container using the following command. Replace the <PLACEHOLDERS> with your container app's values.

Bash

```
az containerapp exec \
--name <CONTAINER_APP_NAME> \
--resource-group <RESOURCE_GROUP>
```

To connect to a container console in a container app with multiple revisions, replicas, and containers include the following parameters in the `az containerapp exec` command.

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

Argument	Description
<code>--revision</code>	The revision names of the container to connect to.
<code>--replica</code>	The replica name of the container to connect to.
<code>--container</code>	The container name of the container to connect to.

You can get the revision names with the `az containerapp revision list` command. Replace the <PLACEHOLDERS> with your container app's values.

Bash

```
az containerapp revision list \
--name <CONTAINER_APP_NAME> \
--resource-group <RESOURCE_GROUP> \
--query "[].name"
```

Use the `az containerapp replica list` command to get the replica and container names. Replace the <PLACEHOLDERS> with your container app's values.

Bash

```
az containerapp replica list \  
  --name <CONTAINER_APP_NAME> \  
  --resource-group <RESOURCE_GROUP> \  
  --revision <REVISION_NAME> \  
  --query "[].{Containers:properties.containers[].name, Name:name}"
```

Connect to the container console with the `az containerapp exec` command. Replace the <PLACEHOLDERS> with your container app's values.

Bash

```
az containerapp exec \  
  --name <CONTAINER_APP_NAME> \  
  --resource-group <RESOURCE_GROUP> \  
  --revision <REVISION_NAME> \  
  --replica <REPLICA_NAME> \  
  --container <CONTAINER_NAME>
```

[View log streams from the Azure portal](#)

Feedback

Was this page helpful?

 Yes

 No

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

Monitor Azure Container Apps metrics

Article • 09/23/2024

Azure Monitor collects metric data from your container app at regular intervals to help you gain insights into the performance and health of your container app.

The metrics explorer in the Azure portal allows you to visualize the data. You can also retrieve raw metric data through the [Azure CLI](#) and Azure [PowerShell cmdlets](#).

Available metrics

Container Apps provides these basic metrics.

[+] Expand table

Title	Dimensions	Description	Metric ID	Unit
CPU Usage	Replica, Revision	CPU consumed by the container app, in nano cores (1,000,000,000 nanocores = 1 core)	UsageNanoCores	nanocores
Memory Working Set Bytes	Replica, Revision	Container app working set memory used in bytes	WorkingSetBytes	bytes
Network In Bytes	Replica, Revision	Network received bytes	RxBytes	bytes
Network Out Bytes	Replica, Revision	Network transmitted bytes	TxBytes	bytes
Replica count	Revision	Number of active replicas	Replicas	n/a
Replica Restart Count	Replica, Revision	Restarts count of container app replicas	RestartCount	n/a

Title	Dimensions	Description	Metric ID	Unit
Requests	Replica, Revision, Status Code, Status Code Category	Requests processed	Requests	n/a
Reserved Cores	Revision	Number of reserved cores for container app revisions	CoresQuotaUsed	n/a
Resiliency Connection Timeouts	Revision	Total connection timeouts	ResiliencyConnectTimeouts	n/a
Resiliency Ejected Hosts	Revision	Number of currently ejected hosts	ResiliencyEjectedHosts	n/a
Resiliency Ejections Aborted	Revision	Number of ejections aborted due to the max ejection %	ResiliencyEjectionsAborted	n/a
Resiliency Request Retries	Revision	Total request retries	ResiliencyRequestRetries	n/a
Resiliency Request Timeouts	Revision	Total requests that timed out waiting for a response	ResiliencyRequestTimeouts	n/a
Resiliency Requests Pending Connection Pool	Replica	Total requests pending a connection pool connection	ResiliencyRequestsPendingConnectionPool	n/a
Total Reserved Cores	None	Total cores reserved for the container app	TotalCoresQuotaUsed	n/a

The metrics namespace is microsoft.app/containerapps.

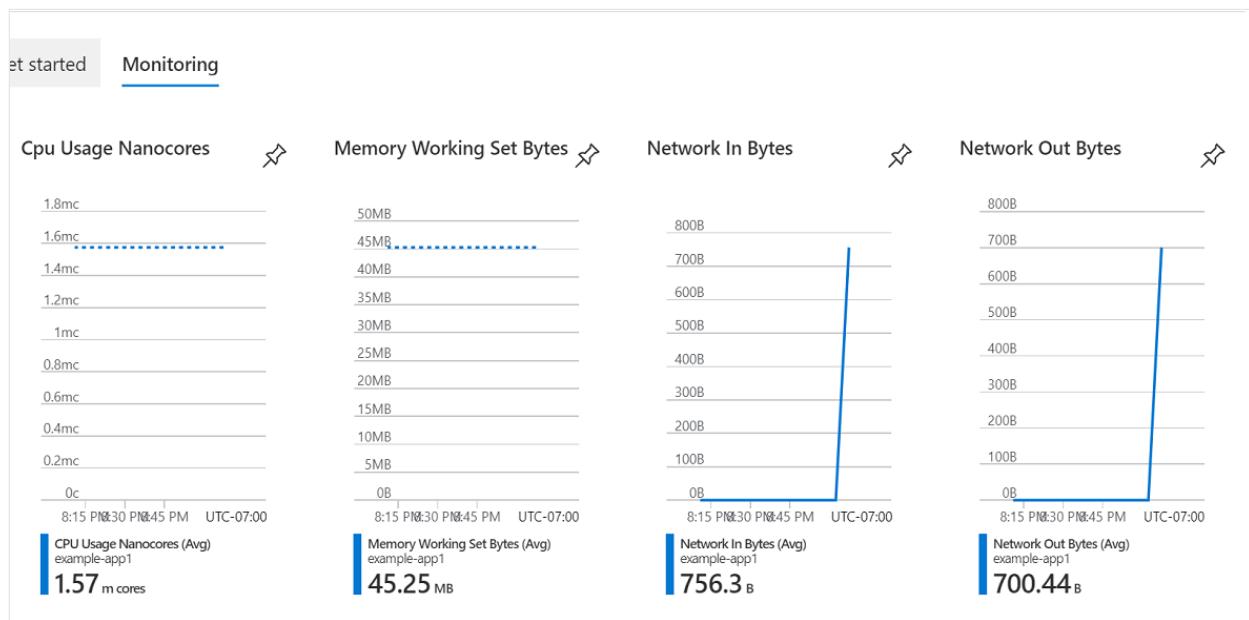
⚠ Note

Replica restart count is the aggregate restart count over the specified time range, not the number of restarts that occurred at a point in time.

More runtime specific metrics are available, [Java metrics](#).

Metrics snapshots

Select the **Monitoring** tab on your app's **Overview** page to display charts showing your container app's current CPU, memory, and network utilization.



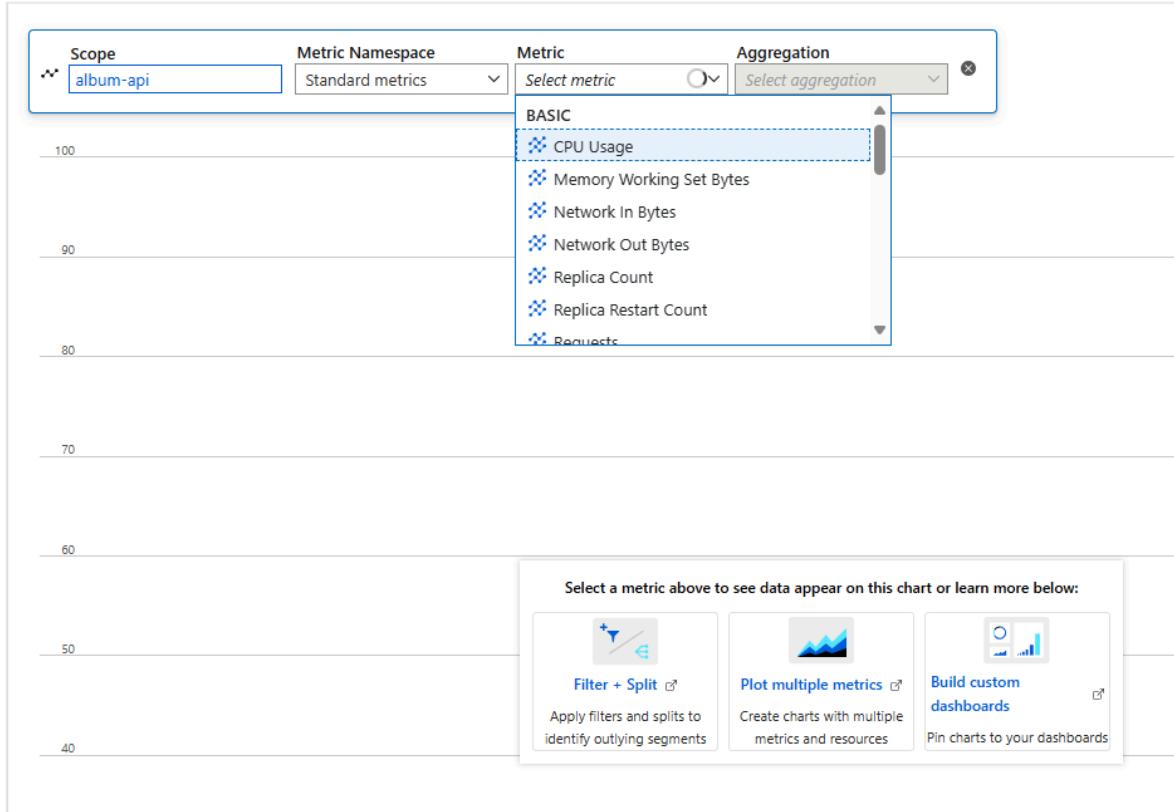
From this view, you can pin one or more charts to your dashboard or select a chart to open it in the metrics explorer.

Using metrics explorer

The Azure Monitor metrics explorer lets you create charts from metric data to help you analyze your container app's resource and network usage over time. You can pin charts to a dashboard or in a shared workbook.

1. Open the metrics explorer in the Azure portal by selecting **Metrics** from the sidebar menu on your container app's page. To learn more about metrics explorer, see [Analyze metrics with Azure Monitor metrics explorer](#).
2. Create a chart by selecting **Metric**. You can modify the chart by changing aggregation, adding more metrics, changing time ranges and intervals, adding

filters, and applying splitting.

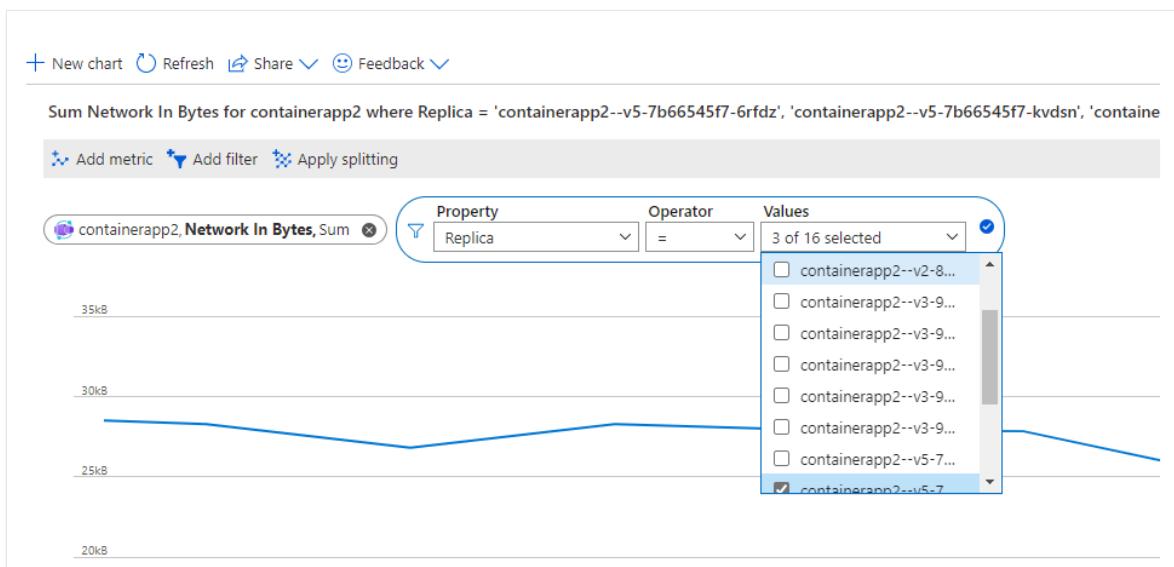


Add filters

Optionally, you can create filters to limit the data shown based on revisions and replicas.

To create a filter:

1. Select **Add filter**.
2. Select a revision or replica from the **Property** list.
3. Select values from the **Value** list.



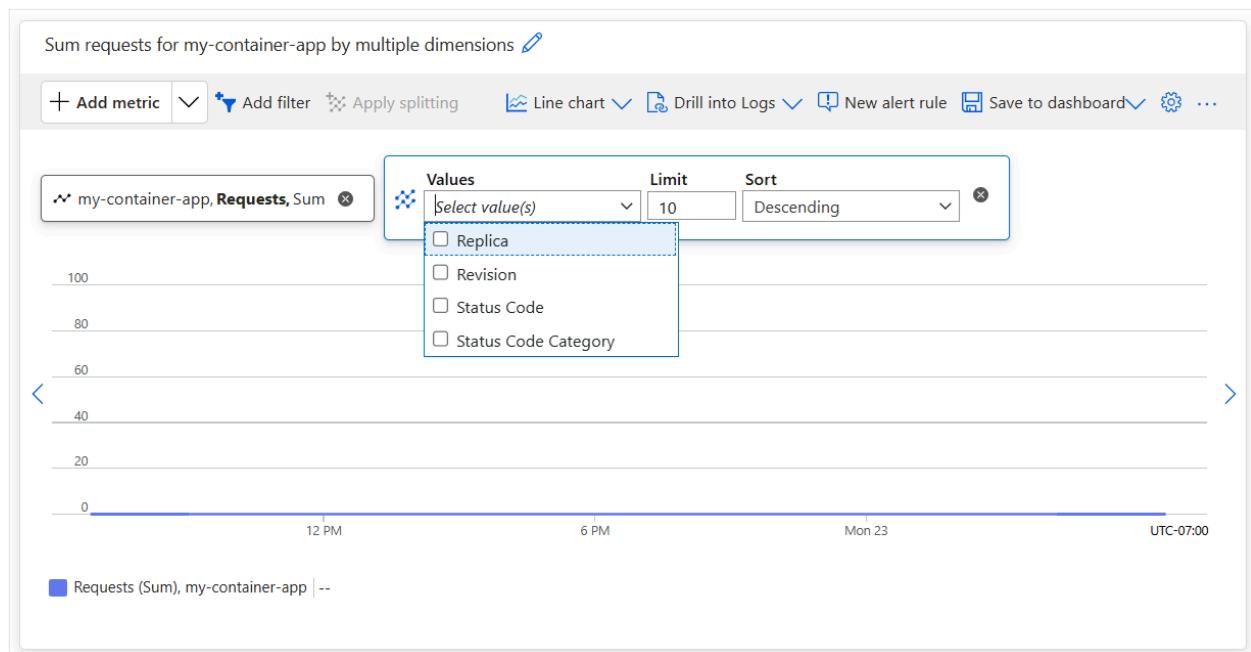
Split metrics

When your chart contains a single metric, you can choose to split the metric information by revision or replica with the exceptions:

- The *Replica count* metric can only split by revision.
- The *Requests* metric can also be split on the status code and status code category.

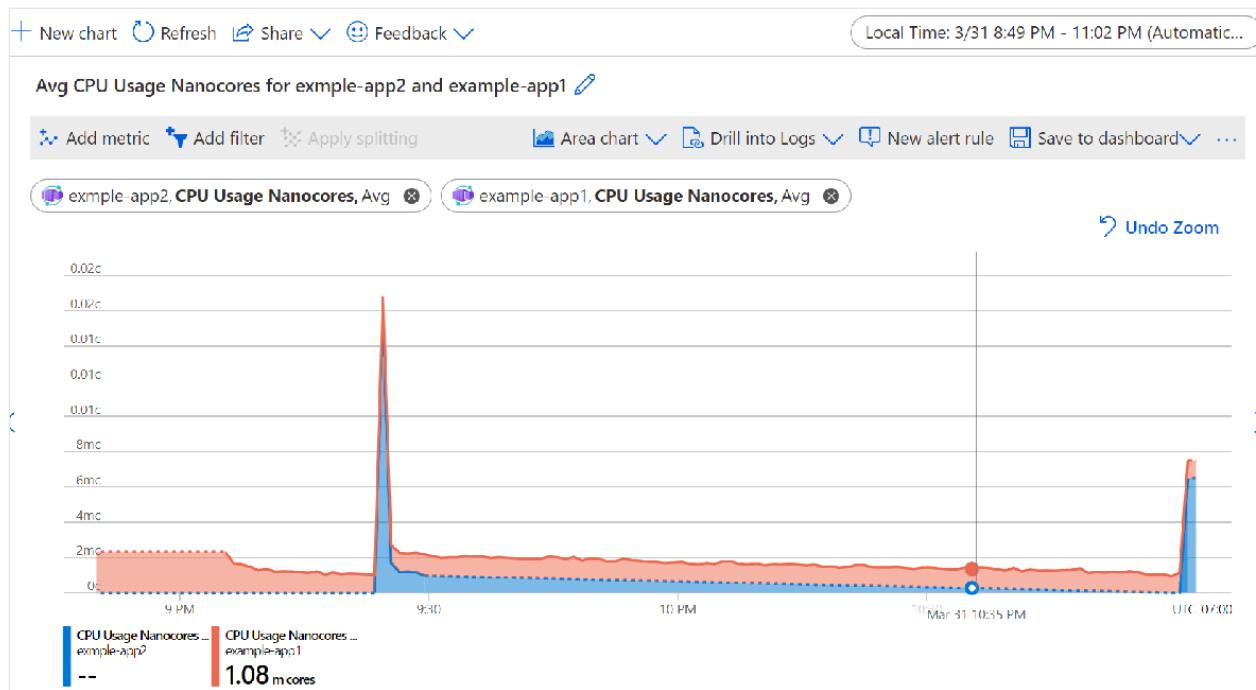
To split by revision or replica:

1. Select **Apply splitting**.
2. From the **Values** drop-down list, select **Revision** or **Replica**.
3. You can set the limit of the number of revisions or replicas to display in the chart.
The default value is 10.
4. You can set sort order to **Ascending** or **Descending**. The default value is **Descending**.



Add scopes

You can add more scopes to view metrics across multiple container apps.



Set up alerts in Azure Container Apps

Feedback

Was this page helpful?

 Yes

No

Provide product feedback | Get help at Microsoft Q&A

Monitor logs in Azure Container Apps with Log Analytics

Article • 04/14/2023

Azure Container Apps is integrated with Azure Monitor Log Analytics to monitor and analyze your container app's logs. When selected as your log monitoring solution, your Container Apps environment includes a Log Analytics workspace that provides a common place to store the system and application log data from all container apps running in the environment.

Log entries are accessible by querying Log Analytics tables through the Azure portal or a command shell using the [Azure CLI](#).

There are two types of logs for Container Apps.

- Console logs, which are emitted by your app.
- System logs, which are emitted by the Container Apps service.

System Logs

The Container Apps service provides system log messages at the container app level. System logs emit the following messages:

Source	Type	Message
Dapr	Info	Successfully created dapr component <component-name> with scope <dapr-component-scope>
Dapr	Info	Successfully updated dapr component <component-name> with scope <component-type>
Dapr	Error	Error creating dapr component <component-name>
Volume Mounts	Info	Successfully mounted volume <volume-name> for revision <revision-scope>
Volume Mounts	Error	Error mounting volume <volume-name>
Domain Binding	Info	Successfully bound Domain <domain> to the container app <container app name>
Authentication	Info	Auth enabled on app. Creating authentication config

Source	Type	Message
Authentication	Info	Auth config created successfully
Traffic weight	Info	Setting a traffic weight of <percentage>% for revision <revision-name>
Revision Provisioning	Info	Creating a new revision: <revision-name>
Revision Provisioning	Info	Successfully provisioned revision <name>
Revision Provisioning	Info	Deactivating Old revisions since 'ActiveRevisionsMode=Single'
Revision Provisioning	Error	Error provisioning revision <revision-name>. ErrorCode: <[ErrImagePull] [Timeout] [ContainerCrashing]>

The system log data is accessible by querying the `ContainerAppSystemLogs_CL` table. The most used Container Apps specific columns in the table are:

Column	Description
<code>ContainerAppName_s</code>	Container app name
<code>EnvironmentName_s</code>	Container Apps environment name
<code>Log_s</code>	Log message
<code>RevisionName_s</code>	Revision name

Console Logs

Console logs originate from the `stderr` and `stdout` messages from the containers in your container app and Dapr sidecars. You can view console logs by querying the `ContainerAppConsoleLogs_CL` table.

Tip

Instrumenting your code with well-defined log messages can help you to understand how your code is performing and to debug issues. To learn more about best practices refer to [Design for operations](#).

The most commonly used Container Apps specific columns in `ContainerAppConsoleLogs_CL` include:

Column	Description
ContainerAppName_s	Container app name
ContainerGroupName_g	Replica name
ContainerId_s	Container identifier
ContainerImage_s	Container image name
EnvironmentName_s	Container Apps environment name
Log_s	Log message
RevisionName_s	Revision name

Query Log with Log Analytics

Log Analytics is a tool in the Azure portal that you can use to view and analyze log data. Using Log Analytics, you can write Kusto queries and then sort, filter, and visualize the results in charts to spot trends and identify issues. You can work interactively with the query results or use them with other features such as alerts, dashboards, and workbooks.

Azure portal

Start Log Analytics from **Logs** in the sidebar menu on your container app page. You can also start Log Analytics from **Monitor>Logs**.

You can query the logs using the tables listed in the **CustomLogs** category **Tables** tab. The tables in this category are the `ContainerAppSystemlogs_CL` and `ContainerAppConsoleLogs_CL` tables.

New Query 1*

workspacecebca9ba3 Select scope

Tables Queries Functions ...

Search Filter Group by: Solution

Collapse all

Favorites

You can add favorites by clicking on the icon

▲ LogManagement

▶ Usage

▲ Custom Logs

▶ ContainerAppConsoleLogs_CL

▶ ContainerAppSystemLogs_CL

Below is a Kusto query that displays console log entries for the container app named *album-api*.

Kusto

```
ContainerAppConsoleLogs_CL
| where ContainerAppName_s == 'album-api'
| project Time=TimeGenerated, AppName=ContainerAppName_s,
Revision=RevisionName_s, Container=ContainerName_s, Message=Log_s
| take 100
```

Below is a Kusto query that displays system log entries for the container app named *album-api*.

Kusto

```
ContainerAppSystemLogs_CL
| where ContainerAppName_s == 'album-api'
| project Time=TimeGenerated, EnvName=EnvironmentName_s,
AppName=ContainerAppName_s, Revision=RevisionName_s, Message=Log_s
| take 100
```

For more information regarding Log Analytics and log queries, see the [Log Analytics tutorial](#).

Azure CLI/PowerShell

Container Apps logs can be queried using the [Azure CLI](#).

These example Azure CLI queries output a table containing log records for the container app name `album-api`. The table columns are specified by the parameters after the `project` operator. The `$WORKSPACE_CUSTOMER_ID` variable contains the GUID of the Log Analytics workspace.

This example queries the `ContainerAppConsoleLogs_CL` table:

Bash

Azure CLI

```
az monitor log-analytics query --workspace $WORKSPACE_CUSTOMER_ID --  
analytics-query "ContainerAppConsoleLogs_CL | where ContainerAppName_s  
== 'album-api' | project Time=TimeGenerated, AppName=ContainerAppName_s,  
Revision=RevisionName_s, Container=ContainerName_s, Message=Log_s,  
LogLevel_s | take 5" --out table
```

This example queries the `ContainerAppSystemLogs_CL` table:

Bash

Azure CLI

```
az monitor log-analytics query --workspace $WORKSPACE_CUSTOMER_ID --  
analytics-query "ContainerAppSystemLogs_CL | where ContainerAppName_s ==  
'album-api' | project Time=TimeGenerated, AppName=ContainerAppName_s,  
Revision=RevisionName_s, Message=Log_s, LogLevel_s | take 5" --out table
```

Next steps

[View log streams from the Azure portal](#)

Set up alerts in Azure Container Apps

Article • 04/14/2023

Azure Monitor alerts notify you so that you can respond quickly to critical issues. There are two types of alerts that you can define:

- [Metric alerts](#) based on Azure Monitor metric data
- [Log alerts](#) based on Azure Monitor Log Analytics data

You can create alert rules from metric charts in the metric explorer and from queries in Log Analytics. You can also define and manage alerts from the [Monitor>Alerts](#) page. To learn more about alerts, refer to [Overview of alerts in Microsoft Azure](#).

The **Alerts** page in the **Monitoring** section on your container app page displays all of your app's alerts. You can filter the list by alert type, resource, time and severity. You can also modify and create new alert rules from this page.

Create metric alert rules

When you create alerts rules based on a metric chart in the metrics explorer, alerts are triggered when the metric data matches alert rule conditions. For more information about creating metrics charts, see [Using metrics explorer](#)

After creating a metric chart, you can create a new alert rule.

1. Select **New alert rule**. The [Create an alert rule](#) page is opened to the **Condition** tab. Here you'll find a *condition* that is populated with the metric chart settings.
2. Select the condition.

The screenshot shows the 'Create an alert rule' page with the 'Condition' tab selected. The page header includes 'Home > my-container-app | Metrics > Create an alert rule ...'. Below the tabs, there is a note: 'Configure when the alert rule should trigger by selecting a signal and defining its logic.' A table lists a single condition: 'Whenever the average usagenanocores is greater than <logic undefined>'. The table columns are 'Condition name', 'Time series monitored', and 'Estimated monthly cost (USD)'. The total cost is listed as '\$ 0.10'.

Condition name	Time series monitored	Estimated monthly cost (USD)
Whenever the average usagenanocores is greater than <logic undefined>	1	\$ 0.10
	1	Total \$ 0.10

[Add condition](#)

3. Modify the **Alert logic** section to set the alert criteria. You can set the alert to trigger when the metric value is greater than, less than, or equal to a threshold value. You can also set the alert to trigger when the metric value is outside of a

range of values.

Configure signal logic

Chart period ⓘ
Over the last 6 hours

CPU Usage (Sum)
my-container-app
9.19 cores

Alert logic

Threshold ⓘ
Static Dynamic

Operator ⓘ
Greater than

Aggregation type * ⓘ
Total

Threshold value * ⓘ
10 ✓

Condition preview
Whenever the total cpu usage is greater than 10

Evaluated based on

Aggregation granularity (Period) * ⓘ
5 minutes

Frequency of evaluation ⓘ
Every 5 Minutes

Done

4. Select **Done**.
5. You can add more conditions to the alert rule by selecting **Add condition** on the **Create an alert rule** page.
6. Select the **Details** tab.
7. Enter a name and description for the alert rule.
8. Select **Review + create**.

9. Select Create.

The screenshot shows the 'Create an alert rule' dialog box. At the top, there are tabs for Scope, Condition, Actions, Details (which is underlined), Tags, and Review + create. Below the tabs, the 'Project details' section is shown, where you can select a subscription and resource group. The 'Alert rule details' section follows, which includes fields for Severity (set to 3 - Informational), Alert rule name (set to 'maximum replica alert'), and Alert rule description (set to 'Alert when there are more than 10 replicas'). A red box highlights the alert rule name and description fields. At the bottom of the dialog, there are buttons for Review + create, Previous, and Next: Tags >.

Add conditions to an alert rule

To add more conditions to your alert rule:

1. Select **Alerts** from the left side menu of your container app page.
2. Select **Alert rules** from the top menu.
3. Select an alert from the table.
4. Select **Add condition** in the **Condition** section.

5. Select from the metrics listed in the **Select a signal** pane.

The screenshot shows the 'Select a signal' pane with the following interface elements:

- Signal type:** Metrics (selected)
- Monitor service:** All
- Displaying:** 1 - 7 signals out of total 7 signals
- Search by signal name:** (empty)
- Table:** A list of 7 metrics:

Signal name	Signal type	Monitor service
Replica Count	Metrics	Platform
Requests	Metrics	Platform
Replica Restart Count	Metrics	Platform
Network In Bytes	Metrics	Platform
Network Out Bytes	Metrics	Platform
CPU Usage Nanocores	Metrics	Platform
Memory Working Set Bytes	Metrics	Platform
- Buttons:** Done

6. Configure the settings for your alert condition. For more information about configuring alerts, see [Manage metric alerts](#).

You can receive individual alerts for specific revisions or replicas by enabling alert splitting and selecting **Revision** or **Replica** from the Dimension name list.

Example of selecting a dimension to split an alert.

The screenshot shows the 'Configure signal logic' pane with the following interface elements:

- Alert name:** 7.92 MB
- Split by dimensions:**
 - Dimension name:** Replica (selected)
 - Operator:** =
 - Dimension values:** A dropdown menu with a search bar "Type to start filtering..." and a list of values:
 - containerapp2--v2-8449f5bff5-fgvm5
 - containerapp2--v2-8449f5bff5-hh5xg
 - containerapp2--v2-8449f5bff5-jj29j
 - containerapp2--v2-8449f5bff5-tksz6
 - containerapp2--v3-98c586c96-fqq8n
 - containerapp2--v3-98c586c96-kl9kr
 - containerapp2--v3-98c586c96-n2fc6
 - Add custom value:** (button)
 - Select dimension:** (dropdown)
 - Alert logic:** (checkbox)
 - Threshold:** Static (selected) / Dynamic
 - Information:** We currently support alert rules with D

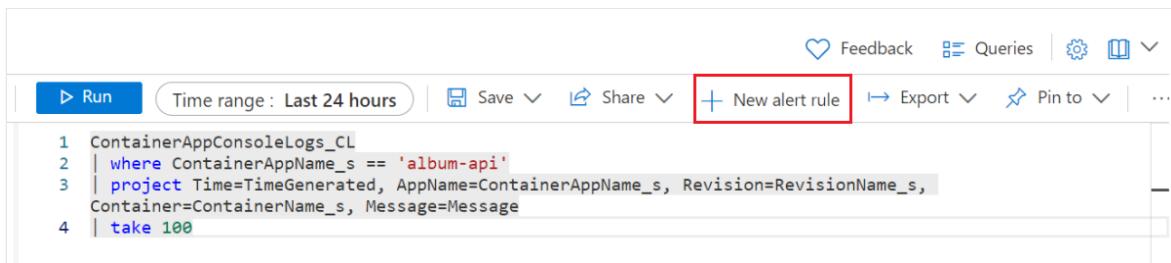
To learn more about configuring alerts, visit [Create a metric alert for an Azure resource](#)

Create log alert rules

You can create log alerts from queries in Log Analytics. When you create an alert rule from a query, the query is run at set intervals triggering alerts when the log data matches the alert rule conditions. To learn more about creating log alert rules, see [Manage log alerts](#).

To create an alert rule:

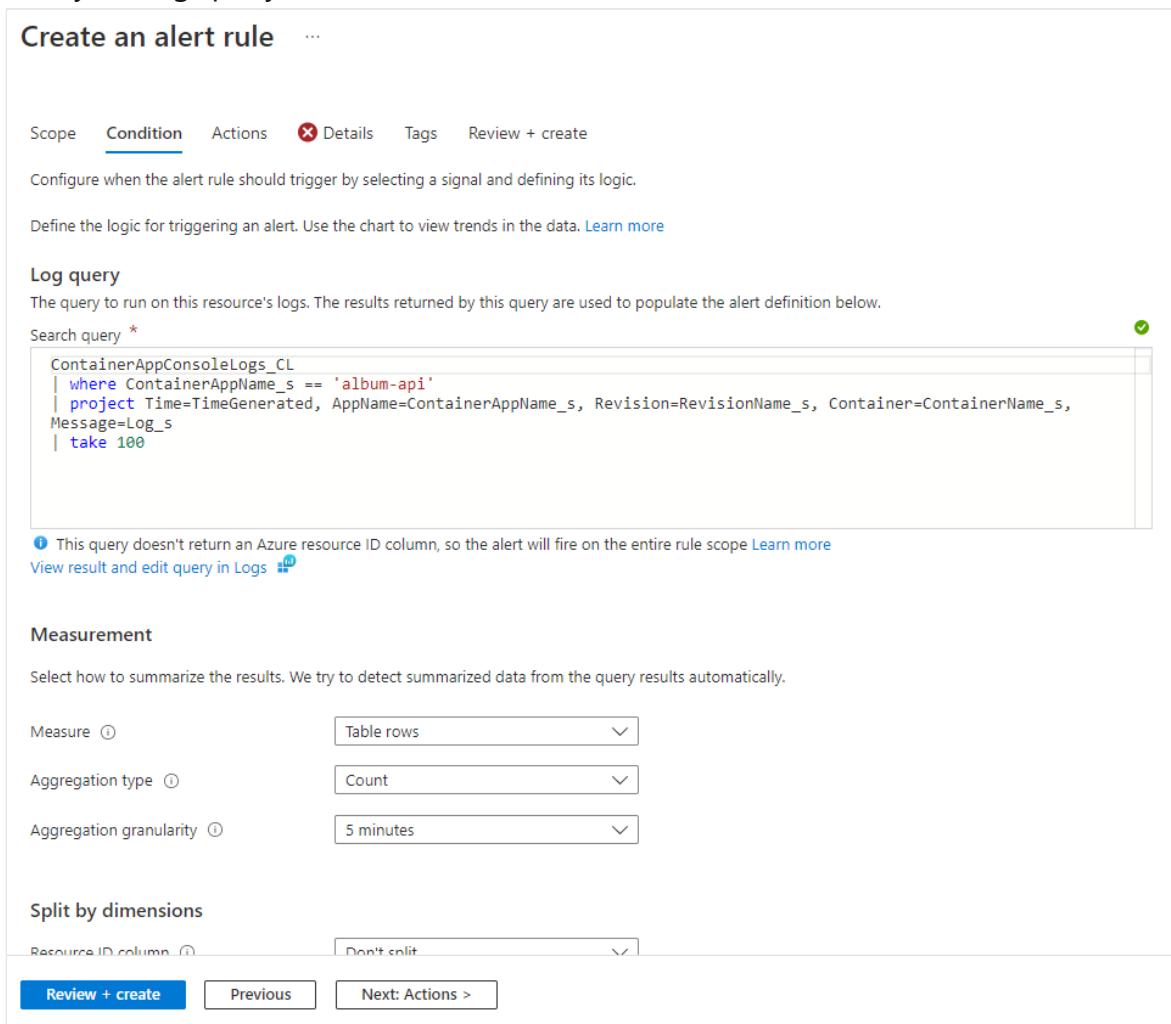
1. First, create and run a query to validate the query.
2. Select **New alert rule**.



A screenshot of the Microsoft Azure Log Analytics workspace. At the top, there's a toolbar with various icons like 'Feedback', 'Queries', 'Save', 'Share', 'Export', 'Pin to', and a gear icon. A red box highlights the '+ New alert rule' button. Below the toolbar, there's a search bar with 'Time range : Last 24 hours'. The main area contains a code editor with the following Log Analytics query:

```
1 ContainerAppConsoleLogs_CL
2 | where ContainerAppName_s == 'album-api'
3 | project Time=TimeGenerated, AppName=ContainerAppName_s, Revision=RevisionName_s,
   Container=ContainerName_s, Message=Message
4 | take 100
```

3. The **Create an alert rule** editor is opened to the **Condition** tab, which is populated with your log query.



The screenshot shows the 'Create an alert rule' editor with the 'Condition' tab selected. The 'Log query' section contains the same Log Analytics query as the previous screenshot. Below it, the 'Measurement' section has three dropdown menus: 'Measure' (set to 'Table rows'), 'Aggregation type' (set to 'Count'), and 'Aggregation granularity' (set to '5 minutes'). The 'Split by dimensions' section has a dropdown menu set to 'Don't split'. At the bottom, there are navigation buttons: 'Review + create', 'Previous', and 'Next: Actions >'. A small note at the bottom left says: 'This query doesn't return an Azure resource ID column, so the alert will fire on the entire rule scope' with a 'Learn more' link.

4. Configure the settings in the **Measurement** section

Measurement

Select how to summarize the results. We try to detect summarized data from the query results automatically.

Measure <small>i</small>	Table rows
Aggregation type <small>i</small>	Count
Aggregation granularity <small>i</small>	5 minutes

5. Optionally, you can enable alert splitting in the alert rule to send individual alerts for each dimension you select in the **Split by dimensions** section of the editor.

Split by dimensions

Resource ID column i Don't split

Use dimensions to monitor specific time series and provide context to the fired alert. Dimensions can be either number or string columns. If you select more than one dimension value, each time series that results from the combination will trigger its own alert and will be charged separately. i

Dimension name	Operator	Dimension values	Add custom value
AppName	=	sample-app1	
Revision	=	sample-app1--v2	
Container	=	simple-hello-world-container	
Message	=	All current and future values	

6. Enter the threshold criteria in the **Alert logic** section.

Alert logic

Operator * i Greater than

Threshold value * i

Frequency of evaluation * i 5 minutes

7. Select the **Details** tab.

8. Enter a name and description for the alert rule.

The screenshot shows the 'Create an alert rule' wizard with the 'Details' tab selected. In the 'Project details' section, the subscription is set to 'Vendor learning support subscription' and the resource group is 'my-container-apps'. In the 'Alert rule details' section, the severity is '3 - Informational', the alert rule name is 'maximum replica alert', and the description is 'Alert when there are more than 10 replicas'. A red box highlights the 'Alert rule name' and 'Description' fields.

Subscription * ⓘ Vendor learning support subscription

Resource group * ⓘ my-container-apps [Create new](#)

Severity * ⓘ 3 - Informational

Alert rule name * ⓘ maximum replica alert

Alert rule description ⓘ Alert when there are more than 10 replicas

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

9. Select **Review + create**.

10. Select **Create**.

[View log streams from the Azure portal](#)

Collect and read OpenTelemetry data in Azure Container Apps (preview)

Article • 08/22/2024

Using an [OpenTelemetry](#) data agent with your Azure Container Apps environment, you can choose to send observability data in an OpenTelemetry format by:

- Piping data from an agent into a desired endpoint. Destination options include Azure Monitor Application Insights, Datadog, and any OpenTelemetry Protocol (OTLP)-compatible endpoint.
- Easily changing destination endpoints without having to reconfigure how they emit data, and without having to manually run an OpenTelemetry agent.

This article shows you how to set up and configure an OpenTelemetry agent for your container app.

Configure an OpenTelemetry agent

OpenTelemetry agents live within your container app environment. You configure agent settings via an ARM template or Bicep calls to the environment, or through the CLI.

Each endpoint type (Azure Monitor Application Insights, DataDog, and OTLP) has specific configuration requirements.

Prerequisites

Enabling the managed OpenTelemetry agent to your environment doesn't automatically mean the agent collects data. Agents only send data based on your configuration settings and instrumenting your code correctly.

Configure source code

Prepare your application to collect data by installing the [OpenTelemetry SDK](#) and follow the OpenTelemetry guidelines to instrument [metrics](#), [logs](#), or [traces](#).

Initialize endpoints

Before you can send data to a collection destination, you first need to create an instance of the destination service. For example, if you want to send data to Azure Monitor Application Insights, you need to create an Application Insights instance ahead of time.

The managed OpenTelemetry agent accepts the following destinations:

- Azure Monitor Application Insights
- Datadog
- Any OTLP endpoint (For example: New Relic or Honeycomb)

The following table shows you what type of data you can send to each destination:

 [Expand table](#)

Destination	Logs	Metrics	Traces
Azure App Insights	Yes	No	Yes
Datadog ↗	No	Yes	Yes
OpenTelemetry ↗ protocol (OTLP) configured endpoint	Yes	Yes	Yes

Azure Monitor Application Insights

The only configuration detail required from Application Insights is the connection string. Once you have the connection string, you can configure the agent via your container app's ARM template or with Azure CLI commands.

ARM template

Before you deploy this template, replace placeholders surrounded by `<>` with your values.

JSON

```
{  
  ...  
  "properties": {  
    "appInsightsConfiguration": {  
      "connectionString": "<YOUR_APP_INSIGHTS_CONNECTION_STRING>"  
    }  
    "openTelemetryConfiguration": {  
      ...  
      "tracesConfiguration": {  
        "destinations": ["appInsights"]  
      },  
    },  
  },  
}
```

```
        "logsConfiguration": {
            "destinations": [ "appInsights" ]
        }
    }
}
```

Datadog

The Datadog agent configuration requires a value for `site` and `key` from your Datadog instance. Gather these values from your Datadog instance according to this table:

[] [Expand table](#)

Datadog agent property	Container Apps configuration property
<code>DD_SITE</code>	<code>site</code>
<code>DD_API_KEY</code>	<code>key</code>

Once you have these configuration details, you can configure the agent via your container app's ARM template or with Azure CLI commands.

ARM template

Before you deploy this template, replace placeholders surrounded by `<>` with your values.

JSON

```
{
...
"properties": {
...
"openTelemetryConfiguration": {
...
"destinationsConfiguration":{
...
"dataDogConfiguration":{
"site": "<YOUR_DATADOG_SUBDOMAIN>.datadoghq.com",
"key": "<YOUR_DATADOG_KEY>"
}
},
"tracesConfiguration":{
"destinations": [ "dataDog" ]
},
"metricsConfiguration": {
```

```
        "destinations": [ "dataDog" ]
    }
}
}
```

OTLP endpoint

An OpenTelemetry protocol (OTLP) endpoint is a telemetry data destination that consumes OpenTelemetry data. In your application configuration, you can add multiple OTLP endpoints. The following example adds two endpoints and sends the following data to these endpoints.

[Expand table](#)

Endpoint name	Data sent to endpoint
otlp1	Metrics and/or traces
otlp2	Logs and/or traces

While you can set up as many OTLP-configured endpoints as you like, each endpoint must have a distinct name.

ARM template

JSON

```
{
  "properties": {
    "appInsightsConfiguration": {},
    "openTelemetryConfiguration": {
      "destinationsConfiguration": {
        "otlpConfigurations": [
          {
            "name": "otlp1",
            "endpoint": "ENDPOINT_URL_1",
            "insecure": false,
            "headers": "api-key-1=key"
          },
          {
            "name": "otlp2",
            "endpoint": "ENDPOINT_URL_2",
            "insecure": true
          }
        ]
      }
    }
  }
},
```

```

    "logsConfiguration": {
      "destinations": ["otlp2"]
    },
    "tracesConfiguration":{
      "destinations": ["otlp1", "otlp2"]
    },
    "metricsConfiguration": {
      "destinations": ["otlp1"]
    }
  }
}

```

[+] Expand table

Name	Description
<code>resource-group</code>	Name of resource group. You can configure the default group using <code>az configure --defaults group=<NAME></code> .
<code>name</code>	Name of the Container Apps environment.
<code>otlp-name</code>	A name you select to identify your OTLP-configured endpoint.
<code>endpoint</code>	The URL of the destination that receives collected data.
<code>insecure</code>	Default true. Defines whether to enable client transport security for the exporter's gRPC connection. If false, the <code>headers</code> parameter is required.
<code>headers</code>	Space-separated values, in 'key=value' format, that provide required information for the OTLP endpoints' security. Example: " <code>api-key=key other-config-value=value</code> ".

Configure Data Destinations

To configure an agent, use the `destinations` array to define which agents your application sends data. Valid keys are either `appInsights`, `dataDog`, or the name of your custom OTLP endpoint. You can control how an agent behaves based off data type and endpoint-related options.

By data type

[+] Expand table

Option	Example
Select a data type.	You can configure logs, metrics, and/or traces individually.
Enable or disable any data type.	You can choose to send only traces and no other data.
Send one data type to multiple endpoints.	You can send logs to both DataDog and an OTLP-configured endpoint.
Send different data types to different locations.	You can send traces to an OTLP endpoint and metrics to DataDog.
Disable sending all data types.	You can choose to not send any data through the OpenTelemetry agent.

By endpoint

- You can only set up one Application Insights and Datadog endpoint each at a time.
- While you can define more than one OTLP-configured endpoint, each one must have a distinct name.

The following example shows how to use an OTLP endpoint named `customDashboard`. It sends:

- traces to app insights and `customDashboard`
- logs to app insights and `customDashboard`
- metrics to DataDog and `customDashboard`

JSON

```
{
  ...
  "properties": {
    ...
    "openTelemetryConfiguration": {
      ...
      "tracesConfiguration": {
        "destinations": [
          "appInsights",
          "customDashboard"
        ]
      },
      "logsConfiguration": {
        "destinations": [
          "appInsights",
          "customDashboard"
        ]
      }
    }
  }
}
```

```

    "metricsConfiguration": {
        "destinations": [
            "dataDog",
            "customDashboard"
        ]
    }
}
}

## Example OpenTelemetry configuration

```

The following example ARM template shows how you might configure your container app to collect telemetry data using Azure Monitor Application Insights, Datadog, and with a custom OTLP agent named `customDashboard`.

Before you deploy this template, replace placeholders surrounded by `<>` with your values.

```

```json
{
 "location": "eastus",
 "properties": {
 "appInsightsConfiguration": {
 "connectionString": "<APP_INSIGHTS_CONNECTION_STRING>"
 },
 "openTelemetryConfiguration": {
 "destinationsConfiguration": {
 "dataDogConfiguration": {
 "site": "datadoghq.com",
 "key": "<YOUR_DATADOG_KEY>"
 },
 "otlpConfigurations": [
 {
 "name": "customDashboard",
 "endpoint": "<OTLP_ENDPOINT_URL>",
 "insecure": true
 }
]
 },
 "tracesConfiguration": {
 "destinations": [
 "appInsights",
 "customDashboard"
]
 },
 "logsConfiguration": {
 "destinations": [
 "appInsights",
 "customDashboard"
]
 },
 "metricsConfiguration": {
 "destinations": [
 "dataDog",

```

```
 "customDashboard"
]
}
}
}
```

## Environment variables

The OpenTelemetry agent automatically injects a set of environment variables into your application at runtime.

The first two environment variables follow standard OpenTelemetry exporter configuration and are used in OTLP standard software development kits. If you explicitly set the environment variable in the container app specification, your value overwrites the automatically injected value.

Learn about the OTLP exporter configuration see, [OTLP Exporter Configuration](#).

[+] [Expand table](#)

Name	Description
OTEL_EXPORTER_OTLP_ENDPOINT	A base endpoint URL for any signal type, with an optionally specified port number. This setting is helpful when you're sending more than one signal to the same endpoint and want one environment variable to control the endpoint. Example: <code>http://otel.service.k8se-apps:4317/</code>
OTEL_EXPORTER_OTLP_PROTOCOL	Specifies the OTLP transport protocol used for all telemetry data. The managed agent only supports <code>grpc</code> . Value: <code>grpc</code> .

The other three environment variables are specific to Azure Container Apps, and are always injected. These variables hold agent's endpoint URLs for each specific data type (logs, metrics, traces).

These variables are only necessary if you're using both the managed OpenTelemetry agent and another OpenTelemetry agent. Using these variables gives you control over how to route data between the different OpenTelemetry agents.

[+] [Expand table](#)

Name	Description	Example
CONTAINERAPP_OTEL_TRACING_GRPC_ENDPOINT	Endpoint URL for trace data only.	<code>http://otel.service.k8se-apps:43178/v1/traces/</code>
CONTAINERAPP_OTEL_LOGGING_GRPC_ENDPOINT	Endpoint URL for log data only.	<code>http://otel.service.k8se-apps:43178/v1/logs/</code>
CONTAINERAPP_OTEL_METRIC_GRPC_ENDPOINT	Endpoint URL for metric data only.	<code>http://otel.service.k8se-apps:43178/v1/metrics/</code>

## OpenTelemetry agent costs

You are [billed](#) for the underlying compute of the agent.

See the destination service for their billing structure and terms. For example, if you send data to both Azure Monitor Application Insights and Datadog, you're responsible for the charges applied by both services.

## Known limitations

- OpenTelemetry agents are in preview.
- System data, such as system logs or Container Apps standard metrics, isn't available to be sent to the OpenTelemetry agent.
- The Application Insights endpoint doesn't accept metrics.
- The Datadog endpoint doesn't accept logs.

## Next steps

[Learn about monitoring and health](#)

## Feedback

Was this page helpful?

Yes

No

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

# Set scaling rules in Azure Container Apps

Article • 06/28/2024

Azure Container Apps manages automatic horizontal scaling through a set of declarative scaling rules. As a container app revision scales out, new instances of the revision are created on-demand. These instances are known as replicas.

Adding or editing scaling rules creates a new revision of your container app. A revision is an immutable snapshot of your container app. To learn which types of changes trigger a new revision, see revision [change types](#).

[Event-driven Container Apps jobs](#) use scaling rules to trigger executions based on events.

## Scale definition

Scaling is the combination of limits, rules, and behavior.

- **Limits** define the minimum and maximum possible number of replicas per revision as your container app scales.

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

Scale limit	Default value	Min value	Max value
Minimum number of replicas per revision	0	0	Maximum replicas configurable are 1,000.
Maximum number of replicas per revision	10	1	Maximum replicas configurable are 1,000.

- **Rules** are the criteria used by Container Apps to decide when to add or remove replicas.

[Scale rules](#) are implemented as HTTP, TCP (Transmission Control Protocol), or custom.

- **Behavior** is the combination of rules and limits to determine scale decisions over time.

[Scale behavior](#) explains how scale decisions are made.

As you define your scaling rules, it's important to consider the following items:

- You aren't billed usage charges if your container app scales to zero.
- Replicas that aren't processing, but remain in memory might be billed at a lower "idle" rate. For more information, see [Billing](#).
- If you want to ensure that an instance of your revision is always running, set the minimum number of replicas to 1 or higher.

## Scale rules

Scaling is driven by three different categories of triggers:

- [HTTP](#): Based on the number of concurrent HTTP requests to your revision.
- [TCP](#): Based on the number of concurrent TCP connections to your revision.
- [Custom](#): Based on CPU, memory, or supported event-driven data sources such as:
  - Azure Service Bus
  - Azure Event Hubs
  - Apache Kafka
  - Redis

If you define more than one scale rule, the container app begins to scale once the first condition of any rules is met.

## HTTP

With an HTTP scaling rule, you have control over the threshold of concurrent HTTP requests that determines how your container app revision scales. Every 15 seconds, the number of concurrent requests is calculated as the number of requests in the past 15 seconds divided by 15. [Container Apps jobs](#) don't support HTTP scaling rules.

In the following example, the revision scales out up to five replicas and can scale in to zero. The scaling property is set to 100 concurrent requests per second.

## Example

Define an HTTP scale rule using the `--scale-rule-http-concurrency` parameter in the [create](#) or [update](#) commands.

[ ] [Expand table](#)

CLI parameter	Description	Default value	Min value	Max value
--scale-rule-type http --scale-rule-name <code>azure-http-rule</code>	When the number of concurrent HTTP requests exceeds this value, then another replica is added. Replicas continue to add to the pool up to the <code>max-replicas</code> amount.	10	1	n/a

#### Azure CLI

```
az containerapp create \
 --name <CONTAINER_APP_NAME> \
 --resource-group <RESOURCE_GROUP> \
 --environment <ENVIRONMENT_NAME> \
 --image <CONTAINER_IMAGE_LOCATION>
 --min-replicas 0 \
 --max-replicas 5 \
 --scale-rule-name azure-http-rule \
 --scale-rule-type http \
 --scale-rule-http-concurrency 100
```

## TCP

With a TCP scaling rule, you have control over the threshold of concurrent TCP connections that determines how your app scales. Every 15 seconds, the number of concurrent connections is calculated as the number of connections in the past 15 seconds divided by 15. [Container Apps jobs](#) don't support TCP scaling rules.

In the following example, the container app revision scales out up to five replicas and can scale in to zero. The scaling threshold is set to 100 concurrent connections per second.

## Example

Define a TCP scale rule using the `--scale-rule-tcp-concurrency` parameter in the [create](#) or [update](#) commands.

[+] [Expand table](#)

CLI parameter	Description	Default value	Min value	Max value
--scale-rule-tcp-concurrency	When the number of concurrent TCP connections exceeds this value, then another replica is added. Replicas continue to be added up to the <code>max-replicas</code> amount as the number of concurrent connections increase.	10	1	n/a

#### Azure CLI

```
az containerapp create \
--name <CONTAINER_APP_NAME> \
--resource-group <RESOURCE_GROUP> \
--environment <ENVIRONMENT_NAME> \
--image <CONTAINER_IMAGE_LOCATION>
--min-replicas 0 \
--max-replicas 5 \
--transport tcp \
--ingress <external/internal> \
--target-port <CONTAINER_TARGET_PORT> \
--scale-rule-name azure-tcp-rule \
--scale-rule-type tcp \
--scale-rule-tcp-concurrency 100
```

## Custom

You can create a custom Container Apps scaling rule based on any [ScaledObject](#)-based [KEDA scaler](#) with these defaults:

[\[ \]](#) Expand table

Defaults	Seconds
Polling interval	30
Cool down period	300

For [event-driven Container Apps jobs](#), you can create a custom scaling rule based on any [ScaledJob](#)-based KEDA scalers.

The following example demonstrates how to create a custom scale rule.

## Example

This example shows how to convert an [Azure Service Bus scaler](#) to a Container Apps scale rule, but you use the same process for any other [ScaledObject](#)-based [KEDA scaler](#) specification.

For authentication, KEDA scaler authentication parameters take [Container Apps secrets](#) or [managed identity](#).

1. From the KEDA scaler specification, find the `type` value.

```
yml

triggers:
- type: azure-servicebus
 metadata:
 queueName: my-queue
 namespace: service-bus-namespace
 messageCount: "5"
```

2. In the CLI command, set the `--scale-rule-type` parameter to the specification `type` value.

```
Bash

az containerapp create \
 --name <CONTAINER_APP_NAME> \
 --resource-group <RESOURCE_GROUP> \
 --environment <ENVIRONMENT_NAME> \
 --image <CONTAINER_IMAGE_LOCATION>
 --min-replicas 0 \
 --max-replicas 5 \
 --secrets "connection-string-secret=<SERVICE_BUS_CONNECTION_STRING>" \
 \
 --scale-rule-name azure-servicebus-queue-rule \
 --scale-rule-type azure-servicebus \
 --scale-rule-metadata "queueName=my-queue" \
 "namespace=service-bus-namespace" \
 "messageCount=5" \
 --scale-rule-auth "connection=connection-string-secret"
```

3. From the KEDA scaler specification, find the `metadata` values.

```
yml

triggers:
- type: azure-servicebus
 metadata:
 queueName: my-queue
 namespace: service-bus-namespace
```

```
messageCount: "5"
```

4. In the CLI command, set the `--scale-rule-metadata` parameter to the metadata values.

You need to transform the values from a YAML format to a key/value pair for use on the command line. Separate each key/value pair with a space.

Bash

```
az containerapp create \
--name <CONTAINER_APP_NAME> \
--resource-group <RESOURCE_GROUP> \
--environment <ENVIRONMENT_NAME> \
--image <CONTAINER_IMAGE_LOCATION>
--min-replicas 0 \
--max-replicas 5 \
--secrets "connection-string-secret=<SERVICE_BUS_CONNECTION_STRING>" \
 \
--scale-rule-name azure-servicebus-queue-rule \
--scale-rule-type azure-servicebus \
--scale-rule-metadata "queueName=my-queue" \
"namespace=service-bus-namespace" \
"messageCount=5" \
--scale-rule-auth "connection=connection-string-secret"
```

## Authentication

Container Apps scale rules support secrets-based authentication. Scale rules for Azure resources, including Azure Queue Storage, Azure Service Bus, and Azure Event Hubs, also support managed identity. Where possible, use managed identity authentication to avoid storing secrets within the app.

### Use secrets

To configure secrets-based authentication for a Container Apps scale rule, you configure the secrets in the container app and reference them in the scale rule.

A KEDA scaler supports secrets in a [TriggerAuthentication](#) which the `authenticationRef` property uses for reference. You can map the `TriggerAuthentication` object to the Container Apps scale rule.

1. Find the `TriggerAuthentication` object referenced by the KEDA `ScaledObject` specification. Identify each `secretTargetRef` of the `TriggerAuthentication` object.

```
yml
```

```
apiVersion: v1
kind: Secret
metadata:
 name: my-secrets
 namespace: my-project
type: Opaque
data:
 connection-string-secret: <SERVICE_BUS_CONNECTION_STRING>

apiVersion: keda.sh/v1alpha1
kind: TriggerAuthentication
metadata:
 name: azure-servicebus-auth
spec:
 secretTargetRef:
 - parameter: connection
 name: my-secrets
 key: connection-string-secret

apiVersion: keda.sh/v1alpha1
kind: ScaledObject
metadata:
 name: azure-servicebus-queue-rule
 namespace: default
spec:
 scaleTargetRef:
 name: my-scale-target
 triggers:
 - type: azure-servicebus
 metadata:
 queueName: my-queue
 namespace: service-bus-namespace
 messageCount: "5"
 authenticationRef:
 name: azure-servicebus-auth
```

2. In your container app, create the `secrets` that match the `secretTargetRef` properties.

3. In the CLI command, set parameters for each `secretTargetRef` entry.

- a. Create a secret entry with the `--secrets` parameter. If there are multiple secrets, separate them with a space.
- b. Create an authentication entry with the `--scale-rule-auth` parameter. If there are multiple entries, separate them with a space.

```
Bash
```

```

az containerapp create \
 --name <CONTAINER_APP_NAME> \
 --resource-group <RESOURCE_GROUP> \
 --environment <ENVIRONMENT_NAME> \
 --image <CONTAINER_IMAGE_LOCATION>
 --min-replicas 0 \
 --max-replicas 5 \
 --secrets "connection-string-secret=<SERVICE_BUS_CONNECTION_STRING>" \
 \
 --scale-rule-name azure-servicebus-queue-rule \
 --scale-rule-type azure-servicebus \
 --scale-rule-metadata "queueName=my-queue" \
 "namespace=service-bus-namespace" \
 "messageCount=5" \
 --scale-rule-auth "connection=connection-string-secret"

```

## Using managed identity

Container Apps scale rules can use managed identity to authenticate with Azure services. The following command creates a container app with a user-assigned managed identity and uses it to authenticate for an Azure Queue scaler.

Bash

```

az containerapp create \
 --resource-group <RESOURCE_GROUP> \
 --name <APP_NAME> \
 --environment <ENVIRONMENT_ID> \
 --user-assigned <USER_ASSIGNED_IDENTITY_ID> \
 --scale-rule-name azure-queue \
 --scale-rule-type azure-queue \
 --scale-rule-metadata "accountName=<AZURE_STORAGE_ACCOUNT_NAME>" \
 "queueName=queue1" "queueLength=1" \
 --scale-rule-identity <USER_ASSIGNED_IDENTITY_ID>

```

Replace placeholders with your values.

## Default scale rule

If you don't create a scale rule, the default scale rule is applied to your container app.

[Expand table](#)

Trigger	Min replicas	Max replicas
HTTP	0	10

## ⓘ Important

Make sure you create a scale rule or set `minReplicas` to 1 or more if you don't enable ingress. If ingress is disabled and you don't define a `minReplicas` or a custom scale rule, then your container app will scale to zero and have no way of starting back up.

# Scale behavior

Scaling behavior has the following defaults:

[\[\] Expand table](#)

Parameter	Value
Polling interval	30 seconds
Cool down period	300 seconds
Scale up stabilization window	0 seconds
Scale down stabilization window	300 seconds
Scale up step	1, 4, 100% of current
Scale down step	100% of current
Scaling algorithm	<pre>desiredReplicas = ceil(currentMetricValue / targetMetricValue)</pre>

- **Polling interval** is how frequently event sources are queried by KEDA. This value doesn't apply to HTTP and TCP scale rules.
- **Cool down period** is how long after the last event was observed before the application scales down to its minimum replica count.
- **Scale up stabilization window** is how long to wait before performing a scale up decision once scale up conditions were met.
- **Scale down stabilization window** is how long to wait before performing a scale down decision once scale down conditions were met.
- **Scale up step** is the rate new instances are added at. It starts with 1, 4, 8, 16, 32, ... up to the configured maximum replica count.
- **Scale down step** is the rate at which replicas are removed. By default 100% of replicas that need to shut down are removed.

- **Scaling algorithm** is the formula used to calculate the current desired number of replicas.

## Example

For the following scale rule:

```
JSON

{
 "minReplicas": 0,
 "maxReplicas": 20,
 "rules": [
 {
 "name": "azure-servicebus-queue-rule",
 "custom": {
 "type": "azure-servicebus",
 "metadata": {
 "queueName": "my-queue",
 "namespace": "service-bus-namespace",
 "messageCount": "5"
 }
 }
 }
]
}
```

As your app scales out, KEDA starts with an empty queue and performs the following steps:

1. Check `my-queue` every 30 seconds.
2. If the queue length equals 0, go back to (1).
3. If the queue length is > 0, scale the app to 1.
4. If the queue length is 50, calculate `desiredReplicas = ceil(50/5) = 10`.
5. Scale app to `min(maxReplicaCount, desiredReplicas, max(4, 2*currentReplicaCount))`
6. Go back to (1).

If the app was scaled to the maximum replica count of 20, scaling goes through the same previous steps. Scale down only happens if the condition was satisfied for 300 seconds (scale down stabilization window). Once the queue length is 0, KEDA waits for 300 seconds (cool down period) before scaling the app to 0.

## Considerations

- In "multiple revisions" mode, adding a new scale trigger creates a new revision of your application but your old revision remains available with the old scale rules.

Use the [Revision management](#) page to manage traffic allocations.

- No usage charges are incurred when an application scales to zero. For more pricing information, see [Billing in Azure Container Apps](#).
- You need to enable data protection for all .NET apps on Azure Container Apps. See [Deploying and scaling an ASP.NET Core app on Azure Container Apps](#) for details.

## Known limitations

- Vertical scaling isn't supported.
- Replica quantities are a target amount, not a guarantee.
- If you're using [Dapr actors](#) to manage states, you should keep in mind that scaling to zero isn't supported. Dapr uses virtual actors to manage asynchronous calls, which means their in-memory representation isn't tied to their identity or lifetime.

## Next steps

[Manage secrets](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#)

# Tutorial: Scale a container app

Article • 08/02/2024

Azure Container Apps manages automatic horizontal scaling through a set of declarative scaling rules. As a container app scales out, new instances of the container app are created on-demand. These instances are known as replicas.

In this tutorial, you add an HTTP scale rule to your container app and observe how your application scales.

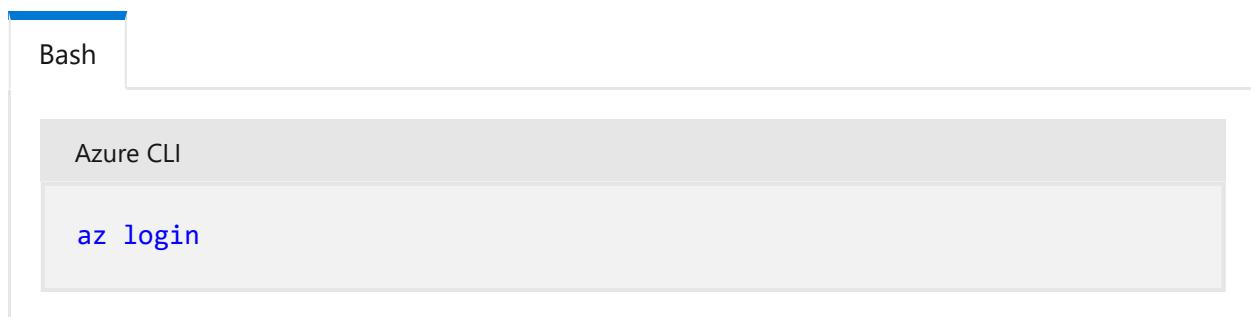
## Prerequisites

 Expand table

Requirement	Instructions
Azure account	If you don't have an Azure account, you can <a href="#">create one for free</a> .  You need the <i>Contributor</i> permission on the Azure subscription to proceed. Refer to <a href="#">Assign Azure roles using the Azure portal</a> for details.
GitHub Account	Get one for <a href="#">free</a> .
Azure CLI	Install the <a href="#">Azure CLI</a> .

## Setup

To sign in to Azure from the CLI, run the following command and follow the prompts to complete the authentication process.



The screenshot shows a terminal window with a blue header bar. The first tab is labeled "Bash". Below it, there is a grey tab labeled "Azure CLI". In the main terminal area, the command `az login` is typed in blue text. The background of the terminal is white.

To ensure you're running the latest version of the CLI, run the upgrade command.



The screenshot shows a terminal window with a blue header bar. The first tab is labeled "Bash". Below it, there is a grey tab. The background of the terminal is white.

Azure CLI

```
az upgrade
```

Next, install or update the Azure Container Apps extension for the CLI.

If you receive errors about missing parameters when you run `az containerapp` commands in Azure CLI or cmdlets from the `Az.App` module in Azure PowerShell, be sure you have the latest version of the Azure Container Apps extension installed.

Bash

Azure CLI

```
az extension add --name containerapp --upgrade
```

#### ⓘ Note

Starting in May 2024, Azure CLI extensions no longer enable preview features by default. To access Container Apps [preview features](#), install the Container Apps extension with `--allow-preview true`.

Azure CLI

```
az extension add --name containerapp --upgrade --allow-preview true
```

Now that the current extension or module is installed, register the `Microsoft.App` and `Microsoft.OperationalInsights` namespaces.

Bash

Azure CLI

```
az provider register --namespace Microsoft.App
```

Azure CLI

```
az provider register --namespace Microsoft.OperationalInsights
```

# Create and deploy the container app

Create and deploy your container app with the `containerapp up` command. This command creates a:

- Resource group
- Container Apps environment
- Log Analytics workspace

If any of these resources already exist, the command uses the existing resources rather than creating new ones.

Lastly, the command creates and deploys the container app using a public container image.

Bash

Azure CLI

```
az containerapp up \
--name my-container-app \
--resource-group my-container-apps \
--location centralus \
--environment 'my-container-apps' \
--image mcr.microsoft.com/k8se/quickstart:latest \
--target-port 8080 \
--ingress external \
--query properties.configuration.ingress.fqdn \
```

## ⓘ Note

Make sure the value for the `--image` parameter is in lower case.

By setting `--ingress` to `external`, you make the container app available to public requests.

The `up` command returns the fully qualified domain name (FQDN) for the container app. Copy this FQDN to a text file. You'll use it in the [Send requests](#) section. Your FQDN looks like the following example:

text

`https://my-container-app.icydune-96848328.centralus.azurecontainerapps.io`

# Add scale rule

Add an HTTP scale rule to your container app by running the `az containerapp update` command.

Bash

Azure CLI

```
az containerapp update \
--name my-container-app \
--resource-group my-container-apps \
--scale-rule-name my-http-scale-rule \
--scale-rule-http-concurrency 1
```

This command adds an HTTP scale rule to your container app with the name `my-http-scale-rule` and a concurrency setting of `1`. If your app receives more than one concurrent HTTP request, the runtime creates replicas of your app to handle the requests.

The `update` command returns the new configuration as a JSON response to verify your request was a success.

## Start log output

You can observe the effects of your application scaling by viewing the logs generated by the Container Apps runtime. Use the `az containerapp logs show` command to start listening for log entries.

Bash

Azure CLI

```
az containerapp logs show \
--name my-container-app \
--resource-group my-container-apps \
--type=system \
--follow=true
```

The `show` command returns entries from the system logs for your container app in real time. You can expect a response like the following example:

## JSON

```
{
 "TimeStamp": "2023-08-01T16:49:03.02752",
 "Log": "Connecting to the container 'my-container-app'..."
}
{
 "TimeStamp": "2023-08-01T16:49:03.04437",
 "Log": "Successfully Connected to container:
 'my-container-app' [Revision: 'my-container-app--9uj51l6',
 Replica: 'my-container-app--9uj51l6-5f96557ffb-5khg9']"
}
{
 "TimeStamp": "2023-08-01T16:47:31.9480811+00:00",
 "Log": "Microsoft.Hosting.Lifetime[14]"
}
{
 "TimeStamp": "2023-08-01T16:47:31.9481264+00:00",
 "Log": "Now listening on: http://[::]:8080"
}
{
 "TimeStamp": "2023-08-01T16:47:31.9490917+00:00",
 "Log": "Microsoft.Hosting.Lifetime[0]"
}
{
 "TimeStamp": "2023-08-01T16:47:31.9491036+00:00",
 "Log": "Application started. Press Ctrl+C to shut down."
}
{
 "TimeStamp": "2023-08-01T16:47:31.949723+00:00",
 "Log": "Microsoft.Hosting.Lifetime[0]"
}
{
 "TimeStamp": "2023-08-01T16:47:31.9497292+00:00",
 "Log": "Hosting environment: Production"
}
{
 "TimeStamp": "2023-08-01T16:47:31.9497325+00:00",
 "Log": "Microsoft.Hosting.Lifetime[0]"
}
{
 "TimeStamp": "2023-08-01T16:47:31.9497367+00:00",
 "Log": "Content root path: /app/"
}
```

For more information, see [az containerapp logs](#).

## Send requests

Bash

Open a new bash shell. Run the following command, replacing <YOUR\_CONTAINER\_APP\_FQDN> with the fully qualified domain name for your container app that you saved from the [Create and deploy the container app](#) section.

Bash

```
seq 1 50 | xargs -Iname -P10 curl "<YOUR_CONTAINER_APP_FQDN>"
```

These commands send 50 requests to your container app in concurrent batches of 10 requests each.

[Expand table](#)

Command or argument	Description
seq 1 50	Generates a sequence of numbers from 1 to 50.
	The pipe operator sends the sequence to the <code>xargs</code> command.
xargs	Runs <code>curl</code> with the specified URL
-Iname	Acts as a placeholder for the output of <code>seq</code> . This argument prevents the return value from being sent to the <code>curl</code> command.
curl	Calls the given URL.
-P10	Instructs <code>xargs</code> to run up to 10 processes at a time.

For more information, see the documentation for:

- [seq ↗](#)
- [xargs ↗](#)
- [curl ↗](#)

In the first shell, where you ran the `az containerapp logs show` command, the output now contains one or more log entries like the following.

JSON

```
{
 "TimeStamp": "2023-08-01 18:09:52 +0000 UTC",
 "Type": "Normal",
 "ContainerAppName": "my-container-app",
 "RevisionName": "my-container-app--9uj51l6",
```

```

 "ReplicaName": "my-container-app--9uj51l6-5f96557ffb-f795d",
 "Msg": "Replica 'my-container-app--9uj51l6-5f96557ffb-f795d' has been
scheduled to run on a node.",
 "Reason": "AssigningReplica",
 "EventSource": "ContainerAppController",
 "Count": 0
}

```

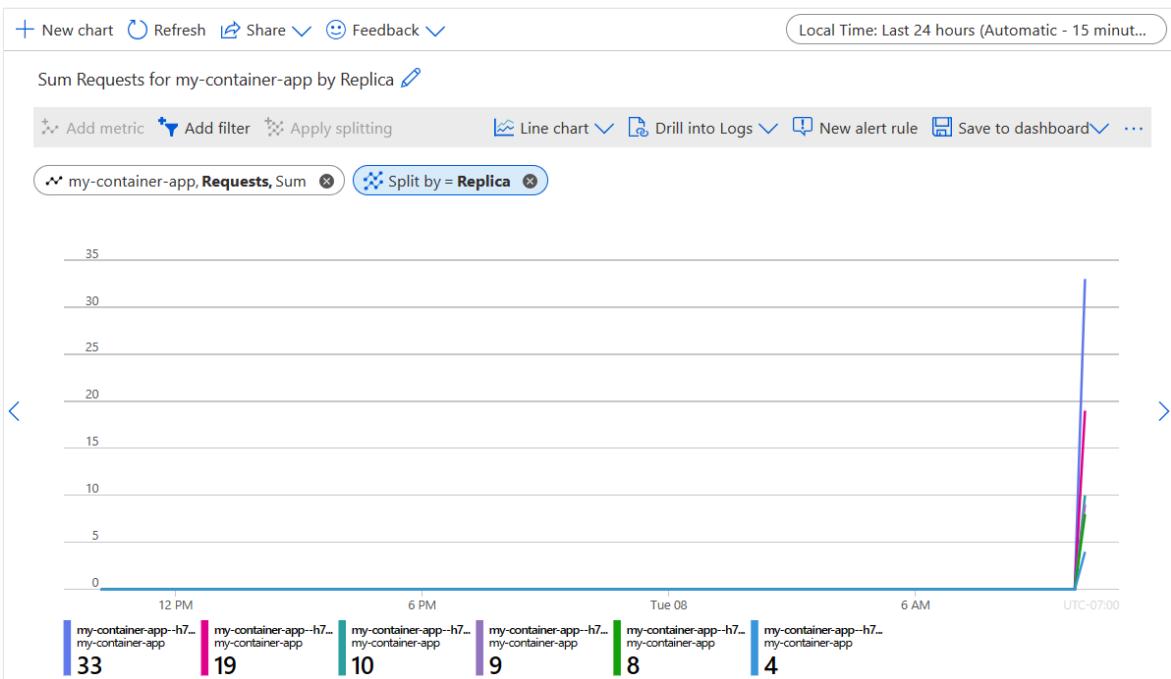
## View scaling in Azure portal (optional)

1. Sign in to the [Azure portal](#).
2. In the **Search** bar at the top, enter **my-container-app**.
3. In the search results, under *Resources*, select **my-container-app**.
4. In the navigation bar at the left, expand **Application** and select **Scale and replicas**.
5. In the *Scale and Replicas* page, select **Replicas**.
6. Your container app now has more than one replica running.

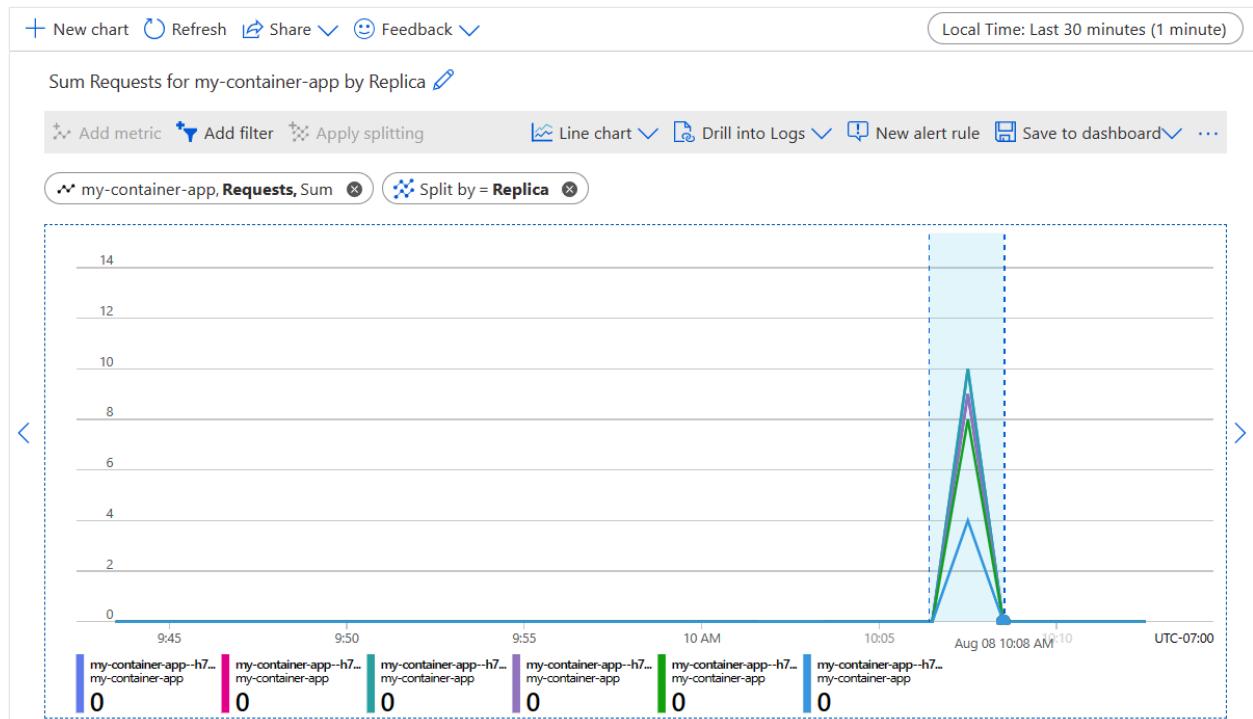
Replica name ↑	Ready ↑	Running status	Restarts ↑	Time created ↑
my-container-app--ztzthcp-858c7fcdd-	1/1	✓ Running	0	8/3/2023, 1:58:33 PM
my-container-app--ztzthcp-858c7fcdd-	1/1	✓ Running	0	8/3/2023, 1:54:29 PM
my-container-app--ztzthcp-858c7fcdd-	1/1	✓ Running	0	8/3/2023, 1:58:33 PM
my-container-app--ztzthcp-858c7fcdd-	1/1	✓ Running	0	8/3/2023, 1:58:33 PM

You may need to select **Refresh** to see the new replicas.

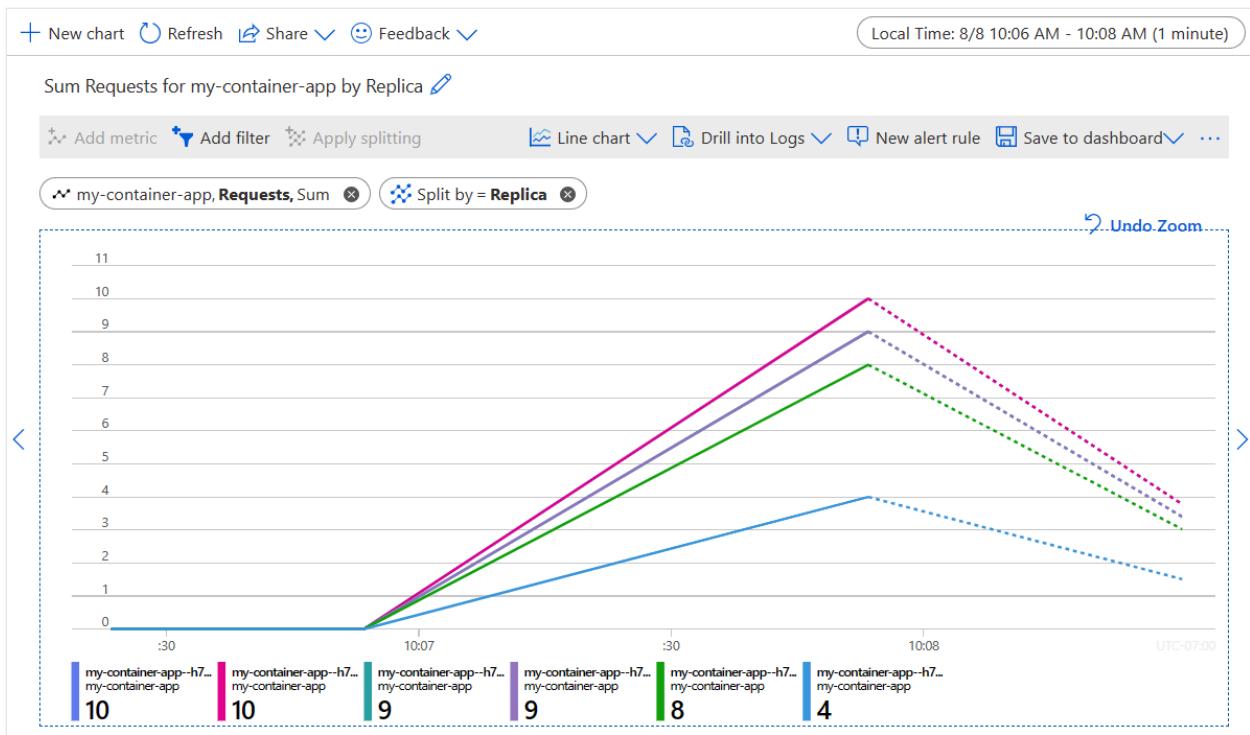
1. In the navigation bar at the left, expand **Monitoring** and select **Metrics**.
2. In the *Metrics* page, set **Metric** to **Requests**.
3. Select **Apply splitting**.
4. Expand the **Values** drop-down and check **Replica**.
5. Select the blue checkmark icon to finish editing the splitting.
6. The graph shows the requests received by your container app, split by replica.



7. By default, the graph scale is set to last 24 hours, with a time granularity of 15 minutes. Select the scale and change it to the last 30 minutes, with a time granularity of one minute. Select the **Apply** button.
8. Select on the graph and drag to highlight the recent increase in requests received by your container app.



The following screenshot shows a zoomed view of how the requests received by your container app are divided among replicas.



## Clean up resources

If you're not going to continue to use this application, run the following command to delete the resource group along with all the resources created in this tutorial.

### ⊗ Caution

The following command deletes the specified resource group and all resources contained within it. If resources outside the scope of this tutorial exist in the specified resource group, they will also be deleted.

Azure CLI

```
az group delete --name my-container-apps
```

### 💡 Tip

Having issues? Let us know on GitHub by opening an issue in the [Azure Container Apps repo](#).

## Next steps

[Set scaling rules in Azure Container Apps](#)

---

# Feedback

Was this page helpful?

 Yes

 No

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

# Certificates in Azure Container Apps

Article • 05/10/2024

You can add digital security certificates to secure custom DNS names in Azure Container Apps to support secure communication among your apps.

## Options

The following table lists the options available to manage certificates in Container Apps:

[+] Expand table

Option	Description
<a href="#">Custom domain with a free certificate</a>	A private certificate that's free of charge and easy to use if you just need to secure your custom domain in Container Apps.
<a href="#">Custom domain with an existing certificate</a>	You can upload a private certificate if you already have one.
<a href="#">Certificates from Azure Key Vault</a>	When you use Azure Key Vault, you get features like automatic renewal and notifications for lifecycle events.

## Next steps

[Custom domain names and free managed certificates](#)

# Custom domain names and free managed certificates in Azure Container Apps

Article • 09/19/2024

Azure Container Apps allows you to bind one or more custom domains to a container app. You can automatically configure a free managed certificate for your custom domain.

If you want to set up a custom domain using your own certificate, see [Custom domain names and certificates in Azure Container Apps](#).

## ⓘ Note

If you configure a [custom environment DNS suffix](#), you cannot add a custom domain that contains this suffix to your Container App.

## Free certificate requirements

Azure Container Apps provides a free managed certificate for your custom domain. Without any action required from you, this TLS/SSL server certificate is automatically renewed as long as your app continues to meet the requirements for managed certificates.

The requirements are:

- Enable HTTP ingress and ensure your container app is publicly accessible.
- Must have an A record for apex domains that points to your Container Apps environment's IP address.
- Establish a CNAME record for subdomains that maps directly to the container app's automatically generated domain name. Mapping to an intermediate CNAME value blocks certificate issuance and renewal. Examples of CNAME values are traffic managers, Cloudflare, and similar services.

## ⓘ Note

To ensure the certificate issuance and subsequent renewals proceed successfully, all requirements must be met at all times when the managed certificate is assigned.

## Add a custom domain and managed certificate

1. Navigate to your container app in the [Azure portal](#)
2. Verify that your app has HTTP ingress enabled by selecting **Ingress** in the *Settings* section. If ingress isn't enabled, enable it with these steps:
  - a. Set *HTTP Ingress* to **Enabled**.
  - b. Select the desired *Ingress traffic* setting.
  - c. Enter the *Target port*.
  - d. Select **Save**.
3. Under the *Settings* section, select **Custom domains**.
4. Select **Add custom domain**.
5. In the *Add custom domain and certificate* window, in *TLS/SSL certificate*, select **Managed certificate**.
6. In *domain*, enter the domain you want to add.
7. Select the *Hostname record type* based on the type of your domain.

[ ] [Expand table](#)

Domain type	Record type	Notes
Apex domain	A record	An apex domain is a domain at the root level of your domain. For example, if your DNS zone is <code>contoso.com</code> , then <code>contoso.com</code> is the apex domain.
Subdomain	CNAME	A subdomain is a domain that is part of another domain. For example, if your DNS zone is <code>contoso.com</code> , then <code>www.contoso.com</code> is an example of a subdomain that can be configured in the zone.

8. Using the DNS provider that is hosting your domain, create DNS records based on the *Hostname record type* you selected using the values shown in the *Domain validation* section. The records point the domain to your container app and verify that you're the owner.
  - If you selected *A record*, create the following DNS records:

[Expand table](#)

Record type	Host	Value
A	@	The IP address of your Container Apps environment
TXT	asuid	The domain verification code

- If you selected *CNAME*, create the following DNS records:

[Expand table](#)

Record type	Host	Value
CNAME	The subdomain (for example, <code>www</code> )	The automatically generated <code>&lt;appname&gt;. &lt;region&gt;.azurecontainerapps.io</code> domain of your container app
TXT	<code>asuid.</code> followed by the subdomain (for example, <code>asuid.www</code> )	The domain verification code

9. Select **Validate**.

10. Once validation succeeds, select **Add**.

It might take several minutes to issue the certificate and add the domain to your container app.

11. Once the operation is complete, you see your domain name in the list of custom domains with a status of *Secured*. Navigate to your domain to verify that it's accessible.

## Next steps

[Authentication in Azure Container Apps](#)

---

## Feedback

Was this page helpful?

[Yes](#)[No](#)

# Custom domain names and bring your own certificates in Azure Container Apps

Article • 05/28/2024

Azure Container Apps allows you to bind one or more custom domains to a container app.

- Every domain name must be associated with a TLS/SSL certificate. You can upload your own certificate or use a [free managed certificate](#).
- Certificates are applied to the container app environment and are bound to individual container apps. You must have role-based access to the environment to add certificates.
- [SNI \(Server Name Identification\) domain certificates](#) are required.
- Ingress must be enabled for the container app.

## ⓘ Note

If you configure a [custom environment DNS \(Domain Name System\) suffix](#), you cannot add a custom domain that contains this suffix to your Container App.

## Add a custom domain and certificate

### ⓘ Important

If you are using a new certificate, you must have an existing [SNI domain certificate](#) file available to upload to Azure.

1. Navigate to your container app in the [Azure portal](#)
2. Verify that your app has ingress enabled by selecting **Ingress** in the *Settings* section. If ingress isn't enabled, enable it with these steps:
  - a. Set *HTTP Ingress* to **Enabled**.
  - b. Select the desired *Ingress traffic* setting.
  - c. Enter the *Target port*.
  - d. Select **Save**.
3. Under the *Settings* section, select **Custom domains**.

4. Select the **Add custom domain** button.
5. In the *Add custom domain and certificate* window, in *TLS/SSL certificate*, select **Bring your own certificate**.
6. In *domain*, enter the domain you want to add.
7. Select **Add a certificate**.
8. In the *Add certificate* window, in *Certificate name*, enter a name for this certificate.
9. In *Certificate file* section, browse for the certificate file you want to upload.
10. Select **Validate**.
11. Once validation succeeds, select **Add**.
12. In the *Add custom domain and certificate* window, in *Certificate*, select the certificate you just added.
13. Select the *Hostname record type* based on the type of your domain.

[+] Expand table

Domain type	Record type	Notes
Apex domain	A record	An apex domain is a domain at the root level of your domain. For example, if your DNS (Domain Name System) zone is <code>contoso.com</code> , then <code>contoso.com</code> is the apex domain.
Subdomain	CNAME	A subdomain is a domain that is part of another domain. For example, if your DNS zone is <code>contoso.com</code> , then <code>www.contoso.com</code> is an example of a subdomain that can be configured in the zone.

14. Using the DNS provider that is hosting your domain, create DNS records based on the *Hostname record type* you selected using the values shown in the *Domain validation* section. The records point the domain to your container app and verify that you own it.

- If you selected *A record*, create the following DNS records:

[+] Expand table

Record type	Host	Value
A	@	The IP address of your Container Apps environment

Record type	Host	Value
TXT	asuid	The domain verification code

- If you selected *CNAME*, create the following DNS records:

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

Record type	Host	Value
CNAME	The subdomain (for example, <code>www</code> )	The automatically generated domain of your container app
TXT	<code>asuid.</code> followed by the subdomain (for example, <code>asuid.www</code> )	The domain verification code

15. Select the **Validate** button.
16. Once validation succeeds, select the **Add** button.
17. Once the operation is complete, you see your domain name in the list of custom domains with a status of *Secured*. Navigate to your domain to verify that it's accessible.

#### Note

For container apps in internal Container Apps environments, [additional configuration](#) is required to use custom domains with VNET-scope ingress.

## Managing certificates

You can manage certificates via the Container Apps environment or through an individual container app.

### Environment

The *Certificates* window of the Container Apps environment presents a table of all the certificates associated with the environment.

You can manage your certificates through the following actions:

Action	Description
Add	Select the <b>Add certificate</b> link to add a new certificate.
Delete	Select the trash can icon to remove a certificate.
Renew	The <i>Health status</i> field of the table indicates that a certificate is expiring soon within 60 days of the expiration date. To renew a certificate, select the <b>Renew certificate</b> link to upload a new certificate.

## Container app

The *Custom domains* window of the container app presents a list of custom domains associated with the container app.

You can manage your certificates for an individual domain name by selecting the ellipsis (...) button, which opens the certificate binding window. From the following window, you can select a certificate to bind to the selected domain name.

## Next steps

[Authentication in Azure Container Apps](#)

# Custom environment DNS Suffix in Azure Container Apps

Article • 07/22/2024

Azure Container Apps environment provides a default DNS suffix in the format `<UNIQUE_IDENTIFIER>. <REGION_NAME>. azurecontainerapps.io`. Each container app in the environment generates a domain name based on this DNS suffix. You can configure a custom DNS suffix for your environment.

## ⓘ Note

To configure a custom domain for individual container apps, see [Custom domain names and certificates in Azure Container Apps](#).

If you configure a custom DNS suffix for your environment, traffic to FQDNs (Fully Qualified Domain Names) that use this suffix will resolve to the environment. FQDNs that use this suffix outside of the environment are unreachable.

## Add a custom DNS suffix and certificate

1. Go to your Container Apps environment in the [Azure portal](#) ↗
2. Under the *Settings* section, select **Custom DNS suffix**.
3. In **DNS suffix**, enter the custom DNS suffix for the environment.  
For example, if you enter `example.com`, the container app domain names are in the format `<APP_NAME>.example.com`.
4. In a new browser window, go to your domain provider's website and add the DNS records shown in the *Domain validation* section to your domain.

expand Expand table

Record type	Host	Value	Description
A	<code>*</code> <code>&lt;DNS_SUFFIX&gt;</code>	Environment inbound IP address	Wildcard record configured to the IP address of the environment.

Record type	Host	Value	Description
TXT	asuid. <DNS_SUFFIX>	Validation token	TXT record with the value of the validation token (not required for Container Apps environment with internal load balancer).

5. Back in the *Custom DNS suffix* windows, in Certificate file, browse, and select a certificate for the TLS binding.

 **Important**

You must use an existing wildcard certificate that's valid for the custom DNS suffix you provided.

6. In **Certificate password**, enter the password for the certificate.

7. Select **Save**.

Once the saved operation is complete, the environment is updated with the custom DNS suffix and TLS certificate.

## Next steps

[Custom domains in Azure Container Apps](#)

## Feedback

Was this page helpful?

 Yes

 No

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

# Import certificates from Azure Key Vault to Azure Container Apps

Article • 09/19/2024

You can set up Azure Key Vault to centrally manage your container app's TLS/SSL certificates and handle updates, renewals, and monitoring.

## Prerequisites

An Azure Key Vault resource is required to store your certificate. See [Import a certificate in Azure Key Vault](#) or [Configure certificate auto-rotation in Key Vault](#) to create a Key Vault and add a certificate.

## Exceptions

While the majority of certificate types are supported, there are a few exceptions to keep in mind.

- ECDSA p384 and p521 certificates are not supported.
- Due to how App Services certificates are saved in Key Vault, they cannot be imported using the Azure Portal and require the Azure CLI.

## Enable managed identity for Container Apps environment

Azure Container Apps uses an environment level managed identity to access your Key Vault and import your certificate. To enable system-assigned managed identity, follow these steps:

1. Open the [Azure portal](#) and find your Azure Container Apps environment where you want to import a certificate.
2. From *Settings*, select **Identity**.
3. On the *System assigned* tab, find the *Status* switch and select **On**.
4. Select **Save**, and when the *Enable system assigned managed identity* window appears, select **Yes**.

5. Under the *Permissions* label, select **Azure role assignments** to open the role assignments window.

6. Select **Add role assignment** and enter the following values:

[+] Expand table

Property	Value
Scope	Select <b>Key Vault</b> .
Subscription	Select your Azure subscription.
Resource	Select your vault.
Role	Select <b>Key Vault Secrets User</b> .

7. Select **Save**.

For more detail on RBAC vs. legacy access policies, see [Azure role-based access control \(Azure RBAC\) vs. access policies](#).

## Import certificate from Key Vault

1. Open the Azure portal and go to your Azure Container Apps environment.

2. From *Settings*, select **Certificates**.

3. Select the **Bring your own certificates (.pfx)** tab.

4. Select **Add certificate**.

5. In the *Add certificate* panel, in *Source*, select **Import from Key Vault**.

6. Select **Select key vault certificate** and select the following values:

[+] Expand table

Property	Value
Subscription	Select your Azure subscription.
Key vault	Select your vault.
Certificate	Select your certificate.

 **Note**

If you see an error, "*The operation "List" is not enabled in this key vault's access policy.*", you need to configure an access policy in your Key Vault to allow your user account to list certificates. For more information, see [Assign a Key Vault access policy](#).

7. Select **Select**.

8. In the *Add certificate* panel, in *Managed identity*, select **System assigned**. If you're using a user-assigned managed identity, select your user-assigned managed identity.

9. Select **Add**.

 **Note**

If you receive an error message, verify that the managed identity is assigned the **Key Vault Secrets User** role on the Key Vault.

## Configure a custom domain

After configuring your certificate, you can use it to secure your custom domain. Follow the steps in [Add a custom domain](#) and select the certificate you imported from Key Vault.

## Rotate certificates

When you rotate your certificate in Key Vault, Azure Container Apps automatically updates the certificate in your environment. It takes up to 12 hours for the new certificate to be applied.

## Related

[Certificates in Azure Container Apps](#)

## Feedback

Was this page helpful?

 Yes

 No

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

# Authentication and authorization in Azure Container Apps

Article • 04/21/2024

Azure Container Apps provides built-in authentication and authorization features (sometimes referred to as "Easy Auth"), to secure your external ingress-enabled container app with minimal or no code.

For details surrounding authentication and authorization, refer to the following guides for your choice of provider.

- [Microsoft Entra ID](#)
- [Facebook](#)
- [GitHub](#)
- [Google](#)
- [Twitter](#)
- [Custom OpenID Connect](#)

## 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, 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 Container Apps saves you time and effort by providing out-of-the-box authentication with federated identity providers. These features allow you to focus more time developing your application, and less time on building security systems.

The benefits include:

- Azure Container Apps provides access to various built-in authentication providers.
- The built-in auth features don't require any particular language, SDK, security expertise, or even any code that you have to write.
- You can integrate with multiple providers including Microsoft Entra ID, Facebook, Google, and Twitter.

# Identity providers

Container Apps uses [federated identity](#), in which a third-party identity provider manages the user identities and authentication flow for you. The following identity providers are available by default:

[+] Expand table

Provider	Sign-in endpoint	How-To guidance
Microsoft identity platform	<code>/auth/login/aad</code>	<a href="#">Microsoft identity platform</a>
Facebook	<code>/auth/login/facebook</code>	<a href="#">Facebook</a>
GitHub	<code>/auth/login/github</code>	<a href="#">GitHub</a>
Google	<code>/auth/login/google</code>	<a href="#">Google</a>
Twitter	<code>/auth/login/twitter</code>	<a href="#">Twitter</a>
Any <a href="#">OpenID Connect</a> provider	<code>/auth/login/&lt;providerName&gt;</code>	<a href="#">OpenID Connect</a>

When you use one of these providers, the sign-in endpoint is available for user authentication and authentication token validation from the provider. You can provide your users with any number of these provider options.

## Considerations for using built-in authentication

This feature should be used with HTTPS only. Ensure `allowInsecure` is disabled on your container app's ingress configuration.

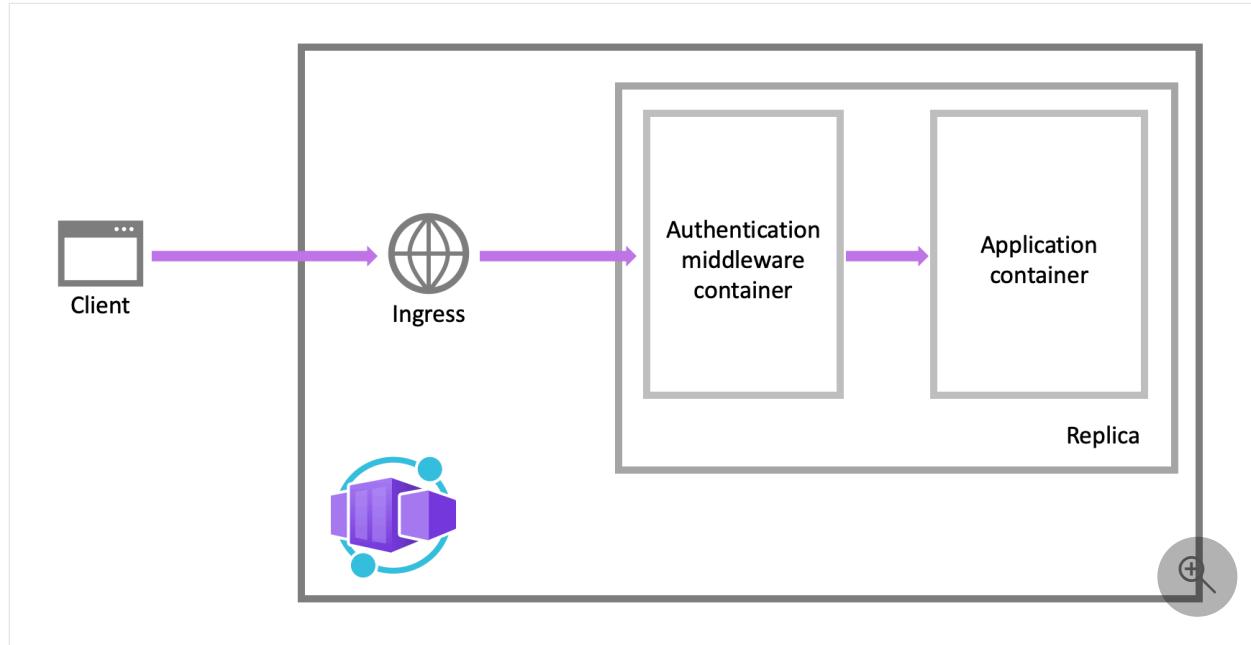
You can configure your container app for authentication with or without restricting access to your site content and APIs. To restrict app access only to authenticated users, set its *Restrict access* setting to **Require authentication**. To authenticate but not restrict access, set its *Restrict access* setting to **Allow unauthenticated access**.

By default, each container app issues its own unique cookie or token for authentication. You can also provide your own signing and encryption keys.

## Feature architecture

The authentication and authorization middleware component is a feature of the platform that runs as a sidecar container on each replica in your application. When

enabled, your application handles each incoming HTTP request after it passes through the security layer.



The platform middleware handles several things for your app:

- Authenticates users and clients with the specified identity providers
- Manages the authenticated session
- Injects identity information into HTTP request headers

The authentication and authorization module runs in a separate container, isolated from your application code. As the security container doesn't run in-process, no direct integration with specific language frameworks is possible. However, relevant information your app needs is provided in request headers as explained in this article.

## 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 (*server-directed flow* or *server flow*):** The application delegates federated sign-in to Container Apps. Delegation is typically the case with browser apps, which presents the provider's sign-in page to the user.
- **With provider SDK (*client-directed flow* or *client flow*):** The application signs users in to the provider manually and then submits the authentication token to Container Apps for validation. This approach is typical for browser-less apps that don't present the provider's sign-in page to the user. An example is a native mobile app that signs users in using the provider's SDK.

Calls from a trusted browser app in Container Apps to another REST API in Container Apps can be authenticated using the server-directed flow. For more information, see [Customize sign in and sign out](#).

The table shows the steps of the authentication flow.

[ ] [Expand table](#)

Step	Without provider SDK	With provider SDK
1. Sign user in	Redirects client to <code>/auth/login/&lt;PROVIDER&gt;</code> .	Client code signs user in directly with provider's SDK and receives an authentication token. For information, see the provider's documentation.
2. Post-authentication	Provider redirects client to <code>/auth/login/&lt;PROVIDER&gt;/callback</code> .	Client code <a href="#">posts token from provider</a> to <code>/auth/login/&lt;PROVIDER&gt;</code> for validation.
3. Establish authenticated session	Container Apps adds authenticated cookie to response.	Container Apps returns its own authentication token to client code.
4. Serve authenticated content	Client includes authentication cookie in subsequent requests (automatically handled by browser).	Client code presents authentication token in <code>x-zumo-auth</code> header.

For client browsers, Container Apps can automatically direct all unauthenticated users to `/auth/login/<PROVIDER>`. You can also present users with one or more `/auth/login/<PROVIDER>` links to sign in to your app using their provider of choice.

## Authorization behavior

In the [Azure portal](#), you can edit your container app's authentication settings to configure it with various behaviors when an incoming request isn't authenticated. The following headings describe the options.

- **Allow unauthenticated access:** This option defers authorization of unauthenticated traffic to your application code. For authenticated requests, Container Apps also passes along authentication information in the HTTP headers. Your app can use information in the headers to make authorization decisions for a request.

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 rejects 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

By default, any user in your Microsoft Entra tenant can request a token for your application from Microsoft Entra ID. You can [configure the application in Microsoft Entra ID](#) if you want to restrict access to your app to a defined set of users.

## Customize sign-in and sign out

Container Apps Authentication provides built-in endpoints for sign in and sign out. When the feature is enabled, these endpoints are available under the `/auth` route prefix on your container app.

## 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

```
Log in with the Microsoft Identity Platform
Log in with Facebook
Log in with Google
Log in with Twitter
```

When the user selects on one of the links, the UI for the respective providers is displayed to the user.

To redirect the user post-sign-in to a custom URL, use the `post_login_redirect_uri` query string parameter (not to be confused with the Redirect URI in your identity provider configuration). For example, to navigate the user to `/Home/Index` after sign-in, use the following HTML code:

#### HTML

```
<a href="/.auth/login/<provider>?post_login_redirect_uri=/Home/Index">Log
in
```

## Client-directed sign-in

In a client-directed sign-in, the application signs in the user to the identity provider using a provider-specific SDK. The application code then submits the resulting authentication token to Container Apps for validation (see [Authentication flow](#)) using an HTTP POST request.

To validate the provider token, container 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:

#### Console

```
POST https://<hostname>.azurecontainerapps.io/.auth/login/aad HTTP/1.1
Content-Type: application/json
```

```
{"id_token":<token>,"access_token":<token>"}
```

The token format varies slightly according to the provider. See the following table for details:

[+] Expand table

Provider value	Required in request body	Comments
aad	{"access_token": " <ACCESS_TOKEN>"}	The <code>id_token</code> , <code>refresh_token</code> , and <code>expires_in</code> properties are optional.
microsoftaccount	{"access_token": " <ACCESS_TOKEN>"} or {"authentication_token": " <TOKEN>"}	<code>authentication_token</code> is preferred over <code>access_token</code> . The <code>expires_in</code> property is optional. When requesting the token from Live services, always request the <code>wl.basic</code> scope.
google	{"id_token": "<ID_TOKEN>"}	The <code>authorization_code</code> property is optional. Providing an <code>authorization_code</code> value adds an access token and a refresh token to the token store. When specified, <code>authorization_code</code> can also optionally be accompanied by a <code>redirect_uri</code> property.
facebook	{"access_token": " <USER_ACCESS_TOKEN>"}	Use a valid <a href="#">user access token</a> from Facebook.
twitter	{"access_token": " <ACCESS_TOKEN>," "access_token_secret": " <ACCES_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:

Console

```
GET https://<hostname>.azurecontainerapps.io/api/products/1
X-ZUMO-AUTH: <authenticationToken_value>
```

## Sign out of a session

Users can sign out by sending a `GET` request to the app's `/.auth/logout` endpoint. The `GET` request conducts the following actions:

- Clears authentication cookies from the current session.
- Deletes the current user's tokens from the token store.
- Performs a server-side sign out on the identity provider for Microsoft Entra ID and Google.

Here's a simple sign out link in a webpage:

HTML

```
Sign out
```

By default, a successful sign out redirects the client to the URL `/.auth/logout/done`. You can change the post-sign-out redirect page by adding the `post_logout_redirect_uri` query parameter. For example:

Console

```
GET /.auth/logout?post_logout_redirect_uri=/index.html
```

Make sure to [encode ↗](#) the value of `post_logout_redirect_uri`.

URL must be hosted in the same domain when using fully qualified URLs.

## Access user claims in application code

For all language frameworks, Container Apps makes the claims in the incoming token available to your application code. The claims are injected into the request headers, which are present whether from an authenticated end user or a client application.

External requests aren't allowed to set these headers, so they're present only if set by Container Apps. Some example headers include:

- X-MS-CLIENT-PRINCIPAL-NAME
- X-MS-CLIENT-PRINCIPAL-ID

Code that is written in any language or framework can get the information that it needs from these headers.

 **Note**

Different language frameworks may present these headers to the app code in different formats, such as lowercase or title case.

## Next steps

Refer to the following articles for details on securing your container app.

- [Microsoft Entra ID](#)
- [Facebook](#)
- [GitHub](#)
- [Google](#)
- [Twitter](#)
- [Custom OpenID Connect](#)

# Enable authentication and authorization in Azure Container Apps with Microsoft Entra ID

Article • 06/14/2024

This article shows you how to configure authentication for Azure Container Apps so that your app signs in users with the [Microsoft identity platform](#) as the authentication provider.

The Container Apps 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](#)

## Option 1: Create a new app registration automatically

This option is designed to make enabling authentication simple and requires just a few steps.

1. Sign in to the [Azure portal](#) and navigate to your app.
2. Select **Authentication** in the menu on the left. Select **Add identity provider**.
3. Select **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 is created and stored as a [secret](#) in the container app.

4. If you're configuring the first identity provider for this application, you're prompted with a **Container Apps authentication settings** section. Otherwise, you move on to the next step.

These options determine how your application responds to unauthenticated requests, and the default selections redirect all requests to sign in with this new provider. You can 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) Select **Next: Permissions** and add any scopes needed by the application. These are added to the app registration, but you can also change them later.

6. Select **Add**.

You're now ready to use the Microsoft identity platform for authentication in your app. The provider is listed on the **Authentication** screen. From there, you can edit or delete this provider configuration.

## Option 2: Use an existing registration created separately

You can also manually register your application for the Microsoft identity platform, customize the registration, and configure Container Apps Authentication with the registration details. This approach is useful when you want to use an app registration from a different Microsoft Entra tenant other than the one in which your application is defined.

### Create an app registration in Microsoft Entra ID for your container app

First, you create your app registration. As you do so, collect the following information that you need later when you configure the authentication in the container app:

- Client ID
- Tenant ID
- Client secret (optional)
- Application ID URI

To register the app, perform the following steps:

1. Sign in to the [Azure portal](#), search for and select **Container Apps**, and then select your app. Note your app's **URL**. You use it to configure your Microsoft Entra app registration.
2. From the portal menu, select **Microsoft Entra ID**, then go to the **App registrations** tab and select **New registration**.
3. In the **Register an application** page, enter a **Name** for your app registration.

4. In **Redirect URI**, select **Web** and type <app-url>/.auth/login/aad/callback. For example, <https://<hostname>.azurecontainerapps.io/.auth/login/aad/callback>.
5. Select **Register**.
6. After the app registration is created, copy the **Application (client) ID** and the **Directory (tenant) ID** for later.
7. Select **Authentication**. Under **Implicit grant and hybrid flows**, enable **ID tokens** to allow OpenID Connect user sign-ins from Container Apps. Select **Save**.
8. (Optional) Select **Branding**. In **Home page URL**, enter the URL of your container app and select **Save**.
9. Select **Expose an API**, and select **Set** next to **Application ID URI**. The ID value uniquely identifies your application when it's used as a resource, which allows requested tokens to grant access. The value is also used as a prefix for scopes you create.

For a single-tenant app, you can use the default value, which is in the form `api://<application-client-id>`. You can also specify a more readable URI like <https://contoso.com/api> based on one of the verified domains for your tenant. For a multitenant app, you must provide a custom URI. To learn more about accepted formats for App ID URIs, see the [app registrations best practices reference](#).
- The value is automatically saved.
10. Select **Add a scope**.
  - a. In **Add a scope**, the **Application ID URI** is the value you set in a previous step. Select **Save and continue**.
  - b. In **Scope name**, enter *user\_impersonation*.
  - 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**.
11. (Optional) To create a client secret, select **Certificates & secrets > Client secrets > New client secret**. Enter a description and expiration and select **Add**. Copy the client secret value shown on the page as the site won't display it to you again.
12. (Optional) To add multiple **Reply URLs**, select **Authentication**.

## Enable Microsoft Entra ID in your container app

1. Sign in to the [Azure portal](#) and navigate to your app.
2. Select **Authentication** in the menu on the left. Select **Add identity provider**.
3. Select **Microsoft** in the identity provider dropdown.
4. For **App registration type**, you can choose to **Pick an existing app registration in this directory** which automatically gathers the necessary app information. If your registration is from another tenant or you don't have permission to view the registration object, choose **Provide the details of an existing app registration**. For this option, you need to fill in the following configuration details:

  Expand table

Field	Description
Application (client) ID	Use the <b>Application (client) ID</b> of the app registration.
Client Secret	Use the client secret you generated in the app registration. With a client secret, hybrid flow is used and the app returns access and refresh tokens. When the client secret isn't set, implicit flow is used and only an ID token is returned. The provider sends the tokens and they're stored in the EasyAuth token store.
Issuer Url	Use < <code>authentication-endpoint</code> >/< <code>TENANT-ID</code> >/v2.0, and replace < <code>authentication-endpoint</code> > with the <a href="#">authentication endpoint for your cloud environment</a> (for example, "https://login.microsoftonline.com" for global Azure), also replacing < <code>TENANT-ID</code> > with the <b>Directory (tenant) ID</b> in which the app registration was created. This value is used to redirect users to the correct Microsoft Entra tenant, and to download the appropriate metadata to determine the appropriate token signing keys and token issuer claim value for example. For applications that use Azure AD v1, omit /v2.0 in the URL.
Allowed Token Audiences	The configured <b>Application (client) ID</b> is <i>always</i> implicitly considered to be an allowed audience. If this value refers to a cloud or server app and you want to accept authentication tokens from a client container app (the authentication token can be retrieved in the <code>X-MS-TOKEN-AAD-ID-TOKEN</code> header), add the <b>Application (client) ID</b> of the client app here.

The client secret is stored as [secrets](#) in your container app.

5. If this is the first identity provider configured for the application, you're also prompted with a **Container Apps authentication settings** section. Otherwise, you 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 sign in with this new provider. You can change customize this behavior now or adjust these settings later from the main **Authentication** screen by choosing **Edit** next to **Authentication settings**. To learn more about these options, see [Authentication flow](#).

## 6. Select Add.

You're now ready to use the Microsoft identity platform for authentication in your app. The provider is listed on the **Authentication** screen. From there, you can edit or delete this provider configuration.

# Configure client apps to access your container app

In the prior section, you registered your container app to authenticate users. In this section, you register native client or daemon apps. They can then request access to APIs exposed by your container app on behalf of users or themselves. Completing the steps in this section isn't required if you only wish to authenticate users.

## Native client application

You can register native clients to request access your container app's APIs on behalf of a signed in user.

1. In the [Azure portal](#), select **Microsoft Entra ID** > **Add** > **App registrations**.
2. In the *Register an application* page, enter a **Name** for your app registration.
3. In **Redirect URI**, select **Public client (mobile & desktop)** and type the URL `<app-url>/auth/login/aad/callback`. For example,  
`https://<hostname>.azurecontainerapps.io/.auth/login/aad/callback`.

### ⓘ Note

For a Microsoft Store application, use the [package SID](#) as the URI instead.

4. Select **Create**.
5. After the app registration is created, copy the value of **Application (client) ID**.

6. Select API permissions > Add a permission > My APIs.
7. Select the app registration you created earlier for your container app. If you don't see the app registration, make sure that you added the **user\_impersonation** scope in [Create an app registration in Microsoft Entra ID for your container app](#).
8. Under Delegated permissions, select **user\_impersonation**, and then select Add permissions.

In this section, you configured a native client application that can request access your container app on behalf of a user.

## Daemon client application (service-to-service calls)

Your application can acquire a token to call a Web API hosted in your container app on behalf of 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. In the [Azure portal](#), select Microsoft Entra ID > Add > App registrations.
2. In the Register an application page, enter a Name for your daemon app registration.
3. For a daemon application, you don't need a Redirect URI so you can keep that empty.
4. Select Create.
5. After the app registration is created, copy the value of Application (client) ID.
6. Select Certificates & secrets > New client secret > Add. Copy the client secret value shown in the page. It isn't shown again.

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 Container Apps Authentication / Authorization validates and uses the token as usual to indicate that the caller (an application in this case, not a user) is authenticated.

This process allows *any* client application in your Microsoft Entra tenant to request an access token and authenticate to the target app. If you also want to enforce *authorization* to allow only certain client applications, you must adjust the configuration.

1. [Define an App Role](#) in the manifest of the app registration representing the container app you want to protect.

2. On the app registration representing the client that needs to be authorized, select **API permissions** > **Add a permission** > **My APIs**.
3. Select the app registration you created earlier. If you don't see the app registration, make sure that you've [added an App Role](#).
4. Under **Application permissions**, select the App Role you created earlier, and then select **Add permissions**.
5. Make sure to select **Grant admin consent** to authorize the client application to request the permission.
6. Similar to the previous scenario (before any roles were added), you can now [request an access token](#) for the same target `resource`, and the access token includes a `roles` claim containing the App Roles that were authorized for the client application.
7. Within the target Container Apps code, you can now validate that the expected roles are present in the token. The Container Apps auth layer doesn't perform the validation steps. For more information, see [Access user claims](#).

In this section, you configured a daemon client application that can access your container app using its own identity.

## Working with authenticated users

Use the following guides for details on working with authenticated users.

- [Customize sign-in and sign-out](#)
- [Access user claims in application code](#)

## Next steps

[Authentication and authorization overview](#)

## Feedback

Was this page helpful?

 Yes

 No

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

# Enable authentication and authorization in Azure Container Apps with Facebook

Article • 05/24/2022

This article shows how to configure Azure Container Apps 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](https://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://<hostname>.azurecontainerapps.io/.auth/login/facebook/callback.
```

Remember to use the hostname of your container 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 container 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. Select **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 **secret** in your container app.

4. If you're configuring the first identity provider for this application, you'll be prompted with a **Container Apps authentication settings** section. Otherwise, you may move on to the next step.

These options determine how your application responds to unauthenticated requests. The default selections redirect all requests to sign in with this new provider. You can change customize this behavior now or adjust these settings later from the main **Authentication** screen by choosing **Edit** next to **Authentication settings**. To learn more about these options, see [Authentication flow](#).

5. (Optional) Select **Next: Scopes** and add any scopes needed by the application.

These scopes are requested when a user signs in for browser-based flows.

6. Select **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.

## Working with authenticated users

Use the following guides for details on working with authenticated users.

- [Customize sign-in and sign-out](#)
- [Access user claims in application code](#)

## Next steps

[Authentication and authorization overview](#)

# Enable authentication and authorization in Azure Container Apps with GitHub

Article • 06/23/2022

This article shows how to configure Azure Container Apps 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://<hostname>.azurecontainerapps.io/.auth/login/github/callback`.
3. On the application page, make note of the **Client ID**, which you'll need later.
4. Under **Client Secrets**, select **Generate a new client secret**.
5. Make note of the client secret value, which you'll 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. Select **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 secret in your container app.

4. If you're configuring the first identity provider for this application, you'll also be prompted with a **Container Apps authentication settings** section. Otherwise, you may move on to the next step.

These options determine how your application responds to unauthenticated requests. The default selections redirect all requests to sign in with this new provider. You can change customize this behavior now or adjust these settings later from the main **Authentication** screen by choosing **Edit** next to **Authentication settings**. To learn more about these options, see [Authentication flow](#).

#### 5. Select **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.

## Working with authenticated users

Use the following guides for details on working with authenticated users.

- [Customize sign-in and sign-out](#)
- [Access user claims in application code](#)

## Next steps

[Authentication and authorization overview](#)

# Enable authentication and authorization in Azure Container Apps with Google

Article • 05/24/2022

This article shows you how to configure Azure Container Apps to use Google as an authentication provider.

To complete the following procedure, you must have a Google account that has a verified email address. To create a new Google account, go to [accounts.google.com](https://accounts.google.com).

## Register your application with Google

1. Follow the Google documentation at [Google Sign-In for server-side apps](#) to create a client ID and client secret. There's no need to make any code changes. Just use the following information:
  - For **Authorized JavaScript Origins**, use  
`https://<hostname>.azurecontainerapps.io` with the name of your app in `<hostname>`.
  - For **Authorized Redirect URI**, use  
`https://<hostname>.azurecontainerapps.io/.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. Select **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 [secret](#) in your container app.

4. If you're configuring the first identity provider for this application, you'll also be prompted with a **Container Apps authentication settings** section. Otherwise, you may move on to the next step.

These options determine how your application responds to unauthenticated requests. The default selections redirect all requests to sign in with this new provider. You can change customize this behavior now or adjust these settings later from the main **Authentication** screen by choosing **Edit** next to **Authentication settings**. To learn more about these options, see [Authentication flow](#).

5. Select **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're 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.

## Working with authenticated users

Use the following guides for details on working with authenticated users.

- [Customize sign-in and sign-out](#)
- [Access user claims in application code](#)

## Next steps

[Authentication and authorization overview](#)

# Enable authentication and authorization in Azure Container Apps with X

Article • 08/09/2024

This article shows how to configure Azure Container Apps to use X as an authentication provider.

To complete the procedure in this article, you need an X account that has a verified email address and phone number. To create a new X account, go to [x.com](#).

## Register your application with X

1. Sign in to the [Azure portal](#) and go to your application. Copy your **URL**. You'll use it to configure your X app.
2. Go to the [X Developers](#) website, sign in with your X account credentials, and select **Create an app**.
3. Enter the **App name** and the **Application description** for your new app. Paste your application's **URL** into the **Website URL** field. In the **Callback URLs** section, enter the HTTPS URL of your container app and append the path `/auth/login/x/callback`. For example,  
`https://<hostname>.azurecontainerapps.io/.auth/login/x/callback`.
4. At the bottom of the page, type at least 100 characters in **Tell us how this app will be used**, then select **Create**. Select **Create** again in the pop-up. The application details are displayed.
5. Select the **Keys and Access Tokens** tab.

Make a note of these values:

- API key
- API secret key

### Important

The API secret key is an important security credential. Do not share this secret with anyone or distribute it with your app.

# Add X information to your application

1. Sign in to the [Azure portal](#) and navigate to your app.
2. Select **Authentication** in the menu on the left. Select **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 `secret` in your container app.
4. If you're configuring the first identity provider for this application, you'll also be prompted with a **Container Apps authentication settings** section. Otherwise, you may move on to the next step.

These options determine how your application responds to unauthenticated requests. The default selections redirect all requests to sign in with this new provider. You can change customize this behavior now or adjust these settings later from the main **Authentication** screen by choosing **Edit** next to **Authentication settings**. To learn more about these options, see [Authentication flow](#).

5. Select **Add**.

You're now ready to use X for authentication in your app. The provider will be listed on the **Authentication** screen. From there, you can edit or delete this provider configuration.

## Working with authenticated users

Use the following guides for details on working with authenticated users.

- [Customize sign-in and sign-out](#)
- [Access user claims in application code](#)

## Next steps

[Authentication and authorization overview](#)

---

## Feedback

Was this page helpful?

 Yes

 No

Provide product feedback  | Get help at Microsoft Q&A

# Enable authentication and authorization in Azure Container Apps with a Custom OpenID Connect provider

Article • 10/16/2024

This article shows you how to configure Azure Container Apps to use a custom authentication provider that adheres to the [OpenID Connect specification](#). OpenID Connect (OIDC) is an industry standard widely adopted by many identity providers (IDPs). You don't need to understand the details of the specification in order to configure your app to use an adherent IDP.

You can configure your app to use one or more OIDC providers. Each must be given a unique alphanumeric name in the configuration, and only one can serve as the default redirect target.

## Register your application with the identity provider

Your provider requires you to register the details of your application with it. One of these steps involves specifying a redirect URI. This redirect URI is of the form `<app-url>/auth/login/<provider-name>/callback`. Each identity provider should provide more instructions on how to complete these steps.

### 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 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 need to collect a **client ID** and **client secret** for your application.

### Important

The client secret is a critical security credential. Do not share this secret with anyone or distribute it within a client application.

Additionally, you need the OpenID Connect metadata for the provider. This information is often exposed via a [configuration metadata document](#), which is the provider's Issuer URL suffixed with `/well-known/openid-configuration`. Make sure to gather this configuration URL.

If you're unable to use a configuration metadata document, you need to gather the following values separately:

- The issuer URL (sometimes shown as `issuer`)
- The [OAuth 2.0 Authorization endpoint](#) (sometimes shown as `authorization_endpoint`)
- The [OAuth 2.0 Token endpoint](#) (sometimes shown as `token_endpoint`)
- The URL of the [OAuth 2.0 JSON Web Key Set](#) document (sometimes shown as `jwks_uri`)

## Add provider information to your application

1. Sign in to the [Azure portal](#) and navigate to your app.
2. Select **Authentication** in the menu on the left. Select **Add identity provider**.
3. Select **OpenID Connect** in the identity provider dropdown.
4. Provide the unique alphanumeric name selected earlier for **OpenID provider name**.
5. If you have the URL for the **metadata document** from the identity provider, provide that value for **Metadata URL**. Otherwise, select the **Provide endpoints separately** option and put each URL gathered from the identity provider in the appropriate field.
6. Provide the earlier collected **Client ID** and **Client Secret** in the appropriate fields.
7. Specify an application setting name for your client secret. Your client secret is stored as a `secret` in your container app.
8. Press the **Add** button to finish setting up the identity provider.

## Working with authenticated users

Use the following guides for details on working with authenticated users.

- [Customize sign-in and sign-out](#)

- Access user claims in application code

## Next steps

[Authentication and authorization overview](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# Manage secrets in Azure Container Apps

Article • 05/23/2023

Azure Container Apps allows your application to securely store sensitive configuration values. Once secrets are defined at the application level, secured values are available to revisions in your container apps. Additionally, you can reference secured values inside scale rules. For information on using secrets with Dapr, refer to [Dapr integration](#).

- Secrets are scoped to an application, outside of any specific revision of an application.
- Adding, removing, or changing secrets doesn't generate new revisions.
- Each application revision can reference one or more secrets.
- Multiple revisions can reference the same secret(s).

An updated or deleted secret doesn't automatically affect existing revisions in your app. When a secret is updated or deleted, you can respond to changes in one of two ways:

1. Deploy a new revision.
2. Restart an existing revision.

Before you delete a secret, deploy a new revision that no longer references the old secret. Then deactivate all revisions that reference the secret.

## Defining secrets

Secrets are defined as a set of name/value pairs. The value of each secret is specified directly or as a reference to a secret stored in Azure Key Vault.

## Store secret value in Container Apps

When you define secrets through the portal, or via different command line options.

Azure portal

1. Go to your container app in the [Azure portal](#).
2. Under the *Settings* section, select **Secrets**.
3. Select **Add**.
4. In the *Add secret* context pane, enter the following information:

- **Name:** The name of the secret.
- **Type:** Select **Container Apps Secret**.
- **Value:** The value of the secret.

5. Select **Add**.

## Reference secret from Key Vault (preview)

When you define a secret, you create a reference to a secret stored in Azure Key Vault. Container Apps automatically retrieves the secret value from Key Vault and makes it available as a secret in your container app.

To reference a secret from Key Vault, you must first enable managed identity in your container app and grant the identity access to the Key Vault secrets.

To enable managed identity in your container app, see [Managed identities](#).

To grant access to Key Vault secrets, [create an access policy](#) in Key Vault for the managed identity you created. Enable the "Get" secret permission on this policy.

Azure portal

1. Go to your container app in the [Azure portal](#).
2. Under the *Settings* section, select **Identity**.
3. In the *System assigned* tab, select **On**.
4. Select **Save** to enable system-assigned managed identity.
5. Under the *Settings* section, select **Secrets**.
6. Select **Add**.
7. In the *Add secret* context pane, enter the following information:
  - **Name:** The name of the secret.
  - **Type:** Select **Key Vault reference**.
  - **Key Vault secret URL:** The URI of your secret in Key Vault.
  - **Identity:** The identity to use to retrieve the secret from Key Vault.
8. Select **Add**.

## Note

If you're using **UDR With Azure Firewall**, you will need to add the `AzureKeyVault` service tag and the `login.microsoft.com` FQDN to the allow list for your firewall. Refer to [configuring UDR with Azure Firewall](#) to decide which additional service tags you need.

## Key Vault secret URI and secret rotation

The Key Vault secret URI must be in one of the following formats:

- `https://myvault.vault.azure.net/secrets/mysecret/ec96f02080254f109c51a1f14cdb1931`: Reference a specific version of a secret.
- `https://myvault.vault.azure.net/secrets/mysecret`: Reference the latest version of a secret.

If a version isn't specified in the URI, then the app uses the latest version that exists in the key vault. When newer versions become available, the app automatically retrieves the latest version within 30 minutes. Any active revisions that reference the secret in an environment variable is automatically restarted to pick up the new value.

For full control of which version of a secret is used, specify the version in the URI.

## Referencing secrets in environment variables

After declaring secrets at the application level as described in the [defining secrets](#) section, you can reference them in environment variables when you create a new revision in your container app. When an environment variable references a secret, its value is populated with the value defined in the secret.

## Example

The following example shows an application that declares a connection string at the application level. This connection is referenced in a container environment variable and in a scale rule.

Azure portal

After you've [defined a secret](#) in your container app, you can reference it in an environment variable when you create a new revision.

1. Go to your container app in the [Azure portal](#).
2. Open the *Revision management* page.
3. Select **Create new revision**.
4. In the *Create and deploy new revision* page, select a container.
5. In the *Environment variables* section, select **Add**.
6. Enter the following information:
  - **Name:** The name of the environment variable.
  - **Source:** Select **Reference a secret**.
  - **Value:** Select the secret you want to reference.
7. Select **Save**.
8. Select **Create** to create the new revision.

## Mounting secrets in a volume

After declaring secrets at the application level as described in the [defining secrets](#) section, you can reference them in volume mounts when you create a new revision in your container app. When you mount secrets in a volume, each secret is mounted as a file in the volume. The file name is the name of the secret, and the file contents are the value of the secret. You can load all secrets in a volume mount, or you can load specific secrets.

## Example

Azure portal

After you've [defined a secret](#) in your container app, you can reference it in a volume mount when you create a new revision.

1. Go to your container app in the [Azure portal](#).
2. Open the *Revision management* page.
3. Select **Create new revision**.
4. In the *Create and deploy new revision* page.

5. Select a container and select **Edit**.
6. In the *Volume mounts* section, expand the **Secrets** section.
7. Select **Create new volume**.
8. Enter the following information:
  - **Name:** mysecrets
  - **Mount all secrets:** enabled

 **Note**

If you want to load specific secrets, disable **Mount all secrets** and select the secrets you want to load.

9. Select **Add**.
10. Under *Volume name*, select **mysecrets**.
11. Under *Mount path*, enter **/mnt/secrets**.
12. Select **Save**.
13. Select **Create** to create the new revision with the volume mount.

## Next steps

[Containers](#)

# Enable an authentication token store in Azure Container Apps

Article • 04/09/2024

Azure Container Apps authentication supports a feature called token store. A token store is a repository of tokens that are associated with the users of your web apps and APIs. You enable a token store by configuring your container app with an Azure Blob Storage container.

Your application code sometimes needs to access data from these providers on the user's behalf, such as:

- Post to an authenticated user's Facebook timeline
- Read a user's corporate data using the Microsoft Graph API

You typically need to write code to collect, store, and refresh tokens in your application. With a token store, you can [retrieve tokens](#) when you need them, and [tell Container Apps to refresh them](#) as they become invalid.

When token store is enabled, the Container Apps authentication system caches ID tokens, access tokens, and refresh tokens the authenticated session, and they're accessible only by the associated user.

## Generate a SAS URL

Before you can create a token store for your container app, you first need an Azure Storage account with a private blob container.

1. Go to your storage account or [create a new one](#) in the Azure portal.
2. Select **Containers** and create a private blob container if necessary.
3. Select the three dots (•••) at the end of the row for the storage container where you want to create your token store.
4. Enter the values appropriate for your needs in the *Generate SAS* window.

Make sure you include the *read*, *write* and *delete* permissions in your definition.

 Note

Make sure you keep track of your SAS expiration dates to ensure access to your container doesn't cease.

5. Select the **Generate SAS token URL** button to generate the SAS URL.
6. Copy the SAS URL and paste it into a text editor for use in a following step.

## Save SAS URL as secret

With SAS URL generated, you can save it in your container app as a secret. Make sure the permissions associated with your store include valid permissions to your blob storage container.

1. Go to your container app in the Azure portal.
2. Select **Secrets**.
3. Select **Add** and enter the following values in the *Add secret* window.

[+] Expand table

Property	Value
Key	Enter a name for your SAS secret.
Type	Select <b>Container Apps secret</b> .
Value	Enter the SAS URL value you generated from your storage container.

## Create a token store

Use the `containerapp auth update` command to associate your Azure Storage account to your container app and create the token store.

In this example, you put your values in place of the placeholder tokens surrounded by `<>` brackets.

Azure CLI

```
az containerapp auth update \
--resource-group <RESOURCE_GROUP_NAME> \
--name <CONTAINER_APP_NAME> \
--sas-url-secret-name <SAS_SECRET_NAME> \
--token-store true
```

Additionally, you can create your store using an [ARM template](#).

## Next steps

[Customize sign in and sign out](#)

# Managed identities in Azure Container Apps

Article • 06/27/2024

A managed identity from Microsoft Entra ID allows your container app to access other Microsoft Entra protected resources. For more about managed identities in Microsoft Entra ID, see [Managed identities for Azure resources](#).

Your container app can be granted two types of identities:

- A **system-assigned identity** is tied to your container app and is deleted when your container app is deleted. An app can only have one system-assigned identity.
- A **user-assigned identity** is a standalone Azure resource that you can assign to your container app and other resources. A container app can have multiple user-assigned identities. User-assigned identities exist until you delete them.

## Why use a managed identity?

You can use a managed identity in a running container app to authenticate to any [service that supports Microsoft Entra authentication](#).

With managed identities:

- Your app connects to resources with the managed identity. You don't need to manage credentials in your container app.
- You can use role-based access control to grant specific permissions to a managed identity.
- System-assigned identities are automatically created and managed. They're deleted when your container app is deleted.
- You can add and delete user-assigned identities and assign them to multiple resources. They're independent of your container app's lifecycle.
- You can use managed identity to [authenticate with a private Azure Container Registry](#) without a username and password to pull containers for your container app.
- You can use a [managed identity to create connections for Dapr-enabled applications via Dapr components](#)

## Common use cases

System-assigned identities are best for workloads that:

- are contained within a single resource
- need independent identities

User-assigned identities are ideal for workloads that:

- run on multiple resources and can share a single identity
- need pre-authorization to a secure resource

## Limitations

[Init containers](#) can't access managed identities in [consumption-only environments](#) and [dedicated workload profile environments](#)

## Configure managed identities

You can configure your managed identities through:

- the Azure portal
- the Azure CLI
- your Azure Resource Manager (ARM) template

When a managed identity is added, deleted, or modified on a running container app, the app doesn't automatically restart and a new revision isn't created.

### ⓘ Note

When adding a managed identity to a container app deployed before April 11, 2022, you must create a new revision.

## Add a system-assigned identity

Azure portal

1. Go to your container app in the Azure portal.
2. From the *Settings* group, select **Identity**.
3. Within the *System assigned* tab, switch *Status* to **On**.
4. Select **Save**.

The screenshot shows the Azure portal interface for a Container App named 'music-store'. In the top navigation bar, there's a search bar and several icons for account management. Below the navigation, the app name 'music-store' is displayed with a key icon and the text 'Container App'. A breadcrumb trail shows 'Dashboard > music-store'. On the left, a sidebar lists various settings: Overview, Access control (IAM), Tags, Diagnose and solve problems, Secrets, Ingress, Continuous deployment, Identity (which is selected and highlighted in grey), and Locks. The main content area has tabs for 'System assigned' (selected) and 'User assigned'. A detailed description of system assigned identities follows. At the bottom of the main area are buttons for Save, Discard, Refresh, and Got feedback?.

## Add a user-assigned identity

Configuring a container app with a user-assigned identity requires that you first create the identity then add its resource identifier to your container app's configuration. You can create user-assigned identities via the Azure portal or the Azure CLI. For information on creating and managing user-assigned identities, see [Manage user-assigned managed identities](#).

Azure portal

First, you'll need to create a user-assigned identity resource.

1. Create a user-assigned managed identity resource according to the steps found in [Manage user-assigned managed identities](#).
2. Go to your container app in the Azure portal.
3. From the *Settings* group, select **Identity**.
4. Within the *User assigned* tab, select **Add**.
5. Search for and select the identity you created earlier.
6. Select **Add**.

The screenshot shows the Azure portal interface for a 'music-store' Container App. On the left, there's a sidebar with various settings like Overview, Access control (IAM), Tags, Diagnose and solve problems, and Identity (which is currently selected). The main area shows a table for 'User assigned' identities, which is currently empty. A modal window titled 'Add user assigned managed identity...' is open, allowing the user to select from a list of available identities. The 'music-store-user-identity' is selected and highlighted with a red box.

## Configure a target resource

For some resources, you need to configure role assignments for your app's managed identity to grant access. Otherwise, calls from your app to services, such as Azure Key Vault and Azure SQL Database, are rejected even when you use a valid token for that identity. To learn more about Azure role-based access control (Azure RBAC), see [What is RBAC?](#). To learn more about which resources support Microsoft Entra tokens, see [Azure services that support Microsoft Entra authentication](#).

### Important

The back-end services for managed identities maintain a cache per resource URI for around 24 hours. If you update the access policy of a particular target resource and immediately retrieve a token for that resource, you may continue to get a cached token with outdated permissions until that token expires. Forcing a token refresh isn't supported.

## Connect to Azure services in app code

With managed identities, an app can obtain tokens to access Azure resources that use Microsoft Entra ID, such as Azure SQL Database, Azure Key Vault, and Azure Storage. These tokens represent the application accessing the resource, and not any specific user of the application.

Container Apps provides an internally accessible [REST endpoint](#) to retrieve tokens. The REST endpoint is available from within the app with a standard HTTP `GET` request, which you can send with a generic HTTP client in your preferred language. For .NET, JavaScript, Java, and Python, the Azure Identity client library provides an abstraction over this REST endpoint. You can connect to other Azure services by adding a credential object to the service-specific client.

 **Note**

When using Azure Identity client library, you need to explicitly specify the user-assigned managed identity client ID.

.NET

 **Note**

When connecting to Azure SQL data sources with [Entity Framework Core](#), consider using [Microsoft.Data.SqlClient](#), which provides special connection strings for managed identity connectivity.

For .NET apps, the simplest way to work with a managed identity is through the [Azure Identity client library for .NET](#). See the following resources for more information:

- [Add Azure Identity client library to your project](#)
- [Access Azure service with a system-assigned identity](#)
- [Access Azure service with a user-assigned identity](#)

The linked examples use `DefaultAzureCredential`. This object is effective in most scenarios as the same pattern works in Azure (with managed identities) and on your local machine (without managed identities).

## Use managed identity for scale rules

You can use managed identities in your scale rules to authenticate with Azure services that support managed identities. To use a managed identity in your scale rule, use the `identity` property instead of the `auth` property in your scale rule. Acceptable values for the `identity` property are either the Azure resource ID of a user-assigned identity, or `system` to use a system-assigned identity.

## Note

Managed identity authentication in scale rules is in public preview. It's available in API version 2024-02-02-preview.

The following ARM template example shows how to use a managed identity with an Azure Queue Storage scale rule:

The queue storage account uses the `accountName` property to identify the storage account, while the `identity` property specifies which managed identity to use. You do not need to use the `auth` property.

JSON

```
"scale": {
 "minReplicas": 1,
 "maxReplicas": 10,
 "rules": [
 {"name": "myQueueRule",
 "azureQueue": {
 "accountName": "mystorageaccount",
 "queueName": "myqueue",
 "queueLength": 2,
 "identity": "<IDENTITY1_RESOURCE_ID>"
 }
 }]
}
```

To learn more about using managed identity with scale rules, see [Set scaling rules in Azure Container Apps](#).

## Control managed identity availability

Container Apps allows you to specify [init containers](#) and main containers. By default, both main and init containers in a consumption workload profile environment can use managed identity to access other Azure services. In consumption-only environments and dedicated workload profile environments, only main containers can use managed identity. Managed identity access tokens are available for every managed identity configured on the container app. However, in some situations only the init container or the main container require access tokens for a managed identity. Other times, you may use a managed identity only to access your Azure Container Registry to pull the container image, and your application itself doesn't need to have access to your Azure Container Registry.

Starting in API version 2024-02-02-preview, you can control which managed identities are available to your container app during the init and main phases to follow the security principle of least privilege. The following options are available:

- **Init**: Available only to init containers. Use this when you want to perform some initialization work that requires a managed identity, but you no longer need the managed identity in the main container. This option is currently only supported in [workload profile consumption environments](#)
  - **Main**: Available only to main containers. Use this if your init container does not need managed identity.
  - **All**: Available to all containers. This value is the default setting.
  - **None**: Not available to any containers. Use this when you have a managed identity that is only used for ACR image pull, scale rules, or Key Vault secrets and does not need to be available to the code running in your containers.

The following ARM template example shows how to configure a container app on a workload profile consumption environment that:

- Restricts the container app's system-assigned identity to main containers only.
  - Restricts a specific user-assigned identity to init containers only.
  - Uses a specific user-assigned identity for Azure Container Registry image pull without allowing the code in the containers to use that managed identity to access the registry. In this example, the containers themselves don't need to access the registry.

This approach limits the resources that can be accessed if a malicious actor were to gain unauthorized access to the containers.

JSON

```
{
 "location": "eastus2",
 "identity": {
 "type": "SystemAssigned, UserAssigned",
 "userAssignedIdentities": {
 "<IDENTITY1_RESOURCE_ID>": {},
 "<ACR_IMAGEPULL_IDENTITY_RESOURCE_ID>": {}
 }
 },
 "properties": {
 "workloadProfileName": "Consumption",
 "environmentId": "<CONTAINER_APPS_ENVIRONMENT_ID>",
 "configuration": {
 "registries": [
 {
 "server": "mvregistry.azurecr.io",
 "username": "mvregistry",
 "password": "XXXXXXXXXX"
 }
]
 }
 }
}
```

```
 "identity": "ACR_IMAGEPULL_IDENTITY_RESOURCE_ID"
 }],
 "identitySettings": [
 {
 "identity": "ACR_IMAGEPULL_IDENTITY_RESOURCE_ID",
 "lifecycle": "None"
 },
 {
 "identity": "<IDENTITY1_RESOURCE_ID>",
 "lifecycle": "Init"
 },
 {
 "identity": "system",
 "lifecycle": "Main"
 }
],
 "template": {
 "containers": [
 {
 "image": "myregistry.azurecr.io/main:1.0",
 "name": "app-main"
 }
],
 "initContainers": [
 {
 "image": "myregistry.azurecr.io/init:1.0",
 "name": "app-init"
 }
]
 }
}
```

## View managed identities

You can show the system-assigned and user-assigned managed identities using the following Azure CLI command. The output shows the managed identity type, tenant IDs and principal IDs of all managed identities assigned to your container app.

Azure CLI

```
az containerapp identity show --name <APP_NAME> --resource-group
<GROUP_NAME>
```

## Remove a managed identity

When you remove a system-assigned identity, it's deleted from Microsoft Entra ID. System-assigned identities are also automatically removed from Microsoft Entra ID

when you delete the container app resource itself. Removing user-assigned managed identities from your container app doesn't remove them from Microsoft Entra ID.

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**. Select **Save**.
  - **User-assigned identity**: Select the **User assigned** tab, select the checkbox for the identity, and select **Remove**. Select **Yes** to confirm.

## Next steps

[Monitor an app](#)

## Feedback

Was this page helpful?

[!\[\]\(f4abd6d2fb5de1e61e9586835841b427\_img.jpg\) Yes](#)

[!\[\]\(9de906d053ac9fa558c6ab7d60bf58ab\_img.jpg\) No](#)

[Provide product feedback ↗](#)

# Azure Container Apps image pull with managed identity

Article • 06/14/2024

You can pull images from private repositories in Microsoft Azure Container Registry using managed identities for authentication to avoid the use of administrative credentials.

You can use a user-assigned or system-assigned managed identity to authenticate with Azure Container Registry.

- With a user-assigned managed identity, you create and manage the identity outside of Azure Container Apps. It can be assigned to multiple Azure resources, including Azure Container Apps.
- With a system-assigned managed identity, the identity is created and managed by Azure Container Apps. It is tied to your container app and is deleted when your app is deleted.
- When possible, you should use a user-assigned managed identity to pull images.

Container Apps checks for a new version of the image whenever a container is started. In Docker or Kubernetes terminology, Container Apps sets each container's image pull policy to `always`.

This article describes how to use the Azure portal to configure your container app to use user-assigned and system-assigned managed identities to pull images from private Azure Container Registry repositories.

## User-assigned managed identity

The following steps describe the process to configure your container app to use a user-assigned managed identity to pull images from private Azure Container Registry repositories.

- Create a container app with a public image.
- Add the user-assigned managed identity to the container app.
- Create a container app revision with a private image and the user-assigned managed identity.

## Prerequisites

- An Azure account with an active subscription.
  - If you don't have one, you [can create one for free](#).
- A private Azure Container Registry containing an image you want to pull.
- Your Azure Container Registry must allow ARM audience tokens for authentication in order to use managed identity to pull images. Use the following command to check if ARM tokens are allowed to access your ACR:

Azure CLI

```
az acr config authentication-as-arm show -r <REGISTRY>
```

If ARM tokens are disallowed, you can allow them with the following command:

Azure CLI

```
az acr config authentication-as-arm update -r <REGISTRY> --status
enabled
```

- Create a user-assigned managed identity. For more information, see [Create a user-assigned managed identity](#).

## Create a container app

Use the following steps to create a container app with the default quickstart image.

1. Navigate to the portal **Home** page.
2. Search for **Container Apps** in the top search bar.
3. Select **Container Apps** in the search results.
4. Select the **Create** button.
5. In the *Basics* tab, do the following actions.

[ ] [Expand table](#)

Setting	Action
Subscription	Select your Azure subscription.
Resource group	Select an existing resource group or create a new one.

Setting	Action
Container app name	Enter a container app name.
Location	Select a location.
Create Container App Environment	Create a new or select an existing environment.

6. Select the **Review + Create** button at the bottom of the **Create Container App** page.

7. Select the **Create** button at the bottom of the **Create Container App** window.

Allow a few minutes for the container app deployment to finish. When deployment is complete, select **Go to resource**.

## Add the user-assigned managed identity

1. Select **Identity** from the left menu.
2. Select the **User assigned** tab.
3. Select the **Add user assigned managed identity** button.
4. Select your subscription.
5. Select the identity you created.
6. Select **Add**.

## Create a container app revision

Create a container app revision with a private image and the system-assigned managed identity.

1. Select **Revision Management** from the left menu.
2. Select **Create new revision**.
3. Select the container image from the **Container Image** table.
4. Enter the information in the *Edit a container* dialog.

 Expand table

Field	Action
Name	Enter a name for the container.

Field	Action
Image source	Select Azure Container Registry.
Authentication	Select Managed Identity.
Identity	Select the identity you created from the drop-down menu.
Registry	Select the registry you want to use from the drop-down menu.
Image	Enter the name of the image you want to use.
Image Tag	Enter the name and tag of the image you want to pull.

### Container details

You can change these settings after creating the Container App.

Name \*

Image source

 Azure Container Registry Docker Hub or other registries

Authentication

 Admin Credentials Managed Identity

Identity \*

Registry \*

(i) Failed to retrieve images for ACR 'containerappimages.azurecr.io' because admin credentials on the ACR are disabled. Please manually enter the image and tag below.

Image \*

Image tag \*

OS type

Linux

Command override (i)

Example: /bin/bash, -c, echo hello; sleep 10...

### Container resource allocation

**Save**

**Cancel**

#### ! Note

If the administrative credentials are not enabled on your Azure Container Registry registry, you will see a warning message displayed and you will need to enter the image name and tag information manually.

5. Select **Save**.

## 6. Select Create from the Create and deploy new revision page.

A new revision will be created and deployed. The portal will automatically attempt to add the `acrpull` role to the user-assigned managed identity. If the role isn't added, you can add it manually.

You can verify that the role was added by checking the identity from the **Identity** pane of the container app page.

1. Select **Identity** from the left menu.
2. Select the **User assigned** tab.
3. Select the user-assigned managed identity.
4. Select **Azure role assignments** from the menu on the managed identity resource page.
5. Verify that the `acrpull` role is assigned to the user-assigned managed identity.

## Create a container app with a private image

If you don't want to start by creating a container app with a public image, you can also do the following.

1. Create a user-assigned managed identity.
2. Add the `acrpull` role to the user-assigned managed identity.
3. Create a container app with a private image and the user-assigned managed identity.

This method is typical in Infrastructure as Code (IaC) scenarios.

## Clean up resources

If you're not going to continue to use this application, you can delete the Azure Container Apps instance and all the associated services by removing the resource group.

### Warning

Deleting the resource group will delete all the resources in the group. If you have other resources in the group, they will also be deleted. If you want to keep the resources, you can delete the container app instance and the container app environment.

1. Select your resource group from the **Overview** section.

2. Select the **Delete resource group** button at the top of the resource group *Overview*.
3. Enter the resource group name in the confirmation dialog.
4. Select **Delete**. The process to delete the resource group may take a few minutes to complete.

## System-assigned managed identity

The method for configuring a system-assigned managed identity in the Azure portal is the same as configuring a user-assigned managed identity. The only difference is that you don't need to create a user-assigned managed identity. Instead, the system-assigned managed identity is created when you create the container app.

The method to configure a system-assigned managed identity in the Azure portal is:

1. Create a container app with a public image.
2. Create a container app revision with a private image and the system-assigned managed identity.

## Prerequisites

- An Azure account with an active subscription.
  - If you don't have one, you [can create one for free](#) ↗.
- A private Azure Container Registry containing an image you want to pull. See [Create a private Azure Container Registry](#).

## Create a container app

Follow these steps to create a container app with the default quickstart image.

1. Navigate to the portal **Home** page.
2. Search for **Container Apps** in the top search bar.
3. Select **Container Apps** in the search results.
4. Select the **Create** button.
5. In the **Basics** tab, do the following actions.

[ ] [Expand table](#)

Setting	Action
Subscription	Select your Azure subscription.
Resource group	Select an existing resource group or create a new one.
Container app name	Enter a container app name.
Location	Select a location.
Create Container App Environment	Create a new or select an existing environment.

6. Select the **Review + Create** button at the bottom of the **Create Container App** page.

7. Select the **Create** button at the bottom of the **Create Container App** page.

Allow a few minutes for the container app deployment to finish. When deployment is complete, select **Go to resource**.

## Edit and deploy a revision

Edit the container to use the image from your private Azure Container Registry, and configure the authentication to use system-assigned identity.

1. The **Containers** from the side menu on the left.
2. Select **Edit and deploy**.
3. Select the *simple-hello-world-container* container from the list.

[ ] Expand table

Setting	Action
Name	Enter the container app name.
Image source	Select <b>Azure Container Registry</b> .
Authentication	Select <b>Managed identity</b> .
Identity	Select <b>System assigned</b> .
Registry	Enter the Registry name.
Image	Enter the image name.

Setting	Action
Image tag	Enter the tag.

## Edit a container

X

Basics    Health probes

### Container details

You can change these settings after creating the Container App.

Name \*

my-container-app

Image source

Azure Container Registry

Docker Hub or other registries

Authentication

Admin Credentials

Managed Identity

Identity \*

System assigned

Registry \*

containerappimages.azurecr.io

i Failed to retrieve images for ACR 'containerappimages.azurecr.io' because admin credentials on the ACR are disabled. Please manually enter the image and tag below.

Image \*

container-app

Image tag \*

latest

OS type

Linux

Command override i

Example: /bin/bash, -c, echo hello; sleep 10...

### Container resource allocation

Save

Cancel

! Note

If the administrative credentials are not enabled on your Azure Container Registry registry, you will see a warning message displayed and you will need to enter the image name and tag information manually.

4. Select **Save** at the bottom of the page.
5. Select **Create** at the bottom of the **Create and deploy new revision** page
6. After a few minutes, select **Refresh** on the **Revision management** page to see the new revision.

A new revision will be created and deployed. The portal will automatically attempt to add the `acrpull` role to the system-assigned managed identity. If the role isn't added, you can add it manually.

You can verify that the role was added by checking the identity in the **Identity** pane of the container app page.

1. Select **Identity** from the left menu.
2. Select the **System assigned** tab.
3. Select **Azure role assignments**.
4. Verify that the `acrpull` role is assigned to the system-assigned managed identity.

## Clean up resources

If you're not going to continue to use this application, you can delete the Azure Container Apps instance and all the associated services by removing the resource group.

### Warning

Deleting the resource group will delete all the resources in the group. If you have other resources in the group, they will also be deleted. If you want to keep the resources, you can delete the container app instance and the container app environment.

1. Select your resource group from the **Overview** section.
2. Select the **Delete resource group** button at the top of the resource group **Overview**.
3. Enter the resource group name in the confirmation dialog.
4. Select **Delete**. The process to delete the resource group may take a few minutes to complete.

# Next steps

[Managed identities in Azure Container Apps](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#)

# Service discovery resiliency (preview)

Article • 08/02/2024

With Azure Container Apps resiliency, you can proactively prevent, detect, and recover from service request failures using simple resiliency policies. In this article, you learn how to configure Azure Container Apps resiliency policies when initiating requests using Azure Container Apps service discovery.

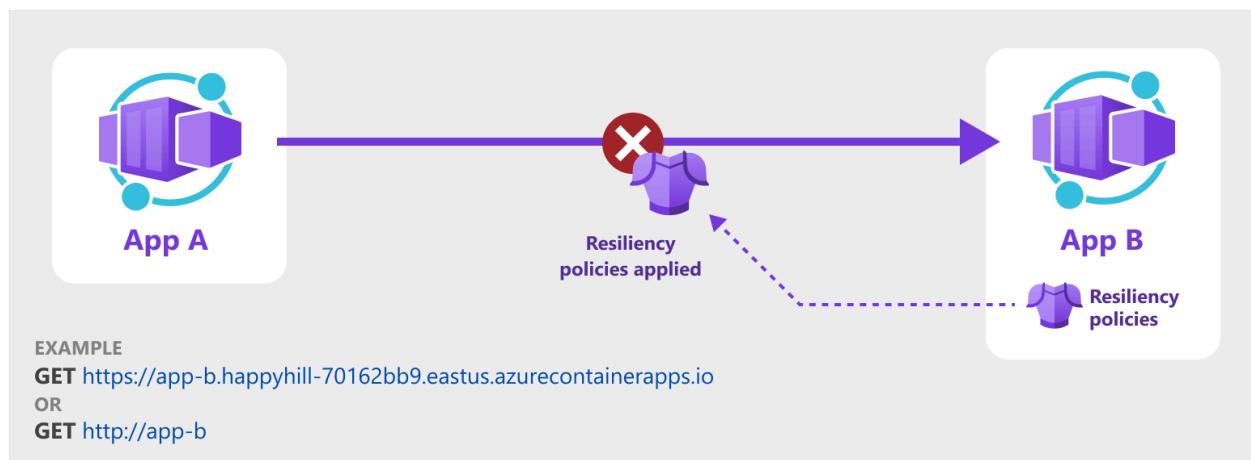
## ⚠ Note

Currently, resiliency policies can't be applied to requests made using the Dapr Service Invocation API.

Policies are in effect for each request to a container app. You can tailor policies to the container app accepting requests with configurations like:

- The number of retries
- Retry and timeout duration
- Retry matches
- Circuit breaker consecutive errors, and others

The following screenshot shows how an application uses a retry policy to attempt to recover from failed requests.



## Supported resiliency policies

- Timeouts
- Retries (HTTP and TCP)
- Circuit breakers
- Connection pools (HTTP and TCP)

# Configure resiliency policies

Whether you configure resiliency policies using Bicep, the CLI, or the Azure portal, you can only apply one policy per container app.

When you apply a policy to a container app, the rules are applied to all requests made to that container app, *not* to requests made from that container app. For example, a retry policy is applied to a container app named `App B`. All inbound requests made to App B automatically retry on failure. However, outbound requests sent by App B aren't guaranteed to retry in failure.

Bicep

The following resiliency example demonstrates all of the available configurations.

Bicep

```
resource myPolicyDoc
'Microsoft.App/containerApps/resiliencyPolicies@2023-11-02-preview' = {
 name: 'my-app-resiliency-policies'
 parent: '${appName}'
 properties: {
 timeoutPolicy: {
 responseTimeoutInSeconds: 15
 connectionTimeoutInSeconds: 5
 }
 httpRetryPolicy: {
 maxRetries: 5
 retryBackOff: {
 initialDelayInMilliseconds: 1000
 maxIntervalInMilliseconds: 10000
 }
 }
 matches: [
 headers: [
 {
 header: 'x-ms-retriable'
 match: {
 exactMatch: 'true'
 }
 }
]
 httpStatusCodes: [
 502
 503
]
 errors: [
 'retriable-status-codes'
 '5xx'
 'reset'
 'connect-failure'
]
 }
}
```

```

 'retriable-4xx'
]
}
tcpRetryPolicy: {
 maxConnectAttempts: 3
}
circuitBreakerPolicy: {
 consecutiveErrors: 5
 intervalInSeconds: 10
 maxEjectionPercent: 50
}
tcpConnectionPool: {
 maxConnections: 100
}
httpConnectionPool: {
 http1MaxPendingRequests: 1024
 http2MaxRequests: 1024
}
}
}
}

```

## Policy specifications

### Timeouts

Timeouts are used to early-terminate long-running operations. The timeout policy includes the following properties.

Bicep

```

properties: {
 timeoutPolicy: {
 responseTimeoutInSeconds: 15
 connectionTimeoutInSeconds: 5
 }
}

```

[Expand table](#)

Metadata	Required	Description	Example
<code>responseTimeoutInSeconds</code>	Yes	Timeout waiting for a response from the container app.	15
<code>connectionTimeoutInSeconds</code>	Yes	Timeout to establish a connection to the container app.	5

# Retries

Define a `tcpRetryPolicy` or an `httpRetryPolicy` strategy for failed operations. The retry policy includes the following configurations.

## httpRetryPolicy

```
Bicep

properties: {
 httpRetryPolicy: {
 maxRetries: 5
 retryBackOff: {
 initialDelayInMilliseconds: 1000
 maxIntervalInMilliseconds: 10000
 }
 matches: {
 headers: [
 {
 header: 'x-ms-retriable'
 match: {
 exactMatch: 'true'
 }
 }
]
 httpStatusCodes: [
 502
 503
]
 errors: [
 'retriable-headers'
 'retriable-status-codes'
]
 }
 }
}
```

[Expand table](#)

Metadata	Required	Description	Example
<code>maxRetries</code>	Yes	Maximum retries to be executed for a failed http-request.	5
<code>retryBackOff</code>	Yes	Monitor the requests and shut off all traffic to the impacted service	N/A

Metadata	Required	Description	Example
		when timeout and retry criteria are met.	
<code>retryBackOff.initialDelayInMilliseconds</code>	Yes	Delay between first error and first retry.	<code>1000</code>
<code>retryBackOff.maxIntervalInMilliseconds</code>	Yes	Maximum delay between retries.	<code>10000</code>
<code>matches</code>	Yes	Set match values to limit when the app should attempt a retry.	<code>headers</code> , <code>httpStatusCodes</code> , <code>errors</code>
<code>matches.headers</code>	Y*	Retry when the error response includes a specific header. *Headers are only required properties if you specify the <code>retriable-headers</code> error property. <a href="#">Learn more about available header matches.</a>	<code>X-Content-Type</code>
<code>matches.httpStatusCodes</code>	Y*	Retry when the response returns a specific status code. *Status codes are only required properties if you specify the <code>retriable-status-codes</code> error property.	<code>502, 503</code>
<code>matches.errors</code>	Yes	Only retries when the app returns a specific error. <a href="#">Learn more about available errors.</a>	<code>connect-failure, reset</code>

## Header matches

If you specified the `retriable-headers` error, you can use the following header match properties to retry when the response includes a specific header.

```
Bicep

matches: {
 headers: [
 {
 header: 'x-ms-retriable'
 match: {
 exactMatch: 'true'
 }
 }
]
}
```

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

Metadata	Description
<code>prefixMatch</code>	Retries are performed based on the prefix of the header value.
<code>exactMatch</code>	Retries are performed based on an exact match of the header value.
<code>suffixMatch</code>	Retries are performed based on the suffix of the header value.
<code>regexMatch</code>	Retries are performed based on a regular expression rule where the header value must match the regex pattern.

## Errors

You can perform retries on any of the following errors:

```
Bicep

matches: {
 errors: [
 'retriable-headers'
 'retriable-status-codes'
 '5xx'
 'reset'
 'connect-failure'
 'retriable-4xx'
]
}
```

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

Metadata	Description
retriable-headers	HTTP response headers that trigger a retry. A retry is performed if any of the header-matches match the response headers. Required if you'd like to retry on any matching headers.
retriable-status-codes	HTTP status codes that should trigger retries. Required if you'd like to retry on any matching status codes.
5xx	Retry if server responds with any 5xx response codes.
reset	Retry if the server doesn't respond.
connect-failure	Retry if a request failed due to a faulty connection with the container app.
retriable-4xx	Retry if the container app responds with a 400-series response code, like 409.

## tcpRetryPolicy

Bicep

```
properties: {
 tcpRetryPolicy: {
 maxConnectAttempts: 3
 }
}
```

[Expand table](#)

Metadata	Required	Description	Example
maxConnectAttempts	Yes	Set the maximum connection attempts ( <code>maxConnectionAttempts</code> ) to retry on failed connections.	<a href="#">3</a>

## Circuit breakers

Circuit breaker policies specify whether a container app replica is temporarily removed from the load balancing pool, based on triggers like the number of consecutive errors.

Bicep

```
properties: {
 circuitBreakerPolicy: {
 consecutiveErrors: 5
 }
}
```

```

 intervalInSeconds: 10
 maxEjectionPercent: 50
 }
}

```

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

Metadata	Required	Description	Example
consecutiveErrors	Yes	Consecutive number of errors before a container app replica is temporarily removed from load balancing.	5
intervalInSeconds	Yes	The amount of time given to determine if a replica is removed or restored from the load balance pool.	10
maxEjectionPercent	Yes	Maximum percent of failing container app replicas to eject from load balancing. Removes at least one host regardless of the value.	50

## Connection pools

Azure Container App's connection pooling maintains a pool of established and reusable connections to container apps. This connection pool reduces the overhead of creating and tearing down individual connections for each request.

Connection pools allow you to specify the maximum number of requests or connections allowed for a service. These limits control the total number of concurrent connections for each service. When this limit is reached, new connections aren't established to that service until existing connections are released or closed. This process of managing connections prevents resources from being overwhelmed by requests and maintains efficient connection management.

## httpConnectionPool

Bicep

```

properties: {
 httpConnectionPool: {
 http1MaxPendingRequests: 1024
 http2MaxRequests: 1024
 }
}

```

[\[\] Expand table](#)

Metadata	Required	Description	Example
<code>http1MaxPendingRequests</code>	Yes	Used for <code>http1</code> requests. Maximum number of open connections to a container app.	<code>1024</code>
<code>http2MaxRequests</code>	Yes	Used for <code>http2</code> requests. Maximum number of concurrent requests to a container app.	<code>1024</code>

## tcpConnectionPool

Bicep

```
properties: {
 tcpConnectionPool: {
 maxConnections: 100
 }
}
```

[\[\] Expand table](#)

Metadata	Required	Description	Example
<code>maxConnections</code>	Yes	Maximum number of concurrent connections to a container app.	<code>100</code>

## Resiliency observability

You can perform resiliency observability via your container app's metrics and system logs.

## Resiliency logs

From the *Monitoring* section of your container app, select **Logs**.



# test-app

Container App



Search

«



Scale and replicas



## Settings

---



Authentication



Secrets



Ingress



Continuous deployment



Custom domains



Dapr



Identity



Service Connector (preview)



CORS



Resiliency (preview)



Locks

## Monitoring

---



Alerts



Metrics



Logs



Log stream



Console



Advisor recommendations

In the Logs pane, write and run a query to find resiliency via your container app system logs. For example, run a query similar to the following to search for resiliency events and show their:

- Time stamp
- Environment name
- Container app name
- Resiliency type and reason
- Log messages

```
ContainerAppSystemLogs_CL
| where EventSource_s == "Resiliency"
| project TimeStamp_s, EnvironmentName_s, ContainerAppName_s, Type_s,
EventSource_s, Reason_s, Log_s
```

Click Run to run the query and view results.

The screenshot shows the Azure Log Analytics workspace interface. At the top, there's a header bar with 'New Query 1\*' and other navigation links like 'Feedback', 'Queries', and 'Alerts'. Below the header, the query editor contains the following code:

```
ContainerAppSystemLogs_CL
| where EventSource_s == "Resiliency"
| project TimeStamp_s, EnvironmentName_s, ContainerAppName_s, Type_s,
EventSource_s, Reason_s, Log_s
```

Below the query editor, the results pane displays a table with the following data:

TimeStamp_s	EnvironmentName_s	ContainerAppName_s	Type_s	EventSource_s	Reason_s	Log_s
> 2023-11-06 20:40:06 +0000 UTC	yellowpebble-3c2e9044	flaky-api-test	Normal	Resiliency	ResiliencyUpdate	Container App 'test-app' has applied resiliency ('target')
> 2023-11-06 20:40:06 +0000 UTC	yellowpebble-3c2e9044	flaky-api-test	Normal	Resiliency	ResiliencyUpdate	Container App 'test-app' has applied resiliency ('target')
> 2023-11-06 18:12:13 +0000 UTC	yellowpebble-3c2e9044	flaky-api-test	Normal	Resiliency	ResiliencyUpdate	Container App 'test-app' has applied resiliency ('target')
> 2023-11-06 18:12:13 +0000 UTC	yellowpebble-3c2e9044	flaky-api-test	Normal	Resiliency	ResiliencyUpdate	Container App 'test-app' has applied resiliency ('target')
> 2023-11-06 18:12:13 +0000 UTC	yellowpebble-3c2e9044	flaky-api-test	Normal	Resiliency	ResiliencyUpdate	Container App 'test-app' has applied resiliency ('target')
> 2023-11-06 18:12:13 +0000 UTC	yellowpebble-3c2e9044	flaky-api-test	Normal	Resiliency	ResiliencyUpdate	Container App 'test-app' has applied resiliency ('target')
> 2023-11-06 18:12:13 +0000 UTC	yellowpebble-3c2e9044	flaky-api-test	Normal	Resiliency	ResiliencyUpdate	Container App 'test-app' has applied resiliency ('target')
> 2023-11-06 18:12:13 +0000 UTC	yellowpebble-3c2e9044	flaky-api-test	Normal	Resiliency	ResiliencyUpdate	Container App 'test-app' has applied resiliency ('target')
> 2023-11-06 18:12:17 +0000 UTC	yellowpebble-3c2e9044	flaky-api-test	Normal	Resiliency	ResiliencyUpdate	Container App 'test-app' has applied resiliency ('target')
> 2023-11-06 18:12:17 +0000 UTC	yellowpebble-3c2e9044	flaky-api-test	Normal	Resiliency	ResiliencyUpdate	Container App 'test-app' has applied resiliency ('target')
> 2023-11-06 18:12:38 +0000 UTC	yellowpebble-3c2e9044	flaky-api-test	Normal	Resiliency	ResiliencyUpdate	Container App 'test-app' has applied resiliency ('target')
> 2023-11-06 18:13:08 +0000 UTC	yellowpebble-3c2e9044	flaky-api-test	Normal	Resiliency	ResiliencyUpdate	Container App 'test-app' has applied resiliency ('target')
> 2023-11-06 18:13:08 +0000 UTC	yellowpebble-3c2e9044	flaky-api-test	Normal	Resiliency	ResiliencyUpdate	Container App 'test-app' has applied resiliency ('target')
> 2023-11-06 18:13:08 +0000 UTC	yellowpebble-3c2e9044	flaky-api-test	Normal	Resiliency	ResiliencyUpdate	Container App 'test-app' has applied resiliency ('target')

At the bottom of the results pane, there are status indicators: '0s 636ms | Display time (UTC+00:00)'. On the right side, there are 'Query details' and '1 - 15 of 28'.

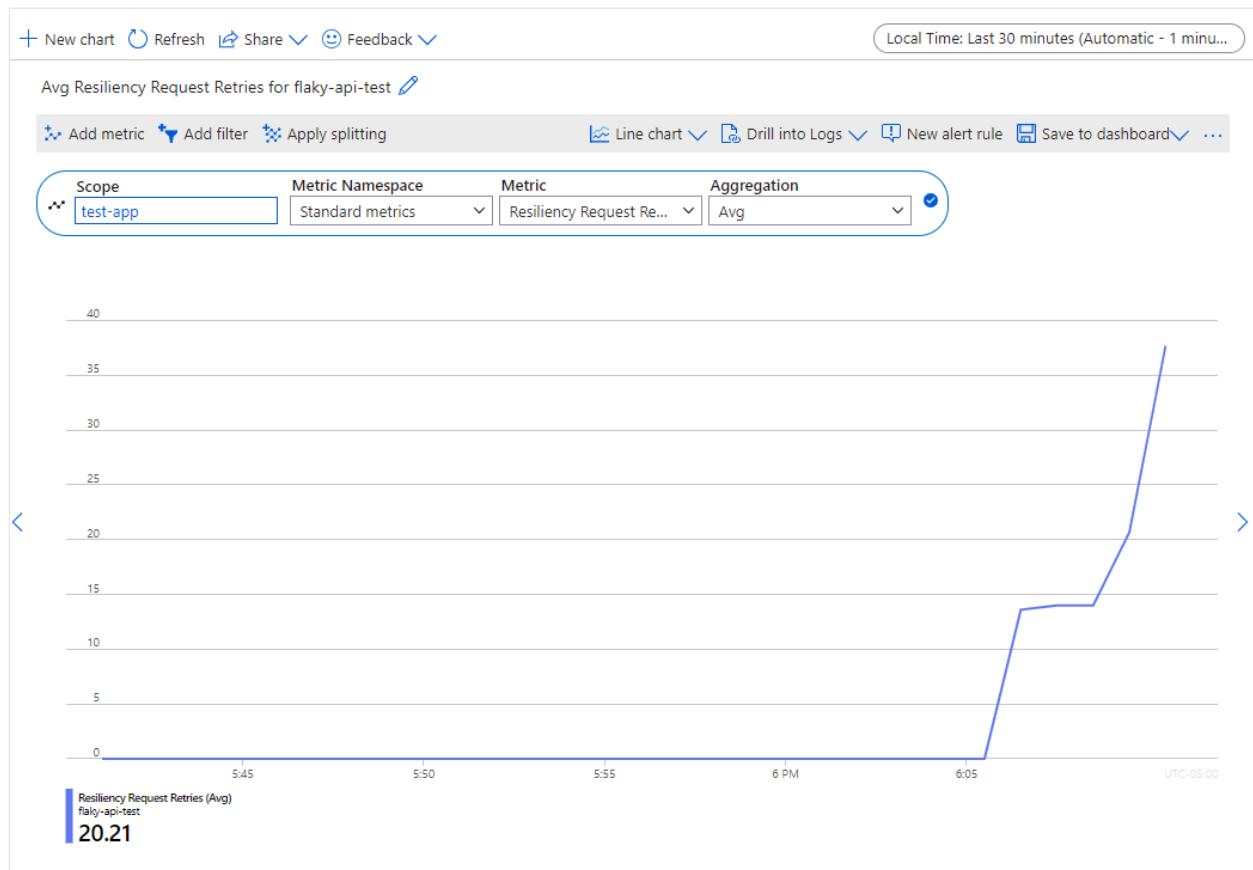
## Resiliency metrics

From the *Monitoring* menu of your container app, select **Metrics**. In the Metrics pane, select the following filters:

- The scope to the name of your container app.
- The **Standard metrics** metrics namespace.
- The resiliency metrics from the drop-down menu.
- How you'd like the data aggregated in the results (by average, by maximum, etc.).
- The time duration (last 30 minutes, last 24 hours, etc.).

The screenshot shows the Azure Container App Metrics pane for a 'test-app' container app. The left sidebar lists various monitoring options under 'Monitoring', with 'Metrics' selected and highlighted with a red box. The main area shows the configuration for a chart. A red box highlights the 'Scope' dropdown set to 'test-app' and the 'Metric Namespace' dropdown set to 'Standard metrics'. A second red box highlights the 'Metric' dropdown menu, which is open and shows several resiliency-related metrics: 'Resiliency Connection Timeouts', 'Resiliency Ejected Hosts', 'Resiliency Request Retries', 'Resiliency Request Timeouts', and 'Resiliency Requests Pending Connectio...'. The 'Resiliency Connection Timeouts' option is selected. The top right of the pane shows a status bar with 'Local Time: Last 24 hours (Automatic)'.

For example, if you set the *Resiliency Request Retries* metric in the *test-app* scope with *Average* aggregation to search within a 30-minute timeframe, the results look like the following:



## Related content

See how resiliency works for [Dapr components in Azure Container Apps](#).

## Feedback

Was this page helpful?

[Yes](#)

[No](#)

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

# Dapr component resiliency (preview)

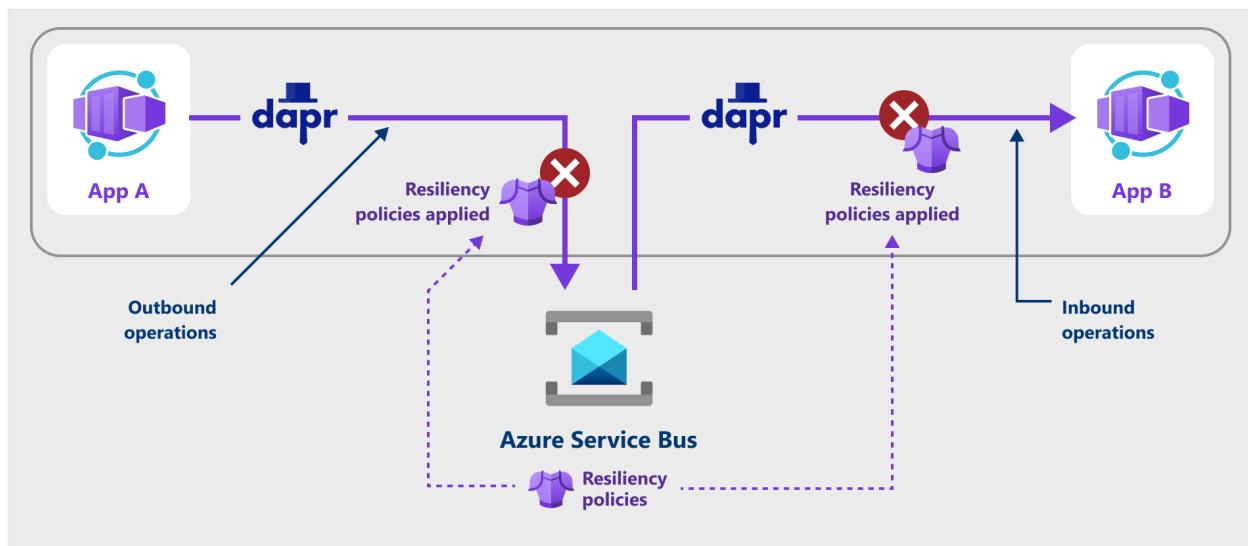
Article • 08/02/2024

Resiliency policies proactively prevent, detect, and recover from your container app failures. In this article, you learn how to apply resiliency policies for applications that use Dapr to integrate with different cloud services, like state stores, pub/sub message brokers, secret stores, and more.

You can configure resiliency policies like retries, timeouts, and circuit breakers for the following outbound and inbound operation directions via a Dapr component:

- **Outbound operations:** Calls from the Dapr sidecar to a component, such as:
  - Persisting or retrieving state
  - Publishing a message
  - Invoking an output binding
- **Inbound operations:** Calls from the Dapr sidecar to your container app, such as:
  - Subscriptions when delivering a message
  - Input bindings delivering an event

The following screenshot shows how an application uses a retry policy to attempt to recover from failed requests.



## Supported resiliency policies

- [Timeouts](#)
- [Retries \(HTTP\)](#)
- [Circuit breakers](#)

# Configure resiliency policies

You can choose whether to create resiliency policies using Bicep, the CLI, or the Azure portal.

Bicep

The following resiliency example demonstrates all of the available configurations.

Bicep

```
resource myPolicyDoc
'Microsoft.App/managedEnvironments/daprComponents/resiliencyPolicies@202
3-11-02-preview' = {
 name: 'my-component-resiliency-policies'
 parent: '${componentName}'
 properties: {
 outboundPolicy: {
 timeoutPolicy: {
 responseTimeoutInSeconds: 15
 }
 httpRetryPolicy: {
 maxRetries: 5
 retryBackOff: {
 initialDelayInMilliseconds: 1000
 maxIntervalInMilliseconds: 10000
 }
 }
 circuitBreakerPolicy: {
 intervalInSeconds: 15
 consecutiveErrors: 10
 timeoutInSeconds: 5
 }
 }
 inboundPolicy: {
 timeoutPolicy: {
 responseTimeoutInSeconds: 15
 }
 httpRetryPolicy: {
 maxRetries: 5
 retryBackOff: {
 initialDelayInMilliseconds: 1000
 maxIntervalInMilliseconds: 10000
 }
 }
 circuitBreakerPolicy: {
 intervalInSeconds: 15
 consecutiveErrors: 10
 timeoutInSeconds: 5
 }
 }
 }
}
```

```
}
```

### ⓘ Important

Once you've applied all the resiliency policies, you need to restart your Dapr applications.

## Policy specifications

### Timeouts

Timeouts are used to early-terminate long-running operations. The timeout policy includes the following properties.

Bicep

```
properties: {
 outbound: {
 timeoutPolicy: {
 responseTimeoutInSeconds: 15
 }
 }
 inbound: {
 timeoutPolicy: {
 responseTimeoutInSeconds: 15
 }
 }
}
```

[\[\] Expand table](#)

Metadata	Required	Description	Example
<code>responseTimeoutInSeconds</code>	Yes	Timeout waiting for a response from the Dapr component.	<code>15</code>

### Retries

Define an `httpRetryPolicy` strategy for failed operations. The retry policy includes the following configurations.

## Bicep

```
properties: {
 outbound: {
 httpRetryPolicy: {
 maxRetries: 5
 retryBackOff: {
 initialDelayInMilliseconds: 1000
 maxIntervalInMilliseconds: 10000
 }
 }
 }
 inbound: {
 httpRetryPolicy: {
 maxRetries: 5
 retryBackOff: {
 initialDelayInMilliseconds: 1000
 maxIntervalInMilliseconds: 10000
 }
 }
 }
}
```

[Expand table](#)

Metadata	Required	Description	Example
<code>maxRetries</code>	Yes	Maximum retries to be executed for a failed http-request.	5
<code>retryBackOff</code>	Yes	Monitor the requests and shut off all traffic to the impacted service when timeout and retry criteria are met.	N/A
<code>retryBackOff.initialDelayInMilliseconds</code>	Yes	Delay between first error and first retry.	1000
<code>retryBackOff.maxIntervalInMilliseconds</code>	Yes	Maximum delay between retries.	10000

## Circuit breakers

Define a `circuitBreakerPolicy` to monitor requests causing elevated failure rates and shut off all traffic to the impacted service when a certain criteria is met.

## Bicep

```

properties: {
 outbound: {
 circuitBreakerPolicy: {
 intervalInSeconds: 15
 consecutiveErrors: 10
 timeoutInSeconds: 5
 }
 },
 inbound: {
 circuitBreakerPolicy: {
 intervalInSeconds: 15
 consecutiveErrors: 10
 timeoutInSeconds: 5
 }
 }
}

```

[ ] [Expand table](#)

Metadata	Required	Description	Example
<code>intervalInSeconds</code>	No	Cyclical period of time (in seconds) used by the circuit breaker to clear its internal counts. If not provided, the interval is set to the same value as provided for <code>timeoutInSeconds</code> .	15
<code>consecutiveErrors</code>	Yes	Number of request errors allowed to occur before the circuit trips and opens.	10
<code>timeoutInSeconds</code>	Yes	Time period (in seconds) of open state, directly after failure.	5

## Circuit breaker process

Specifying `consecutiveErrors` (the circuit trip condition as `consecutiveFailures > $(consecutiveErrors)-1`) sets the number of errors allowed to occur before the circuit trips and opens halfway.

The circuit waits half-open for the `timeoutInSeconds` amount of time, during which the `consecutiveErrors` number of requests must consecutively succeed.

- *If the requests succeed*, the circuit closes.
- *If the requests fail*, the circuit remains in a half-opened state.

If you didn't set any `intervalInSeconds` value, the circuit resets to a closed state after the amount of time you set for `timeoutInSeconds`, regardless of consecutive request

success or failure. If you set `intervalInSeconds` to `0`, the circuit never automatically resets, only moving from half-open to closed state by successfully completing `consecutiveErrors` requests in a row.

If you did set an `intervalInSeconds` value, that determines the amount of time before the circuit is reset to closed state, independent of whether the requests sent in half-opened state succeeded or not.

## Resiliency logs

From the *Monitoring* section of your container app, select **Logs**.



# stageapp

Container App



Search



Scale and replicas

## Settings

---

Authentication

Secrets

Ingress

Continuous deployment

Custom domains

Dapr

Identity

Service Connector (preview)

CORS

Resiliency (preview)

Locks

## Monitoring

---

Alerts

Metrics

Logs

Log stream

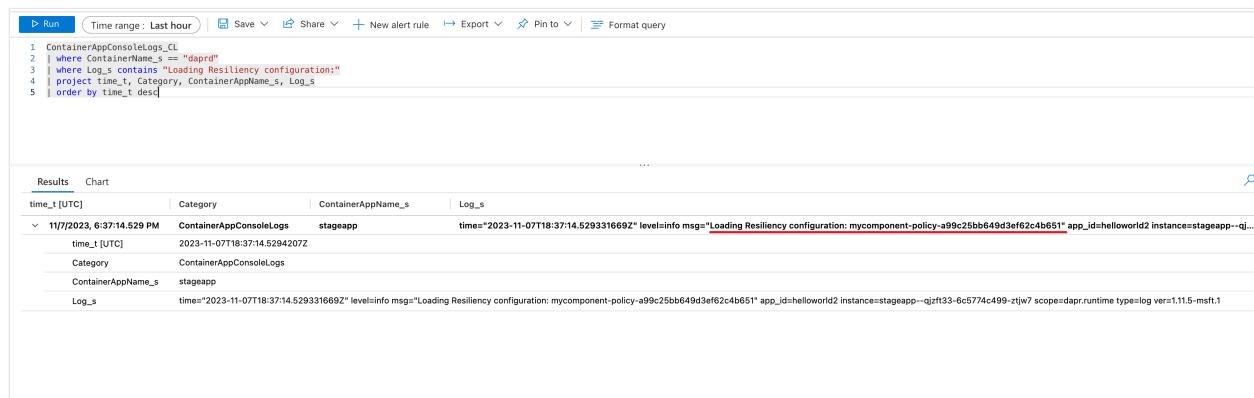
 Console

 Advisor recommendations

In the Logs pane, write and run a query to find resiliency via your container app system logs. For example, to find whether a resiliency policy was loaded:

```
ContainerAppConsoleLogs_CL
| where ContainerName_s == "daprd"
| where Log_s contains "Loading Resiliency configuration:"
| project time_t, Category, ContainerAppName_s, Log_s
| order by time_t desc
```

Click **Run** to run the query and view the result with the log message indicating the policy is loading.



The screenshot shows the Azure portal's Logs pane. At the top, there is a toolbar with buttons for Run, Save, Share, New alert rule, Export, Pin to, and Format query. The Time range is set to Last hour. Below the toolbar, the Kusto query is displayed:

```
ContainerAppConsoleLogs_CL
| where ContainerName_s == "daprd"
| where Log_s contains "Loading Resiliency configuration:"
| project time_t, Category, ContainerAppName_s, Log_s
| order by time_t desc
```

Below the query, there are two tabs: Results and Chart. The Results tab is selected, showing a table with four columns: time\_t [UTC], Category, ContainerAppName\_s, and Log\_s. One log entry is visible:

time_t [UTC]	Category	ContainerAppName_s	Log_s
11/7/2023, 6:37:14.529 PM	ContainerAppConsoleLogs	stageapp	time=2023-11-07T18:37:14.529331669Z level=info msg="Loading Resiliency configuration: mycomponent-policy-a99c25bb649d3ef62c4b651* app_id=helloworld2 instance=stageapp--qjzft33-6c5774c499-ztjw7 scope=dapr.runtime type=log ver=1.11.6-msft.1"

Or, you can find the actual resiliency policy by enabling debugging on your component and using a query similar to the following example:

```
ContainerAppConsoleLogs_CL
| where ContainerName_s == "daprd"
| where Log_s contains "Resiliency configuration (""
| project time_t, Category, ContainerAppName_s, Log_s
| order by time_t desc
```

Click **Run** to run the query and view the resulting log message with the policy configuration.

The screenshot shows the Azure Log Analytics interface with a query results table. The table has columns: time\_1 [UTC], Category, ContainerAppName\_s, and Log\_s. The table contains two rows of log entries. The first row is expanded to show its details.

```

1 ContainerAppConsoleLogs_CL
2 | where ContainerName_s == "daprd"
3 | where Log_s contains "Resiliency configuration"
4 | project time_t, Category, ContainerAppName_s, Log_s
5 | order by time_t desc

```

time_1 [UTC]	Category	ContainerAppName_s	Log_s
11/7/2023, 6:41:45.511 ...	ContainerAppConsoleLogs	stageapp	time="2023-11-07T18:41:45.511590713Z" level=debug msg="Resiliency configuration (mycomponent-policy-a99c25bb649d3ef62c4b651): {"kind":"Resiliency","apiVersion":"dapr.io/v1alpha1","resourceVersion":"10742096","generation":1,"creationTimestamp":"2023-11-07T18:36:17Z","ownerReferences":[{"apiVersion":"k8s.microsoft.com/v1alpha1","kind":"DaprComponentResiliency","name":"mycomponent-policy-a99c25bb649d3ef62c4b651","blockOwnerDeletion":true}],"managedFields":[{"manager":"containerapps","operation":"Update","apiVersion":"dapr.io/v1alpha1","time":2023-11-07T18:36:17Z}],"fieldsType":{"FieldSetV1":{"fieldsV1":{"metadata":{},"ownerReferences":{},"kafka":{},"policies":{},"timeouts":{},"mycomponent-timeout-inbound":{},"targets":{},"components":{},"f-mycomponent":{},"f-inbound":{},"f-outbound":{}}},"spec":{"policies":{},"timeouts":{},"mycomponent-timeout-inbound":{},"outbound":{}}}}} app_id=helloworld2 instance=stageapp--qjft33-640f899c87-h7dq scope=dapr.runtime type=log ver=11.5-mst-1
> 11/7/2023, 6:41:00.718 PM	ContainerAppConsoleLogs	stageapp	time="2023-11-07T18:41:00.718468847Z" level=debug msg="Resiliency configuration (mycomponent-policy-a99c25bb649d3ef62c4b651): {"kind":"Resiliency","apiVersion":"dapr.io/v1alpha1","resourceVersion":"10742096","generation":1,"creationTimestamp":"2023-11-07T18:36:17Z","ownerReferences":[{"apiVersion":"k8s.microsoft.com/v1alpha1","kind":"DaprComponentResiliency","name":"mycomponent-policy-a99c25bb649d3ef62c4b651","blockOwnerDeletion":true}],"managedFields":[{"manager":"containerapps","operation":"Update","apiVersion":"dapr.io/v1alpha1","time":2023-11-07T18:36:17Z}],"fieldsType":{"FieldSetV1":{"fieldsV1":{"metadata":{},"ownerReferences":{},"kafka":{},"policies":{},"timeouts":{},"mycomponent-timeout-inbound":{},"targets":{},"components":{},"f-mycomponent":{},"f-inbound":{},"f-outbound":{}}},"spec":{"policies":{},"timeouts":{},"mycomponent-timeout-inbound":{},"outbound":{}}}}} app_id=helloworld2 instance=stageapp--qjft33-640f899c87-h7dq scope=dapr.runtime type=log ver=11.5-mst-1

## Related content

See how resiliency works for [Service to service communication using Azure Container Apps built in service discovery](#)

## Feedback

Was this page helpful?

Yes

No

Provide product feedback | Get help at Microsoft Q&A

# Reliability in Azure Container Apps

Article • 11/21/2023

This article describes reliability support in [Azure Container Apps](#), and covers both regional resiliency with availability zones and cross-region resiliency with disaster recovery. For a more detailed overview of reliability in Azure, see [Azure reliability](#).

## 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 Container Apps uses [availability zones](#) in regions where they're available to provide high-availability protection for your applications and data from data center failures.

By enabling Container Apps' zone redundancy feature, replicas are automatically distributed across the zones in the region. Traffic is load balanced among the replicas. If a zone outage occurs, traffic is automatically routed to the replicas in the remaining zones.

### Note

There is no extra charge for enabling zone redundancy, but it only provides benefits when you have 2 or more replicas, with 3 or more being ideal since most regions that support zone redundancy have 3 zones.

## Prerequisites

Azure Container Apps offers the same reliability support regardless of your plan type.

Azure Container Apps uses [availability zones](#) in regions where they're available. For a list of regions that support availability zones, see [Availability zone service and regional support](#).

## SLA improvements

There are no increased SLAs for Azure Container Apps. For more information on the Azure Container Apps SLAs, see [Service Level Agreement for Azure Container Apps](#).

## Create a resource with availability zone enabled

### Set up zone redundancy in your Container Apps environment

To take advantage of availability zones, you must enable zone redundancy when you create a Container Apps environment. The environment must include a virtual network with an available subnet. To ensure proper distribution of replicas, set your app's minimum replica count to three.

### Enable zone redundancy via the Azure portal

To create a container app in an environment with zone redundancy enabled using the Azure portal:

1. Navigate to the Azure portal.
2. Search for **Container Apps** in the top search box.
3. Select **Container Apps**.
4. Select **Create New** in the *Container Apps Environment* field to open the *Create Container Apps Environment* panel.
5. Enter the environment name.
6. Select **Enabled** for the *Zone redundancy* field.

Zone redundancy requires a virtual network with an infrastructure subnet. You can choose an existing virtual network or create a new one. When creating a new virtual network, you can accept the values provided for you or customize the settings.

1. Select the **Networking** tab.
2. To assign a custom virtual network name, select **Create New** in the *Virtual Network* field.
3. To assign a custom infrastructure subnet name, select **Create New** in the *Infrastructure subnet* field.
4. You can select **Internal** or **External** for the *Virtual IP*.
5. Select **Create**.

## Create Container Apps Environment

Basics    Monitoring    **Networking**

Selecting your own virtual network allows you to connect your application to other Azure resources or on-premises systems through the same network. [Learn more](#)

**Virtual network**

Use your own virtual network  No  Yes

Virtual network \*

Infrastructure subnet \*

Virtual IP

Internal: The endpoint is an internal load balancer  
 External: Exposes the hosted apps on an internet-accessible IP address

**Create** **Cancel**

## Enable zone redundancy with the Azure CLI

Create a virtual network and infrastructure subnet to include with the Container Apps environment.

When using these commands, replace the <PLACEHOLDERS> with your values.

## ⓘ Note

The Consumption only environment requires a dedicated subnet with a CIDR range of /23 or larger. The workload profiles environment requires a dedicated subnet with a CIDR range of /27 or larger. To learn more about subnet sizing, see the [networking architecture overview](#).

Azure CLI

```
az network vnet create \
--resource-group <RESOURCE_GROUP_NAME> \
--name <VNET_NAME> \
--location <LOCATION> \
--address-prefix 10.0.0.0/16
```

Azure CLI

```
az network vnet subnet create \
--resource-group <RESOURCE_GROUP_NAME> \
--vnet-name <VNET_NAME> \
--name infrastructure \
--address-prefixes 10.0.0.0/21
```

Next, query for the infrastructure subnet ID.

Azure CLI

```
INFRASTRUCTURE_SUBNET=`az network vnet subnet show --resource-group
<RESOURCE_GROUP_NAME> --vnet-name <VNET_NAME> --name infrastructure --
query "id" -o tsv | tr -d '[:space:]'`
```

Finally, create the environment with the `--zone-redundant` parameter. The location must be the same location used when creating the virtual network.

Azure CLI

```
Azure CLI
```

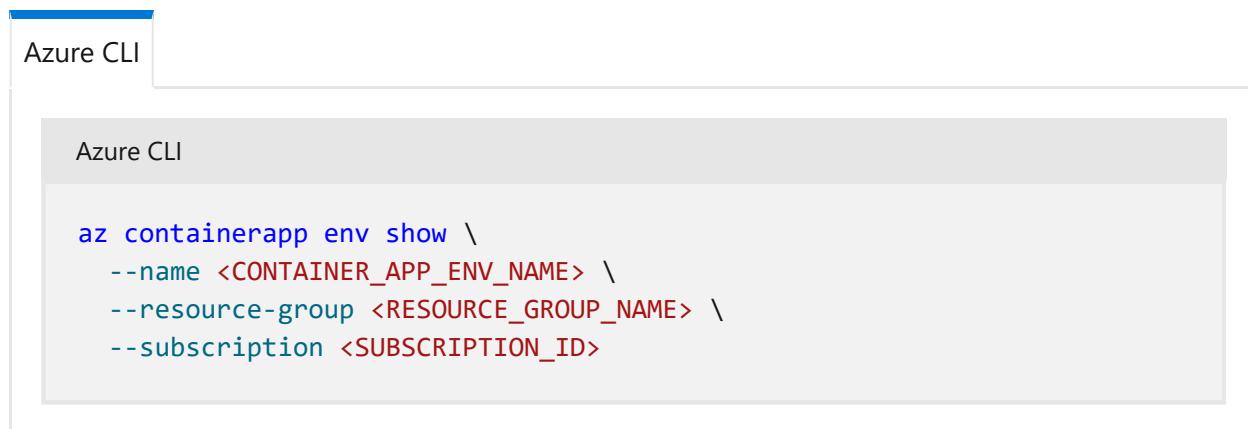
```
az containerapp env create \
--name <CONTAINER_APP_ENV_NAME> \
--resource-group <RESOURCE_GROUP_NAME> \
--location "<LOCATION>" \
--infrastructure-subnet-resource-id $INFRASTRUCTURE_SUBNET \
--zone-redundant
```

## Verify zone redundancy with the Azure CLI

### ⓘ Note

The Azure Portal does not show whether zone redundancy is enabled.

Use the [az container app env show](#) command to verify zone redundancy is enabled for your Container Apps environment.



Azure CLI

```
az containerapp env show \
--name <CONTAINER_APP_ENV_NAME> \
--resource-group <RESOURCE_GROUP_NAME> \
--subscription <SUBSCRIPTION_ID>
```

The command returns a JSON response. Verify the response contains "zoneRedundant": true.

## Safe deployment techniques

When you set up [zone redundancy in your container app](#), replicas are distributed automatically across the zones in the region. After the replicas are distributed, traffic is load balanced among them. If a zone outage occurs, traffic automatically routes to the replicas in the remaining zone.

You should still use safe deployment techniques such as [blue-green deployment](#). Azure Container Apps doesn't provide one-zone-at-a-time deployment or upgrades.

If you have enabled [session affinity](#), and a zone goes down, clients for that zone are routed to new replicas because the previous replicas are no longer available. Any state

associated with the previous replicas is lost.

## Availability zone redeployment and migration

To take advantage of availability zones, enable zone redundancy as you create the Container Apps environment. The environment must include a virtual network with an available subnet. You can't migrate an existing Container Apps environment from nonavailability zone support to availability zone support.

## 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. Before you begin to think about creating your disaster recovery plan, see [Recommendations for designing a disaster recovery strategy](#).

When it comes to DR, Microsoft uses the [shared responsibility model](#). In a shared responsibility model, Microsoft ensures that the baseline infrastructure and platform services are available. At the same time, many Azure services don't automatically replicate data or fall back from a failed region to cross-replicate to another enabled region. For those services, you are responsible for setting up a disaster recovery plan that works for your workload. Most services that run on Azure platform as a service (PaaS) offerings provide features and guidance to support DR and you can use [service-specific features to support fast recovery](#) to help develop your DR plan.

In the unlikely event of a full region outage, you have the option of using one of two strategies:

- **Manual recovery:** Manually deploy to a new region, or wait for the region to recover, and then manually redeploy all environments and apps.
- **Resilient recovery:** First, deploy your container apps in advance to multiple regions. Next, use Azure Front Door or Azure Traffic Manager to handle incoming requests, pointing traffic to your primary region. Then, should an outage occur, you can redirect traffic away from the affected region. For more information, see [Cross-region replication in Azure](#).

### Note

Regardless of which strategy you choose, make sure your deployment configuration files are in source control so you can easily redeploy if necessary.

## More guidance

The following resources can help you create your own disaster recovery plan:

- [Failure and disaster recovery for Azure applications](#)
- [Azure resiliency technical guidance](#)

## Next steps

[Reliability in Azure](#)

# Workload profiles in Azure Container Apps

Article • 01/23/2024

A workload profile determines the amount of compute and memory resources available to the container apps deployed in an environment.

Profiles are configured to fit the different needs of your applications.

[+] Expand table

Profile type	Description	Potential use
Consumption	Automatically added to any new environment.	Apps that don't require specific hardware requirements
Dedicated (General purpose)	Balance of memory and compute resources	Apps that require larger amounts of CPU and/or memory
Dedicated (Memory optimized)	Increased memory resources	Apps that need access to large in-memory data, in-memory machine learning models, or other high memory requirements
Dedicated (GPU enabled) (preview)	GPU enabled with increased memory and compute resources available in West US 3 and North Europe regions.	Apps that require GPU

## ! Note

When using GPU-enabled workload profiles, make sure your application is running the latest version of [CUDA](#).

The Consumption workload profile is the default profile added to every Workload profiles [environment](#) type. You can add Dedicated workload profiles to your environment as you create an environment or after it's created. Workload profiles environments are deployed separately from Consumption only environments.

For each Dedicated workload profile in your environment, you can:

- Select the type and size
- Deploy multiple apps into the profile

- Use autoscaling to add and remove instances based on the needs of the apps
- Limit scaling of the profile to better control costs

You can configure each of your apps to run on any of the workload profiles defined in your Container Apps environment. This configuration is ideal for deploying microservices where each app can run on the appropriate compute infrastructure.

### Note

You can only apply a GPU workload profile to an environment as the environment is created.

## Profile types

There are different types and sizes of workload profiles available by region. By default, each Dedicated plan includes a consumption profile, but you can also add any of the following profiles:

 Expand table

Display name	Name	vCPU	Memory (GiB)	GPU	Category	Allocation
Consumption	consumption	4	8	-	Consumption	per replica
Dedicated-D4	D4	4	16	-	General purpose	per node
Dedicated-D8	D8	8	32	-	General purpose	per node
Dedicated-D16	D16	16	64	-	General purpose	per node
Dedicated-D32	D32	32	128	-	General purpose	per node
Dedicated-E4	E4	4	32	-	Memory optimized	per node
Dedicated-E8	E8	8	64	-	Memory optimized	per node
Dedicated-E16	E16	16	128	-	Memory optimized	per node

Display name	Name	vCPU	Memory (GiB)	GPU	Category	Allocation
Dedicated-E32	E32	32	256	-	Memory optimized	per node
Dedicated-NC24-A100 (preview)	NC24-A100	24	220	1	GPU enabled	per node*
Dedicated-NC48-A100 (preview)	NC48-A100	48	440	2	GPU enabled	per node*
Dedicated-NC96-A100 (preview)	NC96-A100	96	880	4	GPU enabled	per node*

\* Capacity is allocated on a per-case basis. Submit a [support ticket](#) to request the capacity amount required for your application.

Select a workload profile and use the *Name* field when you run `az containerapp env workload-profile set` for the `--workload-profile-type` option.

In addition to different core and memory sizes, workload profiles also have varying image size limits available. To learn more about the image size limits for your container apps, see [hardware reference](#).

The availability of different workload profiles varies by region.

## Resource consumption

You can constrain the memory and CPU usage of each app inside a workload profile, and you can run multiple apps inside a single instance of a workload profile. However, the total amount of resources available to a container app is less than what's allocated to a profile. The difference between allocated and available resources is the amount reserved by the Container Apps runtime.

## Scaling

When demand for new apps or more replicas of an existing app exceeds the profile's current resources, profile instances may be added.

At the same time, if the number of required replicas goes down, profile instances may be removed. You have control over the constraints on the minimum and maximum number of profile instances.

Azure calculates [billing](#) largely based on the number of running profile instances.

## Networking

When you use the workload profile environment, extra networking features that fully secure your ingress and egress networking traffic (such as user defined routes) are available. To learn more about what networking features are supported, see [Networking in Azure Container Apps environment](#). For steps on how to secure your network with Container Apps, see the [lock down your Container App environment section](#).

## Next steps

[Manage workload profiles with the CLI](#)

# Manage workload profiles with the Azure CLI

Article • 08/30/2023

Learn to manage a workload profiles environment using the Azure CLI.

## Create a container app in a profile

By default, your Container Apps environment is created with a managed VNet that is automatically generated for you. Generated VNets are inaccessible to you as they're created in Microsoft's tenant.

Alternatively, you can create an environment with a [custom VNet](#) if you need any of the following features:

- [User defined routes](#)
- Integration with Application Gateway
- Network Security Groups
- Communicating with resources behind private endpoints in your virtual network

Use the following commands to create a workload profiles environment.

### 1. Create *workload profiles* environment

```
Bash

az containerapp env create \
 --enable-workload-profiles \
 --resource-group "<RESOURCE_GROUP>" \
 --name "<NAME>" \
 --location "<LOCATION>"
```

This command can take up to 10 minutes to complete.

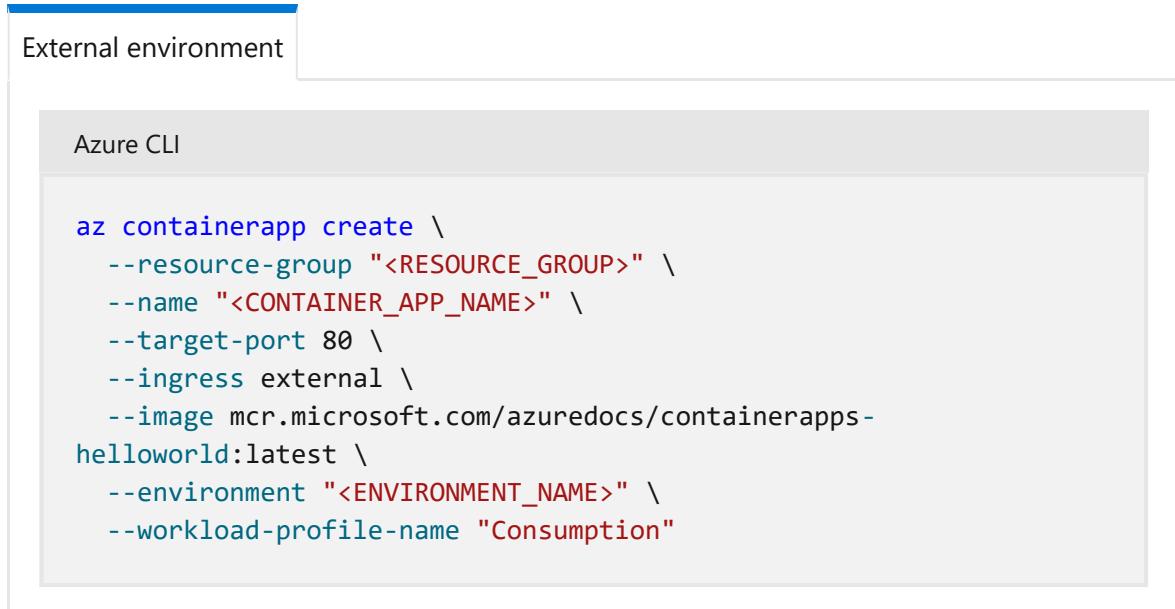
### 2. Check the status of your environment. The following command reports if the environment is created successfully.

```
Bash

az containerapp env show \
 --name "<ENVIRONMENT_NAME>" \
 --resource-group "<RESOURCE_GROUP>"
```

The `provisioningState` needs to report `Succeeded` before moving on to the next command.

### 3. Create a new container app.



The screenshot shows the Azure portal interface. At the top, there's a header bar with the text "External environment". Below it is a section titled "Azure CLI" containing the following command:

```
az containerapp create \
--resource-group "<RESOURCE_GROUP>" \
--name "<CONTAINER_APP_NAME>" \
--target-port 80 \
--ingress external \
--image mcr.microsoft.com/azuredocs/containerapps-
helloworld:latest \
--environment "<ENVIRONMENT_NAME>" \
--workload-profile-name "Consumption"
```

This command deploys the application to the built-in Consumption workload profile. If you want to create an app in a Dedicated profile, you first need to [add the profile to the environment](#).

This command creates the new application in the environment using a specific workload profile.

## Add profiles

Add a new workload profile to an existing environment.



The screenshot shows the Azure portal interface. At the top, there's a header bar with the text "Azure CLI". Below it is a section titled "az containerapp env workload-profile set" containing the following command:

```
az containerapp env workload-profile set \
--resource-group <RESOURCE_GROUP> \
--name <ENVIRONMENT_NAME> \
--workload-profile-type <WORKLOAD_PROFILE_TYPE> \
--workload-profile-name <WORKLOAD_PROFILE_NAME> \
--min-nodes <MIN_NODES> \
--max-nodes <MAX_NODES>
```

The value you select for the `<WORKLOAD_PROFILE_NAME>` placeholder is the workload profile *friendly name*.

Using friendly names allow you to add multiple profiles of the same type to an environment. The friendly name is what you use as you deploy and maintain a container

app in a workload profile.

## Edit profiles

You can modify the minimum and maximum number of nodes used by a workload profile via the `set` command.

Azure CLI

```
az containerapp env workload-profile set \
--resource-group <RESOURCE_GROUP> \
--name <ENV_NAME> \
--workload-profile-type <WORKLOAD_PROFILE_TYPE> \
--workload-profile-name <WORKLOAD_PROFILE_NAME> \
--min-nodes <MIN_NODES> \
--max-nodes <MAX_NODES>
```

## Delete a profile

Use the following command to delete a workload profile.

Azure CLI

```
az containerapp env workload-profile delete \
--resource-group "<RESOURCE_GROUP>" \
--name <ENVIRONMENT_NAME> \
--workload-profile-name <WORKLOAD_PROFILE_NAME>
```

### ⓘ Note

The *Consumption* workload profile can't be deleted.

## Inspect profiles

The following commands allow you to list available profiles in your region and ones used in a specific environment.

## List available workload profiles

Use the `list-supported` command to list the supported workload profiles for your region.

The following Azure CLI command displays the results in a table.

#### Azure CLI

```
az containerapp env workload-profile list-supported \
--location <LOCATION> \
--query "[].{Name: name, Cores: properties.cores, MemoryGiB: properties.memoryGiB, Category: properties.category}" \
-o table
```

The response resembles a table similar to the below example:

#### Output

Name	Cores	MemoryGiB	Category
D4	4	16	GeneralPurpose
D8	8	32	GeneralPurpose
D16	16	64	GeneralPurpose
E4	4	32	MemoryOptimized
E8	8	64	MemoryOptimized
E16	16	128	MemoryOptimized
E32	32	256	MemoryOptimized
Consumption	4	8	Consumption

Select a workload profile and use the *Name* field when you run `az containerapp env workload-profile set` for the `--workload-profile-type` option.

## Show a workload profile

Display details about a workload profile.

#### Azure CLI

```
az containerapp env workload-profile show \
--resource-group <RESOURCE_GROUP> \
--name <ENVIRONMENT_NAME> \
--workload-profile-name <WORKLOAD_PROFILE_NAME>
```

## Next steps

[Workload profiles overview](#)

# Manage a workload profiles in the Azure portal

Article • 08/30/2023

Learn to manage a [workload profiles](#) environment in the Azure portal.

## Create a container app in a workload profile

1. Open the Azure portal.
2. Search for *Container Apps* in the search bar, and select **Container Apps**.
3. Select **Create**.
4. Create a new container app and environment.

**Create Container App**

[Basics](#)   [App settings](#)   [Tags](#)   [Review + create](#)

**Project details**

Subscription \*

Resource group \*  [Create new](#)

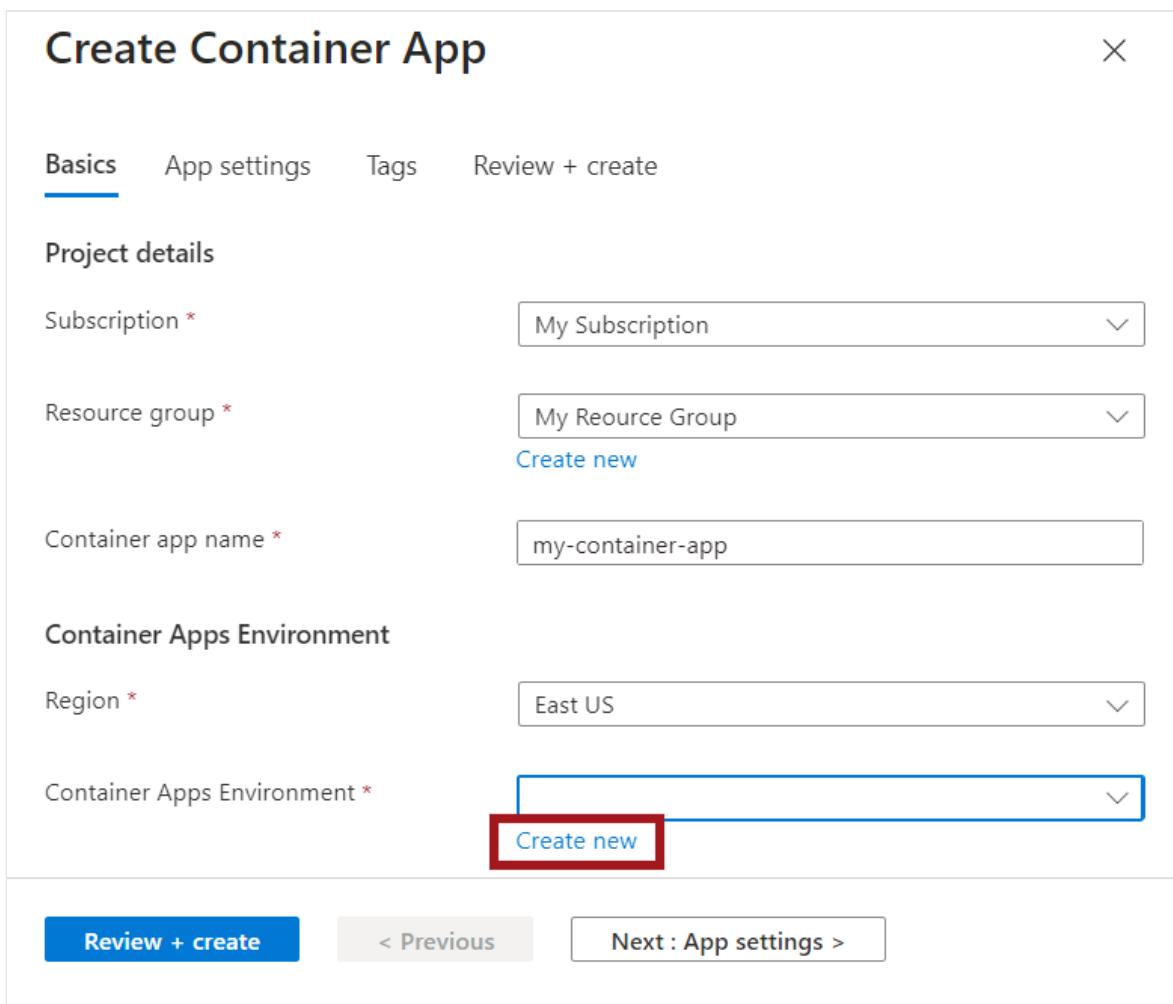
Container app name \*

**Container Apps Environment**

Region \*

Container Apps Environment \*  [Create new](#)

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



Enter the following values to create your new container app.

Property	Value
Subscription	Select your subscription
Resource group	Select or create a resource group
Container app name	Enter your container app name
Region	Select your region.
Container Apps Environment	Select Create New.

## 5. Configure the new environment.

**Create Container Apps Environment** ...

2 Basics Workload profiles Monitoring Networking

The environment is a secure boundary around one or more container apps that can communicate with each other and share a virtual network, logging, and Dapr configuration. [Learn more ↗](#)

**Environment details**

Environment name \*

Environment type \*

1  **Workload Profiles:** Supports the Consumption and Dedicated plans. Run serverless apps with support for scale-to-zero and pay only for resources your apps use. Optionally, run apps with customized hardware and increased cost predictability using Dedicated workload profiles.

**Consumption only:** Supports the Consumption plan. Run serverless apps with support for scale-to-zero and pay only for resources your apps use.

**Zone redundancy**

A Container App Environment can be deployed as a zone redundant service in the regions that support it. This is a deployment time only decision. You can't make Container App Environment zone redundant after it has been deployed. [Learn more ↗](#)

Zone redundancy \*

**Disabled:** Your Container App Environment and the apps in it will not be zone redundant.

**Enabled:** Your Container App Environment and the apps in it will be zone redundant. This requires vNet integration.

**Create** **Cancel**

Enter the following values to create your environment.

Property	Value
Environment name	Enter an environment name.
Environment type	Select <b>Workload profiles</b>

Select the new **Workload profiles** tab at the top of this section.

6. Select the **Add workload profile** button.

## Create Container Apps Environment

Basics    **Workload profiles**    Monitoring    Networking

**+ Add workload profile**    Delete

Name	Scaling	Workload profile size
<input type="checkbox"/> Consumption	-	Up to 4 vCPUs / 8 Gib

**Create**    **Cancel**

7. For *Workload profile name*, enter a name.

8. Next to *Workload profile size*, select **Choose size**.

## Add workload profile

Workload profile name \*

Dedicated-D8

Workload profile size \*

**Choose a size**

Autoscaling instance count range \*

3       5

9. In the *Select a workload profile size* window, select a profile from the list.

## Select a workload profile size

Name	vCPU	RAM (GiB)
▼ General purpose D-series		
Dedicated-D4	4	16
<input checked="" type="checkbox"/> Dedicated-D8	8	32
Dedicated-D16	16	64
Dedicated-D32	32	128
▼ Memory optimized E-series		
Dedicated-E4	4	32
Dedicated-E8	8	64
Dedicated-E16	16	128
Dedicated-E32	32	256

Select

Back

General purpose profiles offer a balanced mix cores vs memory for most applications.

Memory optimized profiles offer specialized hardware with increased memory capabilities.

10. Select the **Select** button.

11. For the *Autoscaling instance count range*, select the minimum and maximum number of instances you want available to this workload profile.

### Add workload profile

X

Workload profile name \*

Workload profile size \*

**Dedicated-D8**  
8 vCPUs, 32 GiB included  
249.357 USD/month  
[Change](#)

Autoscaling instance count range \*

Add Back

12. Select Add.

13. Select **Create**.

14. Select **Review + Create** and wait as Azure validates your configuration options.

15. Select **Create** to create your container app and environment.

## Add profiles

Add a new workload profile to an existing environment.

1. Under the *Settings* section, select **Workload profiles**.

2. Select **Add**.

3. For *Workload profile name*, enter a name.

4. Next to *Workload profile size*, select **Choose size**.

5. In the *Select a workload profile size* window, select a profile from the list.

General purpose profiles offer a balanced mix cores vs memory for most applications.

Memory optimized profiles offer specialized hardware with increased memory or compute capabilities.

6. Select the **Select** button.

7. For the *Autoscaling instance count range*, select the minimum and maximum number of instances you want available to this workload profile.

The screenshot shows the 'Add workload profile' dialog box. It has fields for 'Workload profile name' (set to 'Dedicated-D8'), 'Workload profile size' (set to 'Dedicated-D8' with details: 8 vCPUs, 32 GiB included, 249.357 USD/month), and 'Autoscaling instance count range' (set to a range from 3 to 5). The 'Autoscaling instance count range' field is highlighted with a red border. At the bottom are 'Add' and 'Back' buttons.

Autoscaling instance count range	3	5
----------------------------------	---	---

8. Select **Add**.

# Edit profiles

Under the *Settings* section, select **Workload profiles**.

From this window, you can:

- Adjust the minimum and maximum number of instances available to a profile
- Add new profiles
- Delete existing profiles (except for the consumption profile)

## Delete a profile

Under the *Settings* section, select **Workload profiles**. From this window, you select a profile to delete.

 **Note**

The *Consumption* workload profile can't be deleted.

## Next steps

[Workload profiles overview](#)

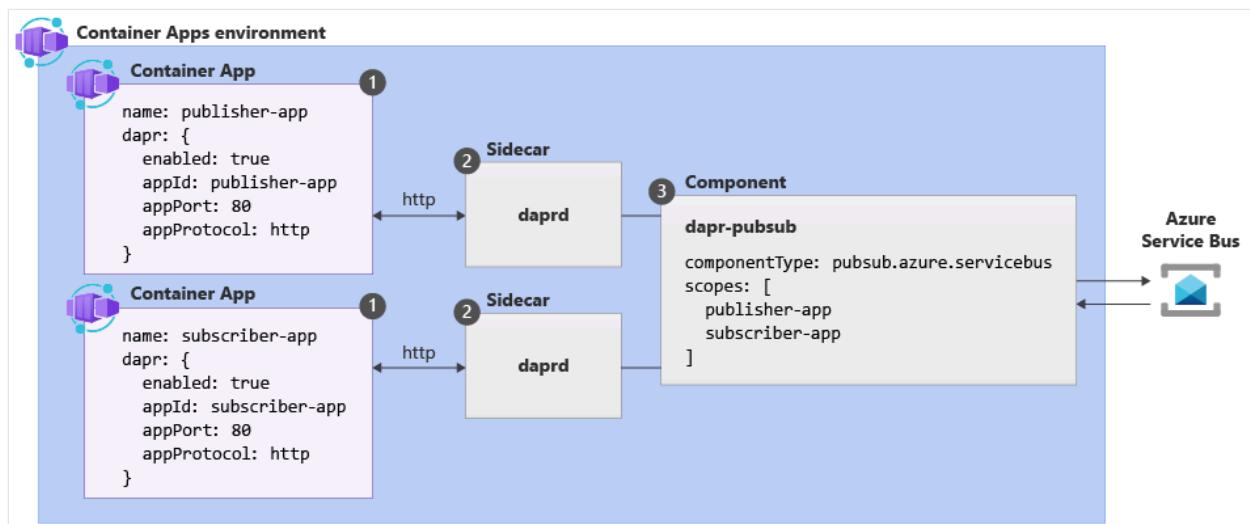
# Microservice APIs powered by Dapr

Article • 10/04/2024

Azure Container Apps provides APIs powered by [Distributed Application Runtime \(Dapr\)](#) that help you write and implement simple, portable, resilient, and secured microservices. Dapr works together with Azure Container Apps as an abstraction layer to provide a low-maintenance and scalable platform. Azure Container Apps offers a selection of fully managed Dapr APIs, components, and features, catered specifically to microservice scenarios. Simply [enable and configure Dapr](#) as usual in your container app environment.

## How the microservices APIs work with your container app

Configure microservices APIs for your container apps environment with a [Dapr-enabled container app](#), a [Dapr component configured for your solution](#), and a Dapr sidecar invoking communication between them. The following diagram demonstrates these core concepts, using the pub/sub API as an example.



[+] [Expand table](#)

Label	Dapr settings	Description
1	Container Apps with Dapr enabled	Dapr is enabled at the container app level by configuring a set of Dapr arguments. These values apply to all revisions of a given container app when running in multiple revisions mode.
2	Dapr	The fully managed Dapr APIs are exposed to each container app through a Dapr sidecar. The Dapr APIs can be invoked from your

Label	Dapr settings	Description
		container app via HTTP or gRPC. The Dapr sidecar runs on HTTP port 3500 and gRPC port 50001.
3	Dapr component configuration	Dapr uses a modular design where functionality is delivered as a component. Dapr components can be shared across multiple container apps. The Dapr app identifiers provided in the scopes array dictate which dapr-enabled container apps load a given component at runtime.

## Supported Dapr APIs, components, and tooling

### Managed APIs

Azure Container Apps offers managed generally available Dapr APIs (building blocks). These APIs are fully managed and supported for use in production environments.

To learn more about using *alpha* Dapr APIs and features, [see the Dapr FAQ](#).



[ ] [Expand table](#)

API	Status	Description
<a href="#">Service-to-service invocation</a> ↗	GA	Discover services and perform reliable, direct service-to-service calls with automatic mTLS authentication and encryption. <a href="#">See known limitations for Dapr service invocation in Azure Container Apps.</a>
<a href="#">State management</a> ↗	GA	Provides state management capabilities for transactions and CRUD operations.

API	Status	Description
<a href="#">Pub/sub</a>	GA	Allows publisher and subscriber container apps to intercommunicate via an intermediary message broker. You can also create declarative subscriptions to a topic using an external component JSON file. <a href="#">Learn more about the declarative pub/sub API.</a>
<a href="#">Bindings</a>	GA	Trigger your applications based on events
<a href="#">Actors</a>	GA	Dapr actors are message-driven, single-threaded, units of work designed to quickly scale. For example, in burst-heavy workload situations.
<a href="#">Observability</a>	GA	Send tracing information to an Application Insights backend.
<a href="#">Secrets</a>	GA	Access secrets from your application code or reference secure values in your Dapr components.
<a href="#">Configuration</a>	GA	Retrieve and subscribe to application configuration items for supported configuration stores.

## Compatible SDKs

Dapr's latest client SDK packages are compatible with Azure Container Apps. You can use any of the [supported, GA Dapr APIs](#) with the following Dapr client SDK versions:

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

Language	SDK version
Java	1.12.0
Go	1.11.0
Python	1.14.0
.NET	1.14.0
JavaScript	3.3.1
Rust	0.15.1

 **Note**

Currently, the Dapr server extensions, actor, and workflow SDK packages are not compatible with Azure Container Apps. [Learn more about all of the Dapr SDK packages.](#)

## Tier 1 versus Tier 2 components

A subset of Dapr components is supported. Within that subset, Dapr components are broken into two support categories: Tier 1 or Tier 2.

- **Tier 1 components:** Stable components that receive immediate investigation in critical (security or serious regression) scenarios. Otherwise, Microsoft collaborates with open source to address in a hotfix or the next regular release.
- **Tier 2 components:** Components that are investigated on a lesser priority, as they're not in stable state or are with a third party provider.

## Tier 1 components

[+] Expand table

API	Component	Type
State management	Azure Cosmos DB	state.azure.cosmosdb
	Azure Blob Storage v1	state.azure.blobstorage
	Azure Table Storage	state.azure.tablestorage
	Microsoft SQL Server	state.sqlserver
Publish & subscribe	Azure Service Bus Queues	pubsub.azure.servicebus.queues
	Azure Service Bus Topics	pubsub.azure.servicebus.topics
	Azure Event Hubs	pubsub.azure.eventhubs
Binding	Azure Storage Queues	bindings.azure.storagequeues
	Azure Service Bus Queues	bindings.azure.servicebusqueues
	Azure Blob Storage	bindings.azure.blobstorage
	Azure Event Hubs	bindings.azure.eventhubs
Secrets management	Azure Key Vault	secretstores.azure.keyvault

## Tier 2 components

[+] Expand table

API	Component	Type
State management	PostgreSQL	state.postgresql
	MySQL & MariaDB	state.mysql
	Redis	state.redis
Publish & subscribe	Apache Kafka	pubsub.kafka
	Redis Streams	pubsub.redis
Binding	Azure Event Grid	bindings.azure.eventgrid
	Azure Cosmos DB	bindings.azure.cosmosdb
	Apache Kafka	bindings.kafka
	PostgreSQL	bindings.postgresql
	Redis	bindings.redis
	Cron	bindings.cron
Configuration	PostgreSQL	configuration.postgresql
	Redis	configuration.redis

## Tooling

Azure Container Apps ensures compatibility with Dapr open source tooling, such as SDKs and the CLI.

## Limitations

- **Dapr Configuration spec:** Any capabilities that require use of the Dapr configuration spec.
- **Any Dapr sidecar annotations not listed in [the Dapr enablement guide](#)**
- **APIs and components support:** Only the Dapr APIs and components [listed as GA, Tier 1, or Tier 2 in this article](#) are supported in Azure Container Apps.
- **Actor reminders:** Require a minReplicas of 1+ to ensure reminders is always active and fires correctly.
- **Jobs:** Dapr isn't supported for jobs.

## Next steps

- [Enable Dapr in your container app.](#)
- [Learn how Dapr components work in Azure Container Apps.](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# Connect to Azure services via Dapr components in the Azure portal

Article • 08/02/2024

You can easily connect Dapr APIs to backing Azure services using a combination of [Service Connector](#) and [Dapr](#). This feature creates Dapr components on your behalf with valid metadata and authenticated identity to access the Azure service.

In this guide, you'll connect Dapr Pub/Sub API to an Azure Service Bus by:

- ✓ Select pub/sub as the API
- ✓ Specify Azure Service Bus as the service and required properties like namespace, queue name, and identity
- ✓ Use your Azure Service Bus pub/sub component!

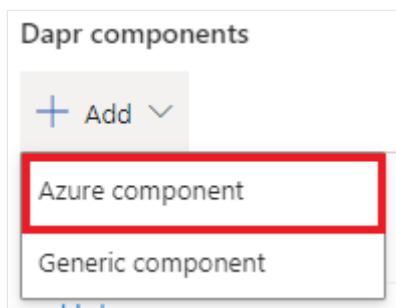
## Prerequisites

- An Azure account with an active subscription. [Create a free Azure account](#).
- [An existing Azure Container App](#).

## Create a Dapr component

Start by navigating to the Dapr component creation feature.

1. In the Azure portal, navigate to your Container Apps environment.
2. In the left-side menu, under **Settings**, select **Dapr components**.
3. From the top menu, select **Add > Azure component** to open the **Add Dapr Component** configuration pane.



**Note**

Currently, creating Dapr components using Service Connector in the Azure portal only works with Azure services (Azure Service Bus, Azure Cosmos DB, etc.). To create non-Azure Dapr components (Redis), use the manual component creation option.

## Provide required metadata

For the component creation tool to map to the required component metadata, you need to provide the required metadata from predefined dropdowns in the **Basics** tab.

For example, for a pub/sub Azure Service Bus component, you'll start with the following fields:

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

Field	Example	Description
Component name	mycomponent	Enter a name for your Dapr component. The name must match the component referenced in your application code.
Building block	Pub/sub	Select the <a href="#">building block/API</a> for your component from the drop-down.
Component type	Service Bus	Select a component type from the drop-down.

The component creation pane populates with different fields depending on the building block and component type you select. For example, the following table and image demonstrate the fields associated with an Azure Service Bus pub/sub component type, but the fields you see may vary.

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

Field	Example	Description
Subscription	My subscription	Select your Azure subscription
Namespace	mynamespace	Select the Service Bus namespace
Authentication	User assigned managed identity	Select the subscription that contains the component you're looking for. Recommended: User assigned managed identity.
User assigned managed identity	testidentity	Select an existing identity from the drop-down. If you don't already have one, you can create a new

Field	Example	Description
		managed identity client ID.

## Add Dapr Component

With Service Connector

Basics    Metadata + Scopes    Review + Create

### Dapr component details

Component name \* (i)

Building Block \* (i)

Pubsub ▼

Select a component type \* (i)

Service Bus ▼

Subscription \* (i)

My subscription ▼

Namespace \* (i)

mynamespace ▼

[Create new](#)

Authentication \* (i)

User assigned managed identity (Recommended) ▼

User assigned managed identity \*

testidentity ▼

[Create new](#)

[Next : Metadata & Scopes](#)
[Cancel](#)

### What happened?

Now that you've filled out these required fields, they'll automatically map to the required component metadata. In this Service Bus example, the only required metadata is the connection string. The component creation tool takes the information you provided and maps the input to create a connection string in the component YAML file.

## Provide optional metadata

While the component creation tool automatically populates all required metadata for the component, you can also customize the component by adding optional metadata.

1. Select **Next : Metadata + Scopes**.
2. Under **Metadata**, select **Add** to select extra, optional metadata for your Dapr component from a drop-down of supported fields.
3. Under **Scopes**, select **Add** or type in the app IDs for the container apps that you want to load this component.
  - By default, when the scope is unspecified, Dapr applies the component to all app IDs.
4. Select **Review + Create** to review the component values.
5. Select **Create**.

## Save the component YAML

Once the component has been added to the Container Apps environment, the portal displays the YAML (or Bicep) for the component.

1. Copy and save the YAML file for future use.
  2. Select **Done** to exit the configuration pane.
- You can then check the YAML/Bicep artifact into a repo and recreate it outside of the portal experience.

### Note

When using Managed Identity, the selected identification is assigned to all containers apps in scope and target services.

## Manage Dapr components

1. In your Container Apps environment, go to **Settings > Dapr components**.
2. The Dapr components that are tied to your Container Apps environment are listed on this page. Review the list and select the **Delete** icon to delete a component, or

select a component's name to review or edit its details.

Name ↑	Type ↑	Delete
mycomponent	bindings.azure.servicebusqueues	

## Next steps

[Enable Dapr on your container apps.](#)

## Related links

Learn more about:

- [Using Dapr with Azure Container Apps](#)
- [Connecting to cloud services using Service Connector](#)

---

## Feedback

Was this page helpful?

Yes    No

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

# Enable Dapr on your container app

Article • 12/21/2023

You can configure Dapr using various [arguments and annotations](#) based on the runtime context. Azure Container Apps provides three channels through which you can enable Dapr:

- [The Azure CLI](#)
- [Infrastructure as Code \(IaC\) templates](#), like Bicep or Azure Resource Manager (ARM) templates
- [The Azure portal](#)

The following table outlines the currently supported list of Dapr sidecar configurations for enabling Dapr in Azure Container Apps:

[+] Expand table

Container Apps CLI	Template field	Description
--enable-dapr	dapr.enabled	Enables Dapr on the container app.
--dapr-app-port	dapr.appPort	The port your application is listening on which is used by Dapr for communicating to your application
--dapr-app-protocol	dapr.appProtocol	Tells Dapr which protocol your application is using. Valid options are <code>http</code> or <code>grpc</code> . Default is <code>http</code> .
--dapr-app-id	dapr.appId	A unique Dapr identifier for your container app used for service discovery, state encapsulation and the pub/sub consumer ID.
--dapr-max-request-size	dapr.httpMaxRequestBodySize	Set the max size of request body http and grpc servers to handle uploading of large files. Default is 4 MB.
--dapr-read-buffer-size	dapr.httpReadBufferSize	Set the max size of http header read buffer in to handle when sending multi-KB headers. The default 4 KB.
--dapr-api-logging	dapr.enableApiLogging	Enables viewing the API calls from your application to the Dapr sidecar.
--dapr-log-level	dapr.logLevel	Set the log level for the Dapr sidecar. Allowed values: debug, error, info, warn. Default is <code>info</code> .

# Using the CLI

You can enable Dapr on your container app using the Azure CLI.

```
Azure CLI
az containerapp dapr enable
```

For more information and examples, see the [reference documentation](#).

# Using Bicep or ARM

When using an IaC template, specify the following arguments in the `properties.configuration` section of the container app resource definition.

```
Bicep
Bicep
dapr: {
 enabled: true
 appId: 'nodeapp'
 appProtocol: 'http'
 appPort: 3000
}
```

The above Dapr configuration values are considered application-scope changes. When you run a container app in multiple-revision mode, changes to these settings don't create a new revision. Instead, all existing revisions are restarted to ensure they're configured with the most up-to-date values.

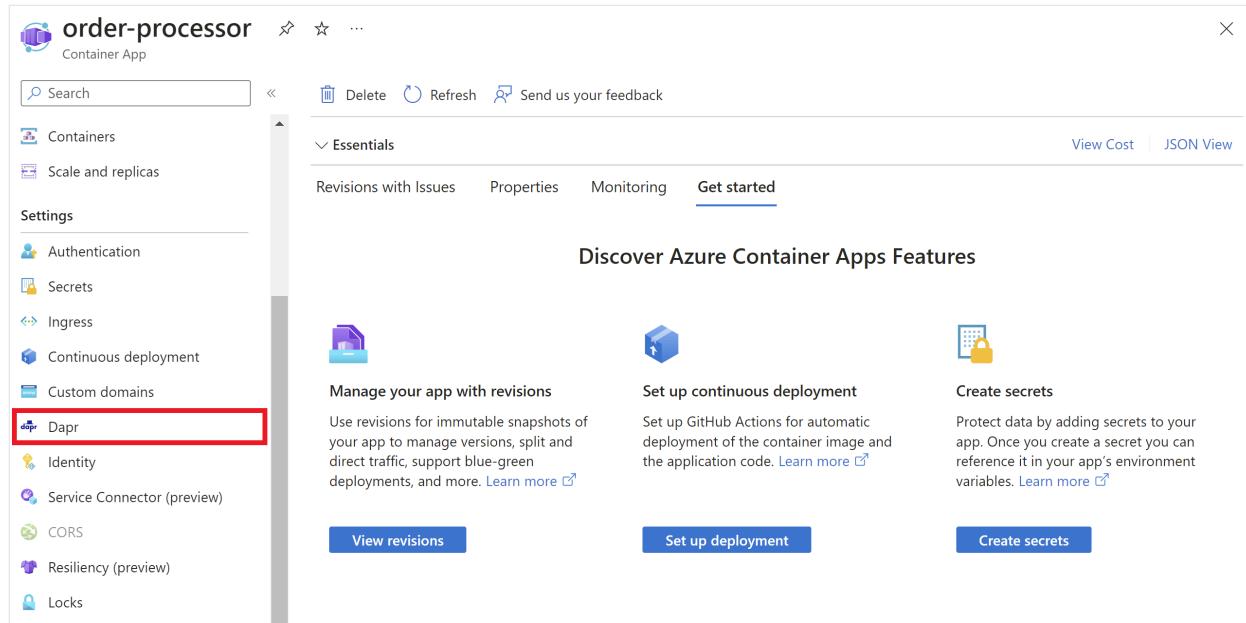
# Using the Azure portal

You can also enable Dapr via the portal view of your container apps.

## ⓘ Note

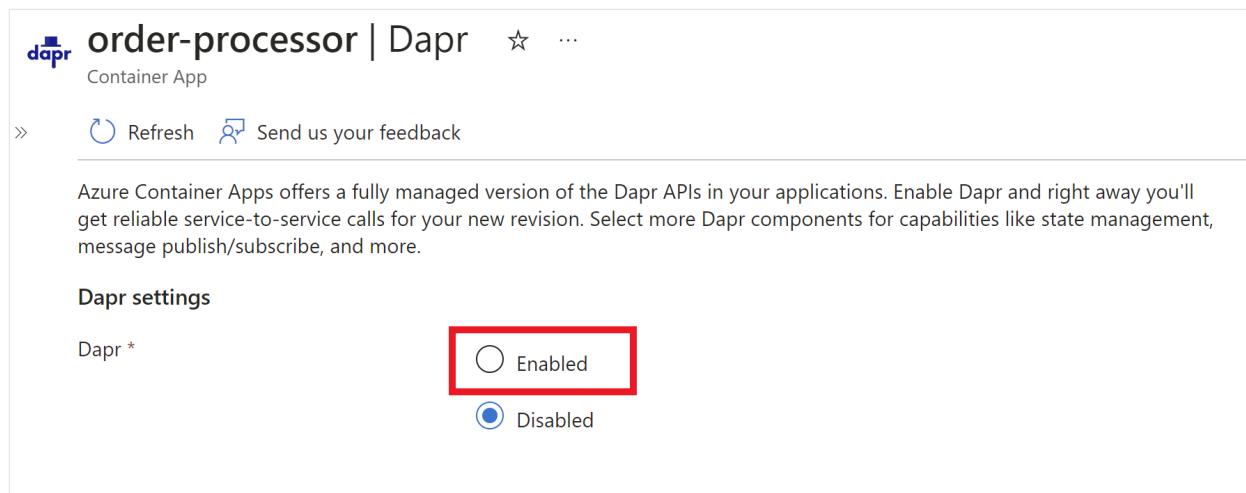
Before you start, make sure you've already created your own Dapr components. [You can connect Dapr components via your container app environment in the portal.](#)

Navigate to your container app in the Azure portal and select **Dapr** under **Settings** in the left side menu.



The screenshot shows the Azure Container Apps portal interface. On the left, there's a sidebar titled 'order-processor' with a 'Container App' icon. The sidebar lists various settings: Containers, Scale and replicas, Settings (expanded), Authentication, Secrets, Ingress, Continuous deployment, Custom domains, Dapr (highlighted with a red box), Identity, Service Connector (preview), CORS, Resiliency (preview), and Locks. At the top right, there are buttons for Delete, Refresh, and Send us your feedback. Below the sidebar, the main area has tabs for Essentials, Revisions with Issues, Properties, Monitoring, and Get started (which is underlined). A section titled 'Discover Azure Container Apps Features' contains three cards: 'Manage your app with revisions' (with a 'View revisions' button), 'Set up continuous deployment' (with a 'Set up deployment' button), and 'Create secrets'. At the bottom right of the main area, there's a 'Create secrets' button.

By default, Dapr is disabled. Select **Enabled** to expand the Dapr settings.



The screenshot shows the 'Dapr settings' page for the 'order-processor' container app. At the top, it says 'order-processor | Dapr' and 'Container App'. Below that are 'Refresh' and 'Send us your feedback' buttons. A text block explains that Azure Container Apps offers a fully managed version of the Dapr APIs. Under 'Dapr settings', there's a field labeled 'Dapr \*' with two radio buttons: 'Enabled' (highlighted with a red box) and 'Disabled'.

Enter the component App ID and select the appropriate headings. If applicable, under the **Components** header, select the link to add and manage your Dapr components to the container app environment.

 **order-processor | Dapr** ⭐ ...

Container App

»  Refresh  Send us your feedback

Dapr \*  Enabled  Disabled

App Id \* ⓘ

App port ⓘ

Protocol ⓘ  HTTP  GRPC

HTTP read buffer size ⓘ  KB

HTTP max request size ⓘ  MB

Log level  ▾

API logging ⓘ

**Components**

In order for your app to use a specific Dapr component, it must be added to your Container Apps Environment with the proper scopes setup to give your app access to it. [Click here to manage your Dapr components.](#)

Name ↑	Type ↑
--------	--------

## Next steps

Try working with Dapr and Azure Container Apps using one of the following tutorials:

- [Microservices communication using Dapr Pub/Sub](#)
- [Event-driven work using Dapr Bindings](#)
- [Microservices communication using Dapr Service Invocation](#)

## Feedback

Was this page helpful?

 Yes

 No

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

# Dapr components in Azure Container Apps

Article • 09/23/2024

Dapr uses a modular design where functionality is delivered as a [component](#). The use of Dapr components is optional and dictated exclusively by the needs of your application.

Dapr components in container apps are environment-level resources that:

- Can provide a pluggable abstraction model for connecting to supporting external services.
- Can be shared across container apps or scoped to specific container apps.
- Can use Dapr secrets to securely retrieve configuration metadata.

In this guide, you learn how to configure Dapr components for your Azure Container Apps services.

## Component schema

In the Dapr open-source project, all components conform to the following basic [schema](#).

```
YAML

apiVersion: dapr.io/v1alpha1
kind: Component
metadata:
 name: [COMPONENT-NAME]
 namespace: [COMPONENT-NAMESPACE]
spec:
 type: [COMPONENT-TYPE]
 version: v1
 initTimeout: [TIMEOUT-DURATION]
 ignoreErrors: [BOOLEAN]
 metadata:
 - name: [METADATA-NAME]
 value: [METADATA-VALUE]
```

In Azure Container Apps, the above schema is slightly simplified to support Dapr components and remove unnecessary fields, including `apiVersion`, `kind`, and redundant `metadata` and `spec` properties.

## YAML

```
componentType: [COMPONENT-TYPE]
version: v1
initTimeout: [TIMEOUT-DURATION]
ignoreErrors: [BOOLEAN]
metadata:
 - name: [METADATA-NAME]
 value: [METADATA-VALUE]
```

# Component scopes

By default, all Dapr-enabled container apps within the same environment load the full set of deployed components. To ensure only the appropriate container apps load components at runtime, application scopes should be used. In the following example, the component is only loaded by the two Dapr-enabled container apps with Dapr application IDs `APP-ID-1` and `APP-ID-2`:

## YAML

```
componentType: [COMPONENT-TYPE]
version: v1
initTimeout: [TIMEOUT-DURATION]
ignoreErrors: [BOOLEAN]
metadata:
 - name: [METADATA-NAME]
 value: [METADATA-VALUE]
scopes:
 - [APP-ID-1]
 - [APP-ID-2]
```

### ⓘ Note

Dapr component scopes correspond to the Dapr application ID of a container app, not the container app name.

# Connecting to external services via Dapr

There are a few approaches supported in container apps to securely establish connections to external services for Dapr components.

## 1. Using managed identity

2. Using a Dapr secret store component reference by creating either:

- An Azure Key Vault secret store, which uses managed identity, or
- Platform-Managed Kubernetes secrets

## Using managed identity

For Azure-hosted services, Dapr can use [the managed identity of the scoped container apps](#) to authenticate to the backend service provider. When using managed identity, you don't need to include secret information in a component manifest. Using managed identity is preferred as it eliminates storage of sensitive input in components and doesn't require managing a secret store.

### Note

The `azureClientId` metadata field (the client ID of the managed identity) is required for any component authenticating with user-assigned managed identity.

## Using a Dapr secret store component reference

When you create Dapr components for non-Entra ID enabled services, certain metadata fields require sensitive input values. The recommended approach for retrieving these secrets is to reference an existing Dapr secret store component that securely accesses secret information.

To set up a reference:

1. [Create a Dapr secret store component using the Azure Container Apps schema](#).  
The component type for all supported Dapr secret stores begins with  
`secretstores..`.
2. [Create extra components \(as needed\) which reference the Dapr secret store component](#) you created to retrieve the sensitive metadata input.

## Creating a Dapr secret store component

When creating a secret store component in Azure Container Apps, you can provide sensitive information in the metadata section in either of the following ways:

- For an [Azure Key Vault secret store](#), use managed identity to establish the connection.
- For [non-Azure secret stores](#), use platform-managed Kubernetes secrets that are defined directly as part of the component manifest.

## Azure Key Vault secret stores

The following component showcases the simplest possible secret store configuration using an Azure Key Vault secret store. In this example, publisher and subscriber applications are configured to both have a system or user-assigned managed identity with appropriate permissions on the Azure Key Vault instance.

YAML

```
componentType: secretstores.azure.keyvault
version: v1
metadata:
 - name: vaultName
 value: [your_keyvault_name]
 - name: azureEnvironment
 value: "AZUREPUBLICCLOUD"
 - name: azureClientId # Only required for authenticating user-assigned
 managed identity
 value: [your_managed_identity_client_id]
scopes:
 - publisher-app
 - subscriber-app
```

## Platform-managed Kubernetes secrets

Kubernetes secrets, Local environment variables, and Local file Dapr secret stores aren't supported in Azure Container Apps. As an alternative for the upstream Dapr default Kubernetes secret store, Azure Container Apps provides a platform-managed approach for creating and leveraging Kubernetes secrets.

This component configuration defines the sensitive value as a secret parameter that can be referenced from the metadata section. This approach can be used to connect to non-Azure services or in dev/test scenarios for quickly deploying components via the CLI without setting up a secret store or managed identity.

YAML

```
componentType: secretstores.azure.keyvault
version: v1
metadata:
 - name: vaultName
 value: [your_keyvault_name]
 - name: azureEnvironment
 value: "AZUREPUBLICCLOUD"
 - name: azureTenantId
 value: "[your_tenant_id]"
 - name: azureClientId
 value: "[your_client_id]"
```

```
- name: azureClientSecret
 secretRef: azClientSecret
secrets:
- name: azClientSecret
 value: "[your_client_secret]"
scopes:
- publisher-app
- subscriber-app
```

## Referencing Dapr secret store components

Once you [create a Dapr secret store using one of the previous approaches](#), you can reference that secret store from other Dapr components in the same environment. The following example demonstrates using Entra ID authentication.

YAML

```
componentType: pubsub.azure.servicebus.queue
version: v1
secretStoreComponent: "[your_secret_store_name]"
metadata:
- name: namespaceName
 # Required when using Azure Authentication.
 # Must be a fully-qualified domain name
 value: "[your_servicebus_namespace.servicebus.windows.net]"
- name: azureTenantId
 value: "[your_tenant_id]"
- name: azureClientId
 value: "[your_client_id]"
- name: azureClientSecret
 secretRef: azClientSecret
scopes:
- publisher-app
- subscriber-app
```

## Component examples

YAML

To create a Dapr component via the Container Apps CLI, you can use a container apps YAML manifest. When configuring multiple components, you must create and apply a separate YAML file for each component.

Azure CLI

```
az containerapp env dapr-component set --name ENVIRONMENT_NAME --resource-group RESOURCE_GROUP_NAME --dapr-component-name pubsub --yaml "./pubsub.yaml"
```

#### YAML

```
pubsub.yaml for Azure Service Bus component
componentType: pubsub.azure.servicebus.queue
version: v1
secretStoreComponent: "my-secret-store"
metadata:
 - name: namespaceName
 # Required when using Azure Authentication.
 # Must be a fully-qualified domain name
 value: "[your_servicebus_namespace.servicebus.windows.net]"
 - name: azureTenantId
 value: "[your_tenant_id]"
 - name: azureClientId
 value: "[your_client_id]"
 - name: azureClientSecret
 secretRef: azClientSecret
scopes:
 - publisher-app
 - subscriber-app
```

## Next steps

Learn how to set Dapr component resiliency.

## Feedback

Was this page helpful?

 Yes

 No

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

# Dapr component resiliency (preview)

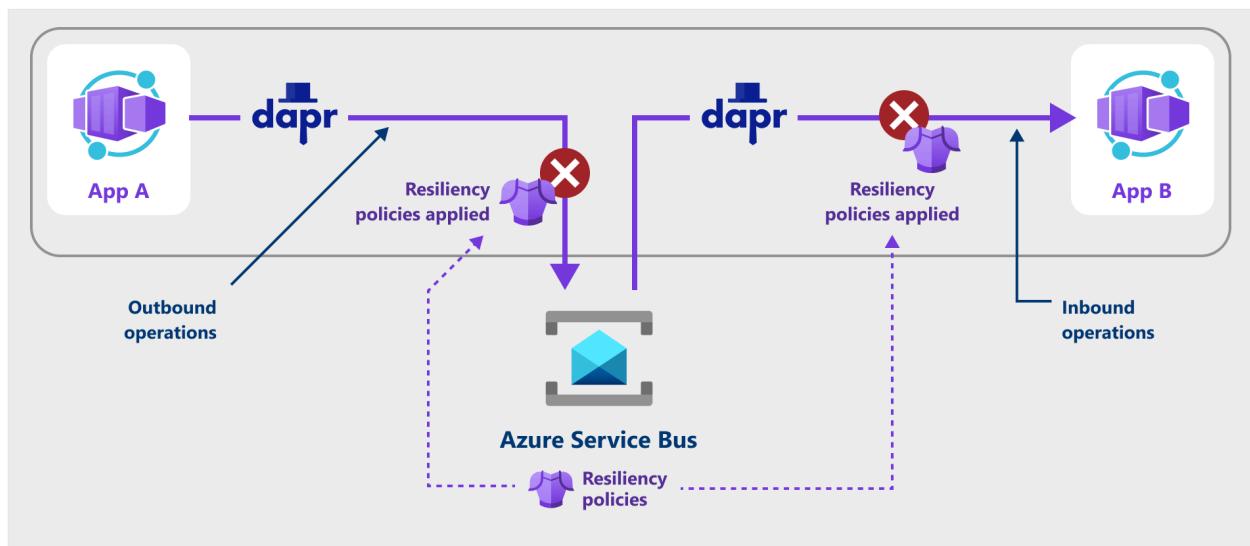
Article • 08/02/2024

Resiliency policies proactively prevent, detect, and recover from your container app failures. In this article, you learn how to apply resiliency policies for applications that use Dapr to integrate with different cloud services, like state stores, pub/sub message brokers, secret stores, and more.

You can configure resiliency policies like retries, timeouts, and circuit breakers for the following outbound and inbound operation directions via a Dapr component:

- **Outbound operations:** Calls from the Dapr sidecar to a component, such as:
  - Persisting or retrieving state
  - Publishing a message
  - Invoking an output binding
- **Inbound operations:** Calls from the Dapr sidecar to your container app, such as:
  - Subscriptions when delivering a message
  - Input bindings delivering an event

The following screenshot shows how an application uses a retry policy to attempt to recover from failed requests.



## Supported resiliency policies

- [Timeouts](#)
- [Retries \(HTTP\)](#)
- [Circuit breakers](#)

# Configure resiliency policies

You can choose whether to create resiliency policies using Bicep, the CLI, or the Azure portal.

Bicep

The following resiliency example demonstrates all of the available configurations.

Bicep

```
resource myPolicyDoc
'Microsoft.App/managedEnvironments/daprComponents/resiliencyPolicies@202
3-11-02-preview' = {
 name: 'my-component-resiliency-policies'
 parent: '${componentName}'
 properties: {
 outboundPolicy: {
 timeoutPolicy: {
 responseTimeoutInSeconds: 15
 }
 httpRetryPolicy: {
 maxRetries: 5
 retryBackOff: {
 initialDelayInMilliseconds: 1000
 maxIntervalInMilliseconds: 10000
 }
 }
 circuitBreakerPolicy: {
 intervalInSeconds: 15
 consecutiveErrors: 10
 timeoutInSeconds: 5
 }
 }
 inboundPolicy: {
 timeoutPolicy: {
 responseTimeoutInSeconds: 15
 }
 httpRetryPolicy: {
 maxRetries: 5
 retryBackOff: {
 initialDelayInMilliseconds: 1000
 maxIntervalInMilliseconds: 10000
 }
 }
 circuitBreakerPolicy: {
 intervalInSeconds: 15
 consecutiveErrors: 10
 timeoutInSeconds: 5
 }
 }
 }
}
```

```
}
```

### ⓘ Important

Once you've applied all the resiliency policies, you need to restart your Dapr applications.

## Policy specifications

### Timeouts

Timeouts are used to early-terminate long-running operations. The timeout policy includes the following properties.

Bicep

```
properties: {
 outbound: {
 timeoutPolicy: {
 responseTimeoutInSeconds: 15
 }
 }
 inbound: {
 timeoutPolicy: {
 responseTimeoutInSeconds: 15
 }
 }
}
```

[\[\] Expand table](#)

Metadata	Required	Description	Example
<code>responseTimeoutInSeconds</code>	Yes	Timeout waiting for a response from the Dapr component.	<code>15</code>

### Retries

Define an `httpRetryPolicy` strategy for failed operations. The retry policy includes the following configurations.

## Bicep

```
properties: {
 outbound: {
 httpRetryPolicy: {
 maxRetries: 5
 retryBackOff: {
 initialDelayInMilliseconds: 1000
 maxIntervalInMilliseconds: 10000
 }
 }
 }
 inbound: {
 httpRetryPolicy: {
 maxRetries: 5
 retryBackOff: {
 initialDelayInMilliseconds: 1000
 maxIntervalInMilliseconds: 10000
 }
 }
 }
}
```

[Expand table](#)

Metadata	Required	Description	Example
<code>maxRetries</code>	Yes	Maximum retries to be executed for a failed http-request.	5
<code>retryBackOff</code>	Yes	Monitor the requests and shut off all traffic to the impacted service when timeout and retry criteria are met.	N/A
<code>retryBackOff.initialDelayInMilliseconds</code>	Yes	Delay between first error and first retry.	1000
<code>retryBackOff.maxIntervalInMilliseconds</code>	Yes	Maximum delay between retries.	10000

## Circuit breakers

Define a `circuitBreakerPolicy` to monitor requests causing elevated failure rates and shut off all traffic to the impacted service when a certain criteria is met.

## Bicep

```

properties: {
 outbound: {
 circuitBreakerPolicy: {
 intervalInSeconds: 15
 consecutiveErrors: 10
 timeoutInSeconds: 5
 }
 },
 inbound: {
 circuitBreakerPolicy: {
 intervalInSeconds: 15
 consecutiveErrors: 10
 timeoutInSeconds: 5
 }
 }
}

```

[ ] [Expand table](#)

Metadata	Required	Description	Example
<code>intervalInSeconds</code>	No	Cyclical period of time (in seconds) used by the circuit breaker to clear its internal counts. If not provided, the interval is set to the same value as provided for <code>timeoutInSeconds</code> .	15
<code>consecutiveErrors</code>	Yes	Number of request errors allowed to occur before the circuit trips and opens.	10
<code>timeoutInSeconds</code>	Yes	Time period (in seconds) of open state, directly after failure.	5

## Circuit breaker process

Specifying `consecutiveErrors` (the circuit trip condition as `consecutiveFailures > $(consecutiveErrors)-1`) sets the number of errors allowed to occur before the circuit trips and opens halfway.

The circuit waits half-open for the `timeoutInSeconds` amount of time, during which the `consecutiveErrors` number of requests must consecutively succeed.

- *If the requests succeed*, the circuit closes.
- *If the requests fail*, the circuit remains in a half-opened state.

If you didn't set any `intervalInSeconds` value, the circuit resets to a closed state after the amount of time you set for `timeoutInSeconds`, regardless of consecutive request

success or failure. If you set `intervalInSeconds` to `0`, the circuit never automatically resets, only moving from half-open to closed state by successfully completing `consecutiveErrors` requests in a row.

If you did set an `intervalInSeconds` value, that determines the amount of time before the circuit is reset to closed state, independent of whether the requests sent in half-opened state succeeded or not.

## Resiliency logs

From the *Monitoring* section of your container app, select **Logs**.



# stageapp

Container App



Search



Scale and replicas

## Settings

---

Authentication

Secrets

Ingress

Continuous deployment

Custom domains

Dapr

Identity

Service Connector (preview)

CORS

Resiliency (preview)

Locks

## Monitoring

---

Alerts

Metrics

Logs

Log stream

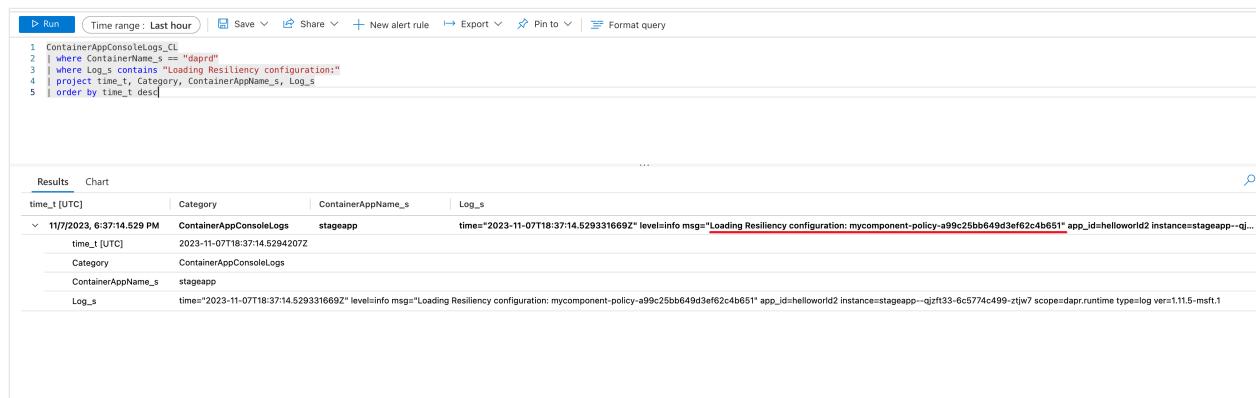
 Console

 Advisor recommendations

In the Logs pane, write and run a query to find resiliency via your container app system logs. For example, to find whether a resiliency policy was loaded:

```
ContainerAppConsoleLogs_CL
| where ContainerName_s == "daprd"
| where Log_s contains "Loading Resiliency configuration:"
| project time_t, Category, ContainerAppName_s, Log_s
| order by time_t desc
```

Click **Run** to run the query and view the result with the log message indicating the policy is loading.



The screenshot shows the Azure portal's Logs pane. At the top, there is a toolbar with buttons for Run, Save, Share, New alert rule, Export, Pin to, and Format query. The Time range is set to Last hour. Below the toolbar, the query is displayed:

```
1 ContainerAppConsoleLogs_CL
2 | where ContainerName_s == "daprd"
3 | where Log_s contains "Loading Resiliency configuration:"
4 | project time_t, Category, ContainerAppName_s, Log_s
5 | order by time_t desc
```

Below the query, there are two tabs: Results and Chart. The Results tab is selected. It displays a table with four columns: time\_t [UTC], Category, ContainerAppName\_s, and Log\_s. One log entry is shown:

time_t [UTC]	Category	ContainerAppName_s	Log_s
11/7/2023, 6:37:14.529 PM	ContainerAppConsoleLogs	stageapp	time=2023-11-07T18:37:14.529331669Z level=info msg="Loading Resiliency configuration: mycomponent-policy-a99c25bb649d3ef62c4b651* app_id=helloworld2 instance=stageapp--qjzft33-6c5774c499-ztjw7 scope=dapr.runtime type=log ver=1.11.6-msft.1"

Or, you can find the actual resiliency policy by enabling debugging on your component and using a query similar to the following example:

```
ContainerAppConsoleLogs_CL
| where ContainerName_s == "daprd"
| where Log_s contains "Resiliency configuration (""
| project time_t, Category, ContainerAppName_s, Log_s
| order by time_t desc
```

Click **Run** to run the query and view the resulting log message with the policy configuration.

The screenshot shows the Azure Log Analytics interface with a query results table. The table has columns: time\_1 [UTC], Category, ContainerAppName\_s, and Log\_s. One row of data is visible:

```

1 ContainerAppConsoleLogs_CL
2 | where ContainerName_s == "daprd"
3 | where Log_s contains "Resiliency configuration"
4 | project time_t, Category, ContainerAppName_s, Log_s
5 | order by time_t desc

```

...  
time\_1 [UTC] Category ContainerAppName\_s Log\_s  
11/7/2023, 6:41:45.511 ... ContainerAppConsoleLogs stageapp time="2023-11-07T18:41:45.511590713Z" level=debug msg="Resiliency configuration (mycomponent-policy-a99c25bb649d3ef62c4b651): {"kind":"Resiliency","apiVersion":"dapr.io/v1alpha1","name":"mycomponent-policy-a99c25bb649d3ef62c4b651","resourceVersion":"10742096","generation":1,"creationTimestamp":"2023-11-07T18:36:17Z","ownerReferences":[{"apiVersion":"k8s.microsoft.com/v1alpha1","kind":"DaprComponentResiliency","name":"mycomponent-policy-a99c25bb649d3ef62c4b651","blockOwnerDeletion":true,"managedFields":[{"manager":"containerapps","operation":"Update","apiVersion":"dapr.io/v1alpha1","time":2023-11-07T18:36:17Z,"fieldsType":"FieldsV1","fieldsV1":{"metadata":{},"ownerReferences":[]}}],"policies":[]}, {"kind":"Timeout","apiVersion":"dapr.io/v1alpha1","targets":[]}, {"kind":"Component","apiVersion":"dapr.io/v1alpha1","name":"mycomponent","version":1.0.0,"spec":{}}, {"kind":"Outbound","apiVersion":"dapr.io/v1alpha1","outbound":[]}, {"kind":"Inbound","apiVersion":"dapr.io/v1alpha1","inbound":[]}}, app\_id=helloworld2 instance=stageapp--qjft33-640f899c87-h7dq scope=dapr.runtime type=log ver=11.5-mst.1  
> 11/7/2023, 6:41:00.718 PM ContainerAppConsoleLogs stageapp time="2023-11-07T18:41:00.718468847Z" level=debug msg="Resiliency configuration (mycomponent-policy-a99c25bb649d3ef62c4b651): {"kind":"Resiliency","apiVersion":"dapr.io/v1alpha1","name":"mycomponent-policy-a99c25bb649d3ef62c4b651","resourceVersion":"10742096","generation":1,"creationTimestamp":"2023-11-07T18:36:17Z","ownerReferences":[{"apiVersion":"k8s.microsoft.com/v1alpha1","kind":"DaprComponentResiliency","name":"mycomponent-policy-a99c25bb649d3ef62c4b651","blockOwnerDeletion":true,"managedFields":[{"manager":"containerapps","operation":"Update","apiVersion":"dapr.io/v1alpha1","time":2023-11-07T18:36:17Z,"fieldsType":"FieldsV1","fieldsV1":{"metadata":{},"ownerReferences":[]}}],"policies":[]}, {"kind":"Timeout","apiVersion":"dapr.io/v1alpha1","targets":[]}, {"kind":"Component","apiVersion":"dapr.io/v1alpha1","name":"mycomponent","version":1.0.0,"spec":{}}, {"kind":"Outbound","apiVersion":"dapr.io/v1alpha1","outbound":[]}, {"kind":"Inbound","apiVersion":"dapr.io/v1alpha1","inbound":[]}}, app\_id=helloworld2 instance=stageapp--qjft33-640f899c87-h7dq scope=dapr.runtime type=log ver=11.5-mst.1

## Related content

See how resiliency works for [Service to service communication using Azure Container Apps built in service discovery](#)

## Feedback

Was this page helpful?



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

# Scale Dapr applications with KEDA scalers

Article • 09/12/2024

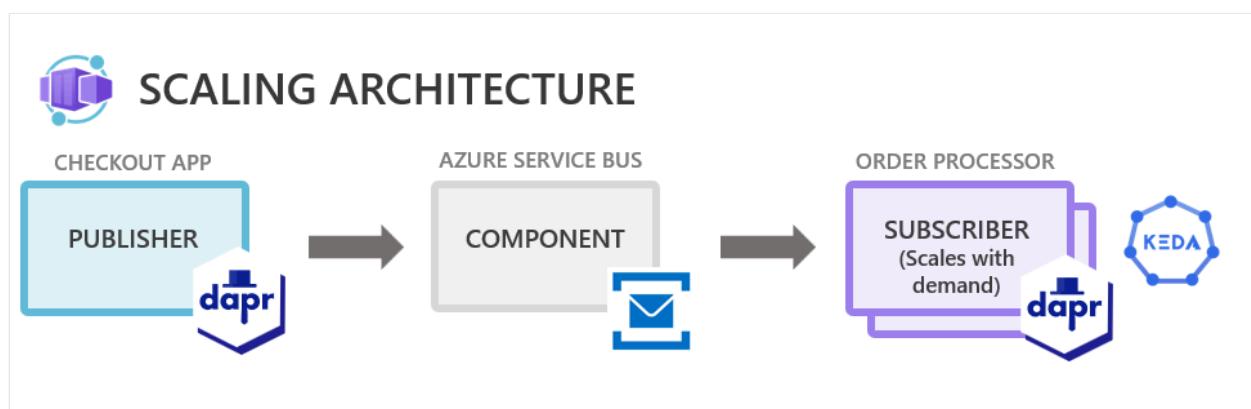
Azure Container Apps automatically scales HTTP traffic to zero. However, to scale non-HTTP traffic (like [Dapr](#) pub/sub and bindings), you can use [KEDA scalers](#) to scale your application and its Dapr sidecar up and down, based on the number of pending inbound events and messages.

This guide demonstrates how to configure the scale rules of a Dapr pub/sub application with a KEDA messaging scaler. For context, refer to the corresponding sample pub/sub applications:

- Microservice communication using pub/sub in [C#](#)
- Microservice communication using pub/sub in [JavaScript](#)
- Microservice communication using pub/sub in [Python](#)

In the above samples, the application uses the following elements:

1. The `checkout` publisher is an application that is meant to run indefinitely and never scale down to zero, despite never receiving any incoming HTTP traffic.
2. The Dapr Azure Service Bus pub/sub component.
3. An `order-processor` subscriber container app picks up messages received via the `orders` topic and processed as they arrive.
4. The scale rule for Azure Service Bus, which is responsible for scaling up the `order-processor` service and its Dapr sidecar when messages start to arrive to the `orders` topic.



Let's take a look at how to apply the scaling rules in a Dapr application.

## Publisher container app

The `checkout` publisher is a headless service that runs indefinitely and never scales down to zero.

By default, [the Container Apps runtime assigns an HTTP-based scale rule to applications](#), which drives scaling based on the number of incoming HTTP requests. In the following example, `minReplicas` is set to `1`. This configuration ensures the container app doesn't follow the default behavior of scaling to zero with no incoming HTTP traffic.

```
Bicep

resource checkout 'Microsoft.App/containerApps@2022-03-01' = {
 name: 'ca-checkout-${resourceToken}'
 location: location
 identity: {
 type: 'SystemAssigned'
 }
 properties: {
 //...
 template: {
 //...
 // Scale the minReplicas to 1
 scale: {
 minReplicas: 1
 maxReplicas: 1
 }
 }
 }
}
```

## Subscriber container app

The following `order-processor` subscriber app includes a custom scale rule that monitors a resource of type `azure-servicebus`. With this rule, the app (and its sidecar) scales up and down as needed based on the number of pending messages in the Bus.

```
Bicep

resource orders 'Microsoft.App/containerApps@2022-03-01' = {
 name: 'ca-orders-${resourceToken}'
 location: location
 tags: union(tags, {
 'azd-service-name': 'orders'
 })
 identity: {
 type: 'SystemAssigned'
 }
 properties: {
 managedEnvironmentId: containerAppsEnvironment.id
 }
}
```

```
configuration: {
 //...
 // Enable Dapr on the container app
 dapr: {
 enabled: true
 appId: 'orders'
 appProtocol: 'http'
 appPort: 5001
 }
 //...
}
template: {
 //...
 // Set the scale property on the order-processor resource
 scale: {
 minReplicas: 0
 maxReplicas: 10
 rules: [
 {
 name: 'topic-based-scaling'
 custom: {
 type: 'azure-servicebus'
 identity: 'system'
 metadata: {
 topicName: 'orders'
 subscriptionName: 'membership-orders'
 messageCount: '30'
 }
 }
 }
]
 }
}
```

# How the scaler works

Notice the `messageCount` property on the scaler's configuration in the subscriber app:

Bicep

```
{
 //...
 properties: {
 //...
 template: {
 //...
 scale: {
 //...
 rules: [
```

```
//...
custom: {
 //...
 metadata: {
 //...
 messageCount: '30'
 }
}
]
```

This property tells the scaler how many messages each instance of the application can process at the same time. In this example, the value is set to `30`, indicating that there should be one instance of the application created for each group of 30 messages waiting in the topic.

For example, if 150 messages are waiting, KEDA scales the app out to five instances. The `maxReplicas` property is set to `10`. Even with a large number of messages in the topic, the scaler never creates more than `10` instances of this application. This setting ensures you don't scale up too much and accrue too much cost.

## Next steps

[Learn more about using Dapr components with Azure Container Apps.](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# Enable token authentication for Dapr requests

Article • 08/28/2024

When [Dapr](#) is enabled for your application in Azure Container Apps, it injects the environment variable `APP_API_TOKEN` into your app's container. Dapr includes the same token in all requests sent to your app, as either:

- An HTTP header (`dapr-api-token`)
- A gRPC metadata option (`dapr-api-token[0]`)

The token is randomly generated and unique per each app and app revision. It can also change at any time. Your application should read the token from the `APP_API_TOKEN` environment variable when it starts up to ensure that it's using the correct token.

You can use this token to authenticate that calls coming into your application are actually coming from the Dapr sidecar, even when listening on public endpoints.

1. The `daprd` container reads and injects it into each call made from Dapr to your application.
2. Your application can then use that token to validate that the request is coming from Dapr.

## Prerequisites

[Dapr-enabled Azure Container App](#)

## Authenticate requests from Dapr

### With Dapr SDKs

If you're using a [Dapr SDK](#), you can use the Dapr authentication methods provided in the open-source SDK repositories.

Once added to your project, the Dapr SDKs validates the token in all incoming requests from Dapr, rejecting calls that don't include the correct token. You don't need to perform any other action.

Incoming requests that don't include the token, or include an incorrect token, are rejected automatically.

## Next steps

[Learn more about the Dapr integration with Azure Container Apps.](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# Tutorial: Microservices communication using Dapr Publish and Subscribe

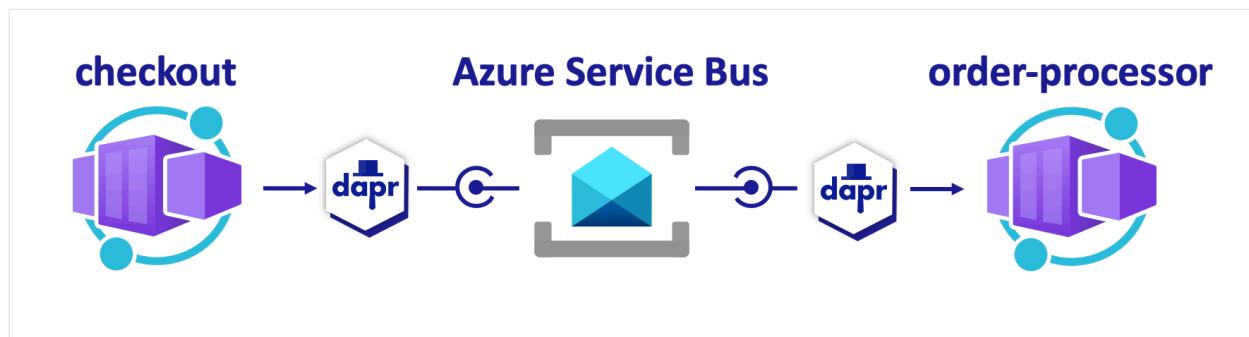
Article • 08/05/2024

In this tutorial, you create publisher and subscriber microservices that leverage [the Dapr Pub/sub API](#) to communicate using messages for event-driven architectures. You'll:

- ✓ Create a publisher microservice and a subscriber microservice that leverage the [Dapr pub/sub API](#) to communicate using messages for event-driven architectures.
- ✓ Deploy the application to Azure Container Apps via the Azure Developer CLI with provided Bicep.

The sample pub/sub project includes:

1. A message generator `checkout` service (publisher) that generates messages of a specific topic.
2. An `order-processor` service (subscriber) that listens for messages from the `checkout` service of a specific topic.



## Prerequisites

- Install [Azure Developer CLI](#)
- Install [Dapr](#) and [init](#) [Dapr](#)
- [Docker Desktop](#)
- Install [Git](#)

## Run the Node.js applications locally

Before deploying the application to Azure Container Apps, run the `order-processor` and `checkout` services locally with Dapr and Azure Service Bus.

# Prepare the project

1. Clone the [sample application](#) to your local machine.

```
Bash
```

```
git clone https://github.com/Azure-Samples/pubsub-dapr-nodejs-servicebus.git
```

2. Navigate into the sample's root directory.

```
Bash
```

```
cd pubsub-dapr-nodejs-servicebus
```

## Run the applications using the Dapr CLI

Start by running the `order-processor` subscriber service.

1. From the sample's root directory, change directories to `order-processor`.

```
Bash
```

```
cd order-processor
```

2. Install the dependencies.

```
Bash
```

```
npm install
```

3. Run the `order-processor` service.

```
Bash
```

```
dapr run --app-port 5001 --app-id order-processing --app-protocol http --dapr-http-port 3501 --resources-path ../components -- npm run start
```

4. In a new terminal window, from the sample's root directory, navigate to the `checkout` publisher service.

```
Bash
```

```
cd checkout
```

## 5. Install the dependencies.

```
Bash
```

```
npm install
```

## 6. Run the `checkout` service.

```
Bash
```

```
dapr run --app-id checkout --app-protocol http --resources-path
./components -- npm run start
```

## Expected output

In both terminals, the `checkout` service publishes 10 messages received by the `order-processor` service before exiting.

`checkout` output:

```
== APP == Published data: {"orderId":1}
== APP == Published data: {"orderId":2}
== APP == Published data: {"orderId":3}
== APP == Published data: {"orderId":4}
== APP == Published data: {"orderId":5}
== APP == Published data: {"orderId":6}
== APP == Published data: {"orderId":7}
== APP == Published data: {"orderId":8}
== APP == Published data: {"orderId":9}
== APP == Published data: {"orderId":10}
```

`order-processor` output:

```
== APP == Subscriber received: {"orderId":1}
== APP == Subscriber received: {"orderId":2}
== APP == Subscriber received: {"orderId":3}
== APP == Subscriber received: {"orderId":4}
== APP == Subscriber received: {"orderId":5}
== APP == Subscriber received: {"orderId":6}
```

```
== APP == Subscriber received: {"orderId":7}
== APP == Subscriber received: {"orderId":8}
== APP == Subscriber received: {"orderId":9}
== APP == Subscriber received: {"orderId":10}
```

7. Make sure both applications have stopped by running the following commands. In the checkout terminal:

```
sh
dapr stop --app-id checkout
```

- In the order-processor terminal:

```
sh
dapr stop --app-id order-processor
```

## Deploy the application template using Azure Developer CLI

Deploy the application to Azure Container Apps using [azd](#).

### Prepare the project

In a new terminal window, navigate into the [sample's](#) root directory.

```
Bash
cd pubsub-dapr-nodejs-servicebus
```

### Provision and deploy using Azure Developer CLI

1. Run `azd init` to initialize the project.

```
Azure Developer CLI
azd init
```

2. When prompted in the terminal, provide the following parameters.

Parameter	Description
Environment Name	Prefix for the resource group created to hold all Azure resources.
Azure Location	The Azure location for your resources.
Azure Subscription	The Azure subscription for your resources.

3. Run `azd up` to provision the infrastructure and deploy the application to Azure Container Apps in a single command.

Azure Developer CLI

```
azd up
```

This process may take some time to complete. As the `azd up` command completes, the CLI output displays two Azure portal links to monitor the deployment progress. The output also demonstrates how `azd up`:

- Creates and configures all necessary Azure resources via the provided Bicep files in the `./infra` directory using `azd provision`. Once provisioned by Azure Developer CLI, you can access these resources via the Azure portal. The files that provision the Azure resources include:
  - `main.parameters.json`
  - `main.bicep`
  - An `app` resources directory organized by functionality
  - A `core` reference library that contains the Bicep modules used by the `azd template`
- Deploys the code using `azd deploy`

## Expected output

Azure Developer CLI

```
Initializing a new project (azd init)
```

```
Provisioning Azure resources (azd provision)
Provisioning Azure resources can take some time
```

```
You can view detailed progress in the Azure Portal:
https://portal.azure.com
```

```
(✓) Done: Resource group: resource-group-name
(✓) Done: Application Insights: app-insights-name
(✓) Done: Portal dashboard: portal-dashboard-name
(✓) Done: Log Analytics workspace: log-analytics-name
(✓) Done: Key vault: key-vault-name
(✓) Done: Container Apps Environment: ca-env-name
(✓) Done: Container App: ca-checkout-name
(✓) Done: Container App: ca-orders-name
```

Deploying services (azd deploy)

```
(✓) Done: Deploying service checkout
(✓) Done: Deploying service orders
- Endpoint: https://ca-orders-
name.endpoint.region.azurecontainerapps.io/
```

SUCCESS: Your Azure app has been deployed!

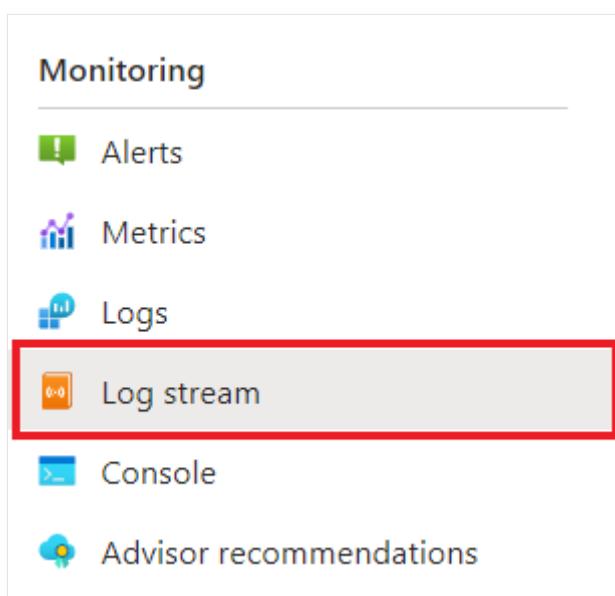
You can view the resources created under the resource group `resource-group-name` in Azure Portal:

<https://portal.azure.com/#@/resource/subscriptions/subscription-id/resourceGroups/resource-group-name/overview>

## Confirm successful deployment

In the Azure portal, verify the `checkout` service is publishing messages to the Azure Service Bus topic.

1. Copy the `checkout` container app name from the terminal output.
2. Sign in to the [Azure portal](#) and search for the container app resource by name.
3. In the Container Apps dashboard, select **Monitoring > Log stream**.



4. Confirm the `checkout` container is logging the same output as in the terminal earlier.

```
Connecting...
2023-03-16T18:31:36.99296 Connecting to the container 'checkoutserv'...
2023-03-16T18:31:37.00856 Successfully Connected to container: 'checkoutserv'
2023-03-16T18:30:56.828036195Z Published data: {"orderId":18}
2023-03-16T18:30:57.840491575Z Published data: {"orderId":19}
2023-03-16T18:30:58.852676997Z Published data: {"orderId":20}
2023-03-16T18:31:19.889447220Z Published data: {"orderId":1}
2023-03-16T18:31:20.960174611Z Published data: {"orderId":2}
2023-03-16T18:31:21.972052331Z Published data: {"orderId":3}
2023-03-16T18:31:22.989037174Z Published data: {"orderId":4}
2023-03-16T18:31:24.005442237Z Published data: {"orderId":5}
2023-03-16T18:31:25.030923118Z Published data: {"orderId":6}
2023-03-16T18:31:26.042206386Z Published data: {"orderId":7}
2023-03-16T18:31:27.064652690Z Published data: {"orderId":8}
2023-03-16T18:31:28.085163847Z Published data: {"orderId":9}
2023-03-16T18:31:29.098112829Z Published data: {"orderId":10}
2023-03-16T18:31:30.130083890Z Published data: {"orderId":11}
2023-03-16T18:31:31.141478370Z Published data: {"orderId":12}
2023-03-16T18:31:32.152914145Z Published data: {"orderId":13}
2023-03-16T18:31:33.178869633Z Published data: {"orderId":14}
2023-03-16T18:31:34.197171578Z Published data: {"orderId":15}
2023-03-16T18:31:35.215483774Z Published data: {"orderId":16}
2023-03-16T18:31:36.239403619Z Published data: {"orderId":17}
2023-03-16T18:31:37.250161693Z Published data: {"orderId":18}
2023-03-16T18:31:38.271098752Z Published data: {"orderId":19}
2023-03-16T18:31:39.282488421Z Published data: {"orderId":20}
2023-03-16T18:32:00.299556190Z Published data: {"orderId":1}
2023-03-16T18:32:01.313715936Z Published data: {"orderId":2}
2023-03-16T18:32:02.324658618Z Published data: {"orderId":3}
2023-03-16T18:32:03.359539159Z Published data: {"orderId":4}
2023-03-16T18:32:04.370340378Z Published data: {"orderId":5}
2023-03-16T18:32:05.391758104Z Published data: {"orderId":6}
2023-03-16T18:32:06.416020741Z Published data: {"orderId":7}
2023-03-16T18:32:07.436700983Z Published data: {"orderId":8}
2023-03-16T18:32:08.451113620Z Published data: {"orderId":9}
2023-03-16T18:32:09.464197086Z Published data: {"orderId":10}
2023-03-16T18:32:10.480107097Z Published data: {"orderId":11}
2023-03-16T18:32:11.491068984Z Published data: {"orderId":12}
2023-03-16T18:32:12.501874767Z Published data: {"orderId":13}
2023-03-16T18:32:13.513070401Z Published data: {"orderId":14}
2023-03-16T18:32:14.524622724Z Published data: {"orderId":15}
2023-03-16T18:32:15.538885437Z Published data: {"orderId":16}
2023-03-16T18:32:16.550466423Z Published data: {"orderId":17}
2023-03-16T18:32:17.560761280Z Published data: {"orderId":18}
2023-03-16T18:32:18.574345037Z Published data: {"orderId":19}
2023-03-16T18:32:19.592939253Z Published data: {"orderId":20}
```

□

5. Do the same for the `order-processor` service.

```
Connecting...
2023-03-16T18:32:36.51236 Connecting to the container 'orderssvc'...
2023-03-16T18:32:36.52492 Successfully Connected to container: 'orderssvc'
2023-03-16T18:32:00.301362583Z Subscriber received: {"orderId":1}
2023-03-16T18:32:01.318856533Z Subscriber received: {"orderId":2}
2023-03-16T18:32:02.330846489Z Subscriber received: {"orderId":3}
2023-03-16T18:32:03.377727556Z Subscriber received: {"orderId":4}
2023-03-16T18:32:04.372099637Z Subscriber received: {"orderId":5}
2023-03-16T18:32:05.394407385Z Subscriber received: {"orderId":6}
2023-03-16T18:32:06.420501724Z Subscriber received: {"orderId":7}
2023-03-16T18:32:07.441036131Z Subscriber received: {"orderId":8}
2023-03-16T18:32:08.454883619Z Subscriber received: {"orderId":9}
2023-03-16T18:32:09.466191591Z Subscriber received: {"orderId":10}
2023-03-16T18:32:10.488832713Z Subscriber received: {"orderId":11}
2023-03-16T18:32:11.492296399Z Subscriber received: {"orderId":12}
2023-03-16T18:32:12.523131890Z Subscriber received: {"orderId":13}
2023-03-16T18:32:13.518628045Z Subscriber received: {"orderId":14}
2023-03-16T18:32:14.528746484Z Subscriber received: {"orderId":15}
2023-03-16T18:32:15.547271429Z Subscriber received: {"orderId":16}
2023-03-16T18:32:16.558563837Z Subscriber received: {"orderId":17}
2023-03-16T18:32:17.561754545Z Subscriber received: {"orderId":18}
2023-03-16T18:32:18.588262841Z Subscriber received: {"orderId":19}
2023-03-16T18:32:19.642823280Z Subscriber received: {"orderId":20}
2023-03-16T18:32:40.683125039Z Subscriber received: {"orderId":1}
2023-03-16T18:32:41.721052948Z Subscriber received: {"orderId":2}
2023-03-16T18:32:42.751906853Z Subscriber received: {"orderId":3}
2023-03-16T18:32:43.769318986Z Subscriber received: {"orderId":4}
2023-03-16T18:32:44.879218480Z Subscriber received: {"orderId":5}
2023-03-16T18:32:45.926990011Z Subscriber received: {"orderId":6}
2023-03-16T18:32:46.918026086Z Subscriber received: {"orderId":7}
2023-03-16T18:32:47.927076538Z Subscriber received: {"orderId":8}
2023-03-16T18:32:48.942846717Z Subscriber received: {"orderId":9}
2023-03-16T18:32:49.965427846Z Subscriber received: {"orderId":10}
2023-03-16T18:32:51.004109254Z Subscriber received: {"orderId":11}
2023-03-16T18:32:52.009043018Z Subscriber received: {"orderId":12}
2023-03-16T18:32:53.037067302Z Subscriber received: {"orderId":13}
2023-03-16T18:32:54.047611916Z Subscriber received: {"orderId":14}
2023-03-16T18:32:55.058429174Z Subscriber received: {"orderId":15}
2023-03-16T18:32:56.069449392Z Subscriber received: {"orderId":16}
2023-03-16T18:32:57.097965149Z Subscriber received: {"orderId":17}
2023-03-16T18:32:58.094564698Z Subscriber received: {"orderId":18}
2023-03-16T18:32:59.117086229Z Subscriber received: {"orderId":19}
2023-03-16T18:33:00.146536418Z Subscriber received: {"orderId":20}
```

## What happened?

Upon successful completion of the `azd up` command:

- Azure Developer CLI provisioned the Azure resources referenced in the [sample project's ./infra directory](#) to the Azure subscription you specified. You can now view those Azure resources via the Azure portal.
- The app deployed to Azure Container Apps. From the portal, you can browse to the fully functional app.

## Clean up resources

If you're not going to continue to use this application, delete the Azure resources you've provisioned with the following command:

```
azd down
```

## Next steps

- Learn more about [deploying applications to Azure Container Apps](#).
- [Enable token authentication for Dapr requests](#).
- Learn more about [Azure Developer CLI](#) and [making your applications compatible with azd](#).
- [Scale your applications using KEDA scalers](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

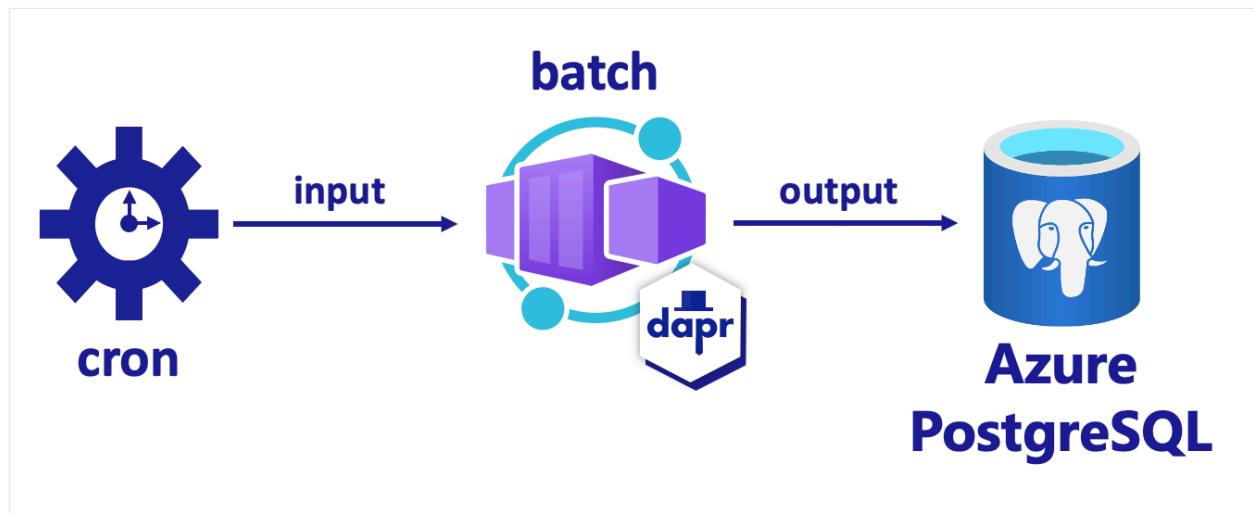
# Tutorial: Event-driven work using Dapr Bindings

Article • 08/05/2024

In this tutorial, you create a microservice to demonstrate [Dapr's Bindings API](#) to work with external systems as inputs and outputs. You'll:

- ✓ Run the application locally with the Dapr CLI.
- ✓ Deploy the application to Azure Container Apps via the Azure Developer CLI with the provided Bicep.

The service listens to input binding events from a system CRON and then outputs the contents of local data to a PostgreSQL output binding.



## Prerequisites

- Install [Azure Developer CLI](#)
- Install [Dapr](#) and [init](#) Dapr
- [Docker Desktop](#)
- Install [Git](#)

## Run the Node.js application locally

Before deploying the application to Azure Container Apps, start by running the PostgreSQL container and JavaScript service locally with [Docker Compose](#) and Dapr.

## Prepare the project

1. Clone the [sample application](#) to your local machine.

```
Bash
```

```
git clone https://github.com/Azure-Samples/bindings-dapr-nodejs-cron-postgres.git
```

2. Navigate into the sample's root directory.

```
Bash
```

```
cd bindings-dapr-nodejs-cron-postgres
```

## Run the application using the Dapr CLI

1. From the sample's root directory, change directories to `db`.

```
Bash
```

```
cd db
```

2. Run the PostgreSQL container with Docker Compose.

```
Bash
```

```
docker compose up -d
```

3. Open a new terminal window and navigate into `/batch` in the sample directory.

```
Bash
```

```
cd bindings-dapr-nodejs-cron-postgres/batch
```

4. Install the dependencies.

```
Bash
```

```
npm install
```

5. Run the JavaScript service application.

```
Bash
```

```
dapr run --app-id batch-sdk --app-port 5002 --dapr-http-port 3500 --resources-path ../components -- node index.js
```

The `dapr run` command runs the binding application locally. Once the application is running successfully, the terminal window shows the output binding data.

## Expected output

The batch service listens to input binding events from a system CRON and then outputs the contents of local data to a PostgreSQL output binding.

```
== APP == {"sql": "insert into orders (orderid, customer, price) values (1, 'John Smith', 100.32);"}
== APP == {"sql": "insert into orders (orderid, customer, price) values (2, 'Jane Bond', 15.4);"}
== APP == {"sql": "insert into orders (orderid, customer, price) values (3, 'Tony James', 35.56);"}
== APP == Finished processing batch
== APP == {"sql": "insert into orders (orderid, customer, price) values (1, 'John Smith', 100.32);"}
== APP == {"sql": "insert into orders (orderid, customer, price) values (2, 'Jane Bond', 15.4);"}
== APP == {"sql": "insert into orders (orderid, customer, price) values (3, 'Tony James', 35.56);"}
== APP == Finished processing batch
== APP == {"sql": "insert into orders (orderid, customer, price) values (1, 'John Smith', 100.32);"}
== APP == {"sql": "insert into orders (orderid, customer, price) values (2, 'Jane Bond', 15.4);"}
== APP == {"sql": "insert into orders (orderid, customer, price) values (3, 'Tony James', 35.56);"}
== APP == Finished processing batch
```

6. In the `./db` terminal, stop the PostgreSQL container.

Bash

```
docker compose stop
```

## Deploy the application template using Azure Developer CLI

Now that you've run the application locally, let's deploy the bindings application to Azure Container Apps using `azd`. During deployment, we will swap the local containerized PostgreSQL for an Azure PostgreSQL component.

## Prepare the project

Navigate into the [sample's](#) root directory.

```
Bash
```

```
cd bindings-dapr-nodejs-cron-postgres
```

## Provision and deploy using Azure Developer CLI

1. Run `azd init` to initialize the project.

```
Azure Developer CLI
```

```
azd init
```

2. When prompted in the terminal, provide the following parameters.

[ Expand table

Parameter	Description
Environment Name	Prefix for the resource group created to hold all Azure resources.
Azure Location	The Azure location for your resources. <a href="#">Make sure you select a location available for Azure PostgreSQL.</a>
Azure Subscription	The Azure subscription for your resources.

3. Run `azd up` to provision the infrastructure and deploy the application to Azure Container Apps in a single command.

```
Azure Developer CLI
```

```
azd up
```

This process may take some time to complete. As the `azd up` command completes, the CLI output displays two Azure portal links to monitor the deployment progress. The output also demonstrates how `azd up`:

- Creates and configures all necessary Azure resources via the provided Bicep files in the `./infra` directory using `azd provision`. Once provisioned by Azure Developer CLI, you can access these resources via the Azure portal. The files that provision the Azure resources include:
  - `main.parameters.json`
  - `main.bicep`
  - An `app` resources directory organized by functionality
  - A `core` reference library that contains the Bicep modules used by the `azd template`
- Deploys the code using `azd deploy`

## Expected output

```
Azure Developer CLI

Initializing a new project (azd init)

Provisioning Azure resources (azd provision)
Provisioning Azure resources can take some time

You can view detailed progress in the Azure Portal:

https://portal.azure.com/#blade/HubsExtension/DeploymentDetailsBlade/overview

(✓) Done: Resource group: resource-group-name
(✓) Done: Log Analytics workspace: log-analytics-name
(✓) Done: Application Insights: app-insights-name
(✓) Done: Portal dashboard: dashboard-name
(✓) Done: Azure Database for PostgreSQL flexible server: postgres-server
(✓) Done: Key vault: key-vault-name
(✓) Done: Container Apps Environment: container-apps-env-name
(✓) Done: Container App: container-app-name

Deploying services (azd deploy)

(✓) Done: Deploying service api
- Endpoint: https://your-container-app-endpoint.region.azurecontainerapps.io/

SUCCESS: Your Azure app has been deployed!
```

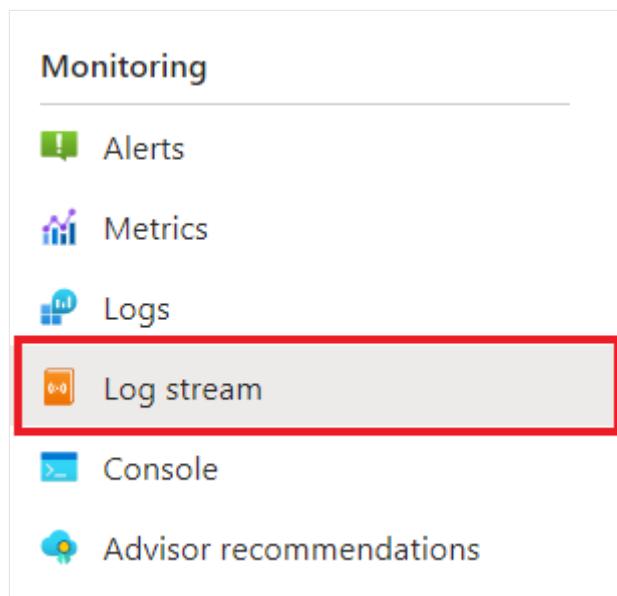
You can view the resources created under the resource group `resource-group-name` in Azure Portal:

<https://portal.azure.com/#@/resource/subscriptions/your-subscription-ID/resourceGroups/your-resource-group/overview>

## Confirm successful deployment

In the Azure portal, verify the batch container app is logging each insert into Azure PostgreSQL every 10 seconds.

1. Copy the Container App name from the terminal output.
2. Sign in to the [Azure portal](#) and search for the Container App resource by name.
3. In the Container App dashboard, select **Monitoring > Log stream**.



4. Confirm the container is logging the same output as in the terminal earlier.

```
Connecting...
2023-03-16T17:43:18.39918 Connecting to the container 'batch'...
2023-03-16T17:43:18.42740 Successfully Connected to container: 'batch'
2023-03-16T17:42:29.002081843Z {"sql": "insert into orders (orderid, customer, price) values (1, 'John Smith', 100.32);"}
2023-03-16T17:42:29.002486595Z {"sql": "insert into orders (orderid, customer, price) values (2, 'Jane Bond', 15.4);"}
2023-03-16T17:42:29.002712521Z {"sql": "insert into orders (orderid, customer, price) values (3, 'Tony James', 35.56);"}
2023-03-16T17:42:29.002949202Z Finished processing batch
2023-03-16T17:42:39.001177919Z {"sql": "insert into orders (orderid, customer, price) values (1, 'John Smith', 100.32);"}
2023-03-16T17:42:39.001611119Z {"sql": "insert into orders (orderid, customer, price) values (2, 'Jane Bond', 15.4);"}
2023-03-16T17:42:39.001863501Z {"sql": "insert into orders (orderid, customer, price) values (3, 'Tony James', 35.56);"}
2023-03-16T17:42:39.002089563Z Finished processing batch
2023-03-16T17:42:49.002300437Z {"sql": "insert into orders (orderid, customer, price) values (1, 'John Smith', 100.32);"}
2023-03-16T17:42:49.002666356Z {"sql": "insert into orders (orderid, customer, price) values (2, 'Jane Bond', 15.4);"}
2023-03-16T17:42:49.002895265Z {"sql": "insert into orders (orderid, customer, price) values (3, 'Tony James', 35.56);"}
2023-03-16T17:42:49.003144817Z Finished processing batch
2023-03-16T17:42:59.001664695Z {"sql": "insert into orders (orderid, customer, price) values (1, 'John Smith', 100.32);"}
2023-03-16T17:42:59.001928185Z {"sql": "insert into orders (orderid, customer, price) values (2, 'Jane Bond', 15.4);"}
2023-03-16T17:42:59.002169928Z {"sql": "insert into orders (orderid, customer, price) values (3, 'Tony James', 35.56);"}
2023-03-16T17:42:59.002379279Z Finished processing batch
2023-03-16T17:43:09.001854673Z {"sql": "insert into orders (orderid, customer, price) values (1, 'John Smith', 100.32);"}
2023-03-16T17:43:09.002184793Z {"sql": "insert into orders (orderid, customer, price) values (2, 'Jane Bond', 15.4);"}
2023-03-16T17:43:09.002425682Z {"sql": "insert into orders (orderid, customer, price) values (3, 'Tony James', 35.56);"}
2023-03-16T17:43:09.002666605Z Finished processing batch
2023-03-16T17:43:19.001859295Z {"sql": "insert into orders (orderid, customer, price) values (1, 'John Smith', 100.32);"}
2023-03-16T17:43:19.002333945Z {"sql": "insert into orders (orderid, customer, price) values (2, 'Jane Bond', 15.4);"}
2023-03-16T17:43:19.002589029Z {"sql": "insert into orders (orderid, customer, price) values (3, 'Tony James', 35.56);"}
2023-03-16T17:43:19.002875744Z Finished processing batch
2023-03-16T17:43:29.001927907Z {"sql": "insert into orders (orderid, customer, price) values (1, 'John Smith', 100.32);"}
2023-03-16T17:43:29.002381054Z {"sql": "insert into orders (orderid, customer, price) values (2, 'Jane Bond', 15.4);"}
2023-03-16T17:43:29.003518597Z {"sql": "insert into orders (orderid, customer, price) values (3, 'Tony James', 35.56);"}
2023-03-16T17:43:29.003551480Z Finished processing batch
```

# What happened?

Upon successful completion of the `azd up` command:

- Azure Developer CLI provisioned the Azure resources referenced in the [sample project's ./infra directory](#) to the Azure subscription you specified. You can now view those Azure resources via the Azure portal.
- The app deployed to Azure Container Apps. From the portal, you can browse the fully functional app.

## Clean up resources

If you're not going to continue to use this application, delete the Azure resources you've provisioned with the following command.

```
Azure Developer CLI
```

```
azd down
```

## Next steps

- Learn more about [deploying microservices using Dapr to Azure Container Apps](#).
- [Enable token authentication for Dapr requests](#).
- Learn more about [Azure Developer CLI](#) and [making your applications compatible with azd](#).
- [Scale your applications using KEDA scalers](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# Tutorial: Microservices communication using Dapr Service Invocation

Article • 08/05/2024

In this tutorial, you create and run two microservices that communicate securely using auto-mTLS and reliably using built-in retries via [the Dapr Service Invocation API](#). You'll:

- ✓ Run the application locally.
- ✓ Deploy the application to Azure Container Apps via the Azure Developer CLI with the provided Bicep.

The sample service invocation project includes:

1. A `checkout` service that uses HTTP proxying on a loop to invoke a request on the `order-processor` service.
2. An `order-processor` service that receives the request from the `checkout` service.



## Prerequisites

- Install [Azure Developer CLI](#)
- [Install](#) and [init](#) Dapr
- [Docker Desktop](#)
- Install [Git](#)

## Run the Node.js applications locally

Before deploying the application to Azure Container Apps, start by running the `order-processor` and `checkout` services locally with Dapr.

## Prepare the project

1. Clone the [sample applications](#) to your local machine.

```
Bash
```

```
git clone https://github.com/Azure-Samples/svc-invoke-dapr-nodejs.git
```

2. Navigate into the sample's root directory.

```
Bash
```

```
cd svc-invoke-dapr-nodejs
```

## Run the applications using the Dapr CLI

Start by running the `order-processor` service.

1. From the sample's root directory, change directories to `order-processor`.

```
Bash
```

```
cd order-processor
```

2. Install the dependencies.

```
Bash
```

```
npm install
```

3. Run the `order-processor` service.

```
Bash
```

```
dapr run --app-port 5001 --app-id order-processor --app-protocol http -
-dapr-http-port 3501 -- npm start
```

4. In a new terminal window, from the sample's root directory, navigate to the `checkout` caller service.

```
Bash
```

```
cd checkout
```

5. Install the dependencies.

```
Bash
```

```
npm install
```

6. Run the `checkout` service.

```
Bash
```

```
dapr run --app-id checkout --app-protocol http --dapr-http-port 3500 -
- npm start
```

## Expected output

In both terminals, the `checkout` service is calling orders to the `order-processor` service in a loop.

`checkout` output:

```
== APP == Order passed: {"orderId":1}
== APP == Order passed: {"orderId":2}
== APP == Order passed: {"orderId":3}
== APP == Order passed: {"orderId":4}
== APP == Order passed: {"orderId":5}
== APP == Order passed: {"orderId":6}
== APP == Order passed: {"orderId":7}
== APP == Order passed: {"orderId":8}
== APP == Order passed: {"orderId":9}
== APP == Order passed: {"orderId":10}
== APP == Order passed: {"orderId":11}
== APP == Order passed: {"orderId":12}
== APP == Order passed: {"orderId":13}
== APP == Order passed: {"orderId":14}
== APP == Order passed: {"orderId":15}
== APP == Order passed: {"orderId":16}
== APP == Order passed: {"orderId":17}
== APP == Order passed: {"orderId":18}
== APP == Order passed: {"orderId":19}
== APP == Order passed: {"orderId":20}
```

`order-processor` output:

```
== APP == Order received: { orderId: 1 }
== APP == Order received: { orderId: 2 }
```

```
== APP == Order received: { orderId: 3 }
== APP == Order received: { orderId: 4 }
== APP == Order received: { orderId: 5 }
== APP == Order received: { orderId: 6 }
== APP == Order received: { orderId: 7 }
== APP == Order received: { orderId: 8 }
== APP == Order received: { orderId: 9 }
== APP == Order received: { orderId: 10 }
== APP == Order received: { orderId: 11 }
== APP == Order received: { orderId: 12 }
== APP == Order received: { orderId: 13 }
== APP == Order received: { orderId: 14 }
== APP == Order received: { orderId: 15 }
== APP == Order received: { orderId: 16 }
== APP == Order received: { orderId: 17 }
== APP == Order received: { orderId: 18 }
== APP == Order received: { orderId: 19 }
== APP == Order received: { orderId: 20 }
```

7. Press `Cmd/Ctrl` + `c` in both terminals to exit out of the service-to-service invocation.

## Deploy the application template using Azure Developer CLI

Deploy the application to Azure Container Apps using [azd](#).

### Prepare the project

In a new terminal window, navigate into the sample's root directory.

```
Bash
```

```
cd svc-invoke-dapr-nodejs
```

### Provision and deploy using Azure Developer CLI

1. Run `azd init` to initialize the project.

```
Azure Developer CLI
```

```
azd init
```

2. When prompted in the terminal, provide the following parameters.

Parameter	Description
Environment Name	Prefix for the resource group created to hold all Azure resources.
Azure Location	The Azure location for your resources.
Azure Subscription	The Azure subscription for your resources.

3. Run `azd up` to provision the infrastructure and deploy the application to Azure Container Apps in a single command.

Azure Developer CLI

```
azd up
```

This process may take some time to complete. As the `azd up` command completes, the CLI output displays two Azure portal links to monitor the deployment progress. The output also demonstrates how `azd up`:

- Creates and configures all necessary Azure resources via the provided Bicep files in the `./infra` directory using `azd provision`. Once provisioned by Azure Developer CLI, you can access these resources via the Azure portal. The files that provision the Azure resources include:
  - `main.parameters.json`
  - `main.bicep`
  - An `app` resources directory organized by functionality
  - A `core` reference library that contains the Bicep modules used by the `azd template`
- Deploys the code using `azd deploy`

## Expected output

Azure Developer CLI

```
Initializing a new project (azd init)
```

```
Provisioning Azure resources (azd provision)
Provisioning Azure resources can take some time
```

```
You can view detailed progress in the Azure Portal:
https://portal.azure.com
```

```
(✓) Done: Resource group: resource-group-name
```

```
(✓) Done: Log Analytics workspace: log-analytics-name
(✓) Done: Application Insights: app-insights-name
(✓) Done: Portal dashboard: dashboard-name
(✓) Done: Container Apps Environment: container-apps-env-name
(✓) Done: Container App: ca-checkout-name
(✓) Done: Container App: ca-order-processor-name
```

Deploying services (azd deploy)

```
(✓) Done: Deploying service api
- Endpoint: https://ca-order-processor-
name.eastus.azurecontainerapps.io/
(✓) Done: Deploying service worker
```

SUCCESS: Your Azure app has been deployed!

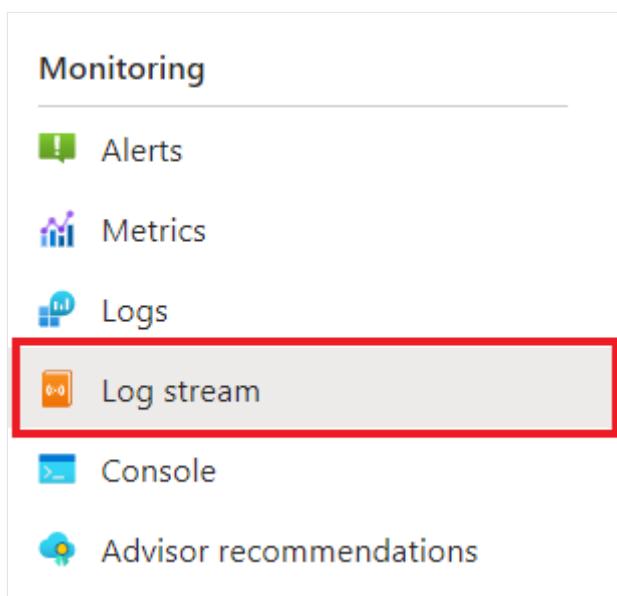
You can view the resources created under the resource group `resource-group-name` in Azure Portal:

<https://portal.azure.com/#@/resource/subscriptions/<your-azure-subscription>/resourceGroups/resource-group-name/overview>

## Confirm successful deployment

In the Azure portal, verify the `checkout` service is passing orders to the `order-processor` service.

1. Copy the `checkout` container app's name from the terminal output.
2. Sign in to the [Azure portal](#) and search for the container app resource by name.
3. In the Container Apps dashboard, select **Monitoring > Log stream**.



4. Confirm the `checkout` container is logging the same output as in the terminal earlier.

```
Connecting...
2023-02-17T17:18:44.44752 Connecting to the container 'checkout'...
2023-02-17T17:18:44.46918 Successfully Connected to container: 'checkout'

2023-02-17T17:18:10.857906291Z Order passed: {"orderId":1}
2023-02-17T17:18:11.863507650Z Order passed: {"orderId":2}
2023-02-17T17:18:12.868358339Z Order passed: {"orderId":3}
2023-02-17T17:18:13.873600279Z Order passed: {"orderId":4}
2023-02-17T17:18:14.878059227Z Order passed: {"orderId":5}
2023-02-17T17:18:15.883034650Z Order passed: {"orderId":6}
2023-02-17T17:18:16.889526921Z Order passed: {"orderId":7}
2023-02-17T17:18:17.894321469Z Order passed: {"orderId":8}
2023-02-17T17:18:18.897729650Z Order passed: {"orderId":9}
2023-02-17T17:18:19.902510277Z Order passed: {"orderId":10}
2023-02-17T17:18:20.908565025Z Order passed: {"orderId":11}
2023-02-17T17:18:21.913487198Z Order passed: {"orderId":12}
2023-02-17T17:18:22.919039820Z Order passed: {"orderId":13}
2023-02-17T17:18:23.924864640Z Order passed: {"orderId":14}
2023-02-17T17:18:24.930955183Z Order passed: {"orderId":15}
2023-02-17T17:18:25.935990278Z Order passed: {"orderId":16}
2023-02-17T17:18:26.941350932Z Order passed: {"orderId":17}
2023-02-17T17:18:27.946330327Z Order passed: {"orderId":18}
2023-02-17T17:18:28.950993193Z Order passed: {"orderId":19}
2023-02-17T17:18:29.954949436Z Order passed: {"orderId":20}
2023-02-17T17:18:50.979894555Z Order passed: {"orderId":1}
2023-02-17T17:18:51.985474976Z Order passed: {"orderId":2}
2023-02-17T17:18:52.990299145Z Order passed: {"orderId":3}
2023-02-17T17:18:53.993957362Z Order passed: {"orderId":4}
2023-02-17T17:18:54.998628641Z Order passed: {"orderId":5}
```

## 5. Do the same for the `order-processor` service.

```
Connecting...
2023-02-17T17:24:16.58999 Connecting to the container 'order-processor'...
2023-02-17T17:24:16.61026 Successfully Connected to container: 'order-processor'
2023-02-17T17:23:36.805887872Z Order received: { orderId: 6 }
2023-02-17T17:23:37.810033680Z Order received: { orderId: 7 }
2023-02-17T17:23:38.814930622Z Order received: { orderId: 8 }
2023-02-17T17:23:39.819613502Z Order received: { orderId: 9 }
2023-02-17T17:23:40.823327552Z Order received: { orderId: 10 }
2023-02-17T17:23:41.828664748Z Order received: { orderId: 11 }
2023-02-17T17:23:42.833989666Z Order received: { orderId: 12 }
2023-02-17T17:23:43.837952569Z Order received: { orderId: 13 }
2023-02-17T17:23:44.842654830Z Order received: { orderId: 14 }
2023-02-17T17:23:45.848068174Z Order received: { orderId: 15 }
2023-02-17T17:23:46.852792769Z Order received: { orderId: 16 }
2023-02-17T17:23:47.857529824Z Order received: { orderId: 17 }
2023-02-17T17:23:48.862937841Z Order received: { orderId: 18 }
2023-02-17T17:23:49.867843896Z Order received: { orderId: 19 }
2023-02-17T17:23:50.872403776Z Order received: { orderId: 20 }
2023-02-17T17:24:11.895782198Z Order received: { orderId: 1 }
2023-02-17T17:24:12.899940853Z Order received: { orderId: 2 }
2023-02-17T17:24:13.904718259Z Order received: { orderId: 3 }
2023-02-17T17:24:14.910750405Z Order received: { orderId: 4 }
2023-02-17T17:24:15.915615615Z Order received: { orderId: 5 }
2023-02-17T17:24:16.919299658Z Order received: { orderId: 6 }
2023-02-17T17:24:17.924038953Z Order received: { orderId: 7 }
2023-02-17T17:24:18.927722608Z Order received: { orderId: 8 }
2023-02-17T17:24:19.933345036Z Order received: { orderId: 9 }
2023-02-17T17:24:20.939099248Z Order received: { orderId: 10 }
```

# What happened?

Upon successful completion of the `azd up` command:

- Azure Developer CLI provisioned the Azure resources referenced in the [sample project's ./infra directory](#) to the Azure subscription you specified. You can now view those Azure resources via the Azure portal.
- The app deployed to Azure Container Apps. From the portal, you can browse the fully functional app.

## Clean up resources

If you're not going to continue to use this application, delete the Azure resources you've provisioned with the following command:

```
Azure Developer CLI
```

```
azd down
```

## Next steps

- Learn more about [deploying Dapr applications to Azure Container Apps](#).
- [Enable token authentication for Dapr requests](#).
- Learn more about [Azure Developer CLI](#) and [making your applications compatible with azd](#).

---

## Feedback

Was this page helpful?

 Yes

 No

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

# Deploy the Dapr extension for Azure Functions in Azure Container Apps (preview)

Article • 08/05/2024

The [Dapr extension for Azure Functions](#) allows you to easily interact with the Dapr APIs from an Azure Function using triggers and bindings. In this guide, you learn how to:

- Create an Azure Redis Cache for use as a Dapr statestore
- Deploy an Azure Container Apps environment to host container apps
- Deploy a Dapr-enabled function on Azure Container Apps:
  - One function that invokes the other service
  - One function that creates an Order and saves it to storage via Dapr statestore
- Verify the interaction between the two apps

## Prerequisites

- [An Azure account with an active subscription.](#)
- [Install Azure CLI](#)

## Set up the environment

1. In the terminal, log in to your Azure subscription.

```
Azure CLI
```

```
az login
```

2. Set up your Azure login with the active subscription you'd like to use.

```
Azure CLI
```

```
az account set --subscription {subscription-id-or-name}
```

3. Clone the [Dapr extension for Azure Functions repo](#).

```
Azure CLI
```

```
git clone https://github.com/Azure/azure-functions-dapr-extension.git
```

# Create resource group

## ⓘ Note

Azure Container Apps support for Functions is currently in preview and available in the following regions.

- Australia East
- Central US
- East US
- East US 2
- North Europe
- South Central US
- UK South
- West Europe
- West US 3

Specifying one of the available regions, create a resource group for your container app.

Azure CLI

```
az group create --name {resourceGroupName} --location {region}
```

# Deploy the Azure Function templates

1. From the root directory, change into the folder holding the template.

Azure CLI

```
cd quickstarts/dotnet-isolated/deploy/aca
```

2. Create a deployment group and specify the template you'd like to deploy.

Azure CLI

```
az deployment group create --resource-group {resourceGroupName} --template-file deploy-quickstart.bicep
```

3. When prompted by the CLI, enter a resource name prefix. The name you choose must be a combination of numbers and lowercase letters, 3 and 24 characters in

length.

```
Please provide string value for 'resourceNamePrefix' (? for help):
{your-resource-name-prefix}
```

The template deploys the following resources and might take a while:

- A Container App Environment
- A Function App
- An Azure Blob Storage Account and a default storage container
- Application Insights
- Log Analytics WorkSpace
- Dapr Component (Azure Redis Cache) for State Management
- The following .NET Dapr-enabled Functions:
  - OrderService
  - CreateNewOrder
  - RetrieveOrder

4. In the Azure portal, navigate to your resource group and select **Deployments** to track the deployment status.

Deployment name	Status	Last modified	Duration	Related events
azure-function	Succeeded	10/13/2023, 9:25:52 AM	1 minute, 26 seconds, 294 millis...	<a href="#">Related events</a>
dapr-components	Succeeded	10/13/2023, 9:24:07 AM	1 second, 115 milliseconds	<a href="#">Related events</a>
Failure-Anomalies-Alert-Rule-Deployment...	Succeeded	10/13/2023, 9:04:40 AM	1 second, 85 milliseconds	<a href="#">Related events</a>
azure-services	Succeeded	10/13/2023, 9:24:00 AM	29 minutes, 41 seconds, 779 millis...	<a href="#">Related events</a>
functionson-on-aca	Succeeded	10/13/2023, 9:25:58 AM	31 minutes, 41 seconds, 54 millis...	<a href="#">Related events</a>
deploy-quickstart	Succeeded	10/13/2023, 9:26:05 AM	31 minutes, 48 seconds, 579 millis...	<a href="#">Related events</a>

## Verify the result

Once the template has deployed successfully, run the following command to initiate an `OrderService` function that triggers the `CreateNewOrder` process. A new order is created and stored in the Redis statestore.

In the command:

- Replace `{quickstart-functionapp-url}` with your actual function app URL. For example: `https://daprext-funcapp.wittyglacier-20884174.eastus.azurecontainerapps.io`.
- Replace `{quickstart-functionapp-name}` with your function app name.

PowerShell

PowerShell

```
Invoke-RestMethod -Uri 'https://'{quickstart-functionapp-
url}.io/api/invoke/{quickstart-functionapp-name}/CreateNewOrder' -Method
POST -Headers @{"Content-Type" = "application/json"} -Body '{
 "data": {
 "value": {
 "orderId": "Order22"
 }
 }
}'
```

## View logs

Data logged via a function app is stored in the `ContainerAppConsoleLogs_CL` custom table in the Log Analytics workspace. Wait a few minutes for the analytics to arrive for the first time before you query the logged data.

You can view logs through the Azure portal or from the command line.

### Via the Azure portal

1. Navigate to your container app environment.
2. In the left side menu, under **Monitoring**, select **Logs**.
3. Run a query like the following to verify your function app is receiving the invoked message from Dapr.

```
ContainerAppsConsoleLogs_CL
| where RevisionName_s == $revision_name
| where Log_s contains "Order22"
| project Log_s
```

The screenshot shows the Azure Log Analytics workspace interface. At the top, there are navigation links: Run, Time range (Last 24 hours), Save, Share, New alert rule, Export, and Pi. Below the header, a query is displayed:

```
1 ContainerAppConsoleLogs_CL
2 | where RevisionName_s == "testdapfunctions-funcapp--mc9ij8f"
3 | where Log_s has "Order22"
4 | project Log_s
```

Below the query, there are tabs for Results and Chart. The Results tab is selected, showing a list of log entries under the heading Log\_s:

- > {"EventType": "MS\_FUNCTION\_AZURE\_MONITOR\_EVENT", "Level": 4, "ResourceId": "testdapfunctions-funcapp.azurewebsites.net", "order": "Order22"}
- > Received Payload CreateNewOrder: {"data": {"value": {"orderId": "Order22"}}}
- > {"EventType": "MS\_FUNCTION\_AZURE\_MONITOR\_EVENT", "Level": 4, "ResourceId": "testdapfunctions-funcapp.azurewebsites.net", "order": "Order22"}
- > Retrieved order: {"orderId": "Order22"}
- > {"EventType": "MS\_FUNCTION\_AZURE\_MONITOR\_EVENT", "Level": 4, "ResourceId": "testdapfunctions-funcapp.azurewebsites.net", "order": "Order22"}

## Via the Azure CLI

Run the following command to view the saved state.

The screenshot shows the Azure CLI interface with the PowerShell tab selected. A command is displayed in the terminal window:

```
Invoke-RestMethod -Uri 'https://{{quickstart-functionapp- url}.io}/api/retrieveorder' -Method GET
```

## Clean up resources

Once you're finished with this tutorial, run the following command to delete your resource group, along with all the resources you created.

```
az group delete --resource-group $RESOURCE_GROUP
```

## Related links

- Learn more about the Dapr extension for Azure Functions
  - Learn more about connecting Dapr components to your container app
- 

## Feedback

Was this page helpful?

 Yes

 No

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

# Tutorial: Deploy a Dapr application to Azure Container Apps using the Azure CLI

Article • 08/21/2024

Dapr [🔗](#) (Distributed Application Runtime) helps developers build resilient, reliable microservices. In this tutorial, a sample Dapr application is deployed to Azure Container Apps.

You learn how to:

- ✓ Create a Container Apps environment for your container apps
- ✓ Create an Azure Blob Storage state store for the container app
- ✓ Deploy two apps that produce and consume messages and persist them in the state store
- ✓ Verify the interaction between the two microservices.

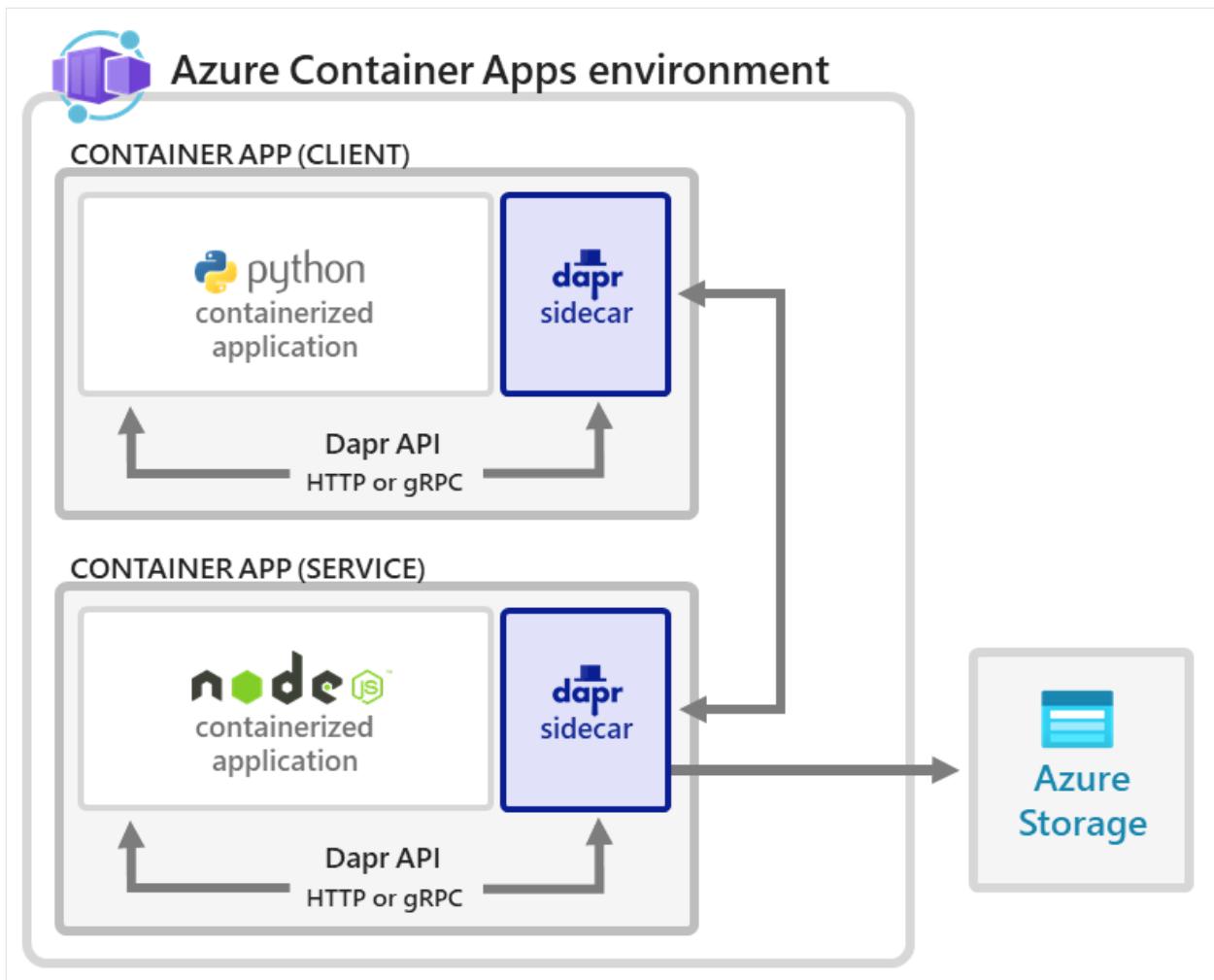
With Azure Container Apps, you get a [fully managed version of the Dapr APIs](#) when building microservices. When you use Dapr in Azure Container Apps, you can enable sidecars to run next to your microservices that provide a rich set of capabilities. Available Dapr APIs include [Service to Service calls](#) [🔗](#), [Pub/Sub](#) [🔗](#), [Event Bindings](#) [🔗](#), [State Stores](#) [🔗](#), and [Actors](#) [🔗](#).

In this tutorial, you deploy the same applications from the Dapr [Hello World](#) [🔗](#) quickstart.

The application consists of:

- A client (Python) container app to generate messages.
- A service (Node) container app to consume and persist those messages in a state store

The following architecture diagram illustrates the components that make up this tutorial:



## Setup

To sign in to Azure from the CLI, run the following command and follow the prompts to complete the authentication process.

```
Bash
az login
```

To ensure you're running the latest version of the CLI, run the upgrade command.

```
Bash
az upgrade
```

Next, install or update the Azure Container Apps extension for the CLI.

If you receive errors about missing parameters when you run `az containerapp` commands in Azure CLI or cmdlets from the `Az.App` module in Azure PowerShell, be sure you have the latest version of the Azure Container Apps extension installed.

Bash

Azure CLI

```
az extension add --name containerapp --upgrade
```

⚠ Note

Starting in May 2024, Azure CLI extensions no longer enable preview features by default. To access Container Apps [preview features](#), install the Container Apps extension with `--allow-preview true`.

Azure CLI

```
az extension add --name containerapp --upgrade --allow-preview true
```

Now that the current extension or module is installed, register the `Microsoft.App` and `Microsoft.OperationalInsights` namespaces.

⚠ Note

Azure Container Apps resources have migrated from the `Microsoft.Web` namespace to the `Microsoft.App` namespace. Refer to [Namespace migration from Microsoft.Web to Microsoft.App in March 2022](#) for more details.

Bash

Azure CLI

```
az provider register --namespace Microsoft.App
```

Azure CLI

```
az provider register --namespace Microsoft.OperationalInsights
```

## Set environment variables

Set the following environment variables. Replace <PLACEHOLDERS> with your values:

Bash

Azure CLI

```
RESOURCE_GROUP="<RESOURCE_GROUP>"
LOCATION="<LOCATION>"
CONTAINERAPPS_ENVIRONMENT="<CONTAINERAPPS_ENVIRONMENT>"
```

## Create an Azure resource group

Create a resource group to organize the services related to your container app deployment.

Bash

Azure CLI

```
az group create \
--name $RESOURCE_GROUP \
--location "$LOCATION"
```

## Create an environment

An environment in Azure Container Apps creates a secure boundary around a group of container apps. Container Apps deployed to the same environment are deployed in the same virtual network and write logs to the same Log Analytics workspace.

To create the environment, run the following command:

Bash

#### Azure CLI

```
az containerapp env create \
--name $CONTAINERAPPS_ENVIRONMENT \
--resource-group $RESOURCE_GROUP \
--location "$LOCATION"
```

## Set up a state store

### Create an Azure Blob Storage account

With the environment deployed, the next step is to deploy an Azure Blob Storage account that is used by one of the microservices to store data. Before deploying the service, you need to choose a name for the storage account. Storage account names must be *unique within Azure*, from 3 to 24 characters in length and must contain numbers and lowercase letters only.

#### Bash

#### Azure CLI

```
STORAGE_ACCOUNT_NAME="<storage account name>"
```

Use the following command to create the Azure Storage account.

#### Bash

#### Azure CLI

```
az storage account create \
--name $STORAGE_ACCOUNT_NAME \
--resource-group $RESOURCE_GROUP \
--location "$LOCATION" \
--sku Standard_RAGRS \
--kind StorageV2
```

## Configure a user-assigned identity for the node app

While Container Apps supports both user-assigned and system-assigned managed identity, a user-assigned identity provides the Dapr-enabled node app with permissions to access the blob storage account.

1. Create a user-assigned identity.

Bash

Azure CLI

```
az identity create --resource-group $RESOURCE_GROUP --name
"nodeAppIdentity" --output json
```

Retrieve the `principalId` and `id` properties and store in variables.

Bash

Azure CLI

```
PRINCIPAL_ID=$(az identity show -n "nodeAppIdentity" --resource-group
$RESOURCE_GROUP --query principalId | tr -d \")
IDENTITY_ID=$(az identity show -n "nodeAppIdentity" --resource-group
$RESOURCE_GROUP --query id | tr -d \")
CLIENT_ID=$(az identity show -n "nodeAppIdentity" --resource-group
$RESOURCE_GROUP --query clientId | tr -d \")
```

2. Assign the `Storage Blob Data Contributor` role to the user-assigned identity

Retrieve the subscription ID for your current subscription.

Bash

Azure CLI

```
SUBSCRIPTION_ID=$(az account show --query id --output tsv)
```

Bash

Azure CLI

```
az role assignment create --assignee $PRINCIPAL_ID \
--role "Storage Blob Data Contributor" \

```

```
--scope
"/subscriptions/$SUBSCRIPTION_ID/resourceGroups/$RESOURCE_GROUP/providers/Microsoft.Storage/storageAccounts/$STORAGE_ACCOUNT_NAME"
```

## Configure the state store component

There are multiple ways to authenticate to external resources via Dapr. This example doesn't use the Dapr Secrets API at runtime, but uses an Azure-based state store.

Therefore, you can forgo creating a secret store component and instead provide direct access from the node app to the blob store using Managed Identity. If you want to use a non-Azure state store or the Dapr Secrets API at runtime, you could create a secret store component. This component would load runtime secrets so you can reference them at runtime.

Open a text editor and create a config file named *statestore.yaml* with the properties that you sourced from the previous steps. This file helps enable your Dapr app to access your state store. The following example shows how your *statestore.yaml* file should look when configured for your Azure Blob Storage account:

YAML

```
statestore.yaml for Azure Blob storage component
componentType: state.azure.blobstorage
version: v1
metadata:
 - name: accountName
 value: "<STORAGE_ACCOUNT_NAME>"
 - name: containerName
 value: mycontainer
 - name: azureClientId
 value: "<MANAGED_IDENTITY_CLIENT_ID>"
scopes:
 - nodeapp
```

To use this file, update the placeholders:

- Replace `<STORAGE_ACCOUNT_NAME>` with the value of the `STORAGE_ACCOUNT_NAME` variable you defined. To obtain its value, run the following command:

Bash

Azure CLI

```
echo $STORAGE_ACCOUNT_NAME
```

- Replace <MANAGED\_IDENTITY\_CLIENT\_ID> with the value of the `CLIENT_ID` variable you defined. To obtain its value, run the following command:

```
Azure CLI
```

```
echo $CLIENT_ID
```

Navigate to the directory in which you stored the component yaml file and run the following command to configure the Dapr component in the Container Apps environment. For more information about configuring Dapr components, see [Configure Dapr components](#).

```
Azure CLI
```

```
az containerapp env dapr-component set \
--name $CONTAINERAPPS_ENVIRONMENT --resource-group $RESOURCE_GROUP \
--dapr-component-name statestore \
--yaml statestore.yaml
```

## Deploy the service application (HTTP web server)

Bash

```
Azure CLI
```

```
az containerapp create \
--name nodeapp \
--resource-group $RESOURCE_GROUP \
--user-assigned $IDENTITY_ID \
--environment $CONTAINERAPPS_ENVIRONMENT \
--image dapriosamples/hello-k8s-node:latest \
--min-replicas 1 \
--max-replicas 1 \
--enable-dapr \
--dapr-app-id nodeapp \
--dapr-app-port 3000 \
--env-vars 'APP_PORT=3000'
```

If you're using an Azure Container Registry, include the `--registry-server <REGISTRY_NAME>.azurecr.io` flag in the command.

By default, the image is pulled from Docker Hub [↗](#).

## Deploy the client application (headless client)

Run the following command to deploy the client container app.

Bash

```
Azure CLI

az containerapp create \
 --name pythonapp \
 --resource-group $RESOURCE_GROUP \
 --environment $CONTAINERAPPS_ENVIRONMENT \
 --image dapriosamples/hello-k8s-python:latest \
 --min-replicas 1 \
 --max-replicas 1 \
 --enable-dapr \
 --dapr-app-id pythonapp
```

If you're using an Azure Container Registry, include the `--registry-server <REGISTRY_NAME>.azurecr.io` flag in the command.

## Verify the results

### Confirm successful state persistence

You can confirm that the services are working correctly by viewing data in your Azure Storage account.

1. Open the [Azure portal](#) [↗](#) in your browser and navigate to your storage account.
2. Select **Containers** left side menu.
3. Select **mycontainer**.
4. Verify that you can see the file named `order` in the container.
5. Select the file.
6. Select the **Edit** tab.
7. Select the **Refresh** button to observe how the data automatically updates.

## View Logs

Logs from container apps are stored in the `ContainerAppConsoleLogs_CL` custom table in the Log Analytics workspace. You can view logs through the Azure portal or via the CLI. There may be a small delay initially for the table to appear in the workspace.

Use the following CLI command to view logs using the command line.

Bash

Azure CLI

```
LOG_ANALYTICS_WORKSPACE_CLIENT_ID=`az containerapp env show --name $CONTAINERAPPS_ENVIRONMENT --resource-group $RESOURCE_GROUP --query properties.appLogsConfiguration.logAnalyticsConfiguration.customerId --out tsv`

az monitor log-analytics query \
--workspace $LOG_ANALYTICS_WORKSPACE_CLIENT_ID \
--analytics-query "ContainerAppConsoleLogs_CL | where ContainerAppName_s == 'nodeapp' and (Log_s contains 'persisted' or Log_s contains 'order') | project ContainerAppName_s, Log_s, TimeGenerated | sort by TimeGenerated | take 5" \
--out table
```

The following output demonstrates the type of response to expect from the CLI command.

Bash

ContainerAppName_s	Log_s	TableName
TimeGenerated		
nodeapp 10-22T21:31:46.184Z	Got a new order! Order ID: 61	PrimaryResult 2021-
nodeapp 10-22T21:31:46.184Z	Successfully persisted state.	PrimaryResult 2021-
nodeapp 10-22T22:01:57.174Z	Got a new order! Order ID: 62	PrimaryResult 2021-
nodeapp 10-22T22:01:57.174Z	Successfully persisted state.	PrimaryResult 2021-
nodeapp 10-22T22:45:44.618Z	Got a new order! Order ID: 63	PrimaryResult 2021-

## Clean up resources

Congratulations! You've completed this tutorial. If you'd like to delete the resources created as a part of this walkthrough, run the following command.

 **Caution**

This command deletes the specified resource group and all resources contained within it. If resources outside the scope of this tutorial exist in the specified resource group, they will also be deleted.

Bash

Azure CLI

```
az group delete --resource-group $RESOURCE_GROUP
```

 **Note**

Since `pythonapp` continuously makes calls to `nodeapp` with messages that get persisted into your configured state store, it is important to complete these cleanup steps to avoid ongoing billable operations.

 **Tip**

Having issues? Let us know on GitHub by opening an issue in the [Azure Container Apps repo](#).

## Next steps

[Application lifecycle management](#)

## Feedback

Was this page helpful?

 Yes

 No

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

# Tutorial: Deploy a Dapr application to Azure Container Apps with an Azure Resource Manager or Bicep template

Article • 08/21/2024

[Dapr](#) (Distributed Application Runtime) is a runtime that helps you build resilient stateless and stateful microservices. In this tutorial, a sample Dapr solution is deployed to Azure Container Apps via an Azure Resource Manager (ARM) or Bicep template.

You learn how to:

- ✓ Create an Azure Blob Storage for use as a Dapr state store
- ✓ Deploy a Container Apps environment to host container apps
- ✓ Deploy two dapr-enabled container apps: one that produces orders and one that consumes orders and stores them
- ✓ Assign a user-assigned identity to a container app and supply it with the appropriate role assignment to authenticate to the Dapr state store
- ✓ Verify the interaction between the two microservices.

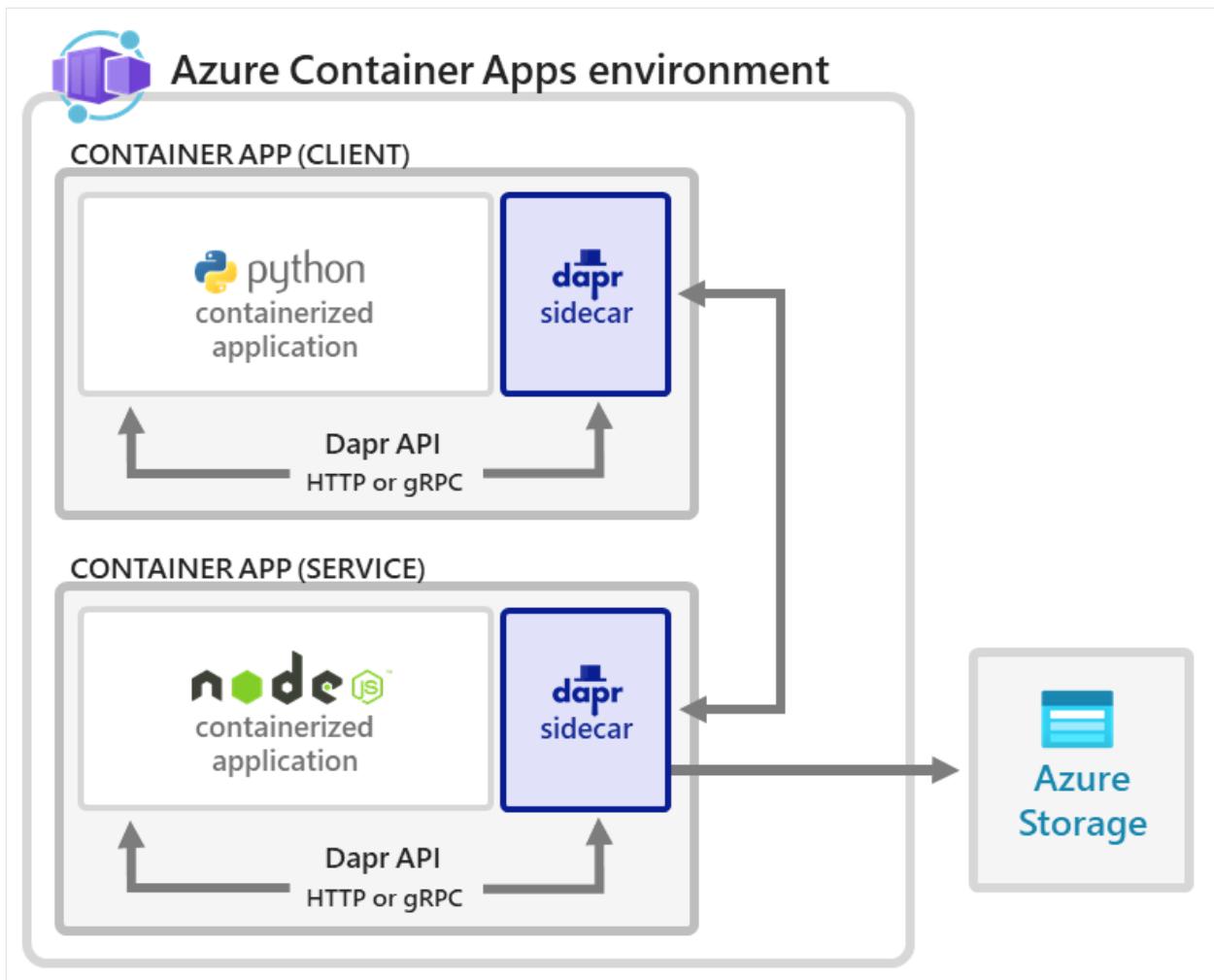
With Azure Container Apps, you get a [fully managed version of the Dapr APIs](#) when building microservices. When you use Dapr in Azure Container Apps, you can enable sidecars to run next to your microservices that provide a rich set of capabilities.

In this tutorial, you deploy the solution from the Dapr [Hello World](#) quickstart.

The application consists of:

- A client (Python) container app to generate messages.
- A service (Node) container app to consume and persist those messages in a state store

The following architecture diagram illustrates the components that make up this tutorial:



## Prerequisites

- Install [Azure CLI](#)
- Install [Git ↗](#)
- An Azure account with an active subscription is required. If you don't already have one, you can [create an account for free ↗](#).
- A GitHub Account. If you don't already have one, sign up for [free ↗](#).

## Setup

To sign in to Azure from the CLI, run the following command and follow the prompts to complete the authentication process.

```
Bash
Azure CLI
```

```
az login
```

To ensure you're running the latest version of the CLI, run the upgrade command.

Bash

Azure CLI

```
az upgrade
```

Next, install or update the Azure Container Apps extension for the CLI.

If you receive errors about missing parameters when you run `az containerapp` commands in Azure CLI or cmdlets from the `Az.App` module in Azure PowerShell, be sure you have the latest version of the Azure Container Apps extension installed.

Bash

Azure CLI

```
az extension add --name containerapp --upgrade
```

#### ⓘ Note

Starting in May 2024, Azure CLI extensions no longer enable preview features by default. To access Container Apps [preview features](#), install the Container Apps extension with `--allow-preview true`.

Azure CLI

```
az extension add --name containerapp --upgrade --allow-preview true
```

Now that the current extension or module is installed, register the `Microsoft.App` and `Microsoft.OperationalInsights` namespaces.

Bash

Azure CLI

```
az provider register --namespace Microsoft.App
```

Azure CLI

```
az provider register --namespace Microsoft.OperationalInsights
```

## Set environment variables

Set the following environment variables. Replace <PLACEHOLDERS> with your values:

Bash

Azure CLI

```
RESOURCE_GROUP="<>RESOURCE_GROUP</>"
LOCATION="<>LOCATION</>"
CONTAINERAPPS_ENVIRONMENT="<>CONTAINERAPPS_ENVIRONMENT</>"
```

## Create an Azure resource group

Create a resource group to organize the services related to your container app deployment.

Bash

Azure CLI

```
az group create \
--name $RESOURCE_GROUP \
--location "$LOCATION"
```

## Prepare the GitHub repository

Go to the repository holding the ARM and Bicep templates that's used to deploy the solution.

Select the **Fork** button at the top of the [repository](#) to fork the repo to your account.

Now you can clone your fork to work with it locally.

Use the following git command to clone your forked repo into the *acadapr-templates* directory.

```
git
```

```
git clone https://github.com/$GITHUB_USERNAME/Tutorial-Deploy-Dapr-Microservices-ACA.git acadapr-templates
```

## Deploy

The template deploys:

- a Container Apps environment
- a Log Analytics workspace associated with the Container Apps environment
- an Application Insights resource for distributed tracing
- a blob storage account and a default storage container
- a Dapr component for the blob storage account
- the node, Dapr-enabled container app with a user-assigned managed identity: [hello-k8s-node](#)
- the python, Dapr-enabled container app: [hello-k8s-python](#)
- a Microsoft Entra ID role assignment for the node app used by the Dapr component to establish a connection to blob storage

Navigate to the *acadapr-templates* directory and run the following command:

Bash

Azure CLI

```
az deployment group create \
--resource-group "$RESOURCE_GROUP" \
--template-file ./azuredeploy.json \
--parameters environment_name="$CONTAINERAPPS_ENVIRONMENT"
```

This command deploys:

- the Container Apps environment and associated Log Analytics workspace for hosting the hello world Dapr solution
- an Application Insights instance for Dapr distributed tracing

- the `nodeapp` app server running on `targetPort: 3000` with Dapr enabled and configured using: `"appId": "nodeapp"` and `"appPort": 3000`, and a user-assigned identity with access to the Azure Blob storage via a Storage Data Contributor role assignment
- A Dapr component of `"type": "state.azure.blobstorage"` scoped for use by the `nodeapp` for storing state
- the Dapr-enabled, headless `pythonapp` that invokes the `nodeapp` service using Dapr service invocation

## Verify the result

### Confirm successful state persistence

You can confirm that the services are working correctly by viewing data in your Azure Storage account.

1. Open the [Azure portal](#) in your browser.
2. Go to the newly created storage account in your resource group.
3. Select **Containers** from the menu on the left side.
4. Select the created container.
5. Verify that you can see the file named `order` in the container.
6. Select the file.
7. Select the **Edit** tab.
8. Select the **Refresh** button to observe updates.

## View Logs

Data logged via a container app are stored in the `ContainerAppConsoleLogs_CL` custom table in the Log Analytics workspace. You can view logs through the Azure portal or from the command line. Wait a few minutes for the analytics to arrive for the first time before you query the logged data.

Use the following command to view logs in bash or PowerShell.

Bash

#### Azure CLI

```
LOG_ANALYTICS_WORKSPACE_CLIENT_ID=`az containerapp env show --name $CONTAINERAPPS_ENVIRONMENT --resource-group $RESOURCE_GROUP --query properties.appLogsConfiguration.logAnalyticsConfiguration.customerId --out tsv`
```

#### Azure CLI

```
az monitor log-analytics query \
--workspace "$LOG_ANALYTICS_WORKSPACE_CLIENT_ID" \
--analytics-query "ContainerAppConsoleLogs_CL | where ContainerAppName_s == 'nodeapp' and (Log_s contains 'persisted' or Log_s contains 'order') | project ContainerAppName_s, Log_s, TimeGenerated | take 5" \
--out table
```

The following output demonstrates the type of response to expect from the command.

#### Console

ContainerAppName_s	Log_s	TableName
TimeGenerated		
nodeapp 10-22T21:31:46.184Z	Got a new order! Order ID: 61	PrimaryResult 2021-
nodeapp 10-22T21:31:46.184Z	Successfully persisted state.	PrimaryResult 2021-
nodeapp 10-22T22:01:57.174Z	Got a new order! Order ID: 62	PrimaryResult 2021-
nodeapp 10-22T22:01:57.174Z	Successfully persisted state.	PrimaryResult 2021-
nodeapp 10-22T22:45:44.618Z	Got a new order! Order ID: 63	PrimaryResult 2021-

## Clean up resources

Once you're done, run the following command to delete your resource group along with all the resources you created in this tutorial.

#### Bash

#### Azure CLI

```
az group delete \
--resource-group $RESOURCE_GROUP
```

### Note

Since `pythonapp` continuously makes calls to `nodeapp` with messages that get persisted into your configured state store, it is important to complete these cleanup steps to avoid ongoing billable operations.

### Tip

Having issues? Let us know on GitHub by opening an issue in the [Azure Container Apps repo](#).

## Next steps

[Application lifecycle management](#)

## Feedback

Was this page helpful?

 Yes

 No

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

# Networking in Azure Container Apps environment

Article • 07/03/2024

Azure Container Apps run in the context of an [environment](#), with its own virtual network (VNet).

By default, your Container App environment is created with a VNet that is automatically generated for you. For fine-grained control over your network, you can provide an existing VNet when you create an environment. Once you create an environment with either a generated or existing VNet, the network type can't be changed.

Generated VNets take on the following characteristics.

They are:

- inaccessible to you as they're created in Microsoft's tenant
- publicly accessible over the internet
- only able to reach internet accessible endpoints

Further, they only support a limited subset of networking capabilities such as ingress IP restrictions and container app level ingress controls.

Use an existing VNet if you need more Azure networking features such as:

- Integration with Application Gateway
- Network Security Groups
- Communication with resources behind private endpoints in your virtual network

The available VNet features depend on your environment selection.

## Environment selection

Container Apps has two different [environment types](#), which share many of the same networking characteristics with some key differences.

[ ] Expand table

Environment type	Description	Supported plan types
Workload profiles	Supports user defined routes (UDR) and egress through NAT Gateway. The minimum required subnet size is /27.	Consumption, Dedicated
Consumption only	Doesn't support user defined routes (UDR), egress through NAT Gateway, peering through a remote gateway, or other custom egress. The minimum required subnet size is /23.	Consumption

## Accessibility levels

You can configure whether your container app allows public ingress or ingress only from within your VNet at the environment level.

[Expand table](#)

Accessibility level	Description
External	Allows your container app to accept public requests. External environments are deployed with a virtual IP on an external, public facing IP address.
Internal	Internal environments have no public endpoints and are deployed with a virtual IP (VIP) mapped to an internal IP address. The internal endpoint is an Azure internal load balancer (ILB) and IP addresses are issued from the custom VNet's list of private IP addresses.

## Custom VNet configuration

As you create a custom VNet, keep in mind the following situations:

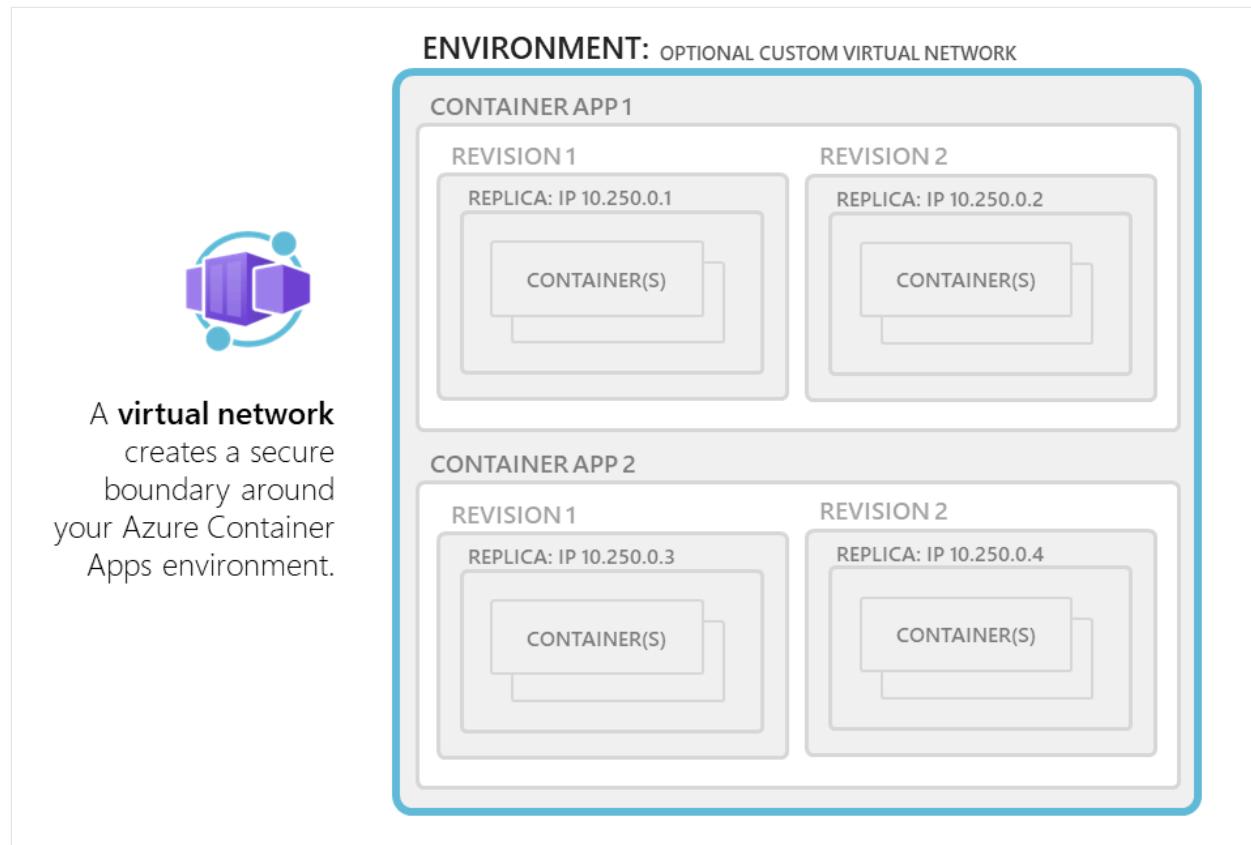
- If you want your container app to restrict all outside access, create an [internal Container Apps environment](#).
- If you use your own VNet, you need to provide a subnet that is dedicated exclusively to the Container App environment you deploy. This subnet isn't available to other services.
- Network addresses are assigned from a subnet range you define as the environment is created.
  - You can define the subnet range used by the Container Apps environment.

- You can restrict inbound requests to the environment exclusively to the VNet by deploying the environment as [internal](#).

#### ⓘ Note

When you provide your own virtual network, additional [managed resources](#) are created. These resources incur costs at their associated rates.

As you begin to design the network around your container app, refer to [Plan virtual networks](#).



#### ⓘ Note

Moving VNets among different resource groups or subscriptions is not allowed if the VNet is in use by a Container Apps environment.

## HTTP edge proxy behavior

Azure Container Apps uses the [Envoy proxy](#) as an edge HTTP proxy. Transport Layer Security (TLS) is terminated on the edge and requests are routed based on their traffic splitting rules and routes traffic to the correct application.

HTTP applications scale based on the number of HTTP requests and connections. Envoy routes internal traffic inside clusters.

Downstream connections support HTTP1.1 and HTTP2 and Envoy automatically detects and upgrades connections if the client connection requires an upgrade.

Upstream connections are defined by setting the `transport` property on the [ingress](#) object.

## Ingress configuration

Under the [ingress](#) section, you can configure the following settings:

- **Accessibility level:** You can set your container app as externally or internally accessible in the environment. An environment variable `CONTAINER_APP_ENV_DNS_SUFFIX` is used to automatically resolve the fully qualified domain name (FQDN) suffix for your environment. When communicating between container apps within the same environment, you may also use the app name. For more information on how to access your apps, see [Ingress in Azure Container Apps](#).
- **Traffic split rules:** You can define traffic splitting rules between different revisions of your application. For more information, see [Traffic splitting](#).

For more information about different networking scenarios, see [Ingress in Azure Container Apps](#).

## Portal dependencies

For every app in Azure Container Apps, there are two URLs.

The Container Apps runtime initially generates a fully qualified domain name (FQDN) used to access your app. See the *Application Url* in the *Overview* window of your container app in the Azure portal for the FQDN of your container app.

A second URL is also generated for you. This location grants access to the log streaming service and the console. If necessary, you may need to add `https://azurecontainerapps.dev/` to the allowlist of your firewall or proxy.

## Ports and IP addresses

The following ports are exposed for inbound connections.

[+] Expand table

Protocol	Port(s)
HTTP/HTTPS	80, 443

IP addresses are broken down into the following types:

[+] Expand table

Type	Description
Public inbound IP address	Used for application traffic in an external deployment, and management traffic in both internal and external deployments.
Outbound public IP	Used as the "from" IP for outbound connections that leave the virtual network. These connections aren't routed down a VPN. Outbound IPs may change over time. Using a NAT gateway or other proxy for outbound traffic from a Container Apps environment is only supported in a <a href="#">workload profiles environment</a> .
Internal load balancer IP address	This address only exists in an <a href="#">internal environment</a> .

## Subnet

Virtual network integration depends on a dedicated subnet. How IP addresses are allocated in a subnet and what subnet sizes are supported depends on which [plan](#) you're using in Azure Container Apps.

Select your subnet size carefully. Subnet sizes can't be modified after you create a Container Apps environment.

Different environment types have different subnet requirements:

Workload profiles environment

- `/27` is the minimum subnet size required for virtual network integration.
- Your subnet must be delegated to `Microsoft.App/environments`.
- When using an external environment with external ingress, inbound traffic routes through the infrastructure's public IP rather than through your subnet.

- Container Apps automatically reserves 12 IP addresses for integration with the subnet. The number of IP addresses required for infrastructure integration doesn't vary based on the scale demands of the environment. Additional IP addresses are allocated according to the following rules depending on the type of workload profile you are using more IP addresses are allocated depending on your environment's workload profile:
  - Dedicated workload profile:** As your container app scales out, each node has one IP address assigned.
  - Consumption workload profile:** Each IP address may be shared among multiple replicas. When planning for how many IP addresses are required for your app, plan for 1 IP address per 10 replicas.
- When you make a [change to a revision](#) in single revision mode, the required address space is doubled for a short period of time in order to support zero downtime deployments. This affects the real, available supported replicas or nodes for a given subnet size. The following table shows both the maximum available addresses per CIDR block and the effect on horizontal scale.

[Expand table](#)

Subnet Size	Available IP Addresses <sup>1</sup>	Max nodes (Dedicated workload profile) <sup>2</sup>	Max replicas (Consumption workload profile) <sup>2</sup>
/23	500	250	2,500
/24	244	122	1,220
/25	116	58	580
/26	52	26	260
/27	20	10	100

<sup>1</sup> The available IP addresses is the size of the subnet minus the 12 IP addresses required for Azure Container Apps infrastructure.

<sup>2</sup> This is accounting for apps in single revision mode.

## Subnet address range restrictions

Subnet address ranges can't overlap with the following ranges reserved by Azure Kubernetes Services:

- 169.254.0.0/16
- 172.30.0.0/16
- 172.31.0.0/16
- 192.0.2.0/24

In addition, a workload profiles environment reserves the following addresses:

- 100.100.0.0/17
- 100.100.128.0/19
- 100.100.160.0/19
- 100.100.192.0/19

## Subnet configuration with CLI

As a Container Apps environment is created, you provide resource IDs for a single subnet.

If you're using the CLI, the parameter to define the subnet resource ID is `infrastructure-subnet-resource-id`. The subnet hosts infrastructure components and user app containers.

If you're using the Azure CLI with a Consumption only environment and the `platformReservedCidr` range is defined, both subnets must not overlap with the IP range defined in `platformReservedCidr`.

## Routes

### User defined routes (UDR)

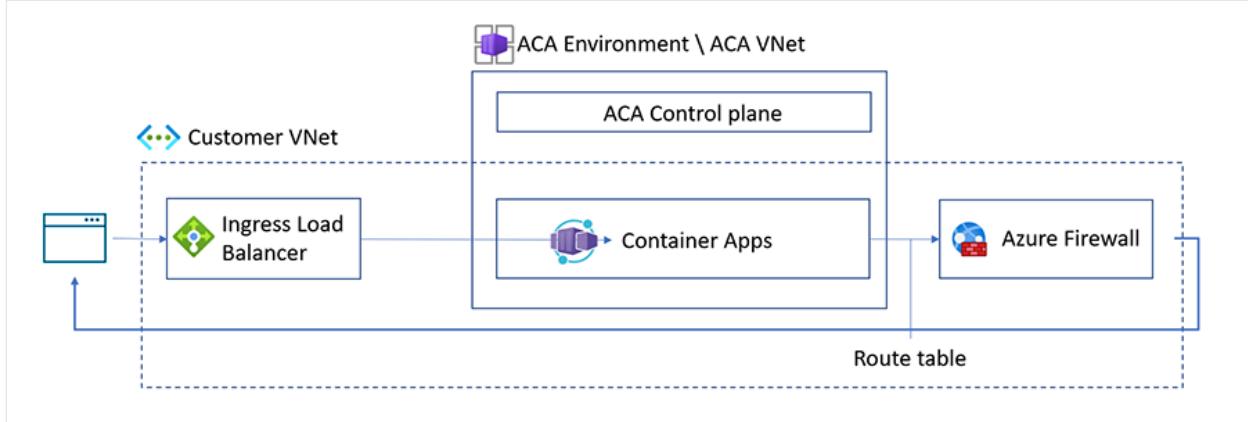
User Defined Routes (UDR) and controlled egress through NAT Gateway are supported in the workload profiles environment. In the Consumption only environment, these features aren't supported.

#### Note

When using UDR with Azure Firewall in Azure Container Apps, you need to add certain FQDN's and service tags to the allowlist for the firewall. To learn more, see

## configuring UDR with Azure Firewall

- You can use UDR with workload profiles environments to restrict outbound traffic from your container app through Azure Firewall or other network appliances.
- Configuring UDR is done outside of the Container Apps environment scope.



Azure creates a default route table for your virtual networks upon create. By implementing a user-defined route table, you can control how traffic is routed within your virtual network. For example, you can create a UDR that routes all traffic to the firewall.

## Configuring UDR with Azure Firewall

User defined routes is only supported in a workload profiles environment. The following application and network rules must be added to the allowlist for your firewall depending on which resources you're using.

### **Note**

For a guide on how to setup UDR with Container Apps to restrict outbound traffic with Azure Firewall, visit the [how to for Container Apps and Azure Firewall](#).

## Application rules

Application rules allow or deny traffic based on the application layer. The following outbound firewall application rules are required based on scenario.

[\[ \] Expand table](#)

Scenarios	FQDNs	Description
All scenarios	<code>mcr.microsoft.com,</code> <code>*.data.mcr.microsoft.com</code>	These FQDNs for Microsoft Container Registry (MCR) are used by Azure Container Apps and either these application rules or the network rules for MCR must be added to the allowlist when using Azure Container Apps with Azure Firewall.
Azure Container Registry (ACR)	<code>Your-ACR-address,</code> <code>*.blob.core.windows.net,</code> <code>login.microsoft.com</code>	These FQDNs are required when using Azure Container Apps with ACR and Azure Firewall.
Azure Key Vault	<code>Your-Azure-Key-Vault-address,</code> <code>login.microsoft.com</code>	These FQDNs are required in addition to the service tag required for the network rule for Azure Key Vault.
Managed Identity	<code>*.identity.azure.net,</code> <code>login.microsoftonline.com,</code> <code>*.login.microsoftonline.com,</code> <code>*.login.microsoft.com</code>	These FQDNs are required when using managed identity with Azure Firewall in Azure Container Apps.
Docker Hub Registry	<code>hub.docker.com</code> , <code>registry-1.docker.io</code> , <code>production.cloudflare.docker.com</code>	If you're using <a href="#">Docker Hub registry</a> and want to access it through the firewall, you need to add these FQDNs to the firewall.

## Network rules

Network rules allow or deny traffic based on the network and transport layer. The following outbound firewall network rules are required based on scenario.

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

Scenarios	Service Tag	Description
All scenarios	<code>MicrosoftContainerRegistry</code> , <code>AzureFrontDoorFirstParty</code>	These Service Tags for Microsoft Container Registry (MCR) are used by Azure Container Apps and either these network rules or the application rules for MCR must be added to the allowlist when using Azure Container Apps with Azure Firewall.
Azure Container	<code>AzureContainerRegistry</code> , <code>AzureActiveDirectory</code>	When using ACR with Azure Container Apps, you need to configure these application rules used by Azure Container Registry.

Scenarios	Service Tag	Description
Registry (ACR)		
Azure Key Vault	AzureKeyVault, AzureActiveDirectory	These service tags are required in addition to the FQDN for the application rule for Azure Key Vault.
Managed Identity	AzureActiveDirectory	When using Managed Identity with Azure Container Apps, you'll need to configure these application rules used by Managed Identity.

### ⓘ Note

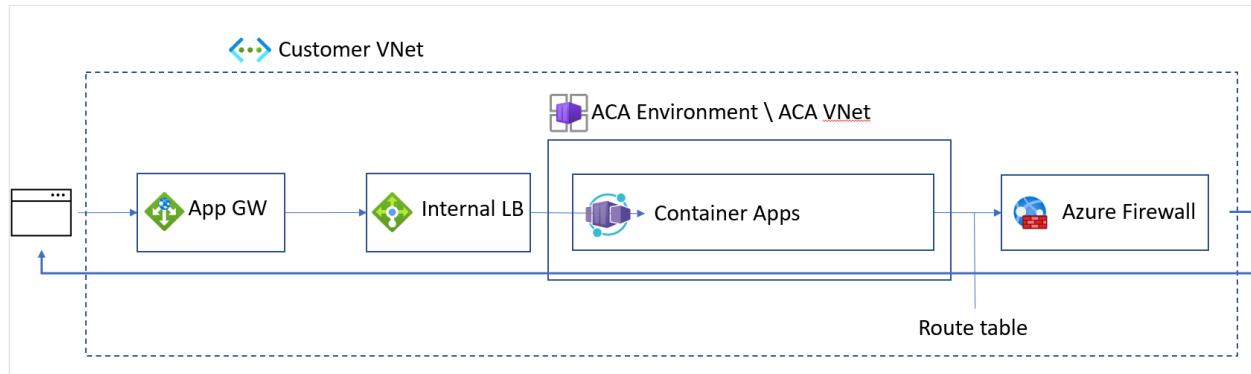
For Azure resources you are using with Azure Firewall not listed in this article, please refer to the [service tags documentation](#).

## NAT gateway integration

You can use NAT Gateway to simplify outbound connectivity for your outbound internet traffic in your virtual network in a workload profiles environment.

When you configure a NAT Gateway on your subnet, the NAT Gateway provides a static public IP address for your environment. All outbound traffic from your container app is routed through the NAT Gateway's static public IP address.

## Environment security



You can fully secure your ingress and egress networking traffic workload profiles environment by taking the following actions:

- Create your internal container app environment in a workload profiles environment. For steps, refer to [Manage workload profiles with the Azure CLI](#).

- Integrate your Container Apps with an [Application Gateway](#).
- Configure UDR to route all traffic through [Azure Firewall](#).

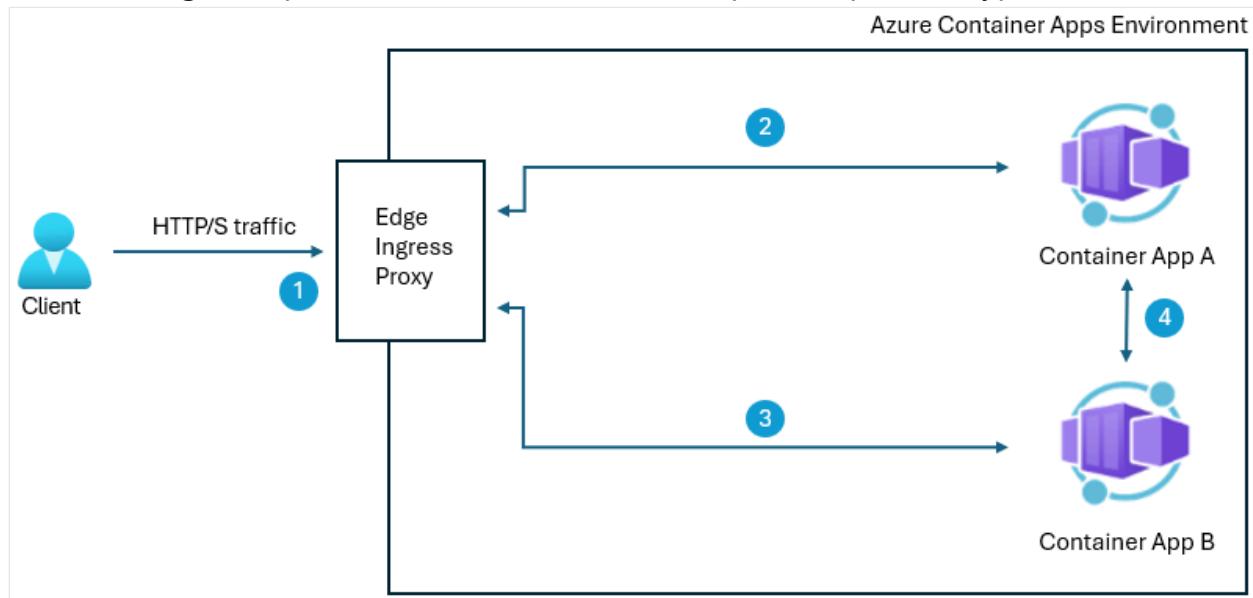
## Peer-to-peer encryption in the Azure Container Apps environment

Azure Container Apps supports peer-to-peer TLS encryption within the environment. Enabling this feature encrypts all network traffic within the environment with a private certificate that is valid within the Azure Container Apps environment scope. These certificates are automatically managed by Azure Container Apps.

 **Note**

By default, peer-to-peer encryption is disabled. Enabling peer-to-peer encryption for your applications may increase response latency and reduce maximum throughput in high-load scenarios.

The following example shows an environment with peer-to-peer encryption enabled.



<sup>1</sup> Inbound TLS traffic is terminated at the ingress proxy on the edge of the environment.

<sup>2</sup> Traffic to and from the ingress proxy within the environment is TLS encrypted with a private certificate and decrypted by the receiver.

<sup>3</sup> Calls made from app A to app B's FQDN are first sent to the edge ingress proxy, and are TLS encrypted.

<sup>4</sup> Calls made from app A to app B using app B's app name are sent directly to app B and are TLS encrypted.

Applications within a Container Apps environment are automatically authenticated. However, the Container Apps runtime doesn't support authorization for access control between applications using the built-in peer-to-peer encryption.

When your apps are communicating with a client outside of the environment, two-way authentication with mTLS is supported. To learn more, see [configure client certificates](#).

#### Azure CLI

You can enable peer-to-peer encryption using the following commands.

On create:

#### Azure CLI

```
az containerapp env create \
 --name <environment-name> \
 --resource-group <resource-group> \
 --location <location> \
 --enable-peer-to-peer-encryption
```

For an existing container app:

#### Azure CLI

```
az containerapp env update \
 --name <environment-name> \
 --resource-group <resource-group> \
 --enable-peer-to-peer-encryption
```

## DNS

- **Custom DNS:** If your VNet uses a custom DNS server instead of the default Azure-provided DNS server, configure your DNS server to forward unresolved DNS queries to 168.63.129.16. [Azure recursive resolvers](#) uses this IP address to resolve requests. When configuring your NSG or firewall, don't block the 168.63.129.16 address, otherwise, your Container Apps environment won't function correctly.
- **VNet-scope ingress:** If you plan to use VNet-scope [ingress](#) in an internal environment, configure your domains in one of the following ways:

- 1. Non-custom domains:** If you don't plan to use a custom domain, create a private DNS zone that resolves the Container Apps environment's default domain to the static IP address of the Container Apps environment. You can use [Azure Private DNS](#) or your own DNS server. If you use Azure Private DNS, create a private DNS Zone named as the Container App environment's default domain (`<UNIQUE_IDENTIFIER>.<REGION_NAME>.azurecontainerapps.io`), with an `A` record. The `A` record contains the name `*<DNS Suffix>` and the static IP address of the Container Apps environment.
- 2. Custom domains:** If you plan to use custom domains and are using an external Container Apps environment, use a publicly resolvable domain to [add a custom domain and certificate](#) to the container app. If you are using an internal Container Apps environment, there is no validation for the DNS binding, as the cluster can only be accessed from within the virtual network. Additionally, create a private DNS zone that resolves the apex domain to the static IP address of the Container Apps environment. You can use [Azure Private DNS](#) or your own DNS server. If you use Azure Private DNS, create a Private DNS Zone named as the apex domain, with an `A` record that points to the static IP address of the Container Apps environment.

The static IP address of the Container Apps environment is available in the Azure portal in **Custom DNS suffix** of the container app page or using the Azure CLI `az containerapp env list` command.

## Managed resources

When you deploy an internal or an external environment into your own network, a new resource group is created in the Azure subscription where your environment is hosted. This resource group contains infrastructure components managed by the Azure Container Apps platform. Don't modify the services in this group or the resource group itself.

## Workload profiles environment

The name of the resource group created in the Azure subscription where your environment is hosted is prefixed with `ME_` by default, and the resource group name *can* be customized as you create your container app environment.

For external environments, the resource group contains a public IP address used specifically for inbound connectivity to your external environment and a load balancer. For internal environments, the resource group only contains a [Load Balancer](#).

In addition to the standard [Azure Container Apps billing](#), you're billed for:

- One standard static [public IP](#) for egress if using an internal or external environment, plus one standard static [public IP](#) for ingress if using an external environment. If you need more public IPs for egress due to SNAT issues, [open a support ticket to request an override](#).
- One standard [load balancer](#).
- The cost of data processed (in GBs) includes both ingress and egress for management operations.

## Consumption only environment

The name of the resource group created in the Azure subscription where your environment is hosted is prefixed with `MC_` by default, and the resource group name *can't* be customized when you create a container app. The resource group contains public IP addresses used specifically for outbound connectivity from your environment and a load balancer.

In addition to the standard [Azure Container Apps billing](#), you're billed for:

- One standard static [public IP](#) for egress. If you need more IPs for egress due to Source Network Address Translation (SNAT) issues, [open a support ticket to request an override](#).
- Two standard [load balancers](#) if using an internal environment, or one standard [load balancer](#) if using an external environment. Each load balancer has fewer than six rules. The cost of data processed (in GBs) includes both ingress and egress for management operations.

## Next steps

- [Deploy with an external environment](#)
- [Deploy with an internal environment](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#)

# Provide a virtual network to an external Azure Container Apps environment

Article • 09/05/2024

The following example shows you how to create a Container Apps environment in an existing virtual network.

Begin by signing in to the [Azure portal](#).

## Create a container app

To create your container app, start at the Azure portal home page.

1. Search for **Container Apps** in the top search bar.
2. Select **Container Apps** in the search results.
3. Select the **Create** button.

### Basics tab

In the *Basics* tab, do the following actions.

1. Enter the following values in the *Project details* section.

  Expand table

Setting	Action
Subscription	Select your Azure subscription.
Resource group	Select <b>Create new</b> and enter <b>my-container-apps</b> .
Container app name	Enter <b>my-container-app</b> .
Deployment source	Select <b>Container image</b> .

### Create an environment

Next, create an environment for your container app.

1. Select the appropriate region.

[ ] Expand table

Setting	Value
Region	Select Central US.

2. In the *Create Container Apps environment* field, select the **Create new** link.

3. In the *Create Container Apps Environment* page on the *Basics* tab, enter the following values:

[ ] Expand table

Setting	Value
Environment name	Enter <b>my-environment</b> .
Environment type	Select <b>Workload profiles</b> .
Zone redundancy	Select <b>Disabled</b>

4. Select the **Monitoring** tab to create a Log Analytics workspace.

5. Select **Azure Log Analytics** as the *Logs Destination*.

6. Select the **Create new** link in the *Log Analytics workspace* field and enter the following values.

[ ] Expand table

Setting	Value
Name	Enter <b>my-container-apps-logs</b> .

The *Location* field is prefilled with *Central US* for you.

7. Select **OK**.

#### (1) Note

You can use an existing virtual network, but a dedicated subnet with a CIDR range of `/23` or larger is required for use with Container Apps when using the Consumption only Architecture. When using a workload profiles environment, a `/27` or larger is required. To learn more about subnet sizing, see the [networking architecture overview](#).

7. Select the **Networking** tab to create a VNET.
8. Select **Yes** next to *Use your own virtual network*.
9. Next to the *Virtual network* box, select the **Create new link** and enter the following value.

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

Setting	Value
Name	Enter <b>my-custom-vnet</b> .

10. Select the **OK** button.
11. Next to the *Infrastructure subnet* box, select the **Create new link** and enter the following values:

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

Setting	Value
Subnet Name	Enter <b>infrastructure-subnet</b> .
Virtual Network Address Block	Keep the default values.
Subnet Address Block	Keep the default values.

12. Select the **OK** button.
13. Under *Virtual IP*, select **External**.
14. Select **Create**.

## Deploy the container app

1. Select **Review and create** at the bottom of the page.

If no errors are found, the *Create* button is enabled.

If there are errors, any tab containing errors is marked with a red dot. Navigate to the appropriate tab. Fields containing an error are highlighted in red. Once all errors are fixed, select **Review and create** again.

2. Select **Create**.

A page with the message *Deployment is in progress* is displayed. Once the deployment is successfully completed, you see the message: *Your deployment is complete.*

## Clean up resources

If you're not going to continue to use this application, you can remove the **my-container-apps** resource group. This deletes the Azure Container Apps instance and all associated services. It also deletes the resource group that the Container Apps service automatically created and which contains the custom network components.

## Next steps

[Managing autoscaling behavior](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# Provide a virtual network to an internal Azure Container Apps environment

Article • 09/05/2024

The following example shows you how to create a Container Apps environment in an existing virtual network.

Begin by signing in to the [Azure portal](#).

## Create a container app

To create your container app, start at the Azure portal home page.

1. Search for **Container Apps** in the top search bar.
2. Select **Container Apps** in the search results.
3. Select the **Create** button.

### Basics tab

In the *Basics* tab, do the following actions.

1. Enter the following values in the *Project details* section.

  Expand table

Setting	Action
Subscription	Select your Azure subscription.
Resource group	Select <b>Create new</b> and enter <b>my-container-apps</b> .
Container app name	Enter <b>my-container-app</b> .
Deployment source	Select <b>Container image</b> .

### Create an environment

Next, create an environment for your container app.

1. Select the appropriate region.

[ ] [Expand table](#)

Setting	Value
Region	Select <b>Central US</b> .

2. In the *Create Container Apps environment* field, select the **Create new** link.

3. In the *Create Container Apps Environment* page on the *Basics* tab, enter the following values:

[ ] [Expand table](#)

Setting	Value
Environment name	Enter <b>my-environment</b> .
Environment type	Select <b>Workload profiles</b> .
Zone redundancy	Select <b>Disabled</b>

4. Select the **Monitoring** tab to create a Log Analytics workspace.

5. Select **Azure Log Analytics** as the *Logs Destination*.

6. Select the **Create new** link in the *Log Analytics workspace* field and enter the following values.

[ ] [Expand table](#)

Setting	Value
Name	Enter <b>my-container-apps-logs</b> .

The *Location* field is prefilled with *Central US* for you.

7. Select **OK**.

#### (1) Note

You can use an existing virtual network, but a dedicated subnet with a CIDR range of `/23` or larger is required for use with Container Apps when using the Consumption only environment. When using the workload profiles environment, a `/27` or larger is required. To learn more about subnet sizing, see the [networking environment overview](#).

7. Select the **Networking** tab to create a VNET.
8. Select **Yes** next to *Use your own virtual network*.
9. Next to the *Virtual network* box, select the **Create new link** and enter the following value.

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

Setting	Value
Name	Enter <b>my-custom-vnet</b> .

10. Select the **OK** button.
11. Next to the *Infrastructure subnet* box, select the **Create new link** and enter the following values:

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

Setting	Value
Subnet Name	Enter <b>infrastructure-subnet</b> .
Virtual Network Address Block	Keep the default values.
Subnet Address Block	Keep the default values.

12. Select the **OK** button.
13. Under *Virtual IP*, select **Internal**.
14. Select **Create**.

## Deploy the container app

1. Select **Review and create** at the bottom of the page.

If no errors are found, the *Create* button is enabled.

If there are errors, any tab containing errors is marked with a red dot. Navigate to the appropriate tab. Fields containing an error are highlighted in red. Once all errors are fixed, select **Review and create** again.

2. Select **Create**.

A page with the message *Deployment is in progress* is displayed. Once the deployment is successfully completed, you see the message: *Your deployment is complete.*

## Clean up resources

If you're not going to continue to use this application, you can delete the Azure Container Apps instance and all the associated services by removing the **my-container-apps** resource group. Deleting this resource group removes the resource group automatically created by the Container Apps service containing the custom network components.

## Additional resources

- To use VNET-scope ingress, you must set up [DNS](#).

## Next steps

[Managing autoscaling behavior](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# Ingress in Azure Container Apps

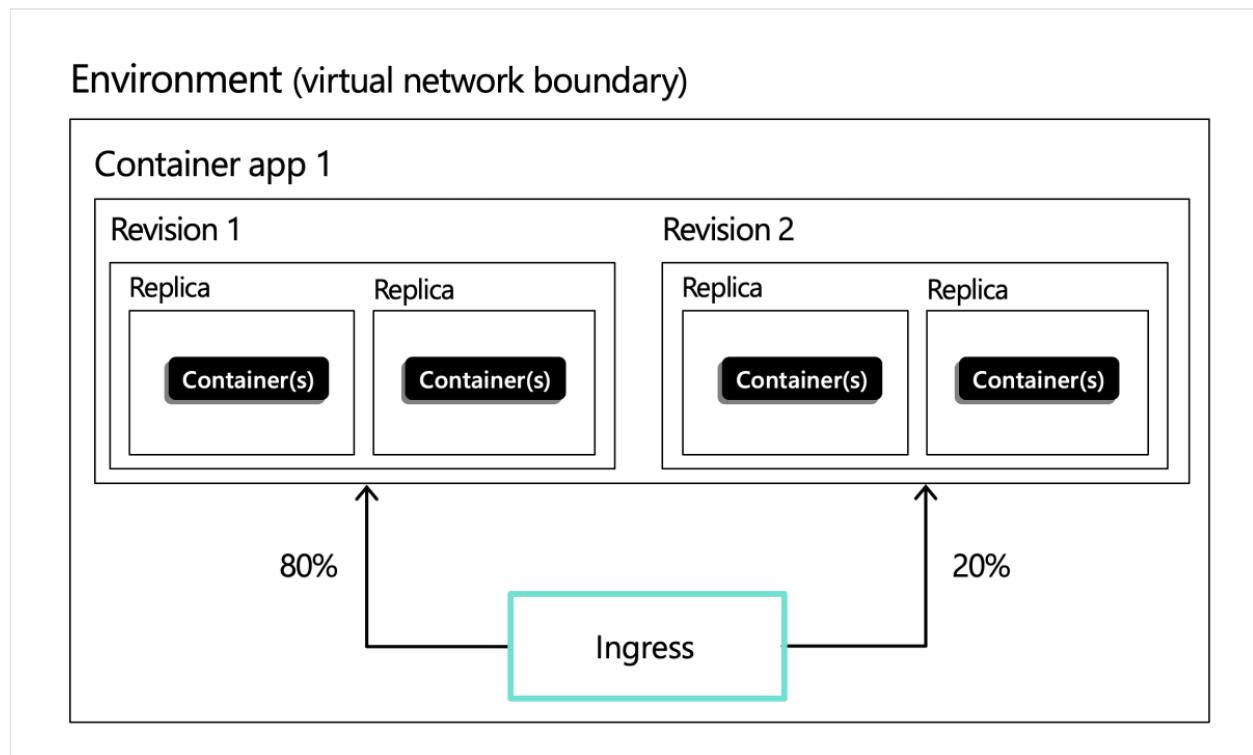
Article • 08/18/2023

Azure Container Apps allows you to expose your container app to the public web, your virtual network (VNET), and other container apps within your environment by enabling ingress. Ingress settings are enforced through a set of rules that control the routing of external and internal traffic to your container app. When you enable ingress, you don't need to create an Azure Load Balancer, public IP address, or any other Azure resources to enable incoming HTTP requests or TCP traffic.

Ingress supports:

- External and internal ingress
- HTTP and TCP ingress types
- Domain names
- IP restrictions
- Authentication
- Traffic splitting between revisions
- Session affinity

Example ingress configuration showing ingress split between two revisions:



For configuration details, see [Configure ingress](#).

## External and internal ingress

When you enable ingress, you can choose between two types of ingress:

- External: Accepts traffic from both the public internet and your container app's internal environment.
- Internal: Allows only internal access from within your container app's environment.

Each container app within an environment can be configured with different ingress settings. For example, in a scenario with multiple microservice apps, to increase security you may have a single container app that receives public requests and passes the requests to a background service. In this scenario, you would configure the public-facing container app with external ingress and the internal-facing container app with internal ingress.

## Protocol types

Container Apps supports two protocols for ingress: HTTP and TCP.

### HTTP

With HTTP ingress enabled, your container app has:

- Support for TLS termination
- Support for HTTP/1.1 and HTTP/2
- Support for WebSocket and gRPC
- HTTPS endpoints that always use TLS 1.2, terminated at the ingress point
- Endpoints that expose ports 80 (for HTTP) and 443 (for HTTPS)
  - By default, HTTP requests to port 80 are automatically redirected to HTTPS on 443
- A fully qualified domain name (FQDN)
- Request timeout is 240 seconds

### HTTP headers

HTTP ingress adds headers to pass metadata about the client request to your container app. For example, the `X-Forwarded-Proto` header is used to identify the protocol that the client used to connect with the Container Apps service. The following table lists the HTTP headers that are relevant to ingress in Container Apps:

Header	Description	Values
<code>X-Forwarded-Proto</code>	Protocol used by the client to connect with the Container	<code>http</code> or <code>https</code>

Header	Description	Values
	Apps service.	
X-Forwarded-For	The IP address of the client that sent the request.	
X-Forwarded-Host	The host name the client used to connect with the Container Apps service.	
X-Forwarded-Client-Cert	The client certificate if <code>clientCertificateMode</code> is set.	Semicolon separated list of Hash, Cert, and Chain. For example: <code>Hash=....;Cert="...";Chain="..."</code>

## TCP

Container Apps supports TCP-based protocols other than HTTP or HTTPS. For example, you can use TCP ingress to expose a container app that uses the [Redis protocol](#).

 **Note**

External TCP ingress is only supported for Container Apps environments that use a **custom VNET**.

With TCP ingress enabled, your container app:

- Is accessible to other container apps in the same environment via its name (defined by the `name` property in the Container Apps resource) and exposed port number.
- Is accessible externally via its fully qualified domain name (FQDN) and exposed port number if the ingress is set to "external".

## Additional TCP ports (preview)

In addition to the main HTTP/TCP port for your container apps, you may expose additional TCP ports to enable applications that accept TCP connections on multiple ports. This feature is in preview.

The following apply to additional TCP ports:

- Additional TCP ports can only be external if the app itself is set as external and the container app is using a custom VNet.

- Any externally exposed additional TCP ports must be unique across the entire Container Apps environment. This includes all external additional TCP ports, external main TCP ports, and 80/443 ports used by built-in HTTP ingress. If the additional ports are internal, the same port can be shared by multiple apps.
- If an exposed port is not provided, the exposed port will default to match the target port.
- Each target port must be unique, and the same target port cannot be exposed on different exposed ports.
- There is a maximum of 5 additional ports per app. If additional ports are required, please open a support request.
- Only the main ingress port supports built-in HTTP features such as CORS and session affinity. When running HTTP on top of the additional TCP ports, these built-in features are not supported.

Visit the [how to article on ingress](#) for more information on how to enable additional ports for your container apps.

## Domain names

You can access your app in the following ways:

- The default fully qualified domain name (FQDN): Each app in a Container Apps environment is automatically assigned an FQDN based on the environment's DNS suffix. To customize an environment's DNS suffix, see [Custom environment DNS Suffix](#).
- A custom domain name: You can configure a custom DNS domain for your Container Apps environment. For more information, see [Custom domain names and certificates](#).
- The app name: You can use the app name for communication between apps in the same environment.

To get the FQDN for your app, see [Location](#).

## IP restrictions

Container Apps supports IP restrictions for ingress. You can create rules to either configure IP addresses that are allowed or denied access to your container app. For more information, see [Configure IP restrictions](#).

## Authentication

Azure Container Apps provides built-in authentication and authorization features to secure your external ingress-enabled container app. For more information, see [Authentication and authorization in Azure Container Apps](#).

You can configure your app to support client certificates (mTLS) for authentication and traffic encryption. For more information, see [Configure client certificates](#).

For details on how to use mTLS for environment level network encryption, see the [networking overview](#).

## Traffic splitting

Containers Apps allows you to split incoming traffic between active revisions. When you define a splitting rule, you assign the percentage of inbound traffic to go to different revisions. For more information, see [Traffic splitting](#).

## Session affinity

Session affinity, also known as sticky sessions, is a feature that allows you to route all HTTP requests from a client to the same container app replica. This feature is useful for stateful applications that require a consistent connection to the same replica. For more information, see [Session affinity](#).

## Cross origin resource sharing (CORS)

By default, any requests made through the browser from a page to a domain that doesn't match the page's origin domain are blocked. To avoid this restriction for services deployed to Container Apps, you can enable cross-origin resource sharing (CORS).

For more information, see [Configure CORS in Azure Container Apps](#).

## Next steps

[Configure ingress](#)

# Configure Ingress for your app in Azure Container Apps

Article • 06/13/2024

This article shows you how to enable [ingress](#) features for your container app. Ingress is an application-wide setting. Changes to ingress settings apply to all revisions simultaneously, and don't generate new revisions.

## Ingress settings

You can set the following ingress template properties:

[+] Expand table

Property	Description	Values	Required
<code>allowInsecure</code>	Allows insecure traffic to your container app. When set to <code>true</code> HTTP requests to port 80 aren't automatically redirected to port 443 using HTTPS, allowing insecure connections.	<code>false</code> (default), <code>true</code> enables insecure connections	No
<code>clientCertificateMode</code>	Client certificate mode for mTLS authentication. Ignore indicates server drops client certificate on forwarding. Accept indicates server forwards client certificate but doesn't require a client certificate. Require indicates server requires a client certificate.	<code>Required</code> , <code>Accept</code> , <code>Ignore</code> (default)	No
<code>customDomains</code>	Custom domain bindings for Container Apps' hostnames. See <a href="#">Custom domains and certificates</a>	An array of bindings	No
<code>exposedPort</code>	(TCP ingress only) The port TCP listens on. If <code>external</code> is <code>true</code> , the value must be unique in the Container Apps environment.	A port number from <code>1</code> to <code>65535</code> . (can't be <code>80</code> or <code>443</code> )	No

Property	Description	Values	Required
<code>external</code>	Allow ingress to your app from outside its Container Apps environment.	<code>true</code> or <code>false</code> (default)	Yes
<code>ipSecurityRestrictions</code>	IP ingress restrictions. See <a href="#">Set up IP ingress restrictions</a>	An array of rules	No
<code>stickySessions.affinity</code>	Enables <a href="#">session affinity</a> .	<code>none</code> (default), <code>sticky</code>	No
<code>targetPort</code>	The port your container listens to for incoming requests.	Set this value to the port number that your container uses. For HTTP ingress, your application ingress endpoint is always exposed on port <code>443</code> .	Yes
<code>traffic</code>	<a href="#">Traffic splitting</a> weights split between revisions.	An array of rules	No
<code>transport</code>	The transport protocol type.	<code>auto</code> (default) detects HTTP/1 or HTTP/2, <code>http</code> for HTTP/1, <code>http2</code> for HTTP/2, <code>tcp</code> for TCP.	No

## Enable ingress

You can configure ingress for your container app using the Azure CLI, an ARM template, or the Azure portal.

This `az containerapp ingress enable` command enables ingress for your container app. You must specify the target port, and you can optionally set the exposed port if your transport type is `tcp`.

### Azure CLI

```
az containerapp ingress enable \
--name <app-name> \
--resource-group <resource-group> \
--target-port <target-port> \
--exposed-port <tcp-exposed-port> \
--transport <transport> \
--type <external>
--allow-insecure
```

```
az containerapp ingress enable
```

ingress arguments:

[Expand table](#)

Option	Property	Description	Values	Required
--type	external	Allow ingress to your app from anywhere, or limit ingress to its internal Container Apps environment.	external or internal	Yes
--allow-insecure	allowInsecure	Allow HTTP connections to your app.		No
--target-port	targetPort	The port your container listens to for incoming requests.	Set this value to the port number that your container uses. Your application ingress endpoint is always exposed on port 443.	Yes
--exposed-port	exposedPort	(TCP ingress only) A port for TCP ingress. If external is true, the value must be unique in the Container Apps environment if ingress is external.	A port number from 1 to 65535. (can't be 80 or 443)	No
--transport	transport	The transport protocol type.	auto (default) detects HTTP/1 or HTTP/2, http for HTTP/1, http2 for HTTP/2, tcp for TCP.	No

## Disable ingress

Disable ingress for your container app by using the `az containerapp ingress disable` command.

Azure CLI

```
az containerapp ingress disable \
--name <app-name> \
--resource-group <resource-group> \
```

## Use other TCP ports

You can expose additional TCP ports from your application. To learn more, see the [ingress concept article](#).

 **Note**

To use this feature, you must have the container apps CLI extension. Run `az extension add -n containerapp` in order to install the latest version of the container apps CLI extension.

Adding other TCP ports can be done through the CLI by referencing a YAML file with your TCP port configurations.

Azure CLI

```
az containerapp create \
 --name <app-name> \
 --resource-group <resource-group> \
 --yaml <your-yaml-file>
```

The following is an example YAML file you can reference in the above CLI command. The configuration for the additional TCP ports is under `additionalPortMappings`.

yml

```
location: northcentralus
name: multiport-example
properties:
 configuration:
 activeRevisionsMode: Single
 ingress:
 additionalPortMappings:
 - exposedPort: 21025
 external: false
 targetPort: 1025
 allowInsecure: false
 external: true
 targetPort: 1080
 traffic:
 - latestRevision: true
 weight: 100
 transport: http
 managedEnvironmentId: <env id>
 template:
 containers:
 - image: maildev/maildev
 name: maildev
 resources:
 cpu: 0.25
```

```
 memory: 0.5Gi
 scale:
 maxReplicas: 1
 minReplicas: 1
 workloadProfileName: Consumption
type: Microsoft.App/containerApps
```

## Next steps

[Ingress in Azure Container Apps](#)

## Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#)

# Set up IP ingress restrictions in Azure Container Apps

Article • 03/30/2023

Azure Container Apps allows you to limit inbound traffic to your container app by configuring IP ingress restrictions via ingress configuration.

There are two types of restrictions:

- *Allow*: Allow inbound traffic only from address ranges you specify in allow rules.
- *Deny*: Deny all inbound traffic only from address ranges you specify in deny rules.

when no IP restriction rules are defined, all inbound traffic is allowed.

IP restrictions rules contain the following properties:

Property	Value	Description
name	string	The name of the rule.
description	string	A description of the rule.
ipAddressRange	IP address range in CIDR format	The IP address range in CIDR notation.
action	Allow or Deny	The action to take for the rule.

The `ipAddressRange` parameter accepts IPv4 addresses. Define each IPv4 address block in [Classless Inter-Domain Routing \(CIDR\)](#) notation.

## ⓘ Note

All rules must be the same type. You cannot combine allow rules and deny rules.

## Manage IP ingress restrictions

You can manage IP access restrictions rules through the Azure portal or Azure CLI.

### Add rules

1. Go to your container app in the Azure portal.
2. Select **Ingress** from the left side menu.

3. Select the IP Security Restrictions Mode toggle to enable IP restrictions. You can choose to allow or deny traffic from the specified IP address ranges.

4. Select Add to create the rule.

The screenshot shows the Azure Container Apps interface for an 'album-app' container app. The 'Ingress' tab is active. On the right, under 'IP Restrictions (Preview)', there's a section for 'IP Security Restrictions Mode' with three options: 'Allow all traffic (default)', 'Allow traffic from IPs configured below, deny all other traffic' (which is selected), and 'Deny traffic from IPs configured below, allow all other traffic'. A red box highlights the 'Add' button at the bottom left of the IP restrictions table. The table below it has columns for 'Source ↑', 'Name ↑', and 'Delete'.

5. Enter values in the following fields:

Field	Description
IPv4 address or range	Enter the IP address or range of IP addresses in CIDR notation. For example, to allow access from a single IP address, use the following format: 10.200.10.2/32.
Name	Enter a name for the rule.
Description	Enter a description for the rule.

6. Select Add.

7. Repeat steps 4-6 to add more rules.

8. When you have finished adding rules, select **Save**.

The screenshot shows the 'IP Restrictions' section of the Azure portal. It includes a search bar, an 'Add' button, and a table with columns for Source and Name. A single rule is listed: '192.168.0.23/32' under Source and 'single-address' under Name. To the right of each entry is a 'Delete' icon. At the bottom are 'Save' and 'Discard' buttons, with 'Save' being highlighted by a red box.

Source ↑	Name ↑	Delete
192.168.0.23/32	single-address	Delete

## Update a rule

1. Go to your container app in the Azure portal.
2. Select **Ingress** from the left side menu.
3. Select the rule you want to update.
4. Change the rule settings.
5. Select **Save** to save the updates.
6. Select **Save** on the Ingress page to save the updated rules.

## Delete a rule

1. Go to your container app in the Azure portal.
2. Select **Ingress** from the left side menu.
3. Select the delete icon next to the rule you want to delete.
4. Select **Save**.

## Next steps

[Configure Ingress](#)

# Configure client certificate authentication in Azure Container Apps

Article • 06/13/2024

Azure Container Apps supports client certificate authentication (also known as mutual TLS or mTLS) that allows access to your container app through two-way authentication. This article shows you how to configure client certificate authorization in Azure Container Apps.

When client certificates are used, the TLS certificates are exchanged between the client and your container app to authenticate identity and encrypt traffic. Client certificates are often used in "zero trust" security models to authorize client access within an organization.

For example, you might want to require a client certificate for a container app that manages sensitive data.

Container Apps accepts client certificates in the PKCS12 format are that issued by a trusted certificate authority (CA), or are self-signed.

## Configure client certificate authorization

To configure support for client certificates, set the `clientCertificateMode` property in your container app template.

The property can be set to one of the following values:

- `require`: The client certificate is required for all requests to the container app.
- `accept`: The client certificate is optional. If the client certificate isn't provided, the request is still accepted.
- `ignore`: The client certificate is ignored.

Ingress passes the client certificate to the container app if `require` or `accept` are set.

The following ARM template example configures ingress to require a client certificate for all requests to the container app.

```
JSON
{
 "properties": {
 "configuration": {
```

```
 "ingress": {
 "clientCertificateMode": "require"
 }
 }
}
```

## Next Steps

[Configure ingress](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#)

# Traffic splitting in Azure Container Apps

Article • 03/30/2023

By default, when ingress is enabled, all traffic is routed to the latest deployed revision. When you enable [multiple revision mode](#) in your container app, you can split incoming traffic between active revisions.

Traffic splitting is useful for testing updates to your container app. You can use traffic splitting to gradually phase in a new revision in [blue-green deployments](#) or in [A/B testing](#).

Traffic splitting is based on the weight (percentage) of traffic that is routed to each revision. The combined weight of all traffic split rules must equal 100%. You can specify revision by revision name or [revision label](#).

This article shows you how to configure traffic splitting rules for your container app. To run the following examples, you need a container app with multiple revisions.

## Configure traffic splitting

Configure traffic splitting between revisions using the [az containerapp ingress traffic set](#) command. You can specify the revisions by name with the `--revision-weight` parameter or by revision label with the `--label-weight` parameter.

The following command sets the traffic weight for each revision to 50%:

```
Azure CLI

az containerapp ingress traffic set \
 --name <APP_NAME> \
 --resource-group <RESOURCE_GROUP> \
 --revision-weight <REVISION_1>=50 <REVISION_2>=50
```

Make sure to replace the placeholder values surrounded by `<>` with your own values.

This command sets the traffic weight for revision `<LABEL_1>` to 80% and revision `<LABEL_2>` to 20%:

```
Azure CLI

az containerapp ingress traffic set \
 --name <APP_NAME> \
 --resource-group <RESOURCE_GROUP> \
 --label-weight <LABEL_1>=80 <LABEL_2>=20
```

```
--label-weight <LABEL_1>=80 <LABEL_2>=20
```

## Use cases

The following scenarios describe configuration settings for common use cases. The examples are shown in JSON format, but you can also use the Azure portal or Azure CLI to configure traffic splitting.

### Rapid iteration

In situations where you're frequently iterating development of your container app, you can set traffic rules to always shift all traffic to the latest deployed revision.

The following example template routes all traffic to the latest deployed revision:

JSON

```
"ingress": {
 "traffic": [
 {
 "latestRevision": true,
 "weight": 100
 }
]
}
```

Once you're satisfied with the latest revision, you can lock traffic to that revision by updating the `ingress` settings to:

JSON

```
"ingress": {
 "traffic": [
 {
 "latestRevision": false, // optional
 "revisionName": "myapp--knowngoodrevision",
 "weight": 100
 }
]
}
```

### Update existing revision

Consider a situation where you have a known good revision that's serving 100% of your traffic, but you want to issue an update to your app. You can deploy and test new revisions using their direct endpoints without affecting the main revision serving the app.

Once you're satisfied with the updated revision, you can shift a portion of traffic to the new revision for testing and verification.

The following template moves 20% of traffic over to the updated revision:

JSON

```
"ingress": {
 "traffic": [
 {
 "revisionName": "myapp--knowngoodrevision",
 "weight": 80
 },
 {
 "revisionName": "myapp--newerrevision",
 "weight": 20
 }
]
}
```

## Staging microservices

When building microservices, you may want to maintain production and staging endpoints for the same app. Use labels to ensure that traffic doesn't switch between different revisions.

The following example template applies labels to different revisions.

JSON

```
"ingress": {
 "traffic": [
 {
 "revisionName": "myapp--knowngoodrevision",
 "weight": 100
 },
 {
 "revisionName": "myapp--98fdgt",
 "weight": 0,
 "label": "staging"
 }
]
}
```

---

## Next steps

[Configure ingress](#)

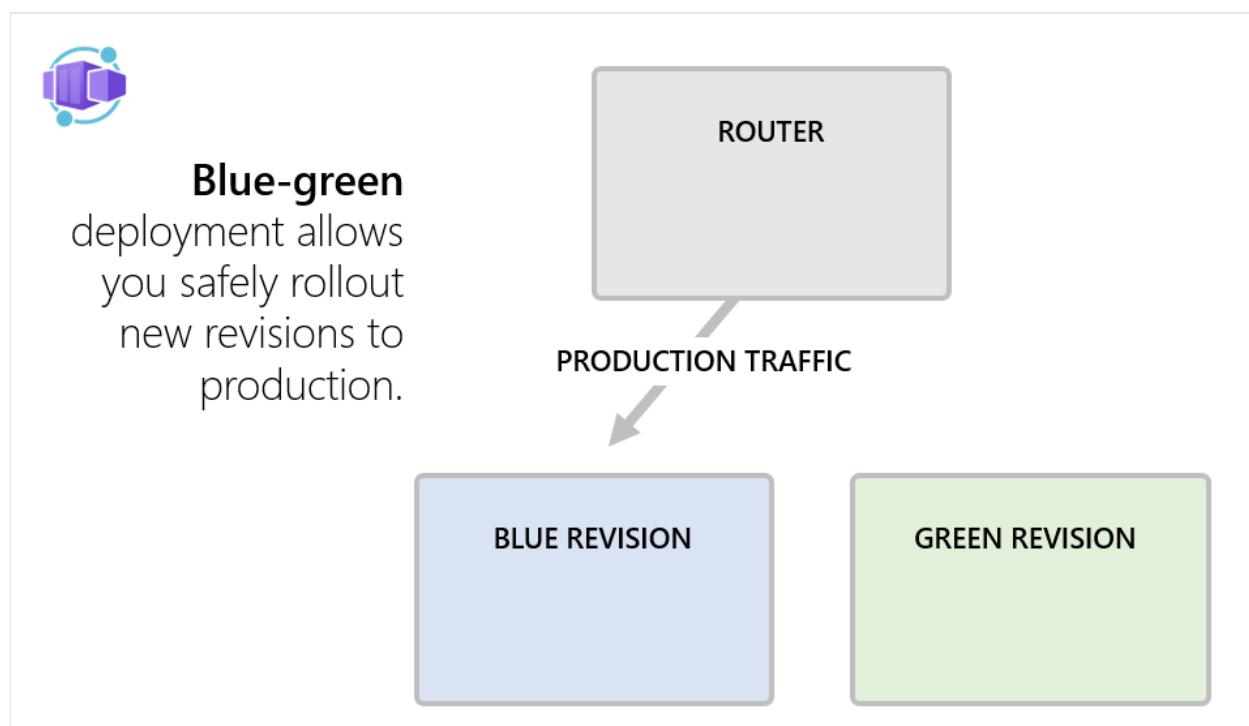
# Blue-Green Deployment in Azure Container Apps

Article • 06/27/2023

[Blue-Green Deployment](#) is a software release strategy that aims to minimize downtime and reduce the risk associated with deploying new versions of an application. In a blue-green deployment, two identical environments, referred to as "blue" and "green," are set up. One environment (blue) is running the current application version and one environment (green) is running the new application version.

Once green environment is tested, the live traffic is directed to it, and the blue environment is used to deploy a new application version during next deployment cycle.

You can enable blue-green deployment in Azure Container Apps by combining [container apps revisions](#), [traffic weights](#), and [revision labels](#).



You use revisions to create instances of the blue and green versions of the application.

Revision	Description
Blue revision	The revision labeled as <i>blue</i> is the currently running and stable version of the application. This revision is the one that users interact with, and it's the target of production traffic.

Revision	Description
Green revision	The revision labeled as <i>green</i> is a copy of the <i>blue</i> revision except it uses a newer version of the app code and possibly new set of environment variables. It doesn't receive any production traffic initially but is accessible via a labeled fully qualified domain name (FQDN).

After you test and verify the new revision, you can then point production traffic to the new revision. If you encounter issues, you can easily roll back to the previous version.

Actions	Description
Testing and verification	The <i>green</i> revision is thoroughly tested and verified to ensure that the new version of the application functions as expected. This testing may involve various tasks, including functional tests, performance tests, and compatibility checks.
Traffic switch	Once the <i>green</i> revision passes all the necessary tests, a traffic switch is performed so that the <i>green</i> revision starts serving production load. This switch is done in a controlled manner, ensuring a smooth transition.
Rollback	If problems occur in the <i>green</i> revision, you can revert the traffic switch, routing traffic back to the stable <i>blue</i> revision. This rollback ensures minimal impact on users if there are issues in the new version. The <i>green</i> revision is still available for the next deployment.
Role change	The roles of the blue and green revisions change after a successful deployment to the <i>green</i> revision. During the next release cycle, the <i>green</i> revision represents the stable production environment while the new version of the application code is deployed and tested in the <i>blue</i> revision.

This article shows you how to implement blue-green deployment in a container app. To run the following examples, you need a container app environment where you can create a new app.

#### ⓘ Note

Refer to [containerapps-blue-green repository](#) for a complete example of a github workflow that implements blue-green deployment for Container Apps.

## Create a container app with multiple active revisions enabled

The container app must have the `configuration.activeRevisionsMode` property set to `multiple` to enable traffic splitting. To get deterministic revision names, you can set the

`template.revisionSuffix` configuration setting to a string value that uniquely identifies a release. For example you can use build numbers, or git commits short hashes.

For the following commands, a set of commit hashes was used.

#### Azure CLI

```
export APP_NAME=<APP_NAME>
export APP_ENVIRONMENT_NAME=<APP_ENVIRONMENT_NAME>
export RESOURCE_GROUP=<RESOURCE_GROUP>

A commitId that is assumed to correspond to the app code currently in
production
export BLUE_COMMIT_ID=fb699ef
A commitId that is assumed to correspond to the new version of the code to
be deployed
export GREEN_COMMIT_ID=c6f1515

create a new app with a new revision
az containerapp create --name $APP_NAME \
--environment $APP_ENVIRONMENT_NAME \
--resource-group $RESOURCE_GROUP \
--image mcr.microsoft.com/k8se/samples/test-app:$BLUE_COMMIT_ID \
--revision-suffix $BLUE_COMMIT_ID \
--env-vars REVISION_COMMIT_ID=$BLUE_COMMIT_ID \
--ingress external \
--target-port 80 \
--revisions-mode multiple

Fix 100% of traffic to the revision
az containerapp ingress traffic set \
--name $APP_NAME \
--resource-group $RESOURCE_GROUP \
--revision-weight APP_NAME-BLUE_COMMIT_ID=100

give that revision a label 'blue'
az containerapp revision label add \
--name $APP_NAME \
--resource-group $RESOURCE_GROUP \
--label blue \
--revision APP_NAME-BLUE_COMMIT_ID
```

## Deploy a new revision and assign labels

The *blue* label currently refers to a revision that takes the production traffic arriving on the app's FQDN. The *green* label refers to a new version of an app that is about to be rolled out into production. A new commit hash identifies the new version of the app code. The following command deploys a new revision for that commit hash and marks it with *green* label.

## Azure CLI

```
#create a second revision for green commitId
az containerapp update --name $APP_NAME \
--resource-group $RESOURCE_GROUP \
--image mcr.microsoft.com/k8se/samples/test-app:$GREEN_COMMIT_ID \
--revision-suffix $GREEN_COMMIT_ID \
--set-env-vars REVISION_COMMIT_ID=$GREEN_COMMIT_ID

#give that revision a 'green' label
az containerapp revision label add \
--name $APP_NAME \
--resource-group $RESOURCE_GROUP \
--label green \
--revision APP_NAME-GREEN_COMMIT_ID
```

The following example shows how the traffic section is configured. The revision with the *blue* `commitId` is taking 100% of production traffic while the newly deployed revision with *green* `commitId` doesn't take any production traffic.

## JSON

```
{
 "traffic": [
 {
 "revisionName": "<APP_NAME>--fb699ef",
 "weight": 100,
 "label": "blue"
 },
 {
 "revisionName": "<APP_NAME>--c6f1515",
 "weight": 0,
 "label": "green"
 }
]
}
```

The newly deployed revision can be tested by using the label-specific FQDN:

## Azure CLI

```
#get the containerapp environment default domain
export APP_DOMAIN=$(az containerapp env show -g $RESOURCE_GROUP -n
$APP_ENVIRONMENT_NAME --query properties.defaultDomain -o tsv | tr -d
'\r\n')

#Test the production FQDN
curl -s https://$APP_NAME.$APP_DOMAIN/api/env | jq | grep COMMIT

#Test the blue lable FQDN
```

```
curl -s https://$APP_NAME---blue.$APP_DOMAIN/api/env | jq | grep COMMIT

#Test the green label FQDN
curl -s https://$APP_NAME---green.$APP_DOMAIN/api/env | jq | grep COMMIT
```

## Send production traffic to the green revision

After confirming that the app code in the *green* revision works as expected, 100% of production traffic is sent to the revision. The *green* revision now becomes the production revision.

Azure CLI

```
set 100% of traffic to green revision
az containerapp ingress traffic set \
 --name $APP_NAME \
 --resource-group $RESOURCE_GROUP \
 --label-weight blue=0 green=100
```

The following example shows how the `traffic` section is configured after this step. The *green* revision with the new application code takes all the user traffic while *blue* revision with the old application version doesn't accept user requests.

JSON

```
{
 "traffic": [
 {
 "revisionName": "<APP_NAME>--c6f1515",
 "weight": 0,
 "label": "blue"
 },
 {
 "revisionName": "<APP_NAME>--fb699ef",
 "weight": 100,
 "label": "green"
 }
]
}
```

## Roll back the deployment if there were problems

If after running in production, the new revision is found to have bugs, you can roll back to the previous good state. After the rollback, 100% of the traffic is sent to the old version in the *blue* revision and that revision is designated as the production revision again.

#### Azure CLI

```
set 100% of traffic to green revision
az containerapp ingress traffic set \
--name $APP_NAME \
--resource-group $RESOURCE_GROUP \
--label-weight blue=100 green=0
```

After the bugs are fixed, the new version of the application is deployed as a *green* revision again. The *green* version eventually becomes the production revision.

## Next deployment cycle

Now the *green* label marks the revision currently running the stable production code.

During the next deployment cycle, the *blue* identifies the revision with the new application version being rolled out to production.

The following commands demonstrate how to prepare for the next deployment cycle.

#### Azure CLI

```
set the new commitId
export BLUE_COMMIT_ID=ad1436b

create a third revision for blue commitId
az containerapp update --name $APP_NAME \
--resource-group $RESOURCE_GROUP \
--image mcr.microsoft.com/k8se/samples/test-app:$BLUE_COMMIT_ID \
--revision-suffix $BLUE_COMMIT_ID \
--set-env-vars REVISION_COMMIT_ID=$BLUE_COMMIT_ID

give that revision a 'blue' label
az containerapp revision label add \
--name $APP_NAME \
--resource-group $RESOURCE_GROUP \
--label blue \
--revision APP_NAME-BLUE_COMMIT_ID
```

## Next steps

## Traffic Weights

# Session Affinity in Azure Container Apps (preview)

Article • 03/30/2023

Session affinity, also known as sticky sessions, is a feature that allows you to route all requests from a client to the same replica. This feature is useful for stateful applications that require a consistent connection to the same replica.

Session stickiness is enforced using HTTP cookies. This feature is available in single revision mode when HTTP ingress is enabled. A client may be routed to a new replica if the previous replica is no longer available.

If your app doesn't require session affinity, we recommend that you don't enable it. With session affinity disabled, ingress distributes requests more evenly across replicas improving the performance of your app.

## ⓘ Note

Session affinity is only supported when your app is in **single revision mode** and the ingress type is HTTP.

This feature is in public preview.

## Configure session affinity

You can enable session affinity when you create your container app via the Azure portal. To enable session affinity:

1. On the **Create Container App** page, select the **App settings** tab.
2. In the **Application ingress settings** section, select **Enabled** for the **Session affinity** setting.

## Create Container App

Name	Value	Delete
<input type="text" value="Enter name"/>	<input type="text" value="Enter value"/>	

**Application ingress settings**

Enable ingress for applications that need an HTTP or TCP endpoint.

Ingress  ⓘ   Enabled

Ingress traffic  **Limited to Container Apps Environment**  
 Limited to VNet: Applies if 'internalOnly' setting is set to true on the Container Apps environment  
 Accepting traffic from anywhere: Applies if 'internalOnly' setting is set to false on the Container Apps environment

Ingress type  ⓘ   **HTTP**  
 TCP

Transport

Insecure connections  Allowed

Target port \* ⓘ

Session affinity  ⓘ   Enabled

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

You can also enable or disable session affinity after your container app is created. To enable session affinity:

1. Go to your app in the portal.
2. Select **Ingress**.
3. You can enable or disable **Session affinity** by selecting or deselecting **Enabled**.
4. Select **Save**.

my-container-app | Ingress

Container App

Search Refresh Send us your feedback

Overview Access control (IAM) Tags Diagnose and solve problems

Enable ingress for applications that need an HTTP or TCP endpoint.

Ingress  Enabled

Limited to Container Apps Environment  
 Limited to VNet: Applies if 'internalOnly' setting is set to true on the Container Apps environment  
 Accepting traffic from anywhere: Applies if 'internalOnly' setting is set to false on the Container Apps environment

Ingress type  HTTP  TCP

Transport Auto

Insecure connections  Allowed

Target port \* 80

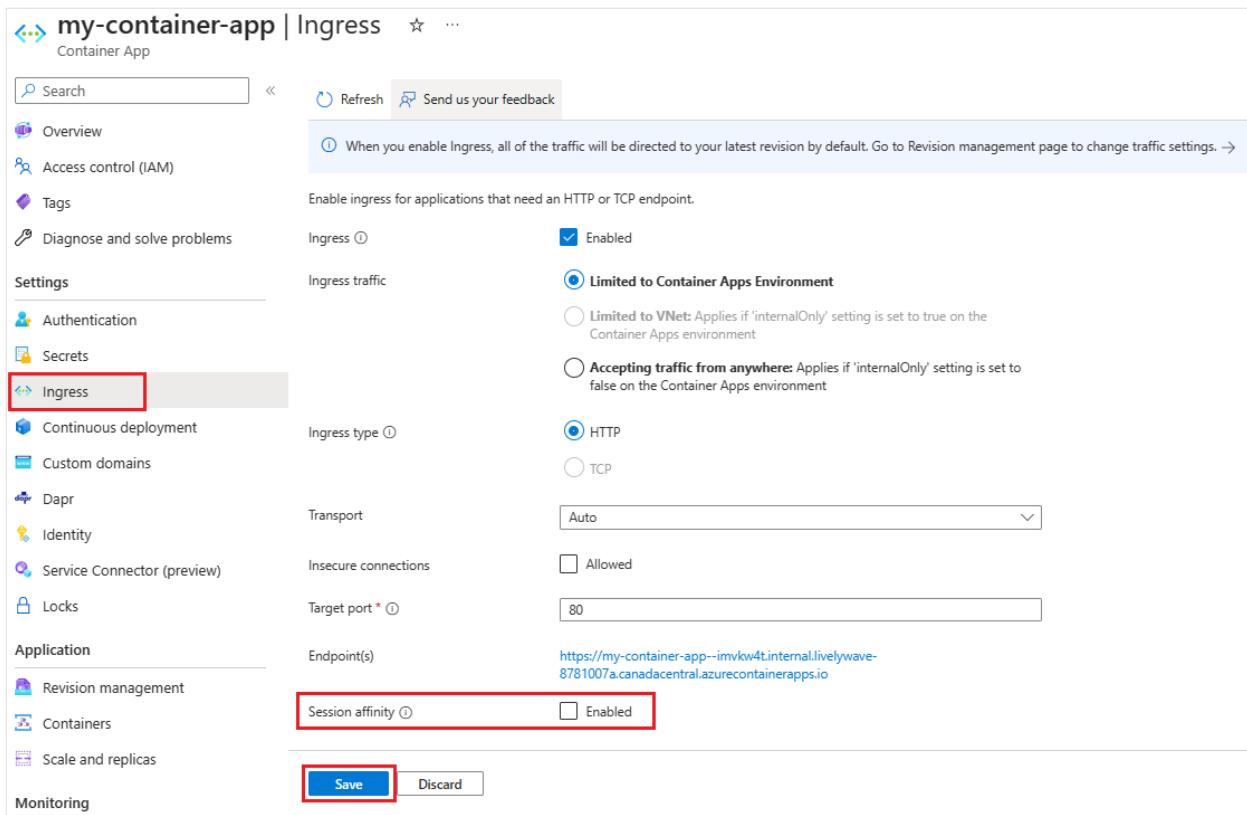
Endpoint(s) https://my-container-app--imvkw4t.internal.livelywave-8781007a.canadacentral.azurecontainerapps.io

Session affinity  Enabled

Save Discard

Continuous deployment Custom domains Dapr Identity Service Connector (preview) Locks Revision management Containers Scale and replicas Monitoring

Ingress



## Next steps

Configure ingress

# Configure cross-origin resource sharing (CORS) for Azure Container Apps

Article • 05/04/2023

By default, requests made through the browser to a domain that doesn't match the page's origin domain are blocked. To avoid this restriction for services deployed to Container Apps, you can enable [CORS](#).

This article shows you how to enable and configure CORS in your container app.

As you enable CORS, you can configure the following settings:

Setting	Explanation
Allow credentials	Indicates whether to return the <a href="#">Access-Control-Allow-Credentials</a> header.
Max age	Configures the <a href="#">Access-Control-Max-Age</a> response header to indicate how long (in seconds) the results of a CORS pre-flight request can be cached.
Allowed origins	List of the origins allowed for cross-origin requests (for example, <code>https://www.contoso.com</code> ). Controls the <a href="#">Access-Control-Allow-Origin</a> response header. Use <code>*</code> to allow all.
Allowed methods	List of HTTP request methods allowed in cross-origin requests. Controls the <a href="#">Access-Control-Allow-Methods</a> response header. Use <code>*</code> to allow all.
Allowed headers	List of the headers allowed in cross-origin requests. Controls the <a href="#">Access-Control-Allow-Headers</a> response header. Use <code>*</code> to allow all.
Expose headers	By default, not all response headers are exposed to client-side JavaScript code in a cross-origin request. Exposed headers are extra headers servers can include in a response. Controls the <a href="#">Access-Control-Expose-Headers</a> response header. Use <code>*</code> to expose all.

For more information, see the Web Hypertext Application Technology Working Group (WHATWG) reference on valid [HTTP responses from a fetch request](#).

## Enable and configure CORS

1. Go to your container app in the Azure portal.
2. Under the settings menu, select *CORS*.

The screenshot shows the CORS configuration for a container app named 'my-container-app'. The left sidebar includes links for Search, Refresh, Locks, Application (Revision management, Containers, Scale and replicas), Monitoring (Alerts, Metrics, Logs, Log stream, Console), and CORS. The main area displays the CORS settings with 'Access-Control-Allow-Credentials' checked and 'Max Age' set to 0. Under 'Allowed Origins', there is one entry: 'https://example.com'. A text input field for adding more origins is shown below. At the bottom are 'Apply' and 'Discard' buttons.

With CORS enabled you can add, edit, and delete values for *Allowed Origins*, *Allowed Methods*, *Allowed Headers*, and *Expose Headers*.

To allow any acceptable values for methods, headers, or origins, enter `*` as the value.

## Next steps

[Configure ingress](#)

# Protect Azure Container Apps with Web Application Firewall on Application Gateway

Article • 04/02/2023

When you host your apps or microservices in Azure Container Apps, you may not always want to publish them directly to the internet. Instead, you may want to expose them through a reverse proxy.

A reverse proxy is a service that sits in front of one or more services, intercepting and directing incoming traffic to the appropriate destination.

Reverse proxies allow you to place services in front of your apps that supports cross-cutting functionality including:

- Routing
- Caching
- Rate limiting
- Load balancing
- Security layers
- Request filtering

This article demonstrates how to protect your container apps using a [Web Application Firewall \(WAF\) on Azure Application Gateway](#) with an internal Container Apps environment.

For more information on networking concepts in Container Apps, see [Networking Environment in Azure Container Apps](#).

## Prerequisites

- **Internal environment with custom VNet:** Have a container app that is on an internal environment and integrated with a custom virtual network. For more information on how to create a custom virtual network integrated app, see [provide a virtual network to an internal Azure Container Apps environment](#).
- **Security certificates:** If you must use TLS/SSL encryption to the application gateway, a valid public certificate that's used to bind to your application gateway is required.

# Retrieve your container app's domain

Use the following steps to retrieve the values of the **default domain** and the **static IP** to set up your Private DNS Zone.

1. From the resource group's *Overview* window in the portal, select your container app.
2. On the *Overview* window for your container app resource, select the link for **Container Apps Environment**
3. On the *Overview* window for your container app environment resource, select **JSON View** in the upper right-hand corner of the page to view the JSON representation of the container apps environment.
4. Copy the values for the **defaultDomain** and **staticIp** properties and paste them into a text editor. You'll create a private DNS zone using these values for the default domain in the next section.

## Create and configure an Azure Private DNS zone

1. On the Azure portal menu or the *Home* page, select **Create a resource**.
2. Search for *Private DNS Zone*, and select **Private DNS Zone** from the search results.
3. Select the **Create** button.
4. Enter the following values:

Setting	Action
Subscription	Select your Azure subscription.
Resource group	Select the resource group of your container app.
Name	Enter the <b>defaultDomain</b> property of the Container Apps Environment from the previous section.
Resource group location	Leave as the default. A value isn't needed as Private DNS Zones are global.

5. Select **Review + create**. After validation finishes, select **Create**.
6. After the private DNS zone is created, select **Go to resource**.

7. In the *Overview* window, select **+Record set**, to add a new record set.

8. In the *Add record set* window, enter the following values:

Setting	Action
Name	Enter *.
Type	Select <b>A-Address Record</b> .
TTL	Keep the default values.
TTL unit	Keep the default values.
IP address	Enter the <b>staticIp</b> property of the Container Apps Environment from the previous section.

9. Select **OK** to create the record set.

10. Select **+Record set** again, to add a second record set.

11. In the *Add record set* window, enter the following values:

Setting	Action
Name	Enter @.
Type	Select <b>A-Address Record</b> .
TTL	Keep the default values.
TTL unit	Keep the default values.
IP address	Enter the <b>staticIp</b> property of the Container Apps Environment from the previous section.

12. Select **OK** to create the record set.

13. Select the **Virtual network links** window from the menu on the left side of the page.

14. Select **+Add** to create a new link with the following values:

Setting	Action
Link name	Enter <b>my-custom-vnet-pdns-link</b> .
I know the resource ID of virtual network	Leave it unchecked.

Setting	Action
Virtual network	Select the virtual network your container app is integrated with.
Enable auto registration	Leave it unchecked.

15. Select **OK** to create the virtual network link.

## Create and configure Azure Application Gateway

### Basics tab

1. Enter the following values in the *Project details* section.

Setting	Action
Subscription	Select your Azure subscription.
Resource group	Select the resource group for your container app.
Application gateway name	Enter <b>my-container-apps-agw</b> .
Region	Select the location where your Container App was provisioned.
Tier	Select <b>WAF V2</b> . You can use <b>Standard V2</b> if you don't need WAF.
Enable autoscaling	Leave as default. For production environments, autoscaling is recommended. See <a href="#">Autoscaling Azure Application Gateway</a> .
Availability zone	Select <b>None</b> . For production environments, <a href="#">Availability Zones</a> are recommended for higher availability.
HTTP2	Keep the default value.
WAF Policy	Select <b>Create new</b> and enter <b>my-waf-policy</b> for the WAF Policy. Select <b>OK</b> . If you chose <b>Standard V2</b> for the tier, skip this step.
Virtual network	Select the virtual network that your container app is integrated with.
Subnet	Select <b>Manage subnet configuration</b> . If you already have a subnet you wish to use, use that instead, and skip to <a href="#">the Frontends section</a> .

- From within the *Subnets* window of *my-custom-vnet*, select **+Subnet** and enter the following values:

Setting	Action
Name	Enter <b>appgateway-subnet</b> .
Subnet address range	Keep the default values.

- For the remainder of the settings, keep the default values.
- Select **Save** to create the new subnet.
- Close the *Subnets* window to return to the *Create application gateway* window.
- Select the following values:

Setting	Action
Subnet	Select the <b>appgateway-subnet</b> you created.

- Select **Next: Frontends**, to proceed.

## Frontends tab

- On the *Frontends* tab, enter the following values:

Setting	Action
Frontend IP address type	Select <b>Public</b> .
Public IP address	Select <b>Add new</b> . Enter <b>my-frontend</b> for the name of your frontend and select <b>OK</b>

### ⓘ Note

For the Application Gateway v2 SKU, there must be a **Public** frontend IP. You can have both a public and a private frontend IP configuration, but a private-only frontend IP configuration with no public IP is currently not supported in the v2 SKU. To learn more, [read here](#).

- Select **Next: Backends**.

## Backends tab

The backend pool is used to route requests to the appropriate backend servers. Backend pools can be composed of any combination of the following resources:

- NICs
- Public IP addresses
- Internal IP addresses
- Virtual Machine Scale Sets
- Fully qualified domain names (FQDN)
- Multi-tenant back-ends like Azure App Service and Container Apps

In this example, you create a backend pool that targets your container app.

1. Select **Add a backend pool**.
2. Open a new tab and navigate to your container app.
3. In the *Overview* window of the Container App, find the **Application Url** and copy it.
4. Return to the *Backends* tab, and enter the following values in the **Add a backend pool** window:

Setting	Action
Name	Enter <b>my-agw-backend-pool</b> .
Add backend pool without targets	Select <b>No</b> .
Target type	Select <b>IP address or FQDN</b> .
Target	Enter the <b>Container App Application Url</b> you copied and remove the <b>https://</b> prefix. This location is the FQDN of your container app.

5. Select **Add**.
6. On the *Backends* tab, select **Next: Configuration**.

## Configuration tab

On the *Configuration* tab, you connect the frontend and backend pool you created using a routing rule.

1. Select **Add a routing rule**. Enter the following values:

<b>Setting</b>	<b>Action</b>
Name	Enter <b>my-agw-routing-rule</b> .
Priority	Enter <b>1</b> .

2. Under Listener tab, enter the following values:

<b>Setting</b>	<b>Action</b>
Listener name	Enter <b>my-agw-listener</b> .
Frontend IP	Select <b>Public</b> .
Protocol	Select <b>HTTPS</b> . If you don't have a certificate you want to use, you can select <b>HTTP</b>
Port	Enter <b>443</b> . If you chose <b>HTTP</b> for your protocol, enter <b>80</b> and skip to the default/custom domain section.
Choose a Certificate	Select <b>Upload a certificate</b> . If your certificate is stored in key vault, you can select <b>Choose a certificate from Key Vault</b> .
Cert name	Enter a name for your certificate.
PFX certificate file	Select your valid public certificate.
Password	Enter your certificate password.

If you want to use the default domain, enter the following values:

<b>Setting</b>	<b>Action</b>
Listener Type	Select <b>Basic</b>
Error page url	Leave as <b>No</b>

Alternatively, if you want to use a custom domain, enter the following values:

<b>Setting</b>	<b>Action</b>
Listener Type	Select <b>Multi site</b>
Host type	Select <b>Single</b>
Host Names	Enter the Custom Domain you wish to use.

Setting	Action
Error page url	Leave as No

3. Select the **Backend targets** tab and enter the following values:

4. Toggle to the *Backend targets* tab and enter the following values:

Setting	Action
Target type	Select <b>my-agw-backend-pool</b> that you created earlier.
Backend settings	Select <b>Add new</b> .

5. In the *Add Backend setting* window, enter the following values:

Setting	Action
Backend settings name	Enter <b>my-agw-backend-setting</b> .
Backend protocol	Select <b>HTTPS</b> .
Backend port	Enter <b>443</b> .
Use well known CA certificate	Select <b>Yes</b> .
Override with new host name	Select <b>Yes</b> .
Host name override	Select <b>Pick host name from backend target</b> .
Create custom probes	Select <b>No</b> .

6. Select **Add**, to add the backend settings.

7. In the *Add a routing rule* window, select **Add** again.

8. Select **Next: Tags**.

9. Select **Next: Review + create**, and then select **Create**.

## Add private link to your Application Gateway

This step is required for internal only container app environments as it allows your Application Gateway to communicate with your Container App on the backend through the virtual network.

1. Once the Application Gateway is created, select **Go to resource**.

2. From the menu on the left, select **Private link**, then select **Add**.

3. Enter the following values:

Setting	Action
Name	Enter <b>my-agw-private-link</b> .
Private link subnet	Select the subnet you wish to create the private link with.
Frontend IP Configuration	Select the frontend IP for your Application Gateway.

4. Under **Private IP address settings** select **Add**.

5. Select **Add** at the bottom of the window.

## Verify the container app

Default domain

1. Find the public IP address for the application gateway on its *Overview* page, or you can search for the address. To search, select **All resources** and enter **my-container-apps-agw-pip** in the search box. Then, select the IP in the search results.
2. Navigate to the public IP address of the application gateway.
3. Your request is automatically routed to the container app, which verifies the application gateway was successfully created.

## Clean up resources

When you no longer need the resources that you created, delete the resource group.

When you delete the resource group, you also remove all the related resources.

To delete the resource group:

1. On the Azure portal menu, select **Resource groups** or search for and select **Resource groups**.
2. On the *Resource groups* page, search for and select **my-container-apps**.

3. On the *Resource group page*, select **Delete resource group**.
4. Enter **my-container-apps** under *TYPE THE RESOURCE GROUP NAME* and then select **Delete**

## Next steps

[Azure Firewall in Azure Container Apps](#)

# Control outbound traffic with user defined routes

Article • 08/29/2023

## ⓘ Note

This feature is only supported for the workload profiles environment type. User defined routes only work with an internal Azure Container Apps environment.

This article shows you how to use user defined routes (UDR) with [Azure Firewall](#) to lock down outbound traffic from your Container Apps to back-end Azure resources or other network resources.

Azure creates a default route table for your virtual networks on create. By implementing a user-defined route table, you can control how traffic is routed within your virtual network. In this guide, your setup UDR on the Container Apps virtual network to restrict outbound traffic with Azure Firewall.

You can also use a NAT gateway or any other third party appliances instead of Azure Firewall.

For more information on networking concepts in Container Apps, see [Networking Environment in Azure Container Apps](#).

## Prerequisites

- **Internal environment:** An internal container app environment on the workload profiles environment that's integrated with a custom virtual network. When you create an internal container app environment, your container app environment has no public IP addresses, and all traffic is routed through the virtual network. For more information, see the [guide for how to create a container app environment on the workload profiles environment](#).
- **curl support:** Your container app must have a container that supports `curl` commands. In this how-to, you use `curl` to verify the container app is deployed correctly. If you don't have a container app with `curl` deployed, you can deploy the following container which supports `curl`,  
`mcr.microsoft.com/k8se/quickstart:latest`.

# Create the firewall subnet

A subnet called **AzureFirewallSubnet** is required in order to deploy a firewall into the integrated virtual network.

1. Open the virtual network that's integrated with your app in the [Azure portal](#).
2. From the menu on the left, select **Subnets**, then select **+ Subnet**.
3. Enter the following values:

Setting	Action
Name	Enter <b>AzureFirewallSubnet</b> .
Subnet address range	Use the default or specify a <a href="#">subnet range /26 or larger</a> .

4. Select **Save**

# Deploy the firewall

1. On the Azure portal menu or the **Home** page, select **Create a resource**.
2. Search for *Firewall*.
3. Select **Firewall**.
4. Select **Create**.
5. On the *Create a Firewall* page, configure the firewall with the following settings.

Setting	Action
Resource group	Enter the same resource group as the integrated virtual network.
Name	Enter a name of your choice
Region	Select the same region as the integrated virtual network.
Firewall policy	Create one by selecting <b>Add new</b> .
Virtual network	Select the integrated virtual network.
Public IP address	Select an existing address or create one by selecting <b>Add new</b> .

6. Select **Review + create**. After validation finishes, select **Create**. The validation step may take a few minutes to complete.

- Once the deployment completes, select **Go to Resource**.
- In the firewall's **Overview** page, copy the **Firewall private IP**. This IP address is used as the next hop address when creating the routing rule for the virtual network.

## Route all traffic to the firewall

Your virtual networks in Azure have default route tables in place when you create the network. By implementing a user-defined route table, you can control how traffic is routed within your virtual network. In the following steps, you create a UDR to route all traffic to your Azure Firewall.

- On the Azure portal menu or the *Home* page, select **Create a resource**.
- Search for **Route tables**.
- Select **Route Tables**.
- Select **Create**.
- Enter the following values:

Setting	Action
Region	Select the region as your virtual network.
Name	Enter a name.
Propagate gateway routes	Select <b>No</b>

- Select **Review + create**. After validation finishes, select **Create**.
- Once the deployment completes, select **Go to Resource**.
- From the menu on the left, select **Routes**, then select **Add** to create a new route table
- Configure the route table with the following settings:

Setting	Action
Address prefix	Enter <i>0.0.0.0/0</i>
Next hop type	Select <i>Virtual appliance</i>
Next hop address	Enter the <i>Firewall Private IP</i> you saved in <a href="#">Deploy the firewall</a> .

10. Select **Add** to create the route.
11. From the menu on the left, select **Subnets**, then select **Associate** to associate your route table with the container app's subnet.
12. Configure the *Associate subnet* with the following values:

Setting	Action
Address prefix	Select the virtual network for your container app.
Next hop type	Select the subnet your for container app.

13. Select **OK**.

## Configure firewall policies

### ⓘ Note

When using UDR with Azure Firewall in Azure Container Apps, you will need to add certain FQDN's and service tags to the allowlist for the firewall. Please refer to [configuring UDR with Azure Firewall](#) to determine which service tags you need.

Now, all outbound traffic from your container app is routed to the firewall. Currently, the firewall still allows all outbound traffic through. In order to manage what outbound traffic is allowed or denied, you need to configure firewall policies.

1. In your *Azure Firewall* resource on the *Overview* page, select **Firewall policy**
2. From the menu on the left of the firewall policy page, select **Application Rules**.
3. Select **Add a rule collection**.
4. Enter the following values for the **Rule Collection**:

Setting	Action
Name	Enter a collection name
Rule collection type	Select <i>Application</i>
Priority	Enter the priority such as 110
Rule collection action	Select <i>Allow</i>

Setting	Action
Rule collection group	Select <i>DefaultApplicationRuleCollectionGroup</i>

5. Under Rules, enter the following values

Setting	Action
Name	Enter a name for the rule
Source type	Select <i>IP Address</i>
Source	Enter *
Protocol	Enter <i>http:80,https:443</i>
Destination Type	Select FQDN.
Destination	Enter <code>mcr.microsoft.com</code> , <code>*.data.mcr.microsoft.com</code> . If you're using ACR, add your ACR address and <code>*.blob.core.windows.net</code> .
Action	Select <i>Allow</i>

#### (!) Note

If you are using [Docker Hub registry](#) and want to access it through your firewall, you will need to add the following FQDNs to your rules destination list: `hub.docker.com`, `registry-1.docker.io`, and `production.cloudflare.docker.com`.

6. Select Add.

## Verify your firewall is blocking outbound traffic

To verify your firewall configuration is set up correctly, you can use the `curl` command from your app's debugging console.

1. Navigate to your Container App that is configured with Azure Firewall.
2. From the menu on the left, select **Console**, then select your container that supports the `curl` command.
3. In the **Choose start up command** menu, select `/bin/sh`, and select **Connect**.

4. In the console, run `curl -s https://mcr.microsoft.com`. You should see a successful response as you added `mcr.microsoft.com` to the allowlist for your firewall policies.
5. Run `curl -s https://<FQDN_ADDRESS>` for a URL that doesn't match any of your destination rules such as `example.com`. The example command would be `curl -s https://example.com`. You should get no response, which indicates that your firewall has blocked the request.

## Next steps

[Authentication in Azure Container Apps](#)

# Securing a custom VNET in Azure Container Apps with Network Security Groups

Article • 03/28/2024

Network Security Groups (NSGs) needed to configure virtual networks closely resemble the settings required by Kubernetes.

You can lock down a network via NSGs with more restrictive rules than the default NSG rules to control all inbound and outbound traffic for the Container Apps environment at the subscription level.

In the workload profiles environment, user-defined routes (UDRs) and [securing outbound traffic with a firewall](#) are supported. When using an external workload profiles environment, inbound traffic to Azure Container Apps is routed through the public IP that exists in the [managed resource group](#) rather than through your subnet. This means that locking down inbound traffic via NSG or Firewall on an external workload profiles environment isn't supported. For more information, see [Networking in Azure Container Apps environments](#).

In the Consumption only environment, custom user-defined routes (UDRs) and ExpressRoutes aren't supported.

## NSG allow rules

The following tables describe how to configure a collection of NSG allow rules. The specific rules required depend on your [environment type](#).

### Inbound

Workload profiles environment

#### ⓘ Note

When using workload profiles, inbound NSG rules only apply for traffic going through your virtual network. If your container apps are set to accept traffic from the public internet, incoming traffic goes through the public endpoint instead of the virtual network.

[+] [Expand table](#)

Protocol	Source	Source ports	Destination	Destination ports	Description
TCP	Your client IPs	*	Your container app's subnet <sup>1</sup>	80, 31080	Allow your Client IPs to access Azure Container Apps when using HTTP. 31080 is the port on which the Container Apps Environment Edge Proxy responds to the HTTP traffic. It is behind the internal load balancer.
TCP	Your client IPs	*	Your container app's subnet <sup>1</sup>	443, 31443	Allow your Client IPs to access Azure Container Apps when using HTTPS. 31443 is the port on which the Container Apps Environment Edge Proxy responds to the HTTPS traffic. It is behind the internal load balancer.
TCP	AzureLoadBalancer	*	Your container app's subnet	30000-32767 <sup>2</sup>	Allow Azure Load Balancer to probe backend pools.

<sup>1</sup> This address is passed as a parameter when you create an environment. For example, 10.0.0.0/21.

<sup>2</sup> The full range is required when creating your Azure Container Apps as a port within the range will be dynamically allocated. Once created, the required ports are two immutable, static values, and you can update your NSG rules.

## Outbound

Workload profiles environment

 Expand table

Protocol	Source	Source ports	Destination	Destination ports	Description
TCP	Your container app's subnet	*	MicrosoftContainerRegistry	443	This is the service tag for Microsoft container registry for system containers.
TCP	Your container app's subnet	*	AzureFrontDoor.FirstParty	443	This is a dependency of the MicrosoftContainerRegistry service tag.
Any	Your container app's subnet	*	Your container app's subnet	*	Allow communication between IPs in your container app's subnet.
TCP	Your container app's subnet	*	AzureActiveDirectory	443	If you're using managed identity, this is required.
TCP	Your container app's subnet	*	AzureMonitor	443	Only required when using Azure Monitor. Allows outbound calls to Azure Monitor.
TCP and UDP	Your container app's subnet	*	168.63.129.16	53	Enables the environment to use Azure DNS to resolve the hostname.
TCP	Your container app's subnet <sup>1</sup>	*	Your Container Registry	Your container registry's port	This is required to communicate with your container registry. For example, when using ACR, you need AzureContainerRegistry and AzureActiveDirectory for the destination, and the port will be your container registry's port unless using private endpoints. <sup>2</sup>
TCP	Your container app's subnet	*	Storage.<Region>	443	Only required when using Azure Container Registry to host your images.

<sup>1</sup> This address is passed as a parameter when you create an environment. For example, `10.0.0.0/21`.

<sup>2</sup> If you're using Azure Container Registry (ACR) with NSGs configured on your virtual network, create a private endpoint on your ACR to allow Azure Container Apps to pull images through the virtual network. You don't need to add an NSG rule for ACR when configured with private endpoints.

## Considerations

- If you're running HTTP servers, you might need to add ports `80` and `443`.
- Don't explicitly deny the Azure DNS address `168.63.128.16` in the outgoing NSG rules, or your Container Apps environment won't be able to function.

# Provide a virtual network to an external Azure Container Apps environment

Article • 09/05/2024

The following example shows you how to create a Container Apps environment in an existing virtual network.

Begin by signing in to the [Azure portal](#).

## Create a container app

To create your container app, start at the Azure portal home page.

1. Search for **Container Apps** in the top search bar.
2. Select **Container Apps** in the search results.
3. Select the **Create** button.

### Basics tab

In the *Basics* tab, do the following actions.

1. Enter the following values in the *Project details* section.

  Expand table

Setting	Action
Subscription	Select your Azure subscription.
Resource group	Select <b>Create new</b> and enter <b>my-container-apps</b> .
Container app name	Enter <b>my-container-app</b> .
Deployment source	Select <b>Container image</b> .

### Create an environment

Next, create an environment for your container app.

1. Select the appropriate region.

[ ] Expand table

Setting	Value
Region	Select Central US.

2. In the *Create Container Apps environment* field, select the **Create new** link.

3. In the *Create Container Apps Environment* page on the *Basics* tab, enter the following values:

[ ] Expand table

Setting	Value
Environment name	Enter <b>my-environment</b> .
Environment type	Select <b>Workload profiles</b> .
Zone redundancy	Select <b>Disabled</b>

4. Select the **Monitoring** tab to create a Log Analytics workspace.

5. Select **Azure Log Analytics** as the *Logs Destination*.

6. Select the **Create new** link in the *Log Analytics workspace* field and enter the following values.

[ ] Expand table

Setting	Value
Name	Enter <b>my-container-apps-logs</b> .

The *Location* field is prefilled with *Central US* for you.

7. Select **OK**.

#### (1) Note

You can use an existing virtual network, but a dedicated subnet with a CIDR range of **/23** or larger is required for use with Container Apps when using the Consumption only Architecture. When using a workload profiles environment, a **/27** or larger is required. To learn more about subnet sizing, see the [networking architecture overview](#).

7. Select the **Networking** tab to create a VNET.
8. Select **Yes** next to *Use your own virtual network*.
9. Next to the *Virtual network* box, select the **Create new link** and enter the following value.

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

Setting	Value
Name	Enter <b>my-custom-vnet</b> .

10. Select the **OK** button.
11. Next to the *Infrastructure subnet* box, select the **Create new link** and enter the following values:

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

Setting	Value
Subnet Name	Enter <b>infrastructure-subnet</b> .
Virtual Network Address Block	Keep the default values.
Subnet Address Block	Keep the default values.

12. Select the **OK** button.
13. Under *Virtual IP*, select **External**.
14. Select **Create**.

## Deploy the container app

1. Select **Review and create** at the bottom of the page.

If no errors are found, the *Create* button is enabled.

If there are errors, any tab containing errors is marked with a red dot. Navigate to the appropriate tab. Fields containing an error are highlighted in red. Once all errors are fixed, select **Review and create** again.

2. Select **Create**.

A page with the message *Deployment is in progress* is displayed. Once the deployment is successfully completed, you see the message: *Your deployment is complete.*

## Clean up resources

If you're not going to continue to use this application, you can remove the **my-container-apps** resource group. This deletes the Azure Container Apps instance and all associated services. It also deletes the resource group that the Container Apps service automatically created and which contains the custom network components.

## Next steps

[Managing autoscaling behavior](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# .NET on Azure Container Apps overview

Article • 06/25/2024

To deploy a .NET application to a cloud native environment like Azure Container Apps, there are decisions you need to make to ensure your application runs smoothly and securely. This guide covers key concepts involved in deploying a .NET application to Azure Container Apps.

Azure Container Apps is a fully managed serverless container service that allows you to run containerized applications without having to manage the underlying infrastructure. Container Apps includes built-in support for features including autoscaling, health checks, and transport layer security (TLS) certificates.

This article details the concepts and concerns important to you as you deploy a .NET application on Azure Container Apps.

## Select a resource type

Container Apps supports two types of resources: apps and jobs. Apps are continuously running services, while jobs are short-lived tasks designed to run to completion.

As you prepare to deploy your app, consider differences between these two application types as their behavior affects how you manage your .NET application. The following table describes the difference in use cases between apps and jobs.

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

Use case	Resource type
An ASP.NET Core web API that serves HTTP requests	App
A .NET Core console application that processes some data, then exits	Job
A continuously running background service that processes messages from a queue	App
An image optimization service that runs only when large images are saved to a storage account.	Job
An application using a framework like Hangfire, Quartz.NET, or the Azure WebJobs SDK	App

# Containerize and deploy your .NET application

For both apps or jobs, you need to build a container image to package your .NET application. For more information on building container image, see [Docker images for ASP.NET Core](#).

Once set up, you can deploy your application to Azure Container Apps by following these guides:

- [Tutorial: Deploy to Azure Container Apps using Visual Studio](#)
- [Quickstart: Build and deploy from a repository to Azure Container Apps](#)
- [Create a job with Azure Container Apps](#)

## Use the HTTP ingress

Azure Container Apps includes a built-in HTTP ingress that allows you to expose your apps to traffic coming from outside the container. The Container Apps ingress sits between your app and the end user. Since the ingress acts as an intermediary, whatever the end user sees ends at the ingress, and whatever your app sees begins at the ingress.

The ingress manages TLS termination and custom domains, eliminating the need for you to manually configure them in your app. Through the ingress, port `443` is exposed for HTTPS traffic, and optionally port `80` for HTTP traffic. The ingress forwards requests to your app at its target port.

If your app needs metadata about the original request, it can use [X-forwarded headers](#).

To learn more, see [HTTP ingress in Azure Container Apps](#).

## Define a target port

To receive traffic, the ingress is configured on a target port where your app listens for traffic.

When ASP.NET Core is running in a container, the application listens to ports as configured in the container image. When you use the [official ASP.NET Core images](#), your app is configured to listen to HTTP on a default port. The default port depends on the ASP.NET Core version.

 [Expand table](#)

Runtime	Target port
ASP.NET Core 7 and earlier	80
ASP.NET Core 8 and later	8080

When you configure the ingress, set the target port to the number corresponding to the container image you're using.

## Define X-forwarded headers

As the ingress handles the original HTTP request, your app sees the ingress as the client. There are some situations where your app needs to know the original client's IP address or the original protocol (HTTP or HTTPS). You can access the protocol and IP information via the request's [X-Forwarded-\\* header](#).

You can read original values from these headers by accessing the `ForwardedHeaders` object.

C#

```
builder.Services.Configure<ForwardedHeadersOptions>(options =>
{
 options.ForwardedHeaders =
 ForwardedHeaders.XForwardedFor | ForwardedHeaders.XForwardedProto;
 options.KnownNetworks.Clear();
 options.KnownProxies.Clear();
});
```

For more information on working with request headers, see [Configure ASP.NET Core to work with proxy servers and load balancers](#).

## Build cloud-native .NET applications

Applications deployed to Container Apps often work best when you build on the foundations of [cloud-native principles](#). The following sections help detail common concerns surrounding cloud-native applications.

## Application configuration

When deploying your .NET application to Azure Container Apps, use environment variables for storing configuration information instead of using `appsettings.json`. This practice allows you to configure your application in different ways in differing

environments. Additionally, using environment variables makes it easier to manage configuration values without having to rebuild and redeploy your container image.

In Azure Container Apps, you [set environment variables](#) when you define your app or job's container. Store sensitive values in secrets and reference them as environment variables. To learn more about managing secrets, see [Manage secrets in Azure Container Apps](#).

## Managed identity

Azure Container Apps supports managed identity, which allows your app to access other Azure services without needing to exchange credentials. To learn more about securely communicating between Azure services, see [Managed identities in Azure Container Apps](#).

## Logging

In a cloud-native environment, logging is crucial for monitoring and troubleshooting your applications. By default, Azure Container Apps uses Azure Log Analytics to collect logs from your containers. You can configure other logging providers. To learn more about application logging, see [Log storage and monitoring options in Azure Container Apps](#).

When you configure a [logging provider](#) that writes logs to the console, Azure Container Apps collects and stores log messages for you.

## Health probes

Azure Container Apps includes built-in support for health probes, which allow you to monitor the health of your applications. If a probe determines your application is in an unhealthy state, then your container is automatically restarted. To learn more about health probes, see [Health probes in Azure Container Apps](#).

For a chance to implement custom logic to determine the health of your application, you can configure a health check endpoint. To learn more about health check endpoints, see [Health checks in ASP.NET Core](#).

## Autoscaling considerations

By default, Azure Container Apps automatically scales your ASP.NET Core apps based on the number of incoming HTTP requests. You can also configure custom autoscaling rules

based on other metrics, such as CPU or memory usage. To learn more about scaling, see [Set scaling rules in Azure Container Apps](#).

In .NET 8.0.4 and later, ASP.NET Core apps that use [data protection](#) are automatically configured to keep protected data accessible to all replicas as the application scales. When your app begins to scale, a key manager handles the writing and sharing keys across multiple revisions. As the app is deployed, the environment variable `autoConfigureDataProtection` is automatically set `true` to enable this feature. For more information on this auto configuration, see [this GitHub pull request ↗](#).

Autoscaling changes the number of replicas of your app based on the rules you define. By default, Container Apps randomly routes incoming traffic to the replicas of your ASP.NET Core app. Since traffic can split among different replicas, your app should be stateless so your application doesn't experience state-related issues.

Features such as anti-forgery, authentication, SignalR, Blazor Server, and Razor Pages depend on data protection require extra configuration to work correctly when scaling to multiple replicas.

## Configure data protection

ASP.NET Core has special features protect and unprotect data, such as session data and anti-forgery tokens. By default, data protection keys are stored on the file system, which isn't suitable for a cloud-native environment.

If you're deploying a .NET Aspire application, data protection is automatically configured for you. In all other situations, you need to [configure data protection manually](#).

## Configure ASP.NET Core SignalR

ASP.NET Core SignalR requires a backplane to distribute messages to multiple server replicas. When deploying your ASP.NET Core app with SignalR to Azure Container Apps, you must configure one of the supported backplanes, such as Azure SignalR Service or Redis. To learn more about backplanes, see [ASP.NET Core SignalR hosting and scaling](#).

## Configure Blazor Server

ASP.NET Core Blazor Server apps store state on the server, which means that each client must be connected to the same server replica during their session. When deploying your Blazor Server app to Azure Container Apps, you must enable sticky sessions to ensure that clients are routed to the same replica. To learn more, see [Session Affinity in Azure Container Apps](#).

## Related information

- Deploy a .NET Aspire app
  - Deploy and scale an ASP.NET Core app
- 

## Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#)

# Deploy a .NET Aspire project to Azure Container Apps

Article • 06/15/2024

.NET Aspire projects are designed to run in containerized environments. Azure Container Apps is a fully managed environment that enables you to run microservices and containerized applications on a serverless platform. This article will walk you through creating a new .NET Aspire solution and deploying it to Microsoft Azure Container Apps using the Azure Developer CLI (`azd`). You'll learn how to complete the following tasks:

- ✓ Provision an Azure resource group and Container Registry
- ✓ Publish the .NET Aspire projects as container images in Azure Container Registry
- ✓ Provision a Redis container in Azure
- ✓ Deploy the apps to an Azure Container Apps environment
- ✓ View application console logs to troubleshoot application issues

## Prerequisites

To work with .NET Aspire, you need the following installed locally:

- [.NET 8.0](#)
- .NET Aspire workload:
  - Installed with the [Visual Studio installer](#) or the [.NET CLI workload](#).
- An OCI compliant container runtime, such as:
  - [Docker Desktop](#) or [Podman](#).
- An Integrated Developer Environment (IDE) or code editor, such as:
  - [Visual Studio 2022](#) version 17.10 or higher (Optional)
  - [Visual Studio Code](#) (Optional)
  - [C# Dev Kit: Extension](#) (Optional)

For more information, see [.NET Aspire setup and tooling](#).

As an alternative to this tutorial and for a more in-depth guide, see [Deploy a .NET Aspire project to Azure Container Apps using azd \(in-depth guide\)](#).

## Deploy .NET Aspire projects with `azd`

With .NET Aspire and Azure Container Apps (ACA), you have a great hosting scenario for building out your cloud-native apps with .NET. We built some great new features into

the Azure Developer CLI (`azd`) specific for making .NET Aspire development and deployment to Azure a friction-free experience. You can still use the Azure CLI and/or Bicep options when you need a granular level of control over your deployments. But for new projects, you won't find an easier path to success for getting a new microservice topology deployed into the cloud.

## Create a .NET Aspire project

As a starting point, this article assumes that you've created a .NET Aspire project from the [.NET Aspire Starter Application template](#). For more information, see [Quickstart: Build your first .NET Aspire project](#).

## Resource naming

When you create new Azure resources, it's important to follow the naming requirements. For Azure Container Apps, the name must be 2-32 characters long and consist of lowercase letters, numbers, and hyphens. The name must start with a letter and end with an alphanumeric character.

For more information, see [Naming rules and restrictions for Azure resources](#).

## Install the Azure Developer CLI

The process for installing `azd` varies based on your operating system, but it is widely available via `winget`, `brew`, `apt`, or directly via `curl`. To install `azd`, see [Install Azure Developer CLI](#).

## Initialize the template

1. Open a new terminal window and `cd` into the *AppHost* project directory of your .NET Aspire solution.
2. Execute the `azd init` command to initialize your project with `azd`, which will inspect the local directory structure and determine the type of app.

```
Azure Developer CLI
```

```
azd init
```

For more information on the `azd init` command, see [azd init](#).

3. Select **Use code in the current directory** when `azd` prompts you with two app initialization options.

Output

```
? How do you want to initialize your app? [Use arrows to move, type to filter]
> Use code in the current directory
 Select a template
```

4. After scanning the directory, `azd` prompts you to confirm that it found the correct .NET Aspire *AppHost* project. Select the **Confirm and continue initializing my app** option.

Output

```
Detected services:

.NET (Aspire)
Detected in:
D:\source\repos\AspireSample\AspireSample.AppHost\AspireSample.AppHost.csproj

azd will generate the files necessary to host your app on Azure using
Azure Container Apps.

? Select an option [Use arrows to move, type to filter]
> Confirm and continue initializing my app
 Cancel and exit
```

5. Enter an environment name, which is used to name provisioned resources in Azure and managing different environments such as `dev` and `prod`.

Output

```
Generating files to run your app on Azure:

(✓) Done: Generating ./azure.yaml
(✓) Done: Generating ./next-steps.md

SUCCESS: Your app is ready for the cloud!
You can provision and deploy your app to Azure by running the azd up
command in this directory. For more information on configuring your
app, see ./next-steps.md
```

`azd` generates a number of files and places them into the working directory. These files are:

- *azure.yaml*: Describes the services of the app, such as .NET Aspire AppHost project, and maps them to Azure resources.
- *.azure/config.json*: Configuration file that informs `azd` what the current active environment is.
- *.azure/aspireazddev/.env*: Contains environment specific overrides.

## Deploy the template

1. Once an `azd` template is initialized, the provisioning and deployment process can be executed as a single command from the *AppHost* project directory using `azd up`:

```
Azure Developer CLI
```

```
azd up
```

2. Select the subscription you'd like to deploy to from the list of available options:

```
Output
```

```
Select an Azure Subscription to use: [Use arrows to move, type to filter]
 1. SampleSubscription01 (xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxx)
 2. SamepleSubscription02 (xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxx)
```

3. Select the desired Azure location to use from the list of available options:

```
Output
```

```
Select an Azure location to use: [Use arrows to move, type to filter]
 42. (US) Central US (centralus)
 43. (US) East US (eastus)
> 44. (US) East US 2 (eastus2)
 46. (US) North Central US (northcentralus)
 47. (US) South Central US (southcentralus)
```

After you make your selections, `azd` executes the provisioning and deployment process.

```
Output
```

```
By default, a service can only be reached from inside the Azure Container Apps environment it is running in. Selecting a service here will also allow it to be reached from the Internet.
```

```
? Select which services to expose to the Internet webfrontend
? Select an Azure Subscription to use: 1. <YOUR SUBSCRIPTION>
```

```
? Select an Azure location to use: 1. <YOUR LOCATION>
```

```
Packaging services (azd package)
```

```
SUCCESS: Your application was packaged for Azure in less than a second.
```

```
Provisioning Azure resources (azd provision)
```

```
Provisioning Azure resources can take some time.
```

```
Subscription: <YOUR SUBSCRIPTION>
```

```
Location: <YOUR LOCATION>
```

```
You can view detailed progress in the Azure Portal:
```

```
<LINK TO DEPLOYMENT>
```

```
(✓) Done: Resource group: <YOUR RESOURCE GROUP>
```

```
(✓) Done: Container Registry: <ID>
```

```
(✓) Done: Log Analytics workspace: <ID>
```

```
(✓) Done: Container Apps Environment: <ID>
```

```
SUCCESS: Your application was provisioned in Azure in 1 minute 13 seconds.
```

```
You can view the resources created under the resource group <YOUR RESOURCE GROUP> in Azure Portal:
```

```
<LINK TO RESOURCE GROUP OVERVIEW>
```

```
Deploying services (azd deploy)
```

```
(✓) Done: Deploying service apiservice
```

```
- Endpoint: <YOUR UNIQUE apiservice APP>.azurecontainerapps.io/
```

```
(✓) Done: Deploying service webfrontend
```

```
- Endpoint: <YOUR UNIQUE webfrontend APP>.azurecontainerapps.io/
```

```
SUCCESS: Your application was deployed to Azure in 1 minute 39 seconds.
```

```
You can view the resources created under the resource group <YOUR RESOURCE GROUP> in Azure Portal:
```

```
<LINK TO RESOURCE GROUP OVERVIEW>
```

```
SUCCESS: Your up workflow to provision and deploy to Azure completed in 3 minutes 50 seconds.
```

The `azd up` command acts as wrapper for the following individual `azd` commands to provision and deploy your resources in a single step:

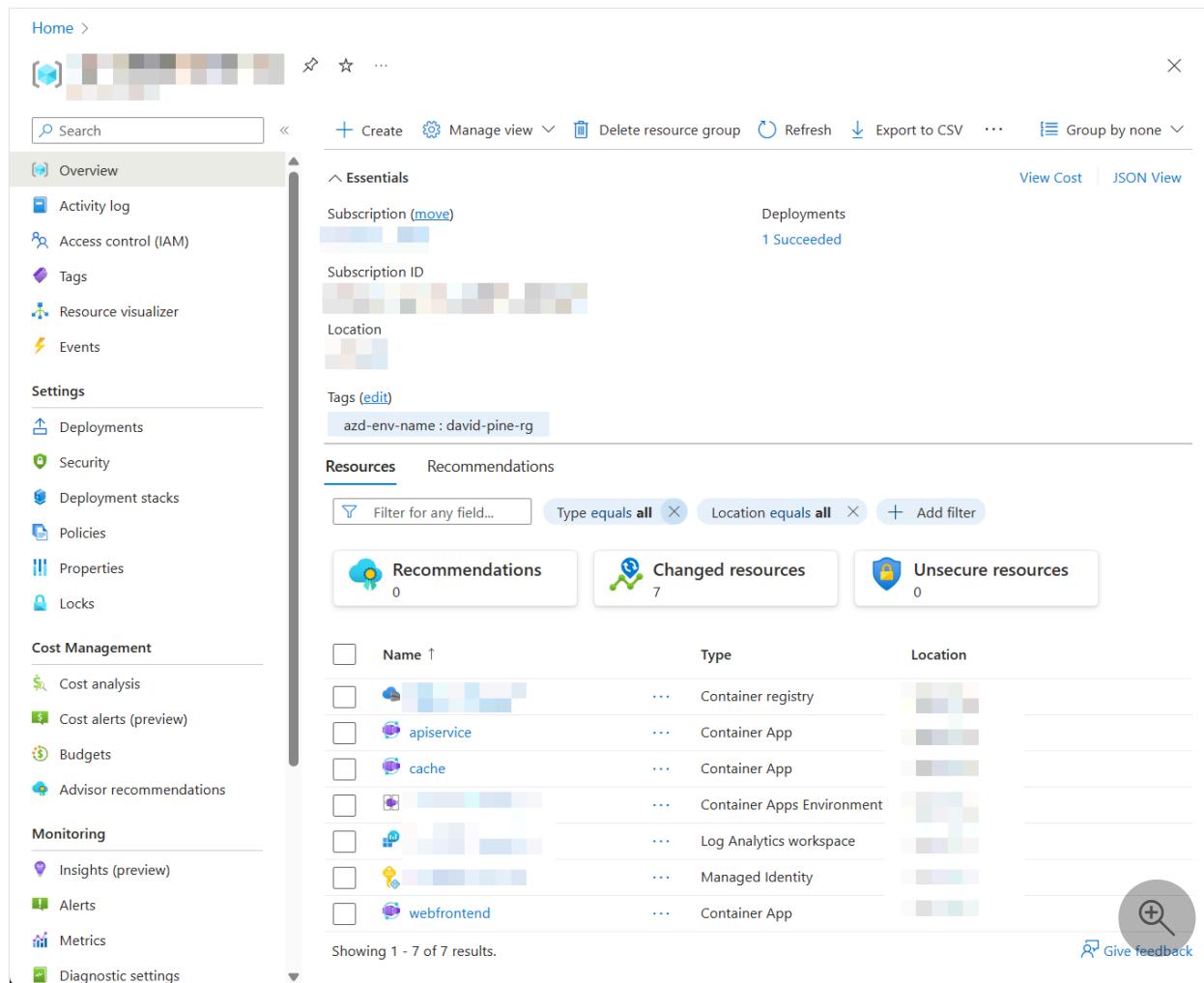
1. **azd package**: The app projects and their dependencies are packaged into containers.
2. **azd provision**: The Azure resources the app will need are provisioned.
3. **azd deploy**: The projects are pushed as containers into an Azure Container Registry instance, and then used to create new revisions of Azure Container Apps in which

the code will be hosted.

When the `azd up` stages complete, your app will be available on Azure, and you can open the Azure portal to explore the resources. `azd` also outputs URLs to access the deployed apps directly.

## Test the deployed app

Now that the app has been provisioned and deployed, you can browse to the Azure portal. In the resource group where you deployed the app, you'll see the three container apps and other resources.

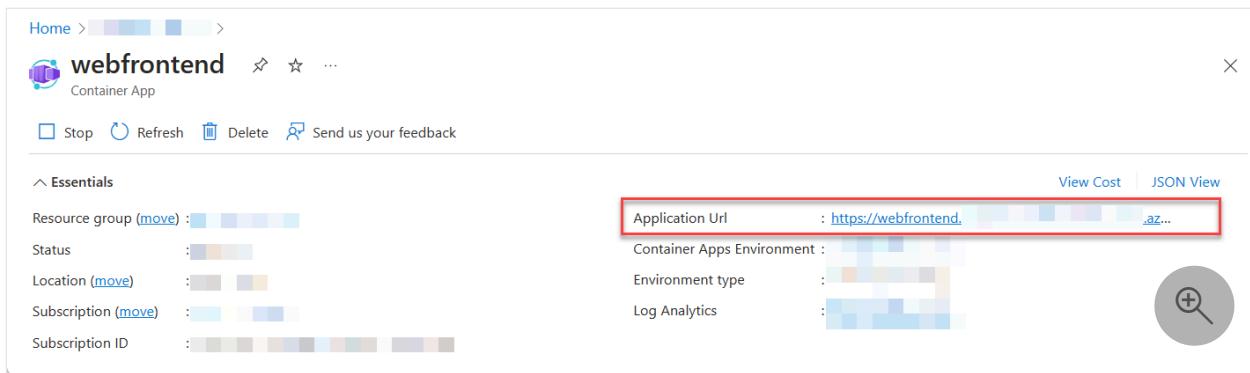


The screenshot shows the Azure portal's resource group overview page. The left sidebar contains navigation links for Home, Overview, Settings, Cost Management, Monitoring, and Diagnostic settings. The main content area displays the following information:

- Essentials:**
  - Subscription (move) - Deployment status: 1 Succeeded
  - Subscription ID
  - Location
  - Tags (edit): azd-env-name : david-pine-rg
- Resources:** A table listing 7 results:

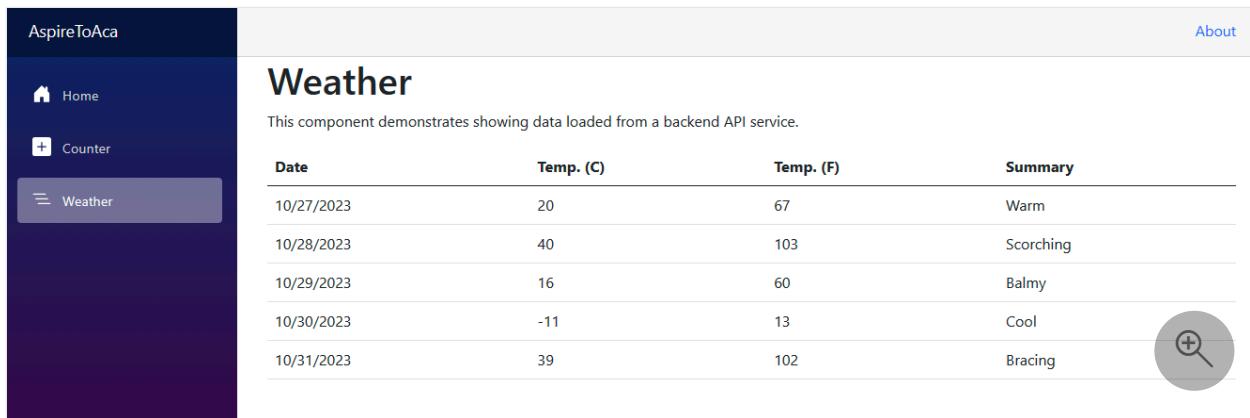
Name ↑	Type	Location
apiservice	Container App	
cache	Container App	
webfrontend	Container App	
Container registry	Container registry	
Container Apps Environment	Container Apps Environment	
Log Analytics workspace	Log Analytics workspace	
Managed Identity	Managed Identity	
- Recommendations:** Three cards: Recommendations (0), Changed resources (7), and Unsecure resources (0).

Click on the `web` Container App to open it up in the portal.



The screenshot shows the Azure portal interface for a 'Container App' named 'webfrontend'. In the top navigation bar, there are links for 'Home', 'Container Apps', 'Container Apps Environment', 'Logs', 'Metrics', 'Events', 'Feedback', and 'View Cost'. Below the navigation, there's a toolbar with 'Stop', 'Refresh', 'Delete', and 'Send us your feedback' buttons. On the left, a sidebar titled 'Essentials' displays resource group, status, location, subscription, and subscription ID. On the right, detailed information is shown for the application, including its URL (<https://webfrontend.az...>), container apps environment, environment type, and log analytics. A search icon is also present.

Click the Application URL link to open the front end in the browser.



The screenshot shows a web application interface titled 'AspireToAca'. The left sidebar has three items: 'Home', 'Counter', and 'Weather', with 'Weather' being the active tab. The main content area is titled 'Weather' and contains a sub-header: 'This component demonstrates showing data loaded from a backend API service.' Below this is a table with five rows of weather data:

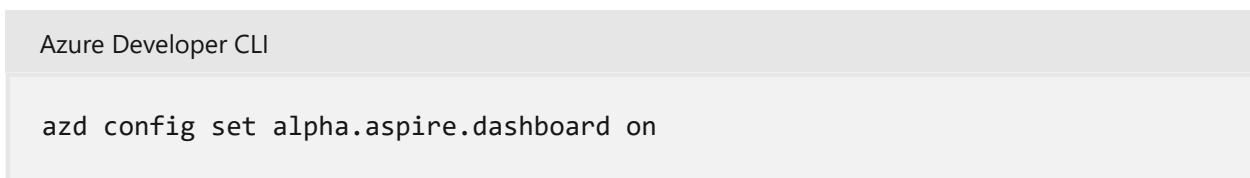
Date	Temp. (C)	Temp. (F)	Summary
10/27/2023	20	67	Warm
10/28/2023	40	103	Scorching
10/29/2023	16	60	Balmy
10/30/2023	-11	13	Cool
10/31/2023	39	102	Bracing

A magnifying glass icon is located in the bottom right corner of the main content area.

When you click the "Weather" node in the navigation bar, the front end `web` container app makes a call to the `apiservice` container app to get data. The front end's output will be cached using the `redis` container app and the [.NET Aspire Redis Output Caching component](#). As you refresh the front end a few times, you'll notice that the weather data is cached. It will update after a few seconds.

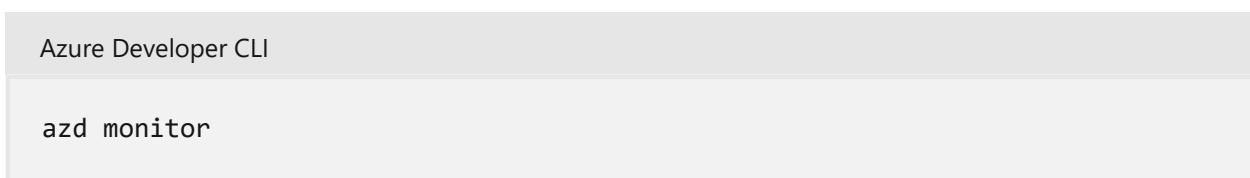
## Deploy the .NET Aspire Dashboard

You can deploy the .NET Aspire dashboard as part of your hosted app. This feature is currently in alpha support, so you must enable the `alpha.aspire.dashboard` feature flag. When enabled, the `azd` output logs print an additional URL to the deployed dashboard.



The screenshot shows the Azure Developer CLI interface. The first section is labeled 'Azure Developer CLI' and contains the command `azd config set alpha.aspire.dashboard on`. The second section is also labeled 'Azure Developer CLI' and contains the command `azd monitor`.

You can also run `azd monitor` to automatically launch the dashboard.



The screenshot shows the Azure Developer CLI interface. The first section is labeled 'Azure Developer CLI' and contains the command `azd config set alpha.aspire.dashboard on`. The second section is also labeled 'Azure Developer CLI' and contains the command `azd monitor`.

# Clean up resources

Run the following Azure CLI command to delete the resource group when you no longer need the Azure resources you created. Deleting the resource group also deletes the resources contained inside of it.

Azure CLI

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

For more information, see [Clean up resources in Azure](#).

 Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see [our contributor guide](#).

.NET

 .NET Aspire feedback

.NET Aspire is an open source project. Select a link to provide feedback:

 Open a documentation issue

 Provide product feedback

# Deploying and scaling an ASP.NET Core app on Azure Container Apps

Article • 04/13/2023

Apps deployed to Azure that experience intermittent high demand benefit from scalability to meet demand. Scalable apps can scale out to ensure capacity during workload peaks and then scale down automatically when the peak drops, which can lower costs. Horizontal scaling (scaling out) adds new instances of a resource, such as VMs or database replicas. This article demonstrates how to deploy a horizontally scalable ASP.NET Core app to [Azure container apps](#) by completing the following tasks:

1. [Set up the sample project](#)
2. [Deploy the app to Azure Container Apps](#)
3. [Scale and troubleshoot the app](#)
4. [Create the Azure Services](#)
5. [Connect the Azure services](#)
6. [Configure and redeploy the app](#)

This article uses Razor Pages, but most of it applies to other ASP.NET Core apps.

In some cases, basic ASP.NET Core apps are able to scale without special considerations. However, apps that utilize certain framework features or architectural patterns require extra configurations, including the following:

- **Secure form submissions:** Razor Pages, MVC and Web API apps often rely on form submissions. By default these apps use [cross site forgery tokens](#) and internal data protection services to secure requests. When deployed to the cloud, these apps must be configured to manage data protection service concerns in a secure, centralized location.
- **SignalR circuits:** Blazor Server apps require the use of a centralized [Azure SignalR service](#) in order to securely scale. These services also utilize the data protection services mentioned previously.
- **Centralized caching or state management services:** Scalable apps may use [Azure Cache for Redis](#) to provide distributed caching. [Azure storage](#) may be needed to store state for frameworks such as [Microsoft Orleans](#), which can help write apps that manage state across many different app instances.

The steps in this article demonstrate how to properly address the preceding concerns by deploying a scalable app to Azure Container Apps. Most of the concepts in this tutorial

also apply when scaling [Azure App Service](#) instances.

## Set up the sample project

Use the GitHub Explorer sample app to follow along with this tutorial. Clone the app from GitHub using the following command:

.NET CLI

```
git clone "https://github.com/dotnet/AspNetCore.Docs.Samples.git"
```

Navigate to the `/tutorials/scalable-razor-apps/start` folder and open the `ScalableRazor.csproj`.

The sample app uses a search form to browse GitHub repositories by name. The form relies on the built-in ASP.NET Core data protection services to handle antiforgery concerns. By default, when the app scales horizontally on Container Apps, the data protection service throws an exception.

## Test the app

1. Launch the app in Visual Studio. The project includes a Docker file, which means that the arrow next to the run button can be selected to start the app using either a Docker Desktop setup or the standard ASP.NET Core local web server.

Use the search form to browse for GitHub repositories by name.

The screenshot shows a web application titled "GitHub searcher!". At the top, there is a navigation bar with links for "GitHub Browser", "Home", and "Privacy". Below the navigation, a search bar contains the text "Microsoft". A green "Search" button is located below the search bar. The main content area displays a table with three columns: "Name", "Description", and "Link". The first row of the table shows the repository "HealthVault-Mobile-iOS-Library". The "Description" column for this row states: "The HealthVault team has recently added the capability to write applications that will run on Mobile Devices and connect directly to the". To the right of the "Link" column for this row is a dark blue "Browse" button.

Name	Description	Link
HealthVault-Mobile-iOS-Library	The HealthVault team has recently added the capability to write applications that will run on Mobile Devices and connect directly to the	<button>Browse</button>

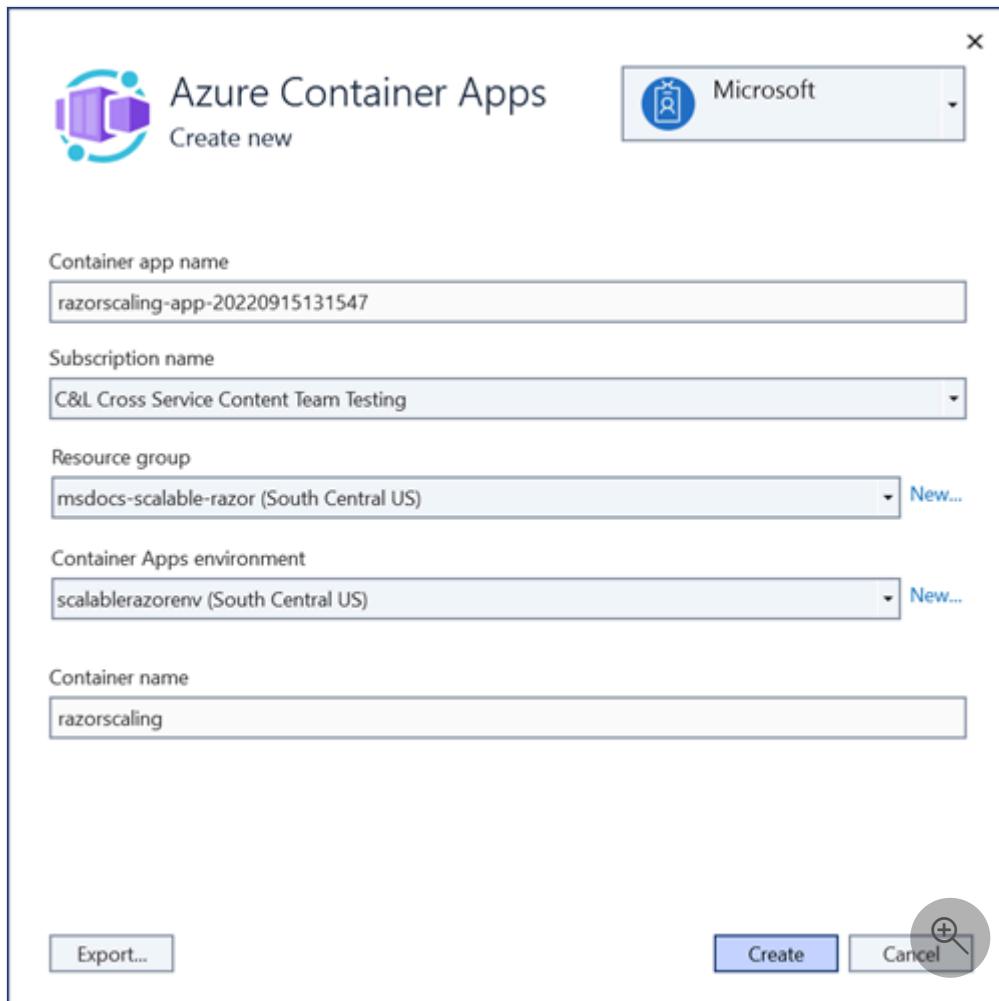
# Deploy the app to Azure Container Apps

Visual Studio is used to deploy the app to Azure Container Apps. Container apps provide a managed service designed to simplify hosting containerized apps and microservices.

## ⓘ Note

Many of the resources created for the app require a location. For this app, location isn't important. A real app should select a location closest to the clients. You may want to select a location near you.

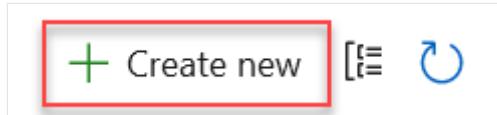
1. In Visual Studio solution explorer, right click on the top level project node and select **Publish**.
2. In the publishing dialog, select **Azure** as the deployment target, and then select **Next**.
3. For the specific target, select **Azure Container Apps (Linux)**, and then select **Next**.
4. Create a new container app to deploy to. Select the green + icon to open a new dialog and enter the following values:



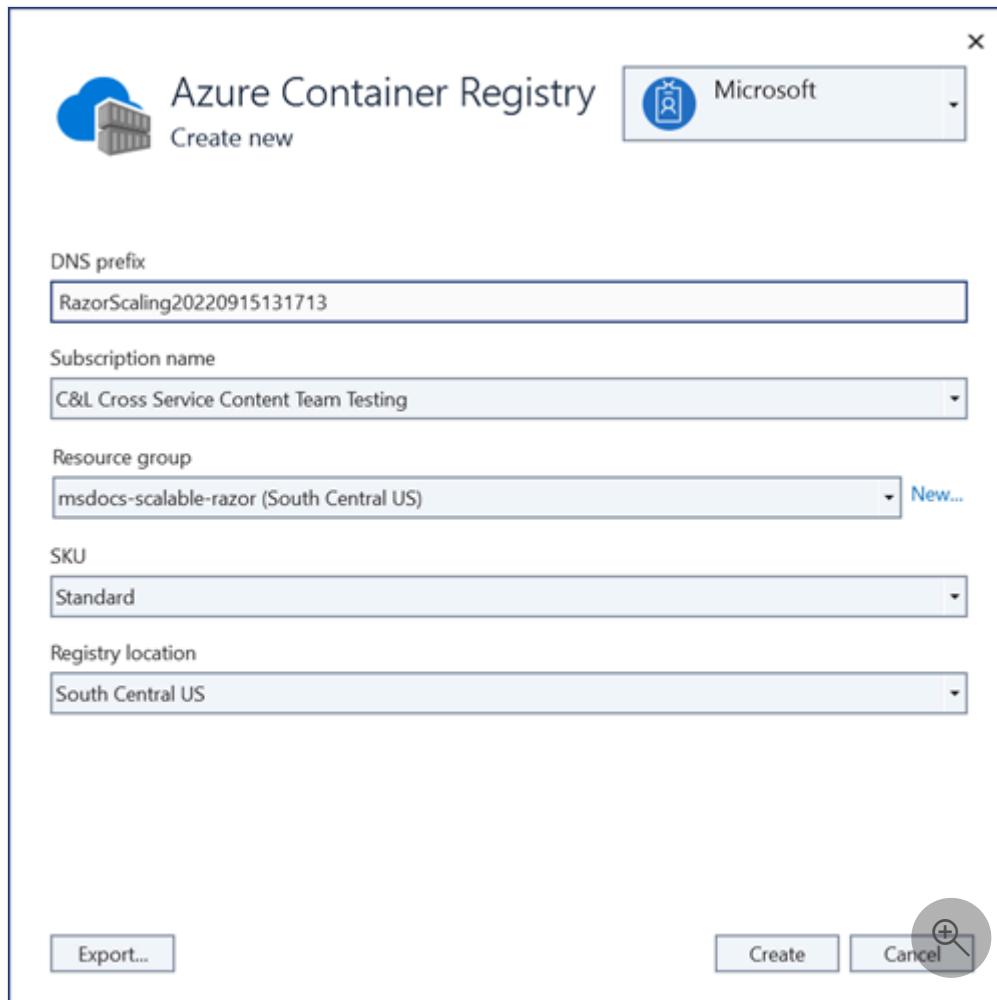
- **Container app name:** Leave the default value or enter a name.
- **Subscription name:** Select the subscription to deploy to.
- **Resource group:** Select **New** and create a new resource group called *msdocs-scalable-razor*.
- **Container apps environment:** Select **New** to open the container apps environment dialog and enter the following values:
  - **Environment name:** Keep the default value.
  - **Location:** Select a location near you.
  - **Azure Log Analytics Workspace:** Select **New** to open the log analytics workspace dialog.
    - **Name:** Leave the default value.
    - **Location:** Select a location near you and then select **Ok** to close the dialog.
  - Select **Ok** to close the container apps environment dialog.
- Select **Create** to close the original container apps dialog. Visual Studio creates the container app resource in Azure.

5. Once the resource is created, make sure it's selected in the list of container apps, and then select **Next**.

6. You'll need to create an Azure Container Registry to store the published image artifact for your app. Select the green + icon on the container registry screen.



7. Leave the default values, and then select **Create**.



8. After the container registry is created, make sure it's selected, and then select finish to close the dialog workflow and display a summary of the publishing profile.

If Visual Studio prompts you to enable the Admin user to access the published docker container, select **Yes**.

9. Select **Publish** in the upper right of the publishing profile summary to deploy the app to Azure.

When the deployment finishes, Visual Studio launches the browser to display the hosted app. Search for **Microsoft** in the form field, and a list of repositories is displayed.

## Scale and troubleshoot the app

The app is currently working without any issues, but we'd like to scale the app across more instances in anticipation of high traffic volumes.

1. In the Azure portal, search for the `razorscaling-app-****` container app in the top level search bar and select it from the results.
2. On the overview page, select **Scale** from the left navigation, and then select **+ Edit and deploy**.
3. On the revisions page, switch to the **Scale** tab.
4. Set both the min and max instances to **4** and then select **Create**. This configuration change guarantees your app is scaled horizontally across four instances.

Navigate back to the app. When the page loads, at first it appears everything is working correctly. However, when a search term is entered and submitted, an error may occur. If an error is not displayed, submit the form several more times.

## Troubleshooting the error

It's not immediately apparent why the search requests are failing. The browser tools indicate a 400 Bad Request response was sent back. However, you can use the logging features of container apps to diagnose errors occurring in your environment.

1. On the overview page of the container app, select **Logs** from the left navigation.
2. On the **Logs** page, close the pop-up that opens and navigate to the **Tables** tab.
3. Expand the **Custom Logs** item to reveal the **ContainerAppConsoleLogs\_CL** node. This table holds various logs for the container app that can be queried to troubleshoot problems.

The screenshot shows the Azure portal interface for a container app named "githubbrowser-app-20220830105907". The left sidebar has a red box around the "Logs" item. The main area shows the "Tables" tab selected under the "Custom Logs" section. A red box highlights the "ContainerAppConsoleLogs\_CL" table. The table has three columns: Computer (string), ContainerAppName\_s (string), and ContainerGroupId\_g (string). There is also a "LogManagement" section with a red box around it.

4. In the query editor, compose a basic query to search the **ContainerAppConsoleLogs\_CL** Logs table for recent exceptions, such as the following script:

KQL

```
ContainerAppConsoleLogs_CL
| where Log_s contains "exception"
| sort by TimeGenerated desc
| limit 500
| project ContainerAppName_s, Log_s
```

The preceding query searches the **ContainerAppConsoleLogs\_CL** table for any rows that contain the word exception. The results are ordered by the time generated, limited to 500 results, and only include the **ContainerAppName\_s** and **Log\_s** columns to make the results easier to read.

5. Select **Run**, a list of results is displayed. Read through the logs and note that most of them are related to antiforgery tokens and cryptography.

The screenshot shows the Azure Data Explorer interface. A query is entered in the KQL editor:

```
ContainerAppConsoleLogs_CL
| where Log_s contains "exception"
| sort by TimeGenerated desc
| limit 500
| project ContainerAppName_s, Log_s
```

The results pane displays a list of log entries. The first few entries are:

ContainerAppName_s	Log_s
githubbrowser-app-20220830105907	System.Security.Cryptography.CryptographicException: The key (or its dependencies) were not found.
githubbrowser-app-20220830105907	End of inner exception stack trace ...
githubbrowser-app-20220830105907	An exception was thrown while deserializing the token.
githubbrowser-app-20220830105907	Microsoft.AspNetCore.Antiforgery.AntiforgeryValidationException: The antiforgery validation failed.
githubbrowser-app-20220830105907	An unhandled exception has occurred while executing the request.
githubbrowser-app-20220830105907	System.InvalidOperationException: An invalid request URI was provided. Either the request URI or the host header must be valid.
githubbrowser-app-20220830105907	at Microsoft.AspNetCore.Diagnostics.ExceptionHandlerMiddleware.<Invoke>()
githubbrowser-app-20220830105907	[41m[30mfail][39m[22m[49m: Microsoft.AspNetCore.Diagnostics.ExceptionHandlerMiddleware
githubbrowser-app-20220830105907	[41m[30mfail][39m[22m[49m: Microsoft.AspNetCore.Diagnostics.ExceptionHandlerMiddleware

### Important

The errors in the app are caused by the .NET data protection services. When multiple instances of the app are running, there is no guarantee that the HTTP POST request to submit the form is routed to the same container that initially loaded the page from the HTTP GET request. If the requests are handled by different instances, the antiforgery tokens aren't handled correctly and an exception occurs.

In the steps ahead, this issue is resolved by centralizing the data protection keys in an Azure storage service and protecting them with Key Vault.

# Create the Azure Services

To resolve the preceding errors, the following services are created and connected to the app:

- **Azure Storage Account:** Handles storing data for the Data Protection Services. Provides a centralized location to store key data as the app scales. Storage accounts can also be used to hold documents, queue data, file shares, and almost any type of blob data.
- **Azure KeyVault:** This service stores secrets for an app, and is used to help manage encryption concerns for the Data Protection Services.

## Create the storage account service

1. In the Azure portal search bar, enter `Storage accounts` and select the matching result.
2. On the storage accounts listing page, select **+ Create**.
3. On the **Basics** tab, enter the following values:
  - **Subscription:** Select the same subscription that you chose for the container app.
  - **Resource Group:** Select the *msdocs-scalable-razor* resource group you created previously.
  - **Storage account name:** Name the account *scalablerazorstorageXXXX* where the X's are random numbers of your choosing. This name must be unique across all of Azure.
  - **Region:** Select the same region you previously selected.
4. Leave the rest of the values at their default and select **Review**. After Azure validates the inputs, select **Create**.

Azure provisions the new storage account. When the task completes, choose **Go to resource** to view the new service.

## Create the storage container

Create a Container to store the app's data protection keys.

1. On the overview page for the new storage account, select **Storage browser** on the left navigation.
2. Select **Blob containers**.
3. Select **+ Add container** to open the **New container** flyout menu.

4. Enter a name of *scalablerazorkeys*, leave the rest of the settings at their defaults, and then select **Create**.

The new containers appear on the page list.

## Create the key vault service

Create a key vault to hold the keys that protect the data in the blob storage container.

1. In the Azure portal search bar, enter **Key Vault** and select the matching result.
2. On the key vault listing page, select **+ Create**.
3. On the **Basics** tab, enter the following values:
  - **Subscription:** Select the same subscription that was previously selected.
  - **Resource Group:** Select the *msdocs-scalabe-razor* resource group previously created.
  - **Key Vault name:** Enter the name *scalablerazorvaultXXXX*.
  - **Region:** Select a region near your location.
4. Leave the rest of the settings at their default, and then select **Review + create**.  
Wait for Azure to validate your settings, and then select **Create**.

Azure provisions the new key vault. When the task completes, select **Go to resource** to view the new service.

## Create the key

Create a secret key to protect the data in the blob storage account.

1. On the main overview page of the key vault, select **Keys** from the left navigation.
2. On the **Create a key** page, select **+ Generate/Import** to open the **Create a key** flyout menu.
3. Enter *razorkey* in the **Name** field. Leave the rest of the settings at their default values and then select **Create**. A new key appears on the key list page.

## Connect the Azure Services

The Container App requires a secure connection to the storage account and the key vault services in order to resolve the data protection errors and scale correctly. The new services are connected together using the following steps:

### Important

Security role assignments through Service Connector and other tools typically take a minute or two to propagate, and in some rare cases can take up to eight minutes.

## Connect the storage account

1. In the Azure portal, navigate to the Container App overview page.
2. On the left navigation, select **Service connector**
3. On the Service Connector page, choose **+ Create** to open the **Creation Connection** flyout panel and enter the following values:
  - **Container:** Select the Container App created previously.
  - **Service type:** Choose **Storage - blob**.
  - **Subscription:** Select the subscription previously used.
  - **Connection name:** Leave the default value.
  - **Storage account:** Select the storage account created previously.
  - **Client type:** Select **.NET**.
4. Select **Next: Authentication** to progress to the next step.
5. Select **System assigned managed identity** and choose **Next: Networking**.
6. Leave the default networking options selected, and then select **Review + Create**.
7. After Azure validates the settings, select **Create**.

The service connector enables a system-assigned managed identity on the container app. It also assigns a role of **Storage Blob Data Contributor** to the identity so it can perform data operations on the storage containers.

## Connect the key vault

1. In the Azure portal, navigate to your Container App overview page.
2. On the left navigation, select **Service connector**.
3. On the Service Connector page, choose **+ Create** to open the **Creation Connection** flyout panel and enter the following values:
  - **Container:** Select the container app created previously.
  - **Service type:** Choose **Key Vault**.
  - **Subscription:** Select the subscription previously used.
  - **Connection name:** Leave the default value.
  - **Key vault:** Select the key vault created previously.
  - **Client type:** Select **.NET**.
4. Select **Next: Authentication** to progress to the next step.
5. Select **System assigned managed identity** and choose **Next: Networking**.

6. Leave the default networking options selected, and then select **Review + Create**.

7. After Azure validates the settings, select **Create**.

The service connector assigns a role to the identity so it can perform data operations on the key vault keys.

## Configure and redeploy the app

The necessary Azure resources have been created. In this section the app code is configured to use the new resources.

1. Install the following NuGet packages:

- **Azure.Identity**: Provides classes to work with the Azure identity and access management services.
- **Microsoft.Extensions.Azure**: Provides helpful extension methods to perform core Azure configurations.
- **Azure.Extensions.AspNetCore.DataProtection.Blobs**: Allows storing ASP.NET Core DataProtection keys in Azure Blob Storage so that keys can be shared across several instances of a web app.
- **Azure.Extensions.AspNetCore.DataProtection.Keys**: Enables protecting keys at rest using the Azure Key Vault Key Encryption/Wrapping feature.

.NET CLI

```
dotnet add package Azure.Identity
dotnet add package Microsoft.Extensions.Azure
dotnet add package Azure.Extensions.AspNetCore.DataProtection.Blobs
dotnet add package Azure.Extensions.AspNetCore.DataProtection.Keys
```

2. Update `Program.cs` with the following highlighted code:

C#

```
using Azure.Identity;
using Microsoft.AspNetCore.DataProtection;
using Microsoft.Extensions.Azure;
var builder = WebApplication.CreateBuilder(args);
var BlobStorageUri = builder.Configuration["AzureURIs:BlobStorage"];
var KeyVaultURI = builder.Configuration["AzureURIs:KeyVault"];

builder.Services.AddRazorPages();
builder.Services.AddHttpClient();
builder.Services.AddServerSideBlazor();
```

```

builder.Services.AddAzureClientsCore();

builder.Services.AddDataProtection()
 .PersistKeysToAzureBlobStorage(new Uri(BlobStorageUri),
 new
DefaultAzureCredential())
 .ProtectKeysWithAzureKeyVault(new Uri(KeyVaultURI),
 new
DefaultAzureCredential());
var app = builder.Build();

if (!app.Environment.IsDevelopment())
{
 app.UseExceptionHandler("/Error");
 app.UseHsts();
}

app.UseHttpsRedirection();
app.UseStaticFiles();

app.UseRouting();

app.UseAuthorization();

app.MapRazorPages();

app.Run();

```

The preceding changes allow the app to manage data protection using a centralized, scalable architecture. `DefaultAzureCredential` discovers the managed identity configurations enabled earlier when the app is redeployed.

Update the placeholders in `AzureURIs` section of the `appsettings.json` file to include the following:

1. Replace the `<storage-account-name>` placeholder with the name of the `scalablerazorstorageXXXX` storage account.
2. Replace the `<container-name>` placeholder with the name of the `scalablerazorkeys` storage container.
3. Replace the `<key-vault-name>` placeholder with the name of the `scalablerazorvaultXXXX` key vault.
4. Replace the `<key-name>` placeholder in the key vault URI with the `razorkey` name created previously.

C#

```
{
 "GitHubURL": "https://api.github.com",
 "Logging": {
 "LogLevel": {
 "Default": "Information",
 "Microsoft.AspNetCore": "Warning"
 }
 },
 "AllowedHosts": "*",
 "AzureURIs": {
 "BlobStorage": "https://<storage-account-name>.blob.core.windows.net/<container-name>/keys.xml",
 "KeyVault": "https://<key-vault-name>.vault.azure.net/keys/<key-name>/"
 }
}
```

## Redeploy the app

The app is now configured correctly to use the Azure services created previously. Redeploy the app for the code changes to be applied.

1. Right click on the project node in the solution explorer and select **Publish**.
2. On the publishing profile summary view, select the **Publish** button in the upper right corner.

Visual Studio redeploys the app to the container apps environment created previously. When the processes finished, the browser launches to the app home page.

Test the app again by searching for *Microsoft* in the search field. The page should now reload with the correct results every time you submit.

## Configure roles for local development

The existing code and configuration of the app can also work while running locally during development. The `DefaultAzureCredential` class configured previously is able to pick up local environment credentials to authenticate to Azure Services. You'll need to assign the same roles to your own account that were assigned to your app's managed identity in order for the authentication to work. This should be the same account you use to log into Visual Studio or the Azure CLI.

## Sign-in to your local development environment

You'll need to be signed in to the Azure CLI, Visual Studio, or Azure PowerShell for your credentials to be picked up by `DefaultAzureCredential`.



A screenshot of the Azure CLI interface. At the top left, it says "Azure CLI". Below that is a grey bar with "Azure CLI" again. In the main area, there is a blue button with the text "az login".

## Assign roles to your developer account

1. In the Azure portal, navigate to the `scalablerazor****` storage account created previously.
2. Select **Access Control (IAM)** from the left navigation.
3. Choose **+ Add** and then **Add role assignment** from the drop-down menu.
4. On the **Add role assignment** page, search for `Storage blob data contributor`, select the matching result, and then select **Next**.
5. Make sure **User, group, or service principal** is selected, and then select **+ Select members**.
6. In the **Select members** flyout, search for your own `user@domain` account and select it from the results.
7. Choose **Next** and then select **Review + assign**. After Azure validates the settings, select **Review + assign** again.

As previously stated, role assignment permissions might take a minute or two to propagate, or in rare cases up to eight minutes.

Repeat the previous steps to assign a role to your account so that it can access the key vault service and secret.

1. In the Azure portal, navigate to the `razorscalingkeys` key vault created previously.
2. Select **Access Control (IAM)** from the left navigation.
3. Choose **+ Add** and then **Add role assignment** from the drop-down menu.
4. On the **Add role assignment** page, search for `Key Vault Crypto Service Encryption User`, select the matching result, and then select **Next**.
5. Make sure **User, group, or service principal** is selected, and then select **+ Select members**.
6. In the **Select members** flyout, search for your own `user@domain` account and select it from the results.

7. Choose **Next** and then select **Review + assign**. After Azure validates your settings, select **Review + assign** again.

You may need to wait again for this role assignment to propagate.

You can then return to Visual Studio and run the app locally. The code should continue to function as expected. `DefaultAzureCredential` uses your existing credentials from Visual Studio or the Azure CLI.

## Reliable web app patterns

See *The Reliable Web App Pattern for .NET* [YouTube videos](#) and [article](#) for guidance on creating a modern, reliable, performant, testable, cost-efficient, and scalable ASP.NET Core app, whether from scratch or refactoring an existing app.

# Java on Azure Container Apps overview

Article • 07/16/2024

Azure Container Apps can run any containerized Java application in the cloud while giving flexible options for how you deploy your applications.

When you use Container Apps for your containerized Java applications, you get:

- **Cost effective scaling:** When you use the [Consumption plan](#), your Java apps can scale to zero. Scaling in when there's little demand for your app automatically drives costs down for your projects.
- **Deployment options:** Azure Container Apps integrates with [Buildpacks](#), which allows you to deploy directly from a Maven build, via artifact files, or with your own Dockerfile.
  - **JAR deployment:** You can deploy your container app directly from a [JAR file](#).
  - **WAR deployment:** You can deploy your container app directly from a [WAR file](#).
  - **IDE support:** You can deploy your container app directly from [IntelliJ](#).
- **Automatic memory fitting:** Container Apps optimizes how the Java Virtual Machine (JVM) [manages memory](#), making the most possible memory available to your Java applications.
- **Build environment variables:** You can configure [custom key-value pairs](#) to control the Java image build from source code.

This article details the information you need to know as you build Java applications on Azure Container Apps.

## Deployment types

Running containerized applications usually means you need to create a Dockerfile for your application, but running Java applications on Container Apps gives you a few options.

[+] [Expand table](#)

Type	Description	Uses Buildpacks	Uses a Dockerfile
Source code build	You can deploy directly to Container Apps from your source code.	Yes	No
Artifact build	You can create a Maven build to deploy to Container Apps	Yes	No
Dockerfile	You can create your Dockerfile manually and take full control over your deployment.	No	Yes

 **Note**

The Buildpacks deployments support JDK versions 8, 11, 17, and 21.

## Application types

Different applications types are implemented either as an individual container app or as a [Container Apps job](#). Use the following table to help you decide which application type is best for your scenario.

Examples listed in this table aren't meant to be exhaustive, but to help you best understand the intent of different application types.

 [Expand table](#)

Type	Examples	Implement as...
Web applications and API endpoints	Spring Boot, Quarkus, Apache Tomcat, and Jetty	An individual container app
Console applications, scheduled tasks, task runners, batch jobs	SparkJobs, ETL tasks, Spring Batch Job, Jenkins pipeline job	A Container Apps job

## Debugging

As you debug your Java application on Container Apps, be sure to inspect the Java [in-process agent](#) for log stream and console debugging messages.

## Troubleshooting

Keep the following items in mind as you develop your Java applications:

- **Default resources:** By default, an app has a half a CPU and 1 GB available.
- **Stateless processes:** As your container app scales in and out, new processes are created and shut down. Make sure to plan ahead so that you write data to shared storage such as databases and file system shares. Don't expect any files written directly to the container file system to be available to any other container.
- **Scale to zero is the default:** If you need to ensure one or more instances of your application are continuously running, make sure you define a [scale rule](#) to best meet your needs.
- **Unexpected behavior:** If your container app fails to build, start, or run, verify that the artifact path is set correctly in your container.
- **Buildpack support issues:** If your Buildpack doesn't support dependencies or the version of Java you require, create your own Dockerfile to deploy your app. You can view a [sample Dockerfile](#) for reference.
- **SIGTERM and SIGINT signals:** By default, the JVM handles `SIGTERM` and `SIGINT` signals and doesn't pass them to the application unless you intercept these signals and handle them in your application accordingly. Container Apps uses both `SIGTERM` and `SIGINT` for process control. If you don't capture these signals, and your application terminates unexpectedly, you might lose these signals unless you persist them to storage.
- **Access to container images:** If you use artifact or source code deployment in combination with the default registry, you don't have direct access to your container images.

## Monitoring

All the [standard observability tools](#) work with your Java application. As you build your Java applications to run on Container Apps, keep in mind the following items:

- **Metrics:** Java Virtual Machine (JVM) metrics are critical for monitoring the health and performance of your Java applications. The data collected includes insights into memory usage, garbage collection, thread count of your JVM. You can check [metrics](#) to help ensure the health and stability of your applications.
- **Logging:** Send application and error messages to `stdout` or `stderr` so they can surface in the log stream. Avoid logging directly to the container's filesystem as is

common when using popular logging services.

- **Performance monitoring configuration:** Deploy performance monitoring services as a separate container in your Container Apps environment so it can directly access your application.

## Diagnostics

Azure Container Apps offers built-in diagnostics tools exclusively for Java developers. This support streamlines the debugging and troubleshooting of Java applications running on Azure Container Apps for enhanced efficiency and eases.

- **Dynamic logger level:** Allows you to access and check different level of log details without code modifications or forcing you to restart your app. You can view [Set dynamic logger level](#) for reference.

## Scaling

If you need to make sure requests from your front-end applications reach the same server, or your front-end app is split between multiple containers, make sure to enable [sticky sessions](#).

## Security

The Container Apps runtime terminates SSL for you inside your Container Apps environment.

## Memory management

To help optimize memory management in your Java application, you can ensure [JVM memory fitting](#) is enabled in your app.

Memory is measured in gibibytes (Gi) and CPU core pairs. The following table shows the range of resources available to your container app.

 Expand table

Threshold	CPU cores	Memory in Gibibytes (Gi)
Minimum	0.25	0.5

Threshold	CPU cores	Memory in Gibibytes (Gi)
Maximum	4	8

Cores are available in 0.25 core increments, with memory available at a 2:1 ratio. For instance, if you require 1.25 cores, you have 2.5 Gi of memory available to your container app.

### ⓘ Note

For apps using JDK versions 9 and lower, make sure to define custom JVM memory settings to match the memory allocation in Azure Container Apps.

## Spring components support

Azure Container Apps offers support for the following Spring Components as managed services:

- **Eureka Server for Spring:** Service registration and discovery are key requirements for maintaining a list of live application instances. Your application uses this list to for routing and load balancing inbound requests. Configuring each client manually takes time and introduces the possibility of human error. Eureka Server simplifies the management of service discovery by functioning as a [service registry](#) where microservices can register themselves and discover other services within the system.
- **Config Server for Spring:** Config Server provides centralized external configuration management for distributed systems. This component designed to address the challenges of [managing configuration settings across multiple microservices](#) in a cloud-native environment.
- **Admin for Spring:** The Admin for Spring managed component provides an administrative interface is designed for Spring Boot web applications that have actuator endpoints. A managed component provides integration and management to your container app by allowing you to bind your container app to the [Admin for Spring component](#).

## Next steps

[Launch your first Java app](#)

---

# Feedback

Was this page helpful?

 Yes

 No

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

# Turn on Java features in Azure Container Apps

Article • 10/14/2024

This guide provides step-by-step instructions for enabling key Java features in Azure Container Apps. By activating these features, you can optimize your Java applications for performance, monitoring, and ease of development.

## Java virtual machine (JVM) metrics

Java virtual machine (JVM) metrics are essential for tracking the performance and health of your Java applications. These metrics offer insights into memory consumption, garbage collection, and thread activity within the JVM. By enabling Java metrics in Azure Container Apps, you can access these detailed metrics in Azure Monitor to proactively optimize application performance and address potential issues.

To turn on Java virtual machine (JVM) metrics in the portal, refer to [Java metrics for Java apps in Azure Container Apps](#).

## Automatic memory fitting

By default, the JVM manages memory conservatively, but Java automatic memory fitting fine-tunes how memory is managed for your Java application. Automatic memory fitting makes more memory available to your Java app, which may potentially boost performance by 10-20% without requiring code changes.

Automatic memory fitting is [enabled by default](#), but you can disable manually.

Disabling automatic memory fitting is currently only available on CLI, please refer to [Disable memory fitting](#).

## Diagnostics

Azure Container Apps provides a built-in diagnostics tool designed specifically for Java developers, which makes debugging and troubleshooting easier and more efficient.

## Dynamic logger level

Enabling dynamic logger level is currently only available on CLI, refer to [Enable JVM diagnostics for your Java applications](#) for details.

## Java components

Azure Container Apps supports Java components as managed services, which allows you to extend the capability of your applications without having to deploy additional code.

### Eureka Server for Spring

Eureka Server for Spring is a service registry that allows microservices to register themselves and discover other services. Available as an Azure Container Apps component, you can bind your container app to a Eureka Server for Spring for automatic registration with the Eureka server.

To use Eureka Server for Spring on portal, refer to [Create the Eureka Server for Spring Java component on Portal](#).

### Config Server for Spring

Config Server for Spring provides a centralized location to make configuration data available to multiple applications.

To use Config Server for Spring on portal, refer to [Create the Config Server for Spring Java component on Portal](#).

### Admin for Spring

The Admin for Spring managed component offers an administrative interface for Spring Boot web applications that expose actuator endpoints.

To use Admin for Spring on portal, refer to [Use the component on Portal](#).

#### 💡 Tip

With Eureka Server for Spring, you can bind Admin for Spring to Eureka Server for Spring, so that it can get application information through Eureka, instead of having to bind individual applications to Admin for Spring. For more information, see [Integrate Admin for Spring with Eureka Server for Spring in Azure Container Apps](#).

# Next steps

[Launch your first Java app](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

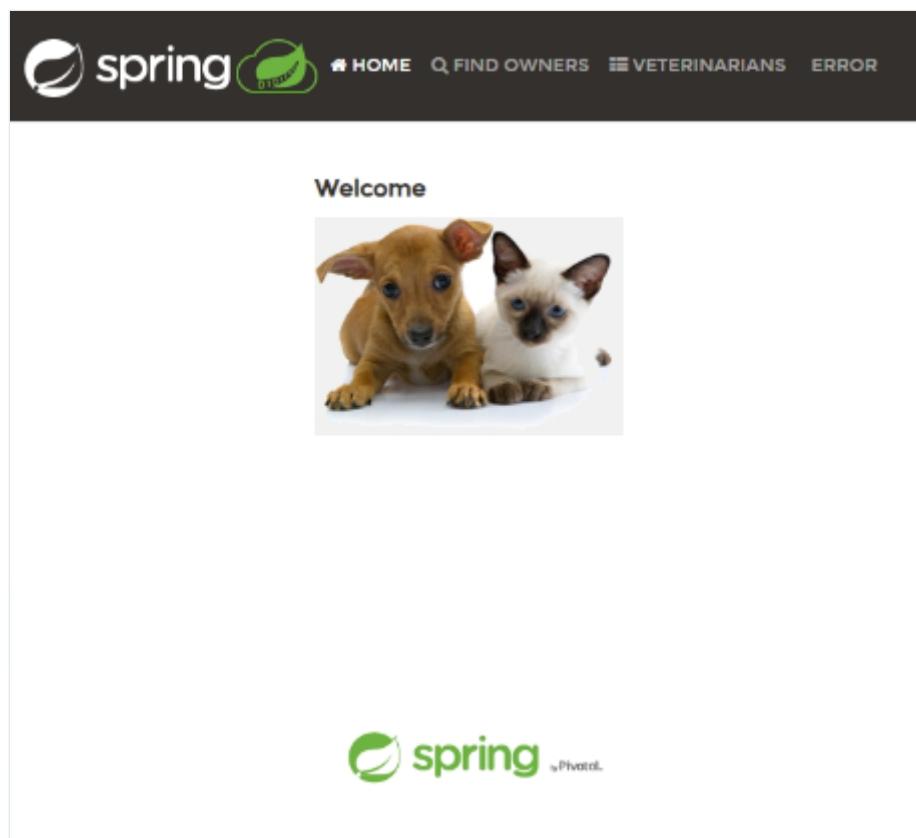
# Quickstart: Launch your first Java application in Azure Container Apps

Article • 06/24/2024

This article shows you how to deploy the Spring PetClinic sample application to run on Azure Container Apps. Rather than manually creating a Dockerfile and directly using a container registry, you can deploy your Java application directly from a Java Archive (JAR) file or a web application archive (WAR) file.

By the end of this tutorial you deploy a web application, which you can manage through the Azure portal.

The following image is a screenshot of how your application looks once deployed to Azure.



## Prerequisites

[ ] Expand table

Requirement	Instructions
Azure account	If you don't have one, <a href="#">create an account for free ↗</a> .

Requirement	Instructions
	You need the <i>Contributor</i> or <i>Owner</i> permission on the Azure subscription to proceed.  Refer to <a href="#">Assign Azure roles using the Azure portal</a> for details.
GitHub Account	Get one for <a href="#">free ↗</a> .
git	<a href="#">Install git ↗</a>
Azure CLI	Install the <a href="#">Azure CLI</a> .
Container Apps CLI extension	Use version <code>0.3.47</code> or higher. Use the <code>az extension add --name containerapp --upgrade --allow-preview</code> command to install the latest version.
Java	Install the <a href="#">Java Development Kit</a> . Use version 17 or later.
Maven	Install the <a href="#">Maven ↗</a> .

## Prepare the project

Clone the Spring PetClinic sample application to your machine.

Bash

```
git clone https://github.com/spring-petclinic/spring-framework-petclinic.git
```

## Build the project

 **Note**

If necessary, you can specify the Tomcat version in the [Java build environment variables](#).

Change into the *spring-framework-petclinic* folder.

Bash

```
cd spring-framework-petclinic
```

Clean the Maven build area, compile the project's code, and create a WAR file, all while skipping any tests.

Bash

```
mvn clean package -DskipTests
```

After you execute the build command, a file named *petclinic.war* is generated in the */target* folder.

## Deploy the project

Deploy the WAR package to Azure Container Apps.

Now you can deploy your WAR file with the `az containerapp up` CLI command.

Azure CLI

```
az containerapp up \
--name <CONTAINER_APP_NAME> \
--resource-group <RESOURCE_GROUP> \
--subscription <SUBSCRIPTION> \
--location <LOCATION> \
--environment <ENVIRONMENT_NAME> \
--artifact <WAR_FILE_PATH_AND_NAME> \
--build-env-vars BP_TOMCAT_VERSION=10.* \
--ingress external \
--target-port 8080 \
--query properties.configuration.ingress.fqdn
```

### ⓘ Note

The default Tomcat version is 9. If you need to change the Tomcat version for compatibility with your application, you can use the `--build-env-vars` `BP_TOMCAT_VERSION=<YOUR_TOMCAT_VERSION>` argument to adjust the version number.

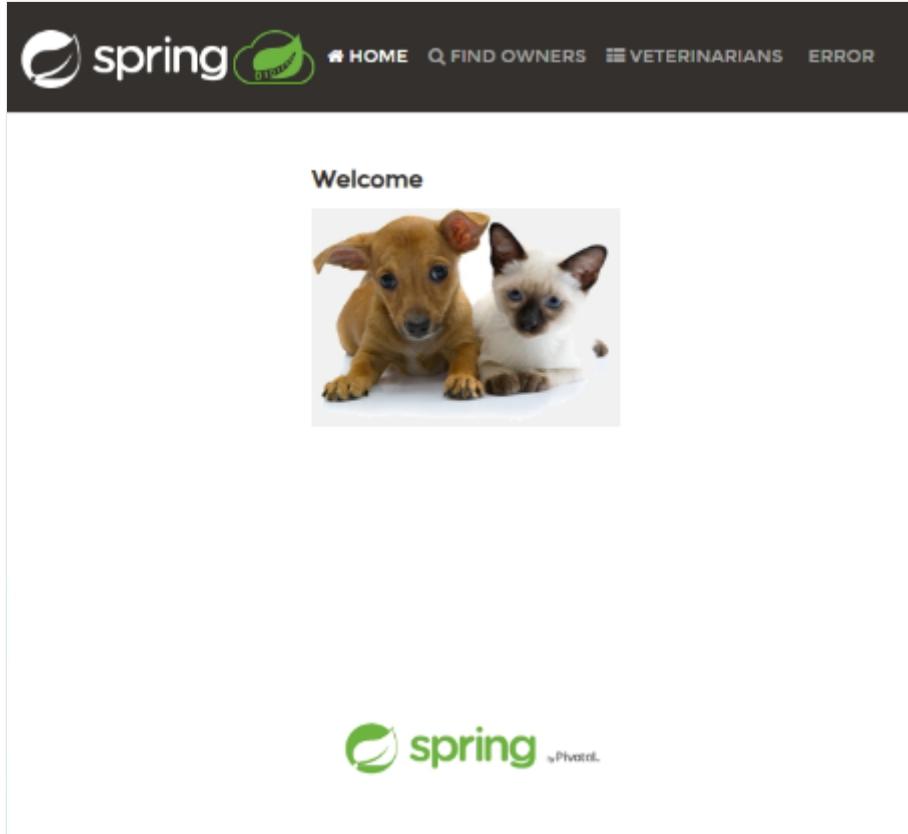
In this example, the Tomcat version is set to `10` (including any minor versions) by setting the `BP_TOMCAT_VERSION=10.*` environment variable.

You can find more applicable build environment variables in [Java build environment variables](#).

## Verify app status

In this example, `containerapp up` command includes the `--query properties.configuration.ingress.fqdn` argument, which returns the fully qualified domain name (FQDN), also known as the app's URL.

View the application by pasting this URL into a browser. Your app should resemble the following screenshot.



## Next steps

[Java build environment variables](#)

## Feedback

Was this page helpful?

[Yes](#)

[No](#)

[Provide product feedback ↗](#)

# Quickstart: Deploy to Azure Container Apps using IntelliJ IDEA

Article • 04/11/2024

This article shows you how to deploy a containerized application to Azure Container Apps using Azure Toolkit for IntelliJ IDEA. The article uses a sample backend web API service that returns a static collection of music albums.

## Prerequisites

- An Azure account with an active subscription. If you don't have a subscription, create a [free account](#) before you begin.
- A supported Java Development Kit (JDK). For more information about the JDKs available for use when developing on Azure, see [Java support on Azure and Azure Stack](#).
- [IntelliJ IDEA](#), Ultimate or Community Edition.
- [Maven 3.5.0+](#).
- A [Docker](#) client.
- The [Azure Toolkit for IntelliJ](#). For more information, see [Install the Azure Toolkit for IntelliJ](#). You also need to sign in to your Azure account for the Azure Toolkit for IntelliJ. For more information, see [Sign-in instructions for the Azure Toolkit for IntelliJ](#).

## Clone the project

1. Use the following commands to clone the sample app and check out the `IDE` branch:

```
git
git clone https://github.com/Azure-Samples/containerapps-albumapi-java
cd containerapps-albumapi-java
git checkout IDE
```

2. Select **Open** to open the project in IntelliJ IDEA.

## Build and run the project locally

1. Use the following command to build the project with [Maven](#):

```
Azure CLI
```

```
mvn clean package -DskipTests
```

2. To verify that the application is running, open a browser and go to <http://localhost:8080/albums>. The page returns a list of JSON objects similar to the output of the following command:

```
Azure CLI
```

```
java -jar target\containerapps-albumapi-java-0.0.1-SNAPSHOT.jar
```

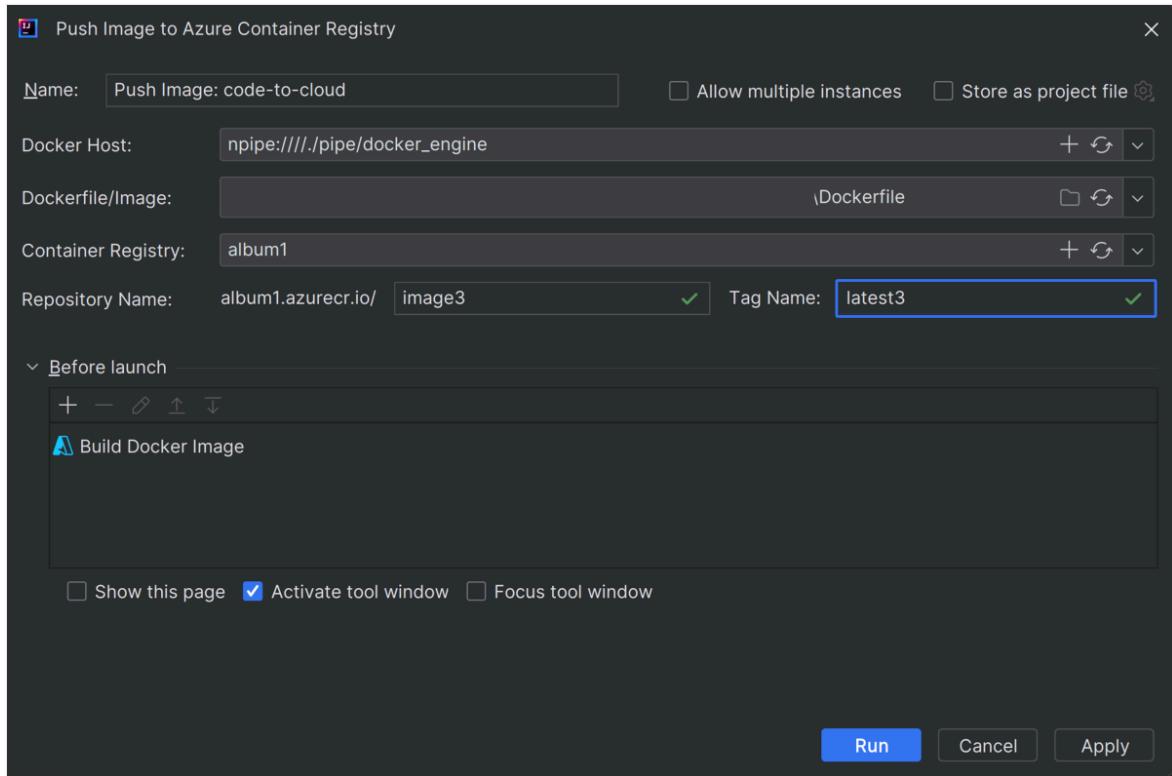
## Push image to an Azure Container Registry

To deploy your project to Azure Container Apps, you need to build the container image and push it to an Azure Container Registry first.

1. In **Azure Explorer** view, expand the **Azure node**, right-click **Container Registries**, and then select **Create in Azure Portal**.
2. On the **Create container registry** page, enter the following information:
  - **Subscription**: Specifies the Azure subscription that you want to use for your container registry.
  - **Resource Group**: Specifies the resource group for your container registry.  
Select one of the following options:
    - **Create New**: Specifies that you want to create a new resource group.
    - **Use Existing**: Specifies that you must select from a list of resource groups that are associated with your Azure account.
  - **Registry Name**: Specifies a name for the new container registry.
  - **Location**: Specifies the region where your container registry is created (for example, **West US**).
  - **SKU**: Specifies the service tier for your container registry. For this tutorial, select **Basic**.
3. Select **Review + create** and verify that the information is correct. Then, select **Create**.
4. On the **Project** tab, navigate to your project and open **Dockerfile**.
5. Select the Azure icon and then select **Push Image to Azure Container Registry**.

6. Select the registry you created in the previous step, fill in the following information, and then select **Run**.

- **Repository Name:** Specifies the name for the repository.
- **Tag Name:** Specifies the version of an image or other artifact.

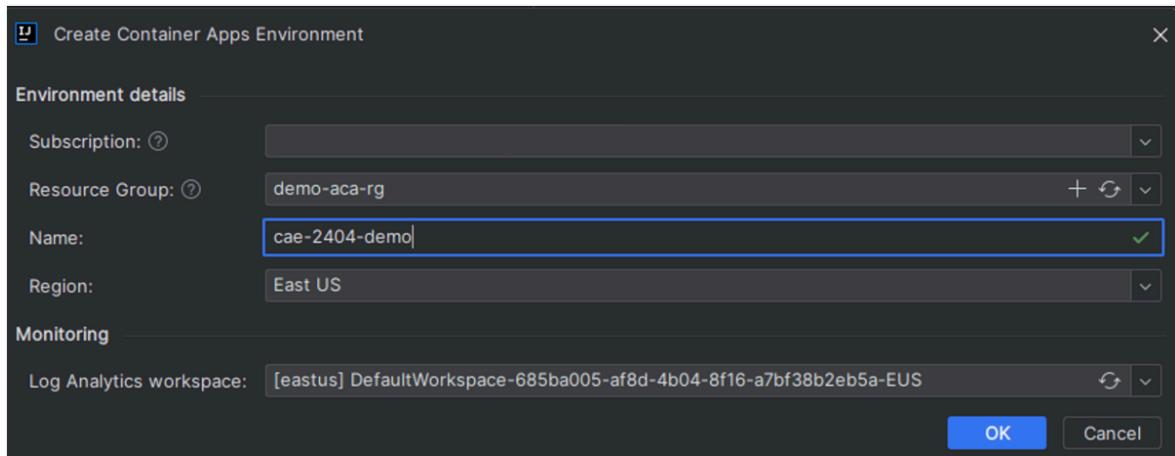


## Create an environment and a container app

Use the following steps to set up your environment and deploy a container app in Azure:

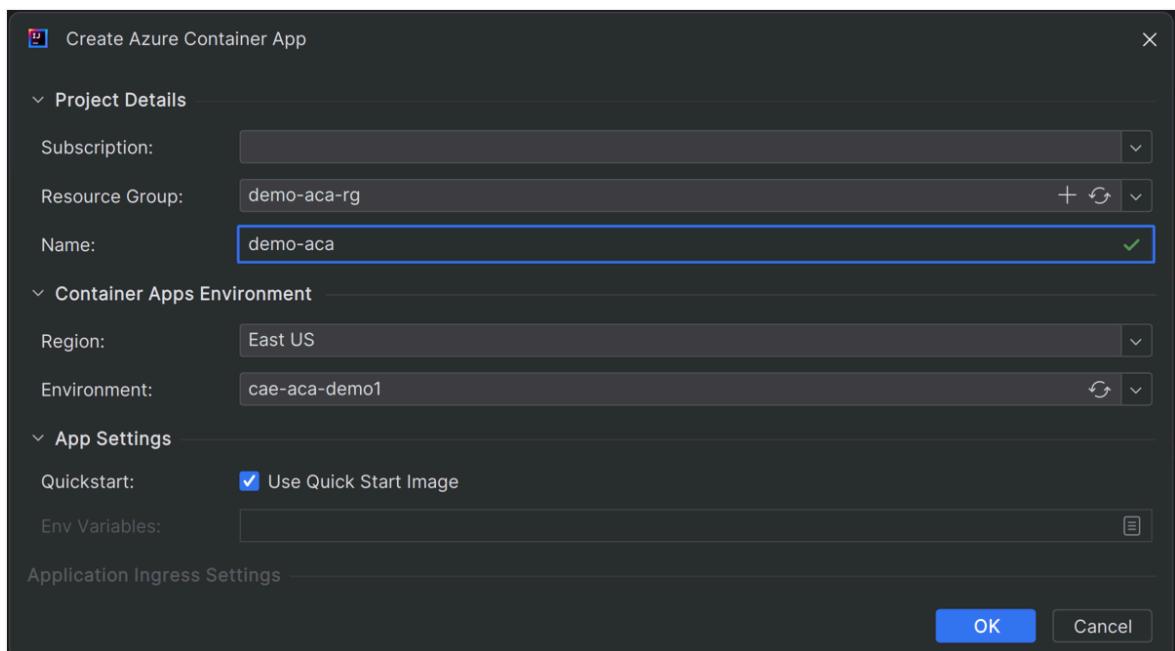
1. Right-click **Container Apps Environment** in **Azure Explorer** view, and then select **Create Container Apps Environment**.
2. On the **Create Container Apps Environment** page, enter the following information, and then select **OK**.
  - **Subscription:** Specifies the Azure subscription that you want to use.
  - **Resource Group:** Specifies the resource group for your container apps. Select one of the following options:
    - **Create New:** Specifies that you want to create a new resource group.
    - **Use Existing:** Specifies that you must select from a list of resource groups that are associated with your Azure account.
  - **Name:** Specifies the name for the new container apps environment.
  - **Region:** Specifies the appropriate region (for example, **East US**).

- **Log Analytics workspace:** Specifies the Log Analytics workspace you want to use or accept the default.



3. Right-click on the container apps environment you created and select **Create > Container App** in Azure Explorer. Enter the following information:

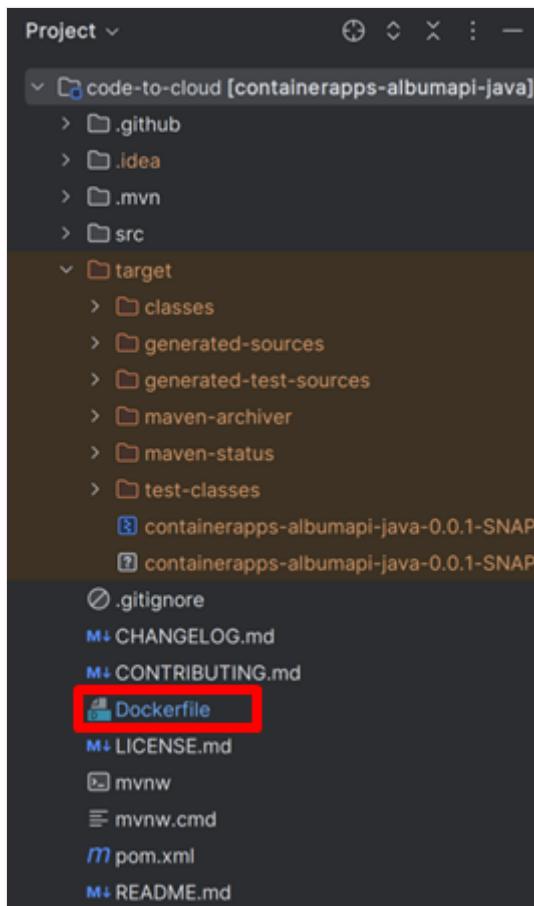
- **Subscription:** Specifies the Azure subscription that you want to use.
- **Resource Group:** Specifies the resource group for your container apps. Select one of the following options:
  - **Create New:** Specifies that you want to create a new resource group.
  - **Use Existing:** Specifies that you must select from a list of resource groups that are associated with your Azure account.
- **Name:** Specifies the name for a new container app.
- **Region:** Specifies the appropriate region (for example, **East US**).
- **Environment:** Specifies the Container Apps Environment you want to use.
- **Quickstart:** Select **Use Quick Start Image**.



4. Select **OK**. The toolkit displays a status message when the app creation succeeds.

# Deploy the container app

1. On the Project tab, navigate to your project and open Dockerfile.



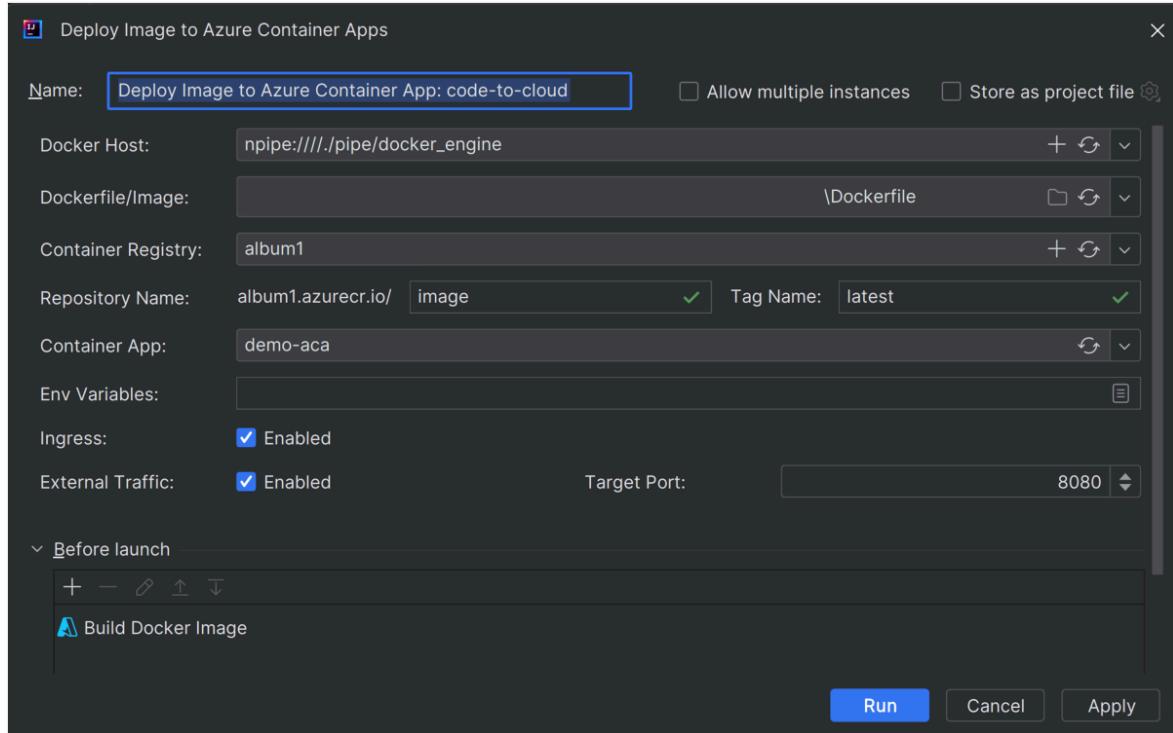
2. Select the Azure icon and then select Deploy Image to Container App.



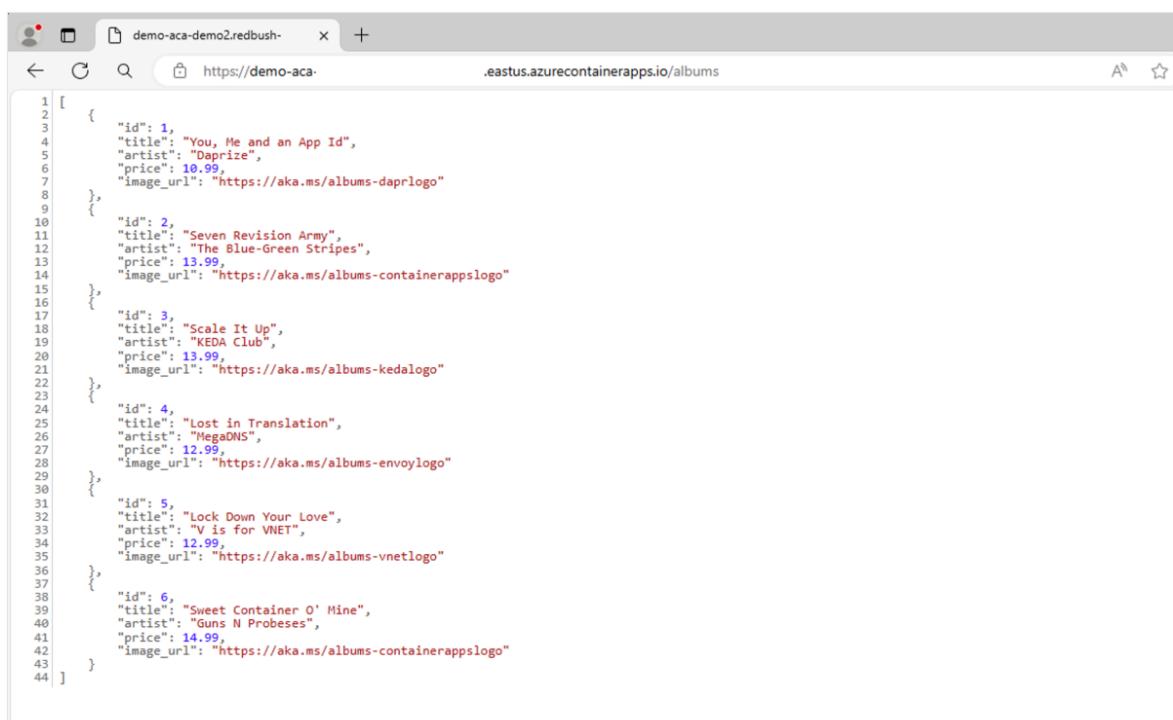
3. On the Deploy Image to Azure Container Apps page, enter the following information, and then select Run.

- **Dockerfile/Image:** Specifies the path of the Dockerfile or accept the default.
- **Container Registry:** Specifies the Container Registry you want to use.
- **Repository Name:** Specifies the repository name you want to use under your Container Registry.
- **Tag Name:** Specifies the tag name you want to use under your Container Registry.
- **Container App:** Specifies the Container App you want to deploy to.

- **Ingress:** Enable ingress for applications that require an HTTP or TCP endpoint. Select **Enable**.
- **External Traffic:** Enable external traffic for applications that need an HTTP or TCP endpoint. Select **Enable**.
- **Target Port:** Set this value to the port number that your container uses. Open port 8080 in this step.



4. After the deployment finishes, the Azure Toolkit for IntelliJ displays a notification. Select **Browse** to open the deployed app in a browser.



In the browser's address bar, append the `/albums` path to the end of the app URL to view data from a sample API request.

## Clean up resources

If you want to clean up and remove an Azure Container Apps resource, you can delete the resource or resource group. Deleting the resource group also deletes any other resources associated with it. Use the following steps to clean up resources:

1. To delete your Azure Container Apps resources, navigate to the left-hand **Azure Explorer** sidebar and locate the **Container Apps Environment** item.
2. Right-click on the Azure Container Apps service you'd like to delete and then select **Delete**.
3. To delete your resource group, visit the [Azure portal](#) and manually delete the resources under your subscription.

## Next steps

- [Java on Azure Container Apps overview](#)
- 

## Feedback

Was this page helpful?



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

# Use memory efficiently for Java apps in Azure Container Apps (preview)

Article • 03/03/2024

The Java Virtual Machine (JVM) uses memory conservatively as it assumes OS memory must be shared among multiple applications. However, your container app can optimize memory usage and make the maximum amount of memory possible available to your application. This memory optimization is known as Java automatic memory fitting. When memory fitting is enabled, Java application performance is typically improved between 10% and 20% without any code changes.

Azure Container Apps provides automatic memory fitting under the following circumstances:

- A single Java application is running in a container.
- Your application is deployed from source code or a JAR file.

Automatic memory fitting is enabled by default, but you can disable manually.

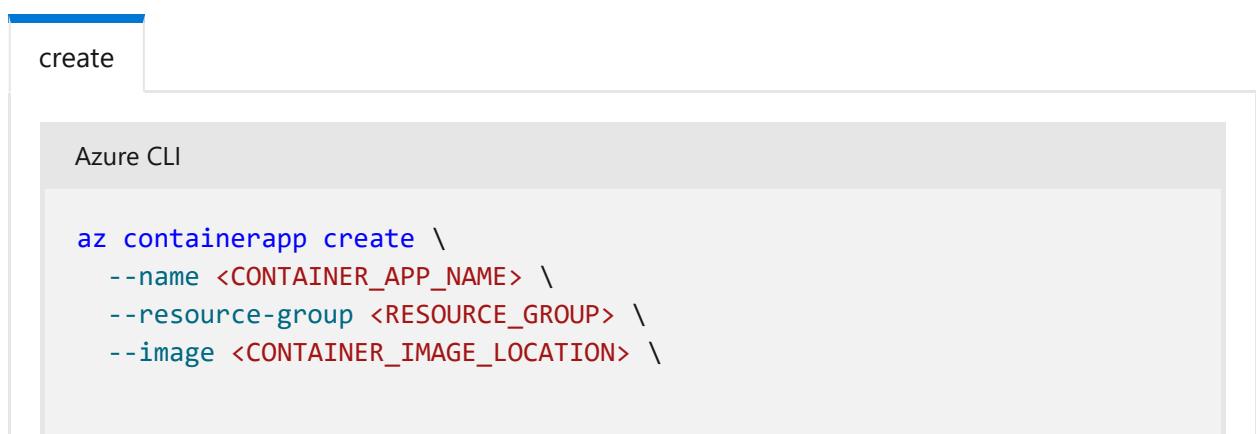
## Disable memory fitting

Automatic memory fitting is helpful in most scenarios, but it might not be ideal for all situations. You can disable memory fitting manually or automatically.

### Manual disable

To disable memory fitting when you create your container app, set the environment variable `BP_JVM_FIT` to `false`.

The following examples show you how to use disable memory fitting with the `create`, `up`, and `update` commands.



```
create

Azure CLI

az containerapp create \
--name <CONTAINER_APP_NAME> \
--resource-group <RESOURCE_GROUP> \
--image <CONTAINER_IMAGE_LOCATION> \
```

```
--environment <ENVIRONMENT_NAME> \
--env-vars BP_JVM_FIT="false"
```

To verify that memory fitting is disabled, check your logs for the following message:

Disabling jvm memory fitting, reason: manually disabled

## Automatic disable

Memory fitting is automatically disabled when any of the following conditions are met:

- **Limited container memory:** Container memory is less than 1 GB.
- **Explicitly set memory options:** When one or more memory settings are specified in environment variables through `JAVA_TOOL_OPTIONS`. Memory setting options include the following values:
  - `-XX:MaxRAMPercentage`
  - `-XX:MinRAMPercentage`
  - `-XX:InitialRAMPercentage`
  - `-XX:MaxMetaspaceSize`
  - `-XX:MetaspaceSize`
  - `-XX:ReservedCodeCacheSize`
  - `-XX:MaxDirectMemorySize`
  - `-Xmx`
  - `-Xms`
  - `-Xss`

For example, memory fitting is automatically disabled if you specify the maximum heap size in an environment variable like shown the following example:

Azure CLI

```
az containerapp update \
--name <CONTAINER_APP_NAME> \
--resource-group <RESOURCE_GROUP> \
--image <CONTAINER_IMAGE_LOCATION> \
--set-env-vars JAVA_TOOL_OPTIONS="-Xmx512m"
```

With memory fitting disabled, you see the following message output to the log:

Disabling jvm memory fitting, reason: use settings specified in  
`JAVA_TOOL_OPTIONS=-Xmx512m` instead Picked up `JAVA_TOOL_OPTIONS: -`

Xmx512m

- **Small non-heap memory size:** Rare cases when the calculated size of heap or nonheap size is too small (less than 200 MB).

## Verify memory fit is enabled

Inspect your [log stream](#) during start-up for a message that references *Calculated JVM Memory Configuration*.

Here's an example message output during start-up.

```
Calculated JVM Memory Configuration: -XX:MaxDirectMemorySize=10M -
Xmx1498277K -XX:MaxMetaspaceSize=86874K -XX:ReservedCodeCacheSize=240M
-Xss1M (Total Memory: 2G, Thread Count: 250, Loaded Class Count: 12924,
Headroom: 0%)
```

```
Picked up JAVA_TOOL_OPTIONS: -XX:MaxDirectMemorySize=10M -Xmx1498277K -
XX:MaxMetaspaceSize=86874K -XX:ReservedCodeCacheSize=240M -Xss1M
```

## Runtime configuration

You can set environment variables to affect the memory fitting behavior.

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

Variable	Unit	Example	Description
BPL_JVM_HEAD_ROOM	Percentage	BPL_JVM_HEAD_ROOM=5	Leave memory space for the system based on the given percentage.
BPL_JVM_THREAD_COUNT	Number	BPL_JVM_THREAD_COUNT=200	The estimated maximum number of threads.
BPL_JVM_CLASS_ADJUSTMENT	Number Percentage	BPL_JVM_CLASS_ADJUSTMENT=10000 BPL_JVM_CLASS_ADJUSTMENT="10%"	Adjust JVM class count by explicit value or percentage.

### Note

Changing these variables does not disable automatic memory fitting.

## Out of memory warning

If you decide to configure the memory settings yourself, you run the risk of encountering an out-of-memory warning.

Here are some possible reasons of why your container could run out of memory:

- Heap memory is greater than the total available memory.
- Nonheap memory is greater than the total available memory.
- Heap + nonheap memory is greater than the total available memory.

If your container runs out of memory, then you encounter the following warning:

OOM Warning: heap memory 1200M is greater than 1G available for allocation (-Xmx1200M)

## Next steps

[Monitor logs with Log Analytics](#)

## Feedback

Was this page helpful?

 Yes

 No

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

# Build environment variables for Java in Azure Container Apps

Article • 03/03/2024

Azure Container Apps uses [Buildpacks](#) to automatically create a container image that allows you to deploy from your source code directly to the cloud. To take control of your build configuration, you can use environment variables to customize parts of your build like the JDK, Maven, and Tomcat. The following article shows you how to configure environment variables to help you take control over builds that automatically create a container for you.

## Supported Java build environment variables

### Configure JDK

Container Apps use [Microsoft Build of OpenJDK](#) to build source code and as the runtime environment. Four LTS JDK versions are supported: 8, 11, 17 and 21.

- For source code build, the default version is JDK 17.
- For a JAR file build, the JDK version is read from the file location `META-INF\MANIFEST.MF` in the JAR, but uses the default JDK version 17 if the specified version isn't available.

Here's a listing of the environment variables used to configure JDK:

[+] [Expand table](#)

Environment variable	Description	Default
<code>BP_JVM_VERSION</code>	Controls the JVM version.	17

### Configure Maven

Container Apps supports building Maven-based applications from source.

Here's a listing of the environment variables used to configure Maven:

[+] [Expand table](#)

Build environment variable	Description	Default
<code>BP_MAVEN_VERSION</code>	Sets the major Maven version. Since Buildpacks only ships a single version of each supported line, updates to the buildpack can change the exact version of Maven installed. If you require a	3

Build environment variable	Description	Default
	specific minor/patch version of Maven, use the Maven wrapper instead.	
BP_MAVEN_BUILD_ARGUMENTS	Defines the arguments passed to Maven. The <code>--batch-mode</code> is prepended to the argument list in environments without a TTY.	<code>-Dmaven.test.skip=true</code> <code>--no-transfer-progress</code> <code>package</code>
BP_MAVEN_ADDITIONAL_BUILD_ARGUMENTS	Defines extra arguments used (for example, <code>-DskipJavadoc</code> appended to <code>BP_MAVEN_BUILD_ARGUMENTS</code> ) to pass to Maven.	
BP_MAVEN_ACTIVE_PROFILES	Comma separated list of active profiles passed to Maven.	
BP_MAVEN_BUILT_MODULE	Designates application artifact that contains the module. By default, the build looks in the root module.	
BP_MAVEN_BUILT_ARTIFACT	Location of the built application artifact. This value supersedes the <code>BP_MAVEN_BUILT_MODULE</code> variable. You can match a single file, multiple files, or a directory through one or more space separated patterns.	<code>target/*.[ejw]ar</code>
BP_MAVEN_POM_FILE	Specifies a custom location to the project's <code>pom.xml</code> file. This value is relative to the root of the project (for example, <code>/workspace</code> ).	<code>pom.xml</code>
BP_MAVEN_DAEMON_ENABLED	Triggers the installation and configuration of Apache <code>maven-mvnd</code> instead of Maven. Set this value to <code>true</code> if you want to the Maven Daemon.	<code>false</code>
BP_MAVEN_SETTINGS_PATH	Specifies a custom location to Maven's <code>settings.xml</code> file.	
BP_INCLUDE_FILES	Colon separated list of glob patterns to match source files. Any matched file is retained in the final image.	
BP_EXCLUDE_FILES	Colon separated list of glob patterns to match source files. Any matched file is removed from the final image. Any include patterns are applied first, and you can use "exclude patterns" to reduce the files included in the build.	
BP_JAVA_INSTALL_NODE	Control whether or not a separate Buildpack installs Yarn and Node.js. If set to <code>true</code> , the Buildpack checks the app	<code>false</code>

Build environment variable	Description	Default
	root or path set by <code>BP_NODE_PROJECT_PATH</code> . The project path looks for either a <code>yarn.lock</code> file, which requires the installation of Yarn and Node.js. If there's a <code>package.json</code> file, then the build only requires Node.js.	
<code>BP_NODE_PROJECT_PATH</code>	Direct the project subdirectory to look for <code>package.json</code> and <code>yarn.lock</code> files.	

## Configure Tomcat

Container Apps supports running war file in Tomcat application server.

Here's a listing of the environment variables used to configure Tomcat:

[Expand table](#)

Build environment variable	Description	Default
<code>BP_TOMCAT_CONTEXT_PATH</code>	The context path where the application is mounted.	Defaults to empty ( <code>ROOT</code> )
<code>BP_TOMCAT_EXT_CONF_SHA256</code>	The SHA256 hash of the external configuration package.	
<code>BP_TOMCAT_ENV_PROPERTY_SOURCE_DISABLED</code>	When set to <code>true</code> , the Buildpack doesn't configure <code>org.apache.tomcat.util.digester.EnvironmentPropertySource</code> . This configuration option is added to support loading configuration from environment variables and referencing them in Tomcat configuration files.	
<code>BP_TOMCAT_EXT_CONF_STRIP</code>	The number of directory levels to strip from the external configuration package. <code>0</code>	
<code>BP_TOMCAT_EXT_CONF_URI</code>	The download URI of the external configuration package.	
<code>BP_TOMCAT_EXT_CONF_VERSION</code>	The version of the external configuration package.	
<code>BP_TOMCAT_VERSION</code>	Used to configure a specific Tomcat version. Supported Tomcat versions include 8, 9, and 10. <code>9.*</code>	

## Configure Cloud Build Service

Here's a listing of the environment variables used to configure a Cloud Build Service:

[Expand table](#)

Build environment variable	Description	Default
ORYX_DISABLE_TELEMETRY	Controls whether or not to disable telemetry collection.	false

## How to configure Java build environment variables

ⓘ Note

To run the following CLI commands, use Container Apps extension version `0.3.47` or higher.

Use the `az extension add --name containerapp --upgrade --allow-preview` command to install the latest version.

You can configure Java build environment variables when you deploy Java application source code via CLI command `az containerapp up`, `az containerapp create`, or `az containerapp update`:

Azure CLI

```
az containerapp up \
--name <CONTAINER_APP_NAME> \
--source <SOURCE_DIRECTORY> \
--build-env-vars <NAME=VALUE NAME=VALUE> \
--resource-group <RESOURCE_GROUP_NAME> \
--environment <ENVIRONMENT_NAME>
```

The `build-env-vars` argument is a list of environment variables for the build, space-separated values in `key=value` format. Here's an example list you can pass in as variables:

Bash

```
BP_JVM_VERSION=21 BP_MAVEN_VERSION=4 "BP_MAVEN_BUILD_ARGUMENTS=-
Dmaven.test.skip=true --no-transfer-progress package"
```

You can also configure the Java build environment variables when you [set up GitHub Actions with Azure CLI in Azure Container Apps](#).

Azure CLI

```
az containerapp github-action add \
--repo-url "https://github.com/<OWNER>/<REPOSITORY_NAME>" \
--build-env-vars <NAME=VALUE NAME=VALUE> \
--branch <BRANCH_NAME> \
--name <CONTAINER_APP_NAME> \
--resource-group <RESOURCE_GROUP> \
--registry-url <URL_TO_CONTAINER_REGISTRY> \
--registry-username <REGISTRY_USER_NAME> \
--registry-password <REGISTRY_PASSWORD> \
--service-principal-client-id <appId> \
--service-principal-client-secret <password> \
```

```
--service-principal-tenant-id <tenant> \
--token <YOUR_GITHUB_PERSONAL_ACCESS_TOKEN>
```

## Next steps

[Build and deploy from a repository](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#)

# Java metrics for Java apps in Azure Container Apps

Article • 09/16/2024

Java Virtual Machine (JVM) metrics are critical for monitoring the health and performance of your Java applications. The data collected includes insights into memory usage, garbage collection, thread count of your JVM. Use the following metrics to help ensure the health and stability of your applications.

## Collected metrics

[Expand table](#)

Category	Description	Metric ID	Unit	
Java	jvm.memory.total.used	Total amount of memory used by heap or nonheap	JvmMemoryTotalUsed	bytes
Java	jvm.memory.total.committed	Total amount of memory guaranteed to be available for heap or nonheap	JvmMemoryTotalCommitted	bytes
Java	jvm.memory.total.limit	Total amount of maximum obtainable memory for heap or nonheap	JvmMemoryTotalLimit	bytes
Java	jvm.memory.used	Amount of memory used by each pool	JvmMemoryUsed	bytes

Category	Title	Description	Metric ID	Unit
Java	jvm.memory.committed	Amount of memory guaranteed to be available for each pool	JvmMemoryCommitted	bytes
Java	jvm.memory.limit	Amount of maximum obtainable memory for each pool	JvmMemoryLimit	bytes
Java	jvm.buffer.memory.usage	Amount of memory used by buffers, such as direct memory	JvmBufferMemoryUsage	bytes
Java	jvm.buffer.memory.limit	Amount of total memory capacity of buffers	JvmBufferMemoryLimit	bytes
Java	jvm.buffer.count	Number of buffers in the memory pool	JvmBufferCount	n/a
Java	jvm.gc.count	Count of JVM garbage collection actions	JvmGcCount	n/a
Java	jvm.gc.duration	Duration of JVM garbage collection actions	JvmGcDuration	milliseconds
Java	jvm.thread.count	Number of executing platform threads	JvmThreadCount	n/a

# Configuration

To make the collection of Java metrics available to your app, configure your container app with some specific settings.

create

In the *Create* window, if you select for *Deployment source* the **Container image** option, then you have access to stack-specific features.

Under the *Development stack-specific features* and for the *Development stack*, select **Java**.

The screenshot shows the 'Create Container App' interface. At the top, there's a breadcrumb navigation: Home > Container Apps > Create Container App. Below that, there's a 'Command override' input field with placeholder text 'Example: /bin/bash, -c, echo hello; sleep 100000'. Under 'Development stack-specific features', there's a section for 'Development stack' with a dropdown menu. The 'Generic' option is currently selected. A red box highlights the 'Java' option in the dropdown list, which has a magnifying glass icon next to it. Below this, there's a 'Container resource allocation' section with a note about choosing a workload profile and a 'more' link.

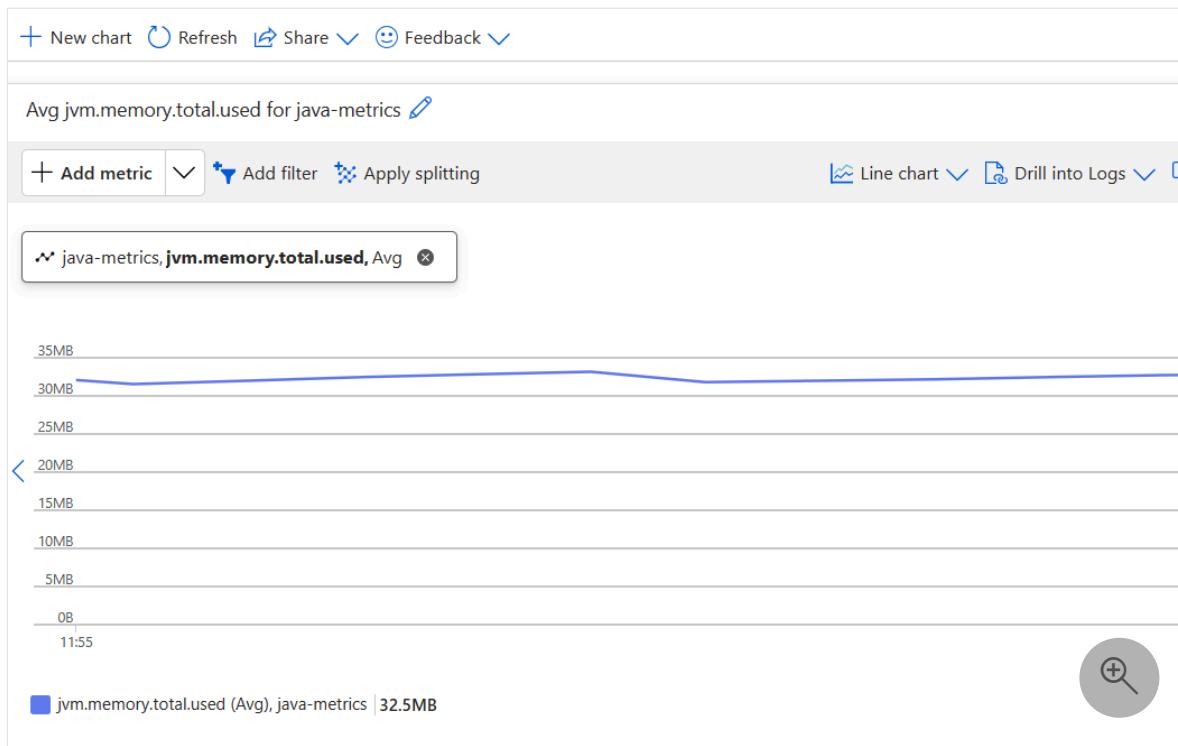
Once you select the Java development stack, the *Customize Java features for your app* window appears. Next to the *Java features* label, select **JVM core metrics**.

## View Java Metrics

Use the following steps to view metrics visualizations for your container app.

1. Go to the Azure portal.
2. Go to your container app.
3. Under the *Monitoring* section, select **Metrics**.

From there, you're presented with a chart that plots the metrics you're tracking in your application.



You can see Java metric names on Azure Monitor, but the data sets show as empty unless the feature is enabled. Refer to the [Configuration](#) section for how to enable it.

## Next steps

[Monitor logs with Log Analytics](#)

## Feedback

Was this page helpful?

[Yes](#)

[No](#)

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

# Configure settings for the Eureka Server for Spring component in Azure Container Apps (preview)

Article • 05/24/2024

Eureka Server for Spring is mechanism for centralized service discovery for microservices. Use the following guidance to learn how to configure and manage your Eureka Server for Spring component.

## Show

You can view the details of an individual component by name using the `show` command.

Before you run the following command, replace placeholders surrounded by `<>` with your values.

Azure CLI

```
az containerapp env java-component eureka-server-for-spring show \
--environment <ENVIRONMENT_NAME> \
--resource-group <RESOURCE_GROUP> \
--name <JAVA_COMPONENT_NAME>
```

## List

You can list all registered Java components using the `list` command.

Before you run the following command, replace placeholders surrounded by `<>` with your values.

Azure CLI

```
az containerapp env java-component list \
--environment <ENVIRONMENT_NAME> \
--resource-group <RESOURCE_GROUP>
```

## Unbind

To remove a binding from a container app, use the `--unbind` option.

Before you run the following command, replace placeholders surrounded by <> with your values.

```
Azure CLI

az containerapp update \
--name <APP_NAME> \
--unbind <JAVA_COMPONENT_NAME> \
--resource-group <RESOURCE_GROUP>
```

## Allowed configuration list for your Eureka Server for Spring

The following list details supported configurations. You can find more details in [Spring Cloud Eureka Server](#).

 **Note**

Please submit support tickets for new feature requests.

## Configuration options

The `az containerapp update` command uses the `--configuration` parameter to control how the Eureka Server for Spring is configured. You can use multiple parameters at once as long as they're separated by a space. You can find more details in [Spring Cloud Eureka Server](#) docs.

The following configuration settings are available on the `eureka.server` configuration property.

 Expand table

Name	Description	Default value
<code>eureka.server.enable-self-preservation</code>	When enabled, the server keeps track of the number of renewals it should receive from the server. Any time, the number of renewals drops below the threshold percentage as defined by <code>eureka.server.renewal-percent-threshold</code> . The default value is set to <code>true</code> in the original Eureka server, but in the Eureka Server Java component,	false

Name	Description	Default value
	the default value is set to <code>false</code> . See <a href="#">Limitations of Eureka Server for Spring Java component</a>	
<code>eureka.server.renewal-percent-threshold</code>	The minimum percentage of renewals that is expected from the clients in the period specified by <code>eureka.server.renewal-threshold-update-interval-ms</code> . If the renewals drop below the threshold, the expirations are disabled if the <code>eureka.server.enable-self-preservation</code> is enabled.	0.85
<code>eureka.server.renewal-threshold-update-interval-ms</code>	The interval with which the threshold as specified in <code>eureka.server.renewal-percent-threshold</code> needs to be updated.	0
<code>eureka.server.expected-client-renewal-interval-seconds</code>	The interval with which clients are expected to send their heartbeats. Defaults to 30 seconds. If clients send heartbeats with different frequency, say, every 15 seconds, then this parameter should be tuned accordingly, otherwise, self-preservation won't work as expected.	30
<code>eureka.server.response-cache-auto-expiration-in-seconds</code>	Gets the time for which the registry payload should be kept in the cache if it is not invalidated by change events.	180
<code>eureka.server.response-cache-update-interval-ms</code>	Gets the time interval with which the payload cache of the client should be updated.	0
<code>eureka.server.use-read-only-response-cache</code>	The <code>com.netflix.eureka.registry.ResponseCache</code> currently uses a two level caching strategy to responses. A <code>readWrite</code> cache with an expiration policy, and a <code>readonly</code> cache that caches without expiry.	true
<code>eureka.server.disable-delta</code>	Checks to see if the delta information can be served to client or not.	false
<code>eureka.server.retention-time-in-m-s-in-delta-queue</code>	Get the time for which the delta information should be cached for the clients to retrieve the value without missing it.	0
<code>eureka.server.delta-retention-timer-interval-in-ms</code>	Get the time interval with which the clean up task should wake up and check for expired delta information.	0
<code>eureka.server.eviction-interval-timer-in-ms</code>	Get the time interval with which the task that expires instances should wake up and run.	60000

Name	Description	Default value
<code>eureka.server.sync-when-timestamp-differs</code>	Checks whether to synchronize instances when timestamp differs.	true
<code>eureka.server.rate-limiter-enabled</code>	Indicates whether the rate limiter should be enabled or disabled.	false
<code>eureka.server.rate-limiter-burst-size</code>	Rate limiter, token bucket algorithm property.	10
<code>eureka.server.rate-limiter-registry-fetch-average-rate</code>	Rate limiter, token bucket algorithm property. Specifies the average enforced request rate.	500
<code>eureka.server.rate-limiter-privileged-clients</code>	A list of certified clients. This is in addition to standard eureka Java clients.	N/A
<code>eureka.server.rate-limiter-throttle-standard-clients</code>	Indicate if rate limit standard clients. If set to false, only non standard clients will be rate limited.	false
<code>eureka.server.rate-limiter-full-fetch-average-rate</code>	Rate limiter, token bucket algorithm property. Specifies the average enforced request rate.	100

## Common configurations

- logging related configurations
  - [logging.level.\\*](#)
  - [logging.group.\\*](#)
  - Any other configurations under `logging.*` namespace should be forbidden, for example, writing log files by using `logging.file` should be forbidden.

## Call between applications

This example shows you how to write Java code to call between applications registered with the Eureka Server for Spring component. When container apps are bound with Eureka, they communicate with each other through the Eureka server.

The example creates two applications, a caller and a callee. Both applications communicate among each other using the Eureka Server for Spring component. The callee application exposes an endpoint that is called by the caller application.

1. Create the callee application. Enable the Eureka client in your Spring Boot application by adding the `@EnableDiscoveryClient` annotation to your main class.

Java

```
@SpringBootApplication
@EnableDiscoveryClient
public class CalleeApplication {
 public static void main(String[] args) {
 SpringApplication.run(CalleeApplication.class, args);
 }
}
```

2. Create an endpoint in the callee application that is called by the caller application.

Java

```
@RestController
public class CalleeController {

 @GetMapping("/call")
 public String calledByCaller() {
 return "Hello from Application callee!";
 }
}
```

3. Set the callee application's name in the application configuration file. For example, `application.yml`.

YAML

```
spring.application.name=callee
```

4. Create the caller application.

Add the `@EnableDiscoveryClient` annotation to enable Eureka client functionality. Also, create a `WebClient.Builder` bean with the `@LoadBalanced` annotation to perform load-balanced calls to other services.

Java

```
@SpringBootApplication
@EnableDiscoveryClient
public class CallerApplication {
 public static void main(String[] args) {
 SpringApplication.run(CallerApplication.class, args);
 }
}
```

```
@Bean
@LoadBalanced
public WebClient.Builder loadBalancedWebClientBuilder() {
 return WebClient.builder();
}
}
```

5. Create a controller in the caller application that uses the `WebClient.Builder` to call the callee application using its application name, callee.

Java

```
@RestController
public class CallerController {
 @Autowired
 private WebClient.Builder webClientBuilder;

 @GetMapping("/call-callee")
 public Mono<String> callCallee() {
 return webClientBuilder.build()
 .get()
 .uri("http://callee/call")
 .retrieve()
 .bodyToMono(String.class);
 }
}
```

Now you have a caller and callee application that communicate with each other using Eureka Server for Spring Java components. Make sure both applications are running and bind with the Eureka server before testing the `/call-callee` endpoint in the caller application.

## Limitations

- The Eureka Server Java component comes with a default configuration, `eureka.server.enable-self-preservation`, set to `false`. This default configuration helps avoid times when instances aren't deleted after self-preservation is enabled. If instances are deleted too early, some requests might be directed to nonexistent instances. If you want to change this setting to `true`, you can overwrite it by setting your own configurations in the Java component.
- The Eureka server has only a single replica and doesn't support scaling, making the peer Eureka server feature unavailable.
- The Eureka dashboard isn't available.

# Next steps

[Tutorial: Connect to a managed Eureka Server for Spring](#)

# Configure the Spring Boot Admin component in Azure Container Apps

Article • 07/16/2024

The Admin for Spring managed component offers an administrative interface for Spring Boot web applications that expose actuator endpoints. This article shows you how to configure and manage your Spring component.

## Show

You can view the details of an individual component by name using the `show` command.

Before you run the following command, replace placeholders surrounded by `<>` with your values.

Azure CLI

```
az containerapp env java-component admin-for-spring show \
--environment <ENVIRONMENT_NAME> \
--resource-group <RESOURCE_GROUP> \
--name <JAVA_COMPONENT_NAME>
```

## Update

You can update the configuration of an Admin for Spring component using the `update` command.

Before you run the following command, replace placeholders surrounded by `<>` with your values. Supported configurations are listed in the [properties list table](#).

Azure CLI

```
az containerapp env java-component admin-for-spring update \
--environment <ENVIRONMENT_NAME> \
--resource-group <RESOURCE_GROUP> \
--name <JAVA_COMPONENT_NAME> \
--configuration <CONFIGURATION_KEY>=<CONFIGURATION_VALUE>
```

## List

You can list all registered Java components using the `list` command.

Before you run the following command, replace placeholders surrounded by `<>` with your values.

Azure CLI

```
az containerapp env java-component list \
--environment <ENVIRONMENT_NAME> \
--resource-group <RESOURCE_GROUP>
```

## Unbind

To remove a binding from a container app, use the `--unbind` option.

Before you run the following command, replace placeholders surrounded by `<>` with your values.

Azure CLI

```
az containerapp update \
--name <APP_NAME> \
--unbind <JAVA_COMPONENT_NAME> \
--resource-group <RESOURCE_GROUP>
```

## Dependency

When you use the admin component in your container app, you need to add the following dependency in your `pom.xml` file. Replace the version number with the latest version available on the [Maven Repository](#).

XML

```
<dependency>
 <groupId>de.codecentric</groupId>
 <version>3.3.2</version>
 <artifactId>spring-boot-admin-starter-client</artifactId>
</dependency>
```

## Configurable properties

Starting with Spring Boot 2, endpoints other than health and info are not exposed by default. You can expose them by adding the following configuration in your `application.properties` file.

properties

```
management.endpoints.web.exposure.include=*
management.endpoint.health.show-details=always
```

## Allowed configuration list for your Admin for Spring

The following list details the admin component properties you can configure for your app. You can find more details in [Spring Boot Admin](#) docs.

 Expand table

Property name	Description	Default value
<code>spring.boot.admin.server.enabled</code>	Enables the Spring Boot Admin Server.	<code>true</code>
<code>spring.boot.admin.context-path</code>	The path prefix where the Admin Server's statics assets and API are served. Relative to the Dispatcher-Servlet.	
<code>spring.boot.admin.monitor.status-interval</code>	Time interval in milliseconds to check the status of instances.	<code>10,000ms</code>
<code>spring.boot.admin.monitor.status-lifetime</code>	Lifetime of status in milliseconds. The status isn't updated as long the last status isn't expired.	10,000 ms
<code>spring.boot.admin.monitor.info-interval</code>	Time interval in milliseconds to check the info of instances.	<code>1m</code>
<code>spring.boot.admin.monitor.info-lifetime</code>	Lifetime of info in minutes. The info isn't as long the last info isn't expired.	<code>1m</code>
<code>spring.boot.admin.monitor.default-timeout</code>	Default timeout when making requests. Individual values for specific	<code>10,000</code>

Property name	Description	Default value
	endpoints can be overridden using <code>spring.boot.admin.monitor.timeout.*</code> .	
<code>spring.boot.admin.monitor.timeout.*</code>	Key-value pairs with the timeout per <code>endpointId</code> .	Defaults to <code>default-timeout</code> value.
<code>spring.boot.admin.monitor.default-retries</code>	Default number of retries for failed requests. Requests that modify data ( <code>PUT</code> , <code>POST</code> , <code>PATCH</code> , <code>DELETE</code> ) are never retried. Individual values for specific endpoints can be overridden using <code>spring.boot.admin.monitor.retries.*</code> .	0
<code>spring.boot.admin.monitor.retries.*</code>	Key-value pairs with the number of retries per <code>endpointId</code> . Requests that modify data ( <code>PUT</code> , <code>POST</code> , <code>PATCH</code> , <code>DELETE</code> ) are never retried.	Defaults to <code>default-retries</code> value.
<code>spring.boot.admin.metadata-keys-to-sanitize</code>	Metadata values for the keys matching these regex patterns used to sanitize in all JSON output. Starting from Spring Boot 3, all actuator values are masked by default. For more information about how to configure the unsanitization process, see <a href="#">(Sanitize Sensitive Values ↴)</a> .	<code>".**password\$", ".\*secret\$", ".\*key\$", ".\*token\$", ".\*credentials.\*\*", ".\*vcap_services\$"</code>
<code>spring.boot.admin.probed-endpoints</code>	For Spring Boot 1.x client applications Spring Boot Admin probes for the specified endpoints using an <code>OPTIONS</code> request. If the path differs from the ID, you can specify this value as <code>id:path</code> (for example: <code>health:ping</code> )	<code>"health", "env", "metrics", "httptrace:trace", "threaddump:dump", "jolokia", "info", "logfile", "refresh", "flyway", "liquibase", "heapspace", "loggers", "auditevents"</code>
<code>spring.boot.admin.instance-proxy.ignored-headers</code>	Headers not to forward when making requests to clients.	<code>"Cookie", "Set-Cookie", "Authorization"</code>
<code>spring.boot.admin.ui.title</code>	The displayed page title.	<code>"Spring Boot Admin"</code>
<code>spring.boot.admin.ui.poll-timer.cache</code>	Polling duration in milliseconds to fetch new cache data.	<code>2500</code>
<code>spring.boot.admin.ui.poll-timer.datasource</code>	Polling duration in milliseconds to fetch new data source data.	<code>2500</code>
<code>spring.boot.admin.ui.poll-timer.gc</code>	Polling duration in milliseconds to fetch new gc data.	<code>2500</code>
<code>spring.boot.admin.ui.poll-timer.process</code>	Polling duration in milliseconds to fetch new process data.	<code>2500</code>
<code>spring.boot.admin.ui.poll-timer.memory</code>	Polling duration in milliseconds to fetch new memory data.	<code>2500</code>
<code>spring.boot.admin.ui.poll-timer.threads</code>	Polling duration in milliseconds to fetch new threads data.	<code>2500</code>
<code>spring.boot.admin.ui.poll-timer.logfile</code>	Polling duration in milliseconds to fetch new logfile data.	<code>1000</code>
<code>spring.boot.admin.ui.enable-toasts</code>	Enables or disables toast notifications.	<code>false</code>
<code>spring.boot.admin.ui.title</code>	Browser's window title value.	<code>""</code>
<code>spring.boot.admin.ui.brand</code>	HTML code rendered in the navigation header and defaults to the Spring Boot Admin label. By default the Spring Boot Admin logo is followed by its name.	<code>""</code>

Property name	Description	Default value
<code>management.scheme</code>	Value that is substituted in the service URL used for accessing the actuator endpoints.	
<code>management.address</code>	Value that is substituted in the service URL used for accessing the actuator endpoints.	
<code>management.port</code>	Value that is substituted in the service URL used for accessing the actuator endpoints.	
<code>management.context-path</code>	Value that is appended to the service URL used for accessing the actuator endpoints.	<code> \${spring.boot.admin.discovery.converter.management-context-path}</code>
<code>health.path</code>	Value that is appended to the service URL used for health checking. Ignored by the <code>EurekaServiceInstanceConverter</code> .	<code> \${spring.boot.admin.discovery.converter.health-endpoint}</code>
<code>spring.boot.admin.discovery.enabled</code>	Enables the <code>DiscoveryClient</code> support for the admin server.	<code>true</code>
<code>spring.boot.admin.discovery.converter.management-context-path</code>	Value that is appended to the <code>service-url</code> of the discovered service when the <code>management-url</code> value is converted by the <code>DefaultServiceInstanceConverter</code> .	<code>/actuator</code>
<code>spring.boot.admin.discovery.converter.health-endpoint-path</code>	Value that is appended to the <code>management-url</code> of the discovered service when the <code>health-url</code> value is converted by the <code>DefaultServiceInstanceConverter</code> .	<code>"health"</code>
<code>spring.boot.admin.discovery.ignored-services</code>	Services that are ignored when using discovery and not registered as application. Supports simple patterns such as <code>"foo*"</code> , <code>"*bar"</code> , <code>"foo*bar*"</code> .	
<code>spring.boot.admin.discovery.services</code>	Services included when using discovery and registered as application. Supports simple patterns such as <code>"foo*"</code> , <code>"*bar"</code> , <code>"foo*bar*"</code> .	<code>"*"</code>
<code>spring.boot.admin.discovery.ignored-instances-metadata</code>	Services ignored if they contain at least one metadata item that matches patterns in this list. Supports patterns such as <code>"discoverable=false"</code> .	
<code>spring.boot.admin.discovery.instances-metadata</code>	Services included if they contain at least one metadata item that matches patterns in list. Supports patterns such as <code>"discoverable=true"</code> .	

## Common configurations

- Logging related configurations:
  - `logging.level.*`
  - `logging.group.*`
  - Any other configurations under `logging.*` namespace should be forbidden. For example, writing log files by using `logging.file` should be forbidden.

# Next steps

[Tutorial: Connect to a managed Admin for Spring](#)

## Related content

- [Tutorial: Integrate the managed Admin for Spring with Eureka Server for Spring](#)

---

## Feedback

Was this page helpful?



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

# Configure settings for the Config Server for Spring component in Azure Container Apps (preview)

Article • 05/24/2024

Config Server for Spring provides a centralized location to make configuration data available to multiple applications. Use the following guidance to learn how to configure and manage your Config Server for Spring component.

## Show

You can view the details of an individual component by name using the `show` command.

Before you run the following command, replace placeholders surrounded by `<>` with your values.

Azure CLI

```
az containerapp env java-component config-server-for-spring show \
--environment <ENVIRONMENT_NAME> \
--resource-group <RESOURCE_GROUP> \
--name <JAVA_COMPONENT_NAME>
```

## List

You can list all registered Java components using the `list` command.

Before you run the following command, replace placeholders surrounded by `<>` with your values.

Azure CLI

```
az containerapp env java-component list \
--environment <ENVIRONMENT_NAME> \
--resource-group <RESOURCE_GROUP>
```

## Bind

Use the `--bind` parameter of the `update` command to create a connection between the Config Server for Spring component and your container app.

Before you run the following command, replace placeholders surrounded by `<>` with your values.

Azure CLI

```
az containerapp update \
--name <CONTAINER_APP_NAME> \
--resource-group <RESOURCE_GROUP> \
--bind <JAVA_COMPONENT_NAME>
```

## Unbind

To break the connection between your container app and the Config Server for Spring component, use the `--unbind` parameter of the `update` command.

Before you run the following command, replace placeholders surrounded by `<>` with your values.

Azure CLI

```
az containerapp update \
--name <CONTAINER_APP_NAME> \
--unbind <JAVA_COMPONENT_NAME> \
--resource-group <RESOURCE_GROUP>
```

## Configuration options

The `az containerapp update` command uses the `--configuration` parameter to control how the Config Server for Spring is configured. You can use multiple parameters at once as long as they're separated by a space. You can find more details in [Spring Cloud Config Server](#) docs.

The following table lists the different configuration values available.

## Git backend configurations

[+] Expand table

Name	Description
<code>spring.cloud.config.server.git.uri</code> <code>spring.cloud.config.server.git.repos.{repoName}.uri</code>	URI of remote repository.
<code>spring.cloud.config.server.git.username</code> <code>spring.cloud.config.server.git.repos.{repoName}.username</code>	Username for authentication with remote repository.
<code>spring.cloud.config.server.git.password</code> <code>spring.cloud.config.server.git.repos.{repoName}.password</code>	Password for authentication with remote repository.
<code>spring.cloud.config.server.git.search-paths</code> <code>spring.cloud.config.server.git.repos.{repoName}.search-paths</code>	Search paths to use within local working copy. By default searches only the root.
<code>spring.cloud.config.server.git.force-pull</code> <code>spring.cloud.config.server.git.repos.{repoName}.force-pull</code>	Flag to indicate that the repository should force pull. If true discard any local changes and take from remote repository.
<code>spring.cloud.config.server.git.default-label</code> <code>spring.cloud.config.server.git.repos.{repoName}.default-label</code>	The default label used for Git is main. If you do not set <code>spring.cloud.config.server.git.default-label</code> and a branch named main does not exist, the config server will by default also try to checkout a branch named master. If you would like to disable the fallback branch behavior you can set <code>spring.cloud.config.server.git.tryMasterBranch</code> to false.
<code>spring.cloud.config.server.git.try-master-branch</code> <code>spring.cloud.config.server.git.repos.{repoName}.try-master-branch</code>	The config server will by default try to checkout a branch named master.
<code>spring.cloud.config.server.git.skip-ssl-validation</code> <code>spring.cloud.config.server.git.repos.{repoName}.skip-ssl-validation</code>	The configuration server's validation of the Git server's SSL certificate can be disabled by setting the <code>git.skipSslValidation</code> property to true.
<code>spring.cloud.config.server.git.clone-on-start</code> <code>spring.cloud.config.server.git.repos.{repoName}.clone-on-start</code>	Flag to indicate that the repository should be cloned on startup (not on demand). Generally leads to slower startup but faster first query.

Name	Description
<code>spring.cloud.config.server.git.timeout</code> <code>spring.cloud.config.server.git.repos.{repoName}.timeout</code>	Timeout (in seconds) for obtaining HTTP or SSH connection (if applicable). Default 5 seconds.
<code>spring.cloud.config.server.git.refresh-rate</code> <code>spring.cloud.config.server.git.repos.{repoName}.refresh-rate</code>	How often the config server will fetch updated configuration data from your Git backend.
<code>spring.cloud.config.server.git.private-key</code> <code>spring.cloud.config.server.git.repos.{repoName}.private-key</code>	Valid SSH private key. Must be set if ignore-local-ssh-settings is true and Git URI is SSH format.
<code>spring.cloud.config.server.git.host-key</code> <code>spring.cloud.config.server.git.repos.{repoName}.host-key</code>	Valid SSH host key. Must be set if host-key-algorithm is also set.
<code>spring.cloud.config.server.git.host-key-algorithm</code> <code>spring.cloud.config.server.git.repos.{repoName}.host-key-algorithm</code>	One of ssh-dss, ssh-rsa, ssh-ed25519, ecdsa-sha2-nistp256, ecdsa-sha2-nistp384, or ecdsa-sha2-nistp521. Must be set if host-key is also set.
<code>spring.cloud.config.server.git.strict-host-key-checking</code> <code>spring.cloud.config.server.git.repos.{repoName}.strict-host-key-checking</code>	true or false. If false, ignore errors with host key.
<code>spring.cloud.config.server.git.repos.{repoName}</code>	URI of remote repository.
<code>spring.cloud.config.server.git.repos.{repoName}.pattern</code>	The pattern format is a comma-separated list of {application}/{profile} names with wildcards. If {application}/{profile} does not match any of the patterns, it uses the default URI defined under.

## Common configurations

- logging related configurations
  - [logging.level.\\*](#)
  - [logging.group.\\*](#)
  - Any other configurations under `logging.*` namespace should be forbidden, for example, writing log files by using `logging.file` should be forbidden.
- `spring.cloud.config.server.overrides`

- Extra map for a property source to be sent to all clients unconditionally.
- **spring.cloud.config.override-none**
  - You can change the priority of all overrides in the client to be more like default values, letting applications supply their own values in environment variables or System properties, by setting the `spring.cloud.config.override-none=true` flag (the default is false) in the remote repository.
- **spring.cloud.config.allow-override**
  - If you enable config first bootstrap, you can allow client applications to override configuration from the config server by placing two properties within the applications configuration coming from the config server.
- **spring.cloud.config.server.health.**
  - You can configure the Health Indicator to check more applications along with custom profiles and custom labels
- **spring.cloud.config.server.accept-empty**
  - You can set `spring.cloud.config.server.accept-empty` to `false` so that the server returns an HTTP `404` status, if the application is not found. By default, this flag is set to `true`.
- **Encryption and decryption (symmetric)**
  - `encrypt.key`
    - It is convenient to use a symmetric key since it is a single property value to configure.
  - `spring.cloud.config.server.encrypt.enabled`
    - You can set this to `false`, to disable server-side decryption.

## Refresh

Services that consume properties need to know about the change before it happens.

The default notification method for Config Server for Spring involves manually triggering the refresh event, such as refresh by call

`https://<YOUR_CONFIG_CLIENT_HOST_NAME>/actuator/refresh`, which may not be feasible if there are many app instances.

Instead, you can automatically refresh values from Config Server by letting the config client poll for changes based on a refresh internal. Use the following steps to automatically refresh values from Config Server.

1. Register a scheduled task to refresh the context in a given interval, as shown in the following example.

```
Java

@Configuration
@AutoConfigureAfter({RefreshAutoConfiguration.class,
RefreshEndpointAutoConfiguration.class})
@EnableScheduling
public class ConfigClientAutoRefreshConfiguration implements
SchedulingConfigurer {
 @Value("${spring.cloud.config.refresh-interval:60}")
 private long refreshInterval;
 @Value("${spring.cloud.config.auto-refresh:false}")
 private boolean autoRefresh;
 private final RefreshEndpoint refreshEndpoint;
 public ConfigClientAutoRefreshConfiguration(RefreshEndpoint
refreshEndpoint) {
 this.refreshEndpoint = refreshEndpoint;
 }
 @Override
 public void configureTasks(ScheduledTaskRegistrar
scheduledTaskRegistrar) {
 if (autoRefresh) {
 // set minimal refresh interval to 5 seconds
 refreshInterval = Math.max(refreshInterval, 5);

 scheduledTaskRegistrar.addFixedRateTask(refreshEndpoint::refresh,
Duration.ofSeconds(refreshInterval));
 }
 }
}
```

2. Enable `autorefresh` and set the appropriate refresh interval in the `application.yml` file. In the following example, the client polls for a configuration change every 60 seconds, which is the minimum value you can set for a refresh interval.

By default, `autorefresh` is set to `false`, and `refresh-interval` is set to 60 seconds.

```
YAML

spring:
 cloud:
 config:
 auto-refresh: true
 refresh-interval: 60
management:
 endpoints:
 web:
 exposure:
```

```
include:
- refresh
```

3. Add `@RefreshScope` in your code. In the following example, the variable `connectTimeout` is automatically refreshed every 60 seconds.

```
Java
```

```
@RestController
@RefreshScope
public class HelloController {
 @Value("${timeout:4000}")
 private String connectTimeout;
}
```

## Encryption and decryption with a symmetric key

### Server-side decryption

By default, server-side encryption is enabled. Use the following steps to enable decryption in your application.

1. Add the encrypted property in your `.properties` file in your git repository.

For example, your file should resemble the following example:

```
message=
{cipher}f43e3df3862ab196a4b367624a7d9b581e1c543610da353fbdd2477d60fb282
f
```

2. Update the Config Server for Spring Java component to use the git repository that has the encrypted property and set the encryption key.

Before you run the following command, replace placeholders surrounded by `<>` with your values.

```
Azure CLI
```

```
az containerapp env java-component config-server-for-spring update \
--environment <ENVIRONMENT_NAME> \
--resource-group <RESOURCE_GROUP> \

```

```
--name <JAVA_COMPONENT_NAME> \
--configuration spring.cloud.config.server.git.uri=<URI>
encrypt.key=randomKey
```

## Client-side decryption

You can use client side decryption of properties by following the steps:

1. Add the encrypted property in your `*.properties` file in your git repository.
2. Update the Config Server for Spring Java component to use the git repository that has the encrypted property and disable server-side decryption.

Before you run the following command, replace placeholders surrounded by `<>` with your values.

Azure CLI

```
az containerapp env java-component config-server-for-spring update \
--environment <ENVIRONMENT_NAME> \
--resource-group <RESOURCE_GROUP> \
--name <JAVA_COMPONENT_NAME> \
--configuration spring.cloud.config.server.git.uri=<URI>
spring.cloud.config.server.encrypt.enabled=false
```

3. In your client app, add the decryption key `ENCRYPT_KEY=randomKey` as an environment variable.

Alternatively, if you include `spring-cloud-starter-bootstrap` on the `classpath`, or set `spring.cloud.bootstrap.enabled=true` as a system property, set `encrypt.key` in `bootstrap.properties`.

Before you run the following command, replace placeholders surrounded by `<>` with your values.

Azure CLI

```
az containerapp update \
--name <APP_NAME> \
--resource-group <RESOURCE_GROUP> \
--set-env-vars "ENCRYPT_KEY=randomKey"
```

```
encrypt:
key: somerandomkey
```

## Next steps

[Tutorial: Connect to a managed Config Server for Spring](#)

# Set dynamic logger level to troubleshoot Java applications in Azure Container Apps (preview)

Article • 05/28/2024

Azure Container Apps platform offers a built-in diagnostics tool exclusively for Java developers to help them debug and troubleshoot their Java applications running on Azure Container Apps more easily and efficiently. One of the key features is a dynamic logger level change, which allows you to access log details that are hidden by default. When enabled, log information is collected without code modifications or forcing you to restart your app when changing log levels.

Before getting started, you need to upgrade Azure Container Apps extension in your Azure CLI to version 0.3.51 or higher.

Azure CLI

```
az extension update --name containerapp
```

## Enable JVM diagnostics for your Java applications

Before using the Java diagnostics tool, you need to first enable Java Virtual Machine (JVM) diagnostics for your Azure Container Apps. This step enables Java diagnostics functionality by injecting an advanced diagnostics agent into your app. Your app might restart during this process.

To take advantage of these diagnostic tools, you can create a new container app with them enabled, or update an existing container app.

To create a new container app with JVM diagnostics enabled, use the following command:

Azure CLI

```
az containerapp create --enable-java-agent \
--environment <ENVIRONMENT_NAME> \
--resource-group <RESOURCE_GROUP> \
--name <CONTAINER_APP_NAME>
```

To update an existing container app, use the following command:

Azure CLI

```
az containerapp update --enable-java-agent \
--resource-group <RESOURCE_GROUP> \
--name <CONTAINER_APP_NAME>
```

## Change runtime logger levels

After enabling JVM diagnostics, you can change runtime log levels for specific loggers in your running Java app without the need to restart your application.

The following sample uses the logger name `org.springframework.boot` with the log level `info`. Make sure to change these values to match your own logger name and level.

Use the following command to adjust log levels for a specific logger:

Azure CLI

```
az containerapp java logger set \
--logger-name "org.springframework.boot" \
--logger-level "info" \
--resource-group <RESOURCE_GROUP> \
--name <CONTAINER_APP_NAME>
```

It may take up to two minutes for the logger level change to take effect. Once complete, you can check the application logs from [log streams](#) or other [log options](#).

## Supported Java logging frameworks

The following Java logging frameworks are supported:

- [Log4j2](#) (only version 2.\*)
- [Logback](#)
- [jboss-logging](#)

## Supported log levels by different logging frameworks

Different logging frameworks support different log levels. In the JVM diagnostics platform, some frameworks are better supported than others. Before changing logging levels, make sure the log levels you're using are supported by both the framework and platform.

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

Framework	OFF	FATAL	ERROR	WARN	INFO	DEBUG	TRACE	ALL
Log4j2	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Logback	Yes	No	Yes	Yes	Yes	Yes	Yes	Yes
jboss-logging	No	Yes	Yes	Yes	Yes	Yes	Yes	No
Platform	Yes	No	Yes	Yes	Yes	Yes	Yes	No

## General visibility of log levels

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

Log Level	FATAL	ERROR	WARN	INFO	DEBUG	TRACE	ALL
OFF							
FATAL	Yes						
ERROR	Yes	Yes					
WARN	Yes	Yes	Yes				
INFO	Yes	Yes	Yes	Yes			
DEBUG	Yes	Yes	Yes	Yes	Yes		
TRACE	Yes	Yes	Yes	Yes	Yes	Yes	
ALL	Yes	Yes	Yes	Yes	Yes	Yes	Yes

For example, if you set log level to `DEBUG`, your app will print logs with level `FATAL`, `ERROR`, `WARN`, `INFO`, `DEBUG` and will NOT print logs with level `TRACE` AND `ALL`.

## Related content

[Log steaming](#)

# Tutorial: Connect to a managed Eureka Server for Spring in Azure Container Apps

Article • 10/14/2024

Eureka Server for Spring is a service registry that allows microservices to register themselves and discover other services. Eureka Server for Spring is available as an Azure Container Apps component. You can bind your container app to Eureka Server for Spring for automatic registration with the Eureka server.

In this tutorial, you learn how to:

- ✓ Create a Eureka Server for Spring Java component.
- ✓ Bind your container app to the Eureka Server for Spring Java component.

## ⓘ Important

This tutorial uses services that can affect your Azure bill. If you decide to follow along, make sure that you delete the resources featured in this article to avoid unexpected billing.

## Prerequisites

To finish this project, you need the following items:

[+] Expand table

Requirement	Instructions
Azure account	An active subscription is required. If you don't have one, you <a href="#">can create one for free ↗</a> .
Azure CLI	Install the <a href="#">Azure CLI</a> .

## Considerations

When you run Eureka Server for Spring in Container Apps, be aware of the following details:

Item	Explanation
Scope	The Eureka Server for Spring component runs in the same environment as the connected container app.
Scaling	The Eureka Server for Spring component can't scale. The scaling properties <code>minReplicas</code> and <code>maxReplicas</code> are both set to <code>1</code> . To achieve high availability, see <a href="#">Create a highly available Eureka Service in Container Apps</a> .
Resources	The container resource allocation for Eureka Server for Spring is fixed. The number of the CPU cores is <code>0.5</code> , and the memory size is <code>1 Gi</code> .
Pricing	The Eureka Server for Spring billing falls under consumption-based pricing. Resources consumed by managed Java components are billed at the active/idle rates. You can delete components that are no longer in use to stop billing.
Binding	Container apps connect to a Eureka Server for Spring component via a binding. The bindings inject configurations into container app environment variables. After a binding is established, the container app can read the configuration values from environment variables and connect to the Eureka Server for Spring component.

## Setup

Before you begin to work with the Eureka Server for Spring component, you first need to create the required resources.

### Azure CLI

Run the following commands to create your resource group in a container app environment.

1. Create variables to support your application configuration. These values are provided for you for the purposes of this lesson.

### Bash

```
export LOCATION=eastus
export RESOURCE_GROUP=my-services-resource-group
export ENVIRONMENT=my-environment
export EUREKA_COMPONENT_NAME=eureka
export APP_NAME=my-eureka-client
export IMAGE="mcr.microsoft.com/javacomponents/samples/sample-
service-eureka-client:latest"
```

Variable	Description
LOCATION	The Azure region location where you create your container app and Java component.
ENVIRONMENT	The container app environment name for your demo application.
RESOURCE_GROUP	The Azure resource group name for your demo application.
EUREKA_COMPONENT_NAME	The name of the Java component created for your container app. In this case, you create a Eureka Server for Spring Java component.
IMAGE	The container image used in your container app.

## 2. Sign in to Azure with the Azure CLI.

Azure CLI

```
az login
```

## 3. Create a resource group.

Azure CLI

```
az group create --name $RESOURCE_GROUP --location $LOCATION
```

## 4. Create your container app environment.

Azure CLI

```
az containerapp env create \
--name $ENVIRONMENT \
--resource-group $RESOURCE_GROUP \
--location $LOCATION
```

# Create the Eureka Server for Spring Java component

Azure CLI

Now that you have an existing environment, you can create your container app and bind it to a Java component instance of Eureka Server for Spring.

1. Create the Eureka Server for Spring Java component.

Azure CLI

```
az containerapp env java-component eureka-server-for-spring create \
 --environment $ENVIRONMENT \
 --resource-group $RESOURCE_GROUP \
 --name $EUREKA_COMPONENT_NAME
```

2. Optional: Update the Eureka Server for Spring Java component configuration.

Azure CLI

```
az containerapp env java-component eureka-server-for-spring update \
 --environment $ENVIRONMENT \
 --resource-group $RESOURCE_GROUP \
 --name $EUREKA_COMPONENT_NAME \
 --configuration eureka.server.renewal-percent-threshold=0.85 \
 eureka.server.eviction-interval-timer-in-ms=10000
```

## Bind your container app to the Eureka Server for Spring Java component

Azure CLI

1. Create the container app and bind it to the Eureka Server for Spring component.

Azure CLI

```
az containerapp create \
 --name $APP_NAME \
 --resource-group $RESOURCE_GROUP \
 --environment $ENVIRONMENT \
 --image $IMAGE \
 --min-replicas 1 \
 --max-replicas 1 \
 --ingress external \
```

```
--target-port 8080 \
--bind $EUREKA_COMPONENT_NAME \
--query properties.configuration.ingress.fqdn
```

2. Copy the URL of your app to a text editor so that you can use it in an upcoming step.

Return to the container app in the portal. Copy the URL of your app to a text editor so that you can use it in an upcoming step.

Go to the `/allRegistrationStatus` route to view all applications that are registered with the Eureka Server for Spring component.

The binding injects several configurations into the application as environment variables, primarily the `eureka.client.service-url.defaultZone` property. This property indicates the internal endpoint of the Eureka Server Java component.

The binding also injects the following properties:

Bash

```
"eureka.client.register-with-eureka": "true"
"eureka.client.fetch-registry": "true"
"eureka.instance.prefer-ip-address": "true"
```

The `eureka.client.register-with-eureka` property is set to `true` to enforce registration with the Eureka server. This registration overwrites the local setting in `application.properties`, from the configuration server and so on. If you want to set it to `false`, you can overwrite it by setting an environment variable in your container app.

The `eureka.instance.prefer-ip-address` property is set to `true` because of the specific domain name system resolution rule in the container app environment. Don't modify this value so that you don't break the binding.

## Optional: Unbind your container app from the Eureka Server for Spring Java component

Azure CLI

To remove a binding from a container app, use the `--unbind` option.

## Azure CLI

```
az containerapp update \
--name $APP_NAME \
--unbind $JAVA_COMPONENT_NAME \
--resource-group $RESOURCE_GROUP
```

# View the application through a dashboard

### ⓘ Important

To view the dashboard, you need to have at least the `Microsoft.App/managedEnvironments/write` role assigned to your account on the managed environment resource. You can explicitly assign the `Owner` or `Contributor` role on the resource. You can also follow the steps to create a custom role definition and assign it to your account.

1. Create a custom role definition.

## Azure CLI

```
az role definition create --role-definition '{
 "Name": "<YOUR_ROLE_NAME>",
 "IsCustom": true,
 "Description": "Can access managed Java Component dashboards in managed environments",
 "Actions": [
 "Microsoft.App/managedEnvironments/write"
],
 "AssignableScopes": ["/subscriptions/<SUBSCRIPTION_ID>"]
}'
```

Make sure to replace the placeholder in between the `<>` brackets in the `AssignableScopes` value with your subscription ID.

2. Assign the custom role to your account on a managed environment resource.

Get the resource ID of the managed environment:

## Azure CLI

```
export ENVIRONMENT_ID=$(az containerapp env show \
--name $ENVIRONMENT --resource-group $RESOURCE_GROUP \
```

```
--query id -o tsv)
```

### 3. Assign the role to your account.

Before you run this command, replace the placeholder in between the <> brackets with your user or service principal ID.

Azure CLI

```
az role assignment create \
--assignee <USER_OR_SERVICE_PRINCIPAL_ID> \
--role "<ROLE_NAME>" \
--scope $ENVIRONMENT_ID
```

#### ⓘ Note

The <USER\_OR\_SERVICE\_PRINCIPAL\_ID> property usually should be the identity that you use to access the Azure portal. The <ROLE\_NAME> property is the name that you assigned in step 1.

### 4. Get the URL of the Eureka Server for Spring dashboard.

Azure CLI

```
az containerapp env java-component eureka-server-for-spring show \
--environment $ENVIRONMENT \
--resource-group $RESOURCE_GROUP \
--name $EUREKA_COMPONENT_NAME \
--query properties.ingress.fqdn -o tsv
```

This command returns the URL that you can use to access the Eureka Server for Spring dashboard. With the dashboard, you can also see your container app, as shown in the following screenshot.

## System Status

Environment	test	Current time	2024-05-30T10:58:19 +0000
Data center	default	Uptime	01:19
		Lease expiration enabled	true
		Renews threshold	1
		Renews (last min)	8

THE SELF PRESERVATION MODE IS TURNED OFF. THIS MAY NOT PROTECT INSTANCE EXPIRY IN CASE OF NETWORK/OTHER PROBLEMS.

## DS Replicas

localhost

## Instances currently registered with Eureka

Application	AMIs	Availability Zones	Status
SAMPLE-CLIENT-APP	n/a (1) (1)		UP (1) - sample-service-eureka-client--seqgzy4-7b776d94cd-fdmwj.sample-client-app:8080

## General Info

Name	Value
total-avail-memory	70mb
num-of-cpus	1
current-memory-usage	46mb (65%)
server-uptime	01:19
registered-replicas	http://localhost:8761/eureka/
unavailable-replicas	http://localhost:8761/eureka/,
available-replicas	



## Optional: Integrate the Eureka Server for Spring and Admin for Spring Java components

If you want to integrate the Eureka Server for Spring and the Admin for Spring Java components, see [Integrate the managed Admin for Spring with Eureka Server for Spring](#).

## Clean up resources

The resources created in this tutorial have an effect on your Azure bill. If you aren't going to use these services long term, run the following command to remove everything you created in this tutorial.

## Azure CLI

```
az group delete \
--resource-group $RESOURCE_GROUP
```

## Next step

[Configure Eureka Server for Spring settings](#)

## Related content

[Integrate the managed Admin for Spring with Eureka Server for Spring](#)

---

## Feedback

Was this page helpful?

 Yes

 No

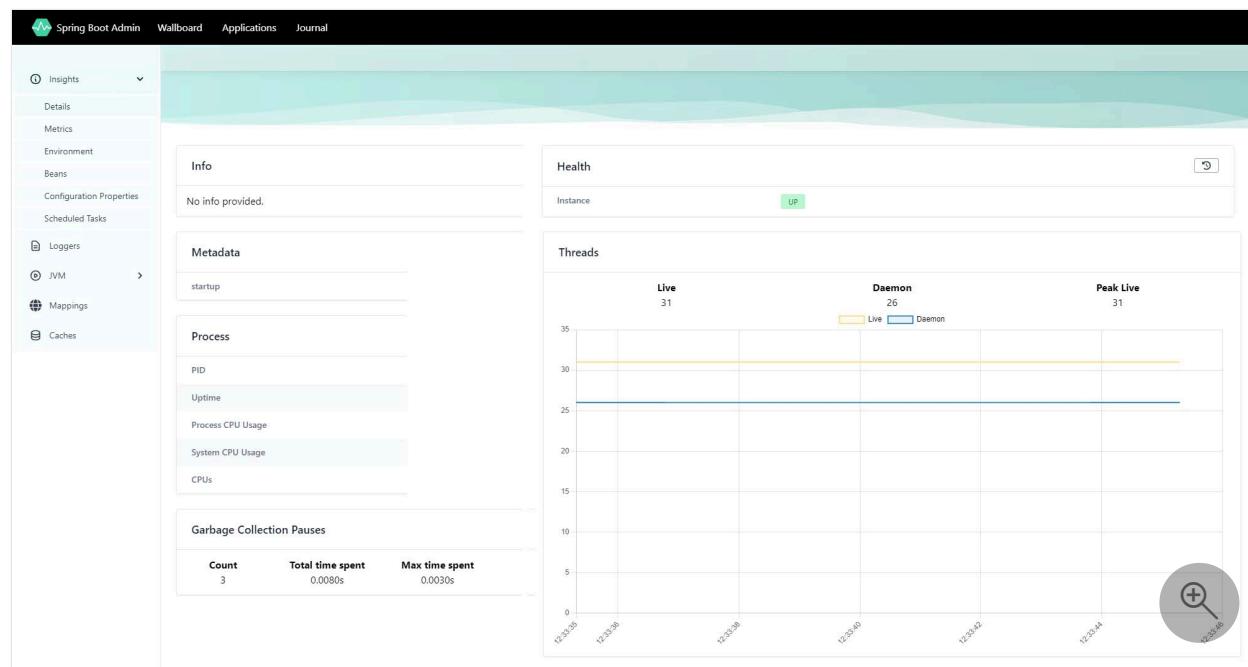
[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

# Tutorial: Connect to a managed Admin for Spring in Azure Container Apps

Article • 10/14/2024

The Admin for Spring managed component offers an administrative interface for Spring Boot web applications that expose actuator endpoints. As a managed component in Azure Container Apps, you can easily bind your container app to Admin for Spring for seamless integration and management.

This tutorial shows you how to create an Admin for Spring Java component and bind it to your container app so that you can monitor and manage your Spring applications with ease.



In this tutorial, you learn how to:

- ✓ Create an Admin for Spring Java component.
- ✓ Bind your container app to an Admin for Spring Java component.

If you want to integrate Admin for Spring with Eureka Server for Spring, see [Integrate Admin for Spring with Eureka Server for Spring in Container Apps](#) instead.

## ⓘ Important

This tutorial uses services that can affect your Azure bill. If you decide to follow along, make sure you delete the resources featured in this article to avoid unexpected billing.

# Prerequisites

To finish this project, you need the following items:

[+] Expand table

Requirement	Instructions
Azure account ↗	An active subscription is required. If you don't have one, you <a href="#">can create one for free ↗</a> .
Azure CLI	Install the <a href="#">Azure CLI</a> .

# Considerations

When you run the Admin for Spring component in Container Apps, be aware of the following details:

[+] Expand table

Item	Explanation
Scope	Components run in the same environment as the connected container app.
Scaling	Component can't scale. The scaling properties <code>minReplicas</code> and <code>maxReplicas</code> are both set to <code>1</code> .
Resources	The container resource allocation for components is fixed. The number of the CPU cores is 0.5, and the memory size is 1 Gi.
Pricing	Component billing falls under consumption-based pricing. Resources consumed by managed components are billed at the active/idle rates. You can delete components that are no longer in use to stop billing.
Binding	Container apps connect to a component via a binding. The bindings inject configurations into container app environment variables. After a binding is established, the container app can read the configuration values from environment variables and connect to the component.

# Setup

Before you begin to work with the Admin for Spring component, you first need to create the required resources.

## Azure CLI

The following commands help you create your resource group and container app environment.

1. Create variables to support your application configuration. These values are provided for you for the purposes of this lesson.

Bash

```
export LOCATION=eastus
export RESOURCE_GROUP=my-resource-group
export ENVIRONMENT=my-environment
export JAVA_COMPONENT_NAME=admin
export APP_NAME=sample-admin-client
export IMAGE="mcr.microsoft.com/javacomponents/samples/sample-
admin-for-spring-client:latest"
```

[+] Expand table

Variable	Description
LOCATION	The Azure region location where you create your container app and Java component.
ENVIRONMENT	The container app environment name for your demo application.
RESOURCE_GROUP	The Azure resource group name for your demo application.
JAVA_COMPONENT_NAME	The name of the Java component created for your container app. In this case, you create an Admin for Spring Java component.
IMAGE	The container image used in your container app.

2. Sign in to Azure with the Azure CLI.

Azure CLI

```
az login
```

3. Create a resource group.

Azure CLI

```
az group create \
--name $RESOURCE_GROUP \
--location $LOCATION \
--query "properties.provisioningState"
```

When you use the `--query` parameter, the response filters down to a simple success or failure message.

#### 4. Create your container app environment.

Azure CLI

```
az containerapp env create \
--name $ENVIRONMENT \
--resource-group $RESOURCE_GROUP \
--location $LOCATION
```

## Use the component

Azure CLI

Now that you have an existing environment, you can create your container app and bind it to a Java component instance of an Admin for Spring component.

#### 1. Create the Admin for Spring Java component.

Azure CLI

```
az containerapp env java-component admin-for-spring create \
--environment $ENVIRONMENT \
--resource-group $RESOURCE_GROUP \
--name $JAVA_COMPONENT_NAME \
--min-replicas 1 \
--max-replicas 1
```

#### 2. Update the Admin for Spring Java component.

Azure CLI

```
az containerapp env java-component admin-for-spring create \
--environment $ENVIRONMENT \
--resource-group $RESOURCE_GROUP \
--name $JAVA_COMPONENT_NAME \
```

```
--min-replicas 2 \
--max-replicas 2
```

## Bind your container app to the Admin for Spring Java component

Azure CLI

1. Create the container app and bind it to the Admin for Spring component.

Azure CLI

```
az containerapp create \
--name $APP_NAME \
--resource-group $RESOURCE_GROUP \
--environment $ENVIRONMENT \
--image $IMAGE \
--min-replicas 1 \
--max-replicas 1 \
--ingress external \
--target-port 8080 \
--bind $JAVA_COMPONENT_NAME
```

The bind operation binds the container app to the Admin for Spring Java component. The container app can now read the configuration values from environment variables, primarily the `SPRING_BOOT_ADMIN_CLIENT_URL` property, and connect to the Admin for Spring component.

The binding also injects the following property:

Bash

```
"SPRING_BOOT_ADMIN_CLIENT_INSTANCE_PREFER-IP": "true",
```

This property indicates that the Admin for Spring component client should prefer the IP address of the container app instance when you connect to the Admin for Spring server.

# Optional: Unbind your container app from the Admin for Spring Java component

Azure CLI

To remove a binding from a container app, use the `--unbind` option.

Azure CLI

```
az containerapp update \
--name $APP_NAME \
--unbind $JAVA_COMPONENT_NAME \
--resource-group $RESOURCE_GROUP
```

## View the dashboard

### Important

To view the dashboard, you need to have at least the `Microsoft.App/managedEnvironments/write` role assigned to your account on the managed environment resource. You can explicitly assign the `Owner` or `Contributor` role on the resource. You can also follow the steps to create a custom role definition and assign it to your account.

1. Create the custom role definition.

Azure CLI

```
az role definition create --role-definition '{
 "Name": "<ROLE_NAME>",
 "IsCustom": true,
 "Description": "Can access managed Java Component dashboards in managed environments",
 "Actions": [
 "Microsoft.App/managedEnvironments/write"
],
 "AssignableScopes": ["/subscriptions/<SUBSCRIPTION_ID>"]
}'
```

Make sure to replace the placeholders in between the `<>` brackets with your values.

2. Assign the custom role to your account on the managed environment resource.

Get the resource ID of the managed environment:

Azure CLI

```
export ENVIRONMENT_ID=$(az containerapp env show \
 --name $ENVIRONMENT --resource-group $RESOURCE_GROUP \
 --query id -o tsv)
```

3. Assign the role to your account.

Before you run this command, replace the placeholder in between the <> brackets with your user or service principal ID.

Azure CLI

```
az role assignment create \
 --assignee <USER_OR_SERVICE_PRINCIPAL_ID> \
 --role "<ROLE_NAME>" \
 --scope $ENVIRONMENT_ID
```

#### ⓘ Note

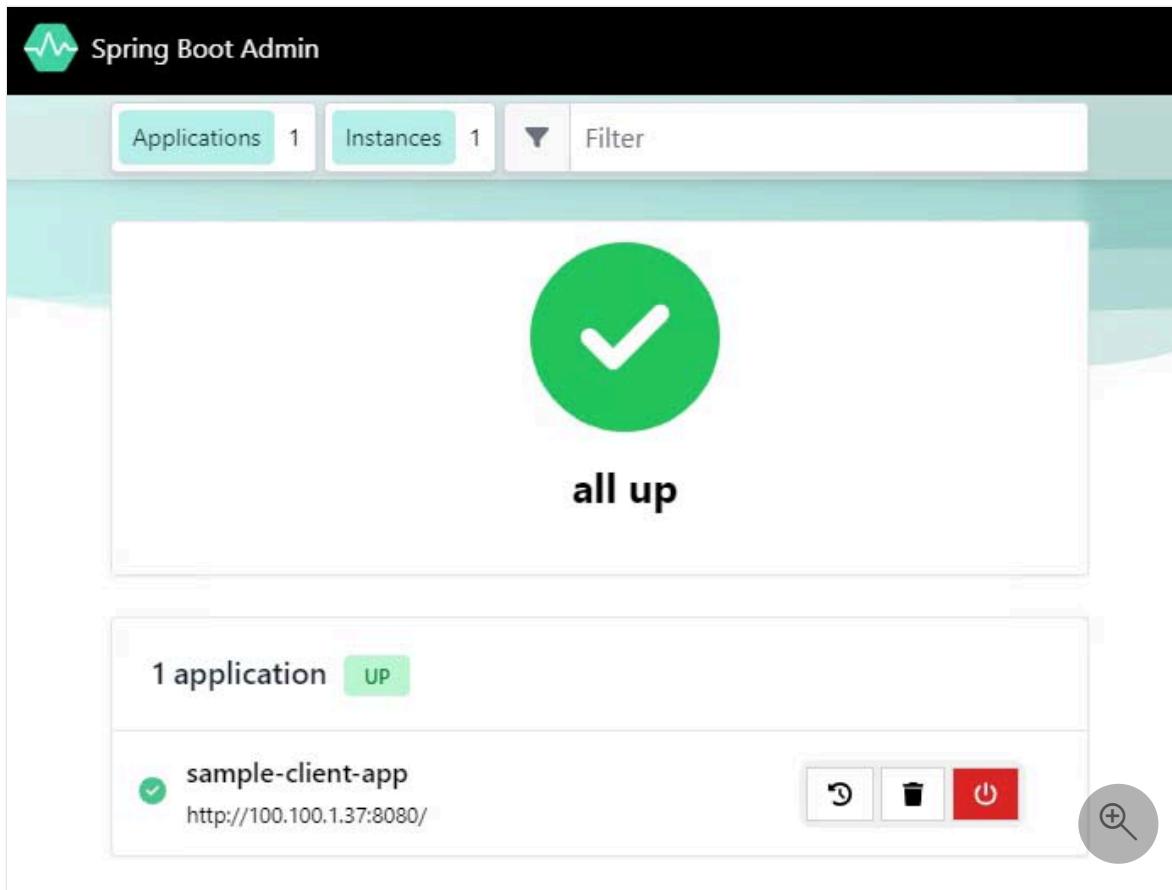
The <USER\_OR\_SERVICE\_PRINCIPAL\_ID> property usually should be the identity that you use to access the Azure portal. The <ROLE\_NAME> property is the name that you assigned in step 1.

4. Get the URL of the Admin for Spring dashboard.

Azure CLI

```
az containerapp env java-component admin-for-spring show \
 --environment $ENVIRONMENT \
 --resource-group $RESOURCE_GROUP \
 --name $JAVA_COMPONENT_NAME \
 --query properties.ingress.fqdn -o tsv
```

This command returns the URL that you can use to access the Admin for Spring dashboard. With the dashboard, you can also see your container app, as shown in the following screenshot.



## Clean up resources

The resources created in this tutorial have an effect on your Azure bill. If you aren't going to use these services long term, run the following command to remove everything you created in this tutorial.

```
Azure CLI

az group delete \
--resource-group $RESOURCE_GROUP
```

## Next step

[Configure Admin for Spring settings](#)

## Related content

[Integrate the managed Admin for Spring with Eureka Server for Spring](#)

# Feedback

Was this page helpful?

 Yes

 No

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

# Tutorial: Connect to a managed Config Server for Spring in Azure Container Apps

Article • 08/27/2024

Config Server for Spring provides a centralized location to make configuration data available to multiple applications. In this article, you learn to connect an app hosted in Azure Container Apps to a Java Config Server for Spring instance.

The Config Server for Spring Java component uses a GitHub repository as the source for configuration settings. Configuration values are made available to your container app via a binding between the component and your container app. As values change in the configuration server, they automatically flow to your application, all without requiring you to recompile or redeploy your application.

In this tutorial, you learn to:

- ✓ Create a Config Server for Spring Java component
- ✓ Bind the Config Server for Spring to your container app
- ✓ Observe configuration values before and after connecting the config server to your application
- ✓ Encrypt and decrypt configuration values with a symmetric key

## ⓘ Important

This tutorial uses services that can affect your Azure bill. If you decide to follow along step-by-step, make sure you delete the resources featured in this article to avoid unexpected billing.

## Prerequisites

To complete this project, you need the following items:

[ ] Expand table

Requirement	Instructions
Azure account	An active subscription is required. If you don't have one, you <a href="#">can create one for free ↗</a> .

Requirement	Instructions
Azure CLI	Install the <a href="#">Azure CLI</a> .

## Considerations

When running in Config Server for Spring in Azure Container Apps, be aware of the following details:

[+] [Expand table](#)

Item	Explanation
Scope	The Config Server for Spring runs in the same environment as the connected container app.
Scaling	To maintain a single source of truth, the Config Server for Spring doesn't scale. The scaling properties <code>minReplicas</code> and <code>maxReplicas</code> are both set to <code>1</code> .
Resources	The container resource allocation for Config Server for Spring is fixed, the number of the CPU cores is <code>0.5</code> , and the memory size is <code>1Gi</code> .
Pricing	The Config Server for Spring billing falls under consumption-based pricing. Resources consumed by managed Java components are billed at the active/idle rates. You can delete components that are no longer in use to stop billing.
Binding	The container app connects to a Config Server for Spring via a binding. The binding injects configurations into container app environment variables. Once a binding is established, the container app can read configuration values from environment variables.

## Setup

Before you begin to work with the Config Server for Spring, you first need to create the required resources.

### Azure CLI

Execute the following commands to create your resource group and Container Apps environment.

1. Create variables to support your application configuration. These values are provided for you for the purposes of this lesson.

Bash

```
export LOCATION=eastus
export RESOURCE_GROUP=my-services-resource-group
export ENVIRONMENT=my-environment
export JAVA_COMPONENT_NAME=configserver
export APP_NAME=my-config-client
export IMAGE="mcr.microsoft.com/javacomponents/samples/sample-service-config-client:latest"
export URI="https://github.com/Azure-Samples/azure-spring-cloud-config-java-aca.git"
```

 Expand table

Variable	Description
LOCATION	The Azure region location where you create your container app and Java component.
ENVIRONMENT	The Azure Container Apps environment name for your demo application.
RESOURCE_GROUP	The Azure resource group name for your demo application.
JAVA_COMPONENT_NAME	The name of the Java component created for your container app. In this case, you create a Config Server for Spring Java component.
IMAGE	The container image used in your container app.
URI	You can replace the URI with your git repo url, if it's private, add the related authentication configurations such as <code>spring.cloud.config.server.git.username</code> and <code>spring.cloud.config.server.git.password</code> .

## 2. Sign in to Azure with the Azure CLI.

```
Azure CLI
```

```
az login
```

## 3. Create a resource group.

```
Azure CLI
```

```
az group create --name $RESOURCE_GROUP --location $LOCATION
```

## 4. Create your container apps environment.

#### Azure CLI

```
az containerapp env create \
--name $ENVIRONMENT \
--resource-group $RESOURCE_GROUP \
--location $LOCATION
```

This environment is used to host both the Config Server for Spring java component and your container app.

## Create the Config Server for Spring Java component

Now that you have a Container Apps environment, you can create your container app and bind it to a Config Server for Spring java component. When you bind your container app, configuration values automatically synchronize from the Config Server component to your application.

#### Azure CLI

1. Create the Config Server for Spring Java component.

#### Azure CLI

```
az containerapp env java-component config-server-for-spring create \
--environment $ENVIRONMENT \
--resource-group $RESOURCE_GROUP \
--name $JAVA_COMPONENT_NAME \
--min-replicas 1 \
--max-replicas 1 \
--configuration spring.cloud.config.server.git.uri=$URI
```

2. Update the Config Server for Spring Java component.

#### Azure CLI

```
az containerapp env java-component config-server-for-spring update \
--environment $ENVIRONMENT \
--resource-group $RESOURCE_GROUP \
--name $JAVA_COMPONENT_NAME \
--min-replicas 2 \
```

```
--max-replicas 2 \
--configuration spring.cloud.config.server.git.uri=$URI
spring.cloud.config.server.git.refresh-rate=60
```

Here, you're telling the component where to find the repository that holds your configuration information via the `uri` property. The `refresh-rate` property tells Container Apps how often to check for changes in your git repository.

## Bind your container app to the Config Server for Spring Java component

Azure CLI

1. Create the container app that consumes configuration data.

Azure CLI

```
az containerapp create \
--name $APP_NAME \
--resource-group $RESOURCE_GROUP \
--environment $ENVIRONMENT \
--image $IMAGE \
--min-replicas 1 \
--max-replicas 1 \
--ingress external \
--target-port 8080 \
--query properties.configuration.ingress.fqdn
```

This command returns the URL of your container app that consumes configuration data. Copy the URL to a text editor so you can use it in a coming step.

If you visit your app in a browser, the `connectTimeout` value returned is the default value of `0`.

2. Bind to the Config Server for Spring.

Now that the container app and Config Server are created, you bind them together with the `update` command to your container app.

Azure CLI

```
az containerapp update \
--name $APP_NAME \
--resource-group $RESOURCE_GROUP \
--bind $JAVA_COMPONENT_NAME
```

The `--bind $JAVA_COMPONENT_NAME` parameter creates the link between your container app and the configuration component.

Once the container app and the Config Server component are bound together, configuration changes are automatically synchronized to the container app.

When you visit the app's URL again, the value of `connectTimeout` is now `10000`. This value comes from the git repo set in the `$URI` variable originally set as the source of the configuration component. Specifically, this value is drawn from the `connectionTimeout` property in the repo's *application.yml* file.

The bind request injects configuration setting into the application as environment variables. These values are now available to the application code to use when fetching configuration settings from the config server.

In this case, the following environment variables are available to the application:

Bash

```
SPRING_CLOUD_CONFIG_URI=http://[JAVA_COMPONENT_INTERNAL_FQDN]:80
SPRING_CLOUD_CONFIG_COMPONENT_URI=http://[JAVA_COMPONENT_INTERNAL_FQDN]:80
SPRING_CONFIG_IMPORT=optional:configserver:$SPRING_CLOUD_CONFIG_URI
```

If you want to customize your own `SPRING_CONFIG_IMPORT`, you can refer to the environment variable `SPRING_CLOUD_CONFIG_COMPONENT_URI`, for example, you can override by command line arguments, like `Java -`

```
Dspring.config.import=optional:configserver:${SPRING_CLOUD_CONFIG_COMPONENT_URI}?
fail-fast=true.
```

You can also remove a binding from your application.

## (Optional) Unbind your container app from the Config Server for Spring Java component

Azure CLI

To remove a binding from a container app, use the `--unbind` option.

Azure CLI

```
az containerapp update \
--name $APP_NAME \
--unbind $JAVA_COMPONENT_NAME \
--resource-group $RESOURCE_GROUP
```

When you visit the app's URL again, the value of `connectTimeout` changes to back to `0`.

## Clean up resources

The resources created in this tutorial have an effect on your Azure bill. If you aren't going to use these services long-term, run the following command to remove everything created in this tutorial.

Azure CLI

```
az group delete \
--resource-group $RESOURCE_GROUP
```

## Next steps

[Customize Config Server for Spring settings](#)

## Feedback

Was this page helpful?

 Yes

 No

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

# Tutorial: Integrate Admin for Spring with Eureka Server for Spring in Azure Container Apps

Article • 07/16/2024

This tutorial will guide you through the process of integrating a managed Admin for Spring with a Eureka Server for Spring within Azure Container Apps.

This article contains some content similar to the "Connect to a managed Admin for Spring in Azure Container Apps" tutorial, but with Eureka Server for Spring, you can bind Admin for Spring to Eureka Server for Spring, so that it can get application information through Eureka, instead of having to bind individual applications to Admin for Spring.

By following this guide, you'll set up a Eureka Server for service discovery and then create an Admin for Spring to manage and monitor your Spring applications registered with the Eureka Server. This setup ensures that other applications only need to bind to the Eureka Server, simplifying the management of your microservices.

In this tutorial, you will learn to:

1. Create a Eureka Server for Spring.
2. Create an Admin for Spring and link it to the Eureka Server.
3. Bind other applications to the Eureka Server for streamlined service discovery and management.

## Prerequisites

To complete this tutorial, you need the following items:

[+] Expand table

Requirement	Instructions
Azure account	An active subscription is required. If you don't have one, you <a href="#">can create one for free</a> .
Azure CLI	Install the <a href="#">Azure CLI</a> .
An existing Eureka Server for Spring Java component	If you don't have one, follow the <a href="#">Create the Eureka Server for Spring</a> section to create one.

# Considerations

When running managed Java components in Azure Container Apps, be aware of the following details:

[+] Expand table

Item	Explanation
Scope	Components run in the same environment as the connected container app.
Scaling	Component can't scale. The scaling properties <code>minReplicas</code> and <code>maxReplicas</code> are both set to <code>1</code> .
Resources	The container resource allocation for components is fixed. The number of the CPU cores is 0.5, and the memory size is 1Gi.
Pricing	Component billing falls under consumption-based pricing. Resources consumed by managed components are billed at the active/idle rates. You can delete components that are no longer in use to stop billing.
Binding	Container apps connect to a component via a binding. The bindings inject configurations into container app environment variables. Once a binding is established, the container app can read the configuration values from environment variables and connect to the component.

# Setup

Before you begin, create the necessary resources by executing the following commands.

1. Create variables to support your application configuration. These values are provided for you for the purposes of this lesson.

Bash

```
export LOCATION=eastus
export RESOURCE_GROUP=my-services-resource-group
export ENVIRONMENT=my-environment
export EUREKA_COMPONENT_NAME=eureka
export ADMIN_COMPONENT_NAME=admin
export CLIENT_APP_NAME=sample-service-eureka-client
export CLIENT_IMAGE="mcr.microsoft.com/javacomponents/samples/sample-admin-for-spring-client:latest"
```

[+] Expand table

Variable	Description
LOCATION	The Azure region location where you create your container app and Java components.
RESOURCE_GROUP	The Azure resource group name for your demo application.
ENVIRONMENT	The Azure Container Apps environment name for your demo application.
EUREKA_COMPONENT_NAME	The name of the Eureka Server Java component.
ADMIN_COMPONENT_NAME	The name of the Admin for Spring Java component.
CLIENT_APP_NAME	The name of the container app that will bind to the Eureka Server.
CLIENT_IMAGE	The container image used in your Eureka Server container app.

## 2. Log in to Azure with the Azure CLI.

```
Azure CLI
```

```
az login
```

## 3. Create a resource group.

```
Azure CLI
```

```
az group create --name $RESOURCE_GROUP --location $LOCATION
```

## 4. Create your container apps environment.

```
Azure CLI
```

```
az containerapp env create \
--name $ENVIRONMENT \
--resource-group $RESOURCE_GROUP \
--location $LOCATION \
--query "properties.provisioningState"
```

Using the `--query` parameter filters the response down to a simple success or failure message.

# Optional: Create the Eureka Server for Spring

If you don't have an existing Eureka Server for Spring, follow the command below to create the Eureka Server Java component. For more information, see [Create the Eureka Server for Spring](java-eureka-server.md#create-the-eureka-server-for-spring-java-component).

#### Azure CLI

```
az containerapp env java-component eureka-server-for-spring create \
--environment $ENVIRONMENT \
--resource-group $RESOURCE_GROUP \
--name $EUREKA_COMPONENT_NAME
```

## Bind the components together

Create the Admin for Spring Java component.

#### Azure CLI

```
az containerapp env java-component admin-for-spring create \
--environment $ENVIRONMENT \
--resource-group $RESOURCE_GROUP \
--name $ADMIN_COMPONENT_NAME \
--bind $EUREKA_COMPONENT_NAME
```

## Bind other apps to the Eureka Server

With the Eureka Server set up, you can now bind other applications to it for service discovery. And you can also monitor and manage these applications in the dashboard of Admin for Spring. Follow the steps below to create and bind a container app to the Eureka Server:

Create the container app and bind it to the Eureka Server.

#### Azure CLI

```
az containerapp create \
--name $CLIENT_APP_NAME \
--resource-group $RESOURCE_GROUP \
--environment $ENVIRONMENT \
--image $CLIENT_IMAGE \
--min-replicas 1 \
--max-replicas 1 \
--ingress external \
```

```
--target-port 8080 \
--bind $EUREKA_COMPONENT_NAME
```

### 💡 Tip

Since the previous steps bound the Admin for Spring component to the Eureka Server for Spring component, the Admin component enables service discovery and allows you to manage it through the Admin for Spring dashboard at the same time.

## View the dashboards

### ⓘ Important

To view the dashboard, you need to have at least the `Microsoft.App/managedEnvironments/write` role assigned to your account on the managed environment resource. You can either explicitly assign `Owner` or `Contributor` role on the resource or follow the steps to create a custom role definition and assign it to your account.

1. Create the custom role definition.

Azure CLI

```
az role definition create --role-definition '{
 "Name": "Java Component Dashboard Access",
 "IsCustom": true,
 "Description": "Can access managed Java Component dashboards in
managed environments",
 "Actions": [
 "Microsoft.App/managedEnvironments/write"
],
 "AssignableScopes": ["/subscriptions/<SUBSCRIPTION_ID>"]
}'
```

Make sure to replace placeholder in between the `<>` brackets in the `AssignableScopes` value with your subscription ID.

2. Assign the custom role to your account on managed environment resource.

Get the resource id of the managed environment.

Azure CLI

```
export ENVIRONMENT_ID=$(az containerapp env show \
 --name $ENVIRONMENT --resource-group $RESOURCE_GROUP \
 --query id -o tsv)
```

### 3. Assign the role to your account.

Before running this command, replace the placeholder in between the <> brackets with your user or service principal ID.

Azure CLI

```
az role assignment create \
 --assignee <USER_OR_SERVICE_PRINCIPAL_ID> \
 --role "Java Component Dashboard Access" \
 --scope $ENVIRONMENT_ID
```

### 4. Get the URL of the Admin for Spring dashboard.

Azure CLI

```
az containerapp env java-component admin-for-spring show \
 --environment $ENVIRONMENT \
 --resource-group $RESOURCE_GROUP \
 --name $ADMIN_COMPONENT_NAME \
 --query properties.ingress.fqdn -o tsv
```

### 5. Get the URL of the Eureka Server for Spring dashboard.

Azure CLI

```
az containerapp env java-component eureka-server-for-spring show \
 --environment $ENVIRONMENT \
 --resource-group $RESOURCE_GROUP \
 --name $EUREKA_COMPONENT_NAME \
 --query properties.ingress.fqdn -o tsv
```

This command returns the URL you can use to access the Eureka Server for Spring dashboard. Through the dashboard, your container app is also to you as shown in the following screenshot.



Spring Boot Admin

Applications 2

Instances 2



Filter



all up

2 applications

UP



SAMPLE-CLIENT-APP

<http://100.100.0.50:8080>



SPRING BOOT ADMIN SERVER

<http://100.100.0.204:8080>



## System Status

Environment	test	Current time	2024-05-30T11:25:29 +0000
Data center	default	Uptime	01:47
		Lease expiration enabled	true
		Renews threshold	3
		Renews (last min)	4

THE SELF PRESERVATION MODE IS TURNED OFF. THIS MAY NOT PROTECT INSTANCE EXPIRY IN CASE OF NETWORK/OTHER PROBLEMS.

## DS Replicas

localhost

### Instances currently registered with Eureka

Application	AMIs	Availability Zones	Status
SAMPLE-CLIENT-APP	n/a (1) (1)		UP (1) - <a href="#">sample-service-eureka-client-ijdy2xy-bc7978c6c-8z1qd:sample-client-app:8080</a>
SPRING BOOT ADMIN SERVER	n/a (1) (1)		UP (1) - <a href="#">admin-azure-java--3qv3mw0-bd9d996c7-7p5w7:Spring Boot Admin Server:8080</a>

## General Info

Name	Value
total-avail-memory	70mb
num-of-cpus	1
current-memory-usage	46mb (65%)
server-uptime	01:47
registered-replicas	<a href="http://localhost:8761/eureka/">http://localhost:8761/eureka/</a>
unavailable-replicas	<a href="http://localhost:8761/eureka/">http://localhost:8761/eureka/</a> ,
available-replicas	

# Clean up resources

The resources created in this tutorial have an effect on your Azure bill. If you aren't going to use these services long-term, run the following command to remove everything created in this tutorial.

## Azure CLI

```
az group delete \
--resource-group $RESOURCE_GROUP
```

# Next steps

[Configure Eureka Server for Spring settings](#)

[Configure Admin for Spring settings](#)

# Feedback

Was this page helpful?

 Yes

 No

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

# Tutorial: Observability of managed Java components in Azure Container Apps

Article • 05/10/2024

Java components include built-in observability features that can give you a holistic view of Java component health throughout its lifecycle. In this tutorial, you learn how to query logs messages generated by a Java component.

## Prerequisites

The following prerequisites are required for this tutorial.

[+] Expand table

Resource	Description
Azure Log Analytics	To use the built-in observability features of managed Java components, ensure you set up Azure Log Analytics to use Log Analytics or <i>Azure Monitor</i> . For more information, see <a href="#">Log storage and monitoring options in Azure Container Apps</a> .
Java component	Make sure to create at least one Java component in your environment, such as <a href="#">Eureka Server</a> or <a href="#">Config Server</a> .

## Query log data

Log Analytics is a tool that helps you view and analyze log data. Using Log Analytics, you can write Kusto queries to retrieve, sort, filter, and visualize log data. These visualizations help you spot trends and identify issues with your application. You can work interactively with the query results or use them with other features such as alerts, dashboards, and workbooks.

1. Open the Azure portal and go to your Azure Log Analytics workspace.
2. Select **Logs** from the sidebar.
3. In the query tab, under the *Tables* section, under *Custom Logs*, select the **ContainerAppSystemlogs\_CL** table.
4. Enter the following Kusto query to display Eureka Server logs for the Spring component.

Kusto

```
ContainerAppSystemLogs_CL
| where ComponentType_s == 'SpringCloudEureka'
| project Time=TimeGenerated, Type=ComponentType_s,
Component=ComponentName_s, Message=Log_s
| take 100
```

The screenshot shows the Azure Monitor Kusto Query Editor interface. On the left, there's a sidebar with tabs for 'Tables', 'Queries', and 'Functions'. Under 'Tables', 'ContainerAppSystemLogs\_CL' is selected and highlighted with a red box. The main area shows a Kusto query and its results. The query is:1 ContainerAppSystemLogs\_CL  
2 | where ComponentType\_s == 'SpringCloudEureka'  
3 | project Time=TimeGenerated, Type=ComponentType\_s,  
Component=ComponentName\_s, Message=Log\_s  
4 | take 100

```
Below the query, the results table has columns: Time [UTC], Type, Component, and Message. It contains three log entries from April 25, 2024:| Time [UTC] | Type | Component | Message |
| --- | --- | --- | --- |
| 4/25/2024, 6:11:06.835 PM | SpringCloudEureka | eureka | Running the evict task with consensus |
| 4/25/2024, 6:09:37.054 PM | SpringCloudEureka | eureka | Disable delta property : false |
| 4/25/2024, 6:09:37.054 PM | SpringCloudEureka | eureka | Single vip registry refresh property : r |

```

5. Select the Run button to run the query.

## Query Java Component Log with Azure monitor

You can query Azure Monitor for monitoring data for your Java component logs.

1. Open the Azure portal and go to your Container Apps environment.
2. From the sidebar, under the *Monitoring* section, select **Logs**.
3. In the query tab, in the *Tables* section, under the *Container Apps* heading, select the **ContainerAppSystemLogs** table.
4. Enter the following Kusto query to display the log entries of Eureka Server for Spring component logs.

Kusto

```
ContainerAppSystemLogs
| where ComponentType == "SpringCloudEureka"
| project Time=TimeGenerated, Type=ComponentType,
Component=ComponentName, Message=Log
| take 100
```

5. Select the Run button to run the query.

# Next steps

[Log storage and monitoring options in Azure Container Apps](#)

# Tutorial: Create a highly available Eureka server component cluster in Azure Container Apps

Article • 08/27/2024

In this tutorial, you learn to create a Eureka service designed to remain operational in the face of failures and high demand. Building a highly available Eureka service ensures the service registry is always available to clients regardless of demand.

Achieving high availability status for Eureka includes linking multiple Eureka server instances together forming a cluster. The cluster provides resources so that if one Eureka server fails, the other services remain available for requests.

In this tutorial, you:

- ✓ Create a Eureka server for Spring components.
- ✓ Bind two Eureka servers for Spring components together into a cluster.
- ✓ Bind applications to both Eureka servers for highly available service discovery.

## Prerequisites

To complete this project, you need the following items:

[+] Expand table

Requirement	Instructions
Azure account	An active subscription is required. If you don't have one, you <a href="#">can create one for free ↗</a> .
Azure CLI	Install the <a href="#">Azure CLI</a> .

## Considerations

When running managed Java components in Azure Container Apps, be aware of the following details:

[+] Expand table

Item	Explanation
Scope	Components run in the same environment as the connected container app.
Scaling	Component can't scale. The scaling properties <code>minReplicas</code> and <code>maxReplicas</code> are both set to <code>1</code> .
Resources	The container resource allocation for components is fixed. The number of the CPU cores is 0.5, and the memory size is 1Gi.
Pricing	Component billing falls under consumption-based pricing. Resources consumed by managed components are billed at the active/idle rates. You can delete components that are no longer in use to stop billing.
Binding	Container apps connect to a component via a binding. The bindings inject configurations into container app environment variables. Once a binding is established, the container app can read the configuration values from environment variables and connect to the component.

## Setup

Use the following steps to create your Eureka service cluster.

1. Create variables that hold application configuration values.

Bash

```
export LOCATION=eastus
export RESOURCE_GROUP=my-services-resource-group
export ENVIRONMENT=my-environment
export EUREKA_COMPONENT_FIRST=eureka01
export EUREKA_COMPONENT_SECOND=eureka02
export APP_NAME=sample-service-eureka-client
export IMAGE="mcr.microsoft.com/javacomponents/samples/sample-service-eureka-client:latest"
```

2. Sign in to Azure with the Azure CLI.

Azure CLI

```
az login
```

3. Create a resource group.

Azure CLI

```
az group create --name $RESOURCE_GROUP --location $LOCATION
```

#### 4. Create your Container Apps environment.

Azure CLI

```
az containerapp env create \
 --name $ENVIRONMENT \
 --resource-group $RESOURCE_GROUP \
 --location $LOCATION
```

## Create a cluster

Next, create two Eureka server instances and link them together as a cluster.

#### 1. Create two Eureka Server for Spring components.

Azure CLI

```
az containerapp env java-component eureka-server-for-spring create \
 --environment $ENVIRONMENT \
 --resource-group $RESOURCE_GROUP \
 --name $EUREKA_COMPONENT_FIRST
```

Azure CLI

```
az containerapp env java-component eureka-server-for-spring create \
 --environment $ENVIRONMENT \
 --resource-group $RESOURCE_GROUP \
 --name $EUREKA_COMPONENT_SECOND
```

## Bind components together

For the Eureka servers to work in a high-availability configuration, they need to be linked together.

#### 1. Bind the first Eureka server to the second.

Azure CLI

```
az containerapp env java-component eureka-server-for-spring update \
 --environment $ENVIRONMENT \
 --resource-group $RESOURCE_GROUP \
 --name $EUREKA_COMPONENT_FIRST
```

```
--name $EUREKA_COMPONENT_FIRST \
--bind $EUREKA_COMPONENT_SECOND
```

- Bind the second Eureka server to the first.

Azure CLI

```
az containerapp env java-component eureka-server-for-spring update \
--environment $ENVIRONMENT \
--resource-group $RESOURCE_GROUP \
--name $EUREKA_COMPONENT_SECOND \
--bind $EUREKA_COMPONENT_FIRST
```

## Deploy and bind the application

With the server components linked together, you can create the container app and binding it to the two Eureka components.

- Create the container app.

Azure CLI

```
az containerapp create \
--name $APP_NAME \
--resource-group $RESOURCE_GROUP \
--environment $ENVIRONMENT \
--image $IMAGE \
--min-replicas 1 \
--max-replicas 1 \
--ingress external \
--target-port 8080
```

- Bind the container app to the first Eureka server component.

Azure CLI

```
az containerapp update \
--name $APP_NAME \
--resource-group $RESOURCE_GROUP \
--bind $EUREKA_COMPONENT_FIRST
```

- Bind the container app to the second Eureka server component.

Azure CLI

```
az containerapp update \
--name $APP_NAME \
--resource-group $RESOURCE_GROUP \
--bind $EUREKA_COMPONENT_SECOND
```

## View the dashboards

### ⓘ Important

To view the dashboard, you need to have at least the `Microsoft.App/managedEnvironments/write` role assigned to your account on the managed environment resource. You can either explicitly assign `Owner` or `Contributor` role on the resource or follow the steps to create a custom role definition and assign it to your account.

1. Create the custom role definition.

Azure CLI

```
az role definition create --role-definition '{
 "Name": "Java Component Dashboard Access",
 "IsCustom": true,
 "Description": "Can access managed Java Component dashboards in
managed environments",
 "Actions": [
 "Microsoft.App/managedEnvironments/write"
],
 "AssignableScopes": ["/subscriptions/<SUBSCRIPTION_ID>"]
}'
```

Make sure to replace placeholder in between the `<>` brackets in the `AssignableScopes` value with your subscription ID.

2. Assign the custom role to your account on managed environment resource.

Get the resource ID of the managed environment.

Azure CLI

```
export ENVIRONMENT_ID=$(az containerapp env show \
--name $ENVIRONMENT --resource-group $RESOURCE_GROUP \
--query id -o tsv)
```

### 3. Assign the role to your account.

Before running this command, replace the placeholder in between the <> brackets with your user or service principal ID.

Azure CLI

```
az role assignment create \
--assignee <USER_OR_SERVICE_PRINCIPAL_ID> \
--role "Java Component Dashboard Access" \
--scope $ENVIRONMENT_ID
```

### 4. Get the URL of the Eureka Server for Spring dashboard.

Azure CLI

```
az containerapp env java-component eureka-server-for-spring show \
--environment $ENVIRONMENT \
--resource-group $RESOURCE_GROUP \
--name $EUREKA_COMPONENT_FIRST \
--query properties.ingress.fqdn -o tsv
```

This command returns the URL you can use to access the Eureka Server for Spring dashboard. Through the dashboard, you can verify that the Eureka server setup consists of two replicas.

## System Status

Environment	test	Current time	2024-08-06T05:35:07 +0000
Data center	default	Uptime	00:01
		Lease expiration enabled	true
		Renews threshold	5
		Renews (last min)	3

THE SELF PRESERVATION MODE IS TURNED OFF. THIS MAY NOT PROTECT INSTANCE EXPIRY IN CASE OF NETWORK/OTHER PROBLEMS.

## DS Replicas

eureka02-azure-java.internal.wittymushroom-d9378d11.azurecontainerapps-test.io

## Instances currently registered with Eureka

Application	AMIs	Availability Zones	Status
SAMPLE-SERVICE-EUREKA-CLIENT	n/a (1)	(1)	UP (1) - sample-service-eureka-client--sesl5cc-568995c88d-tm69x:sample-service-eureka-client_8080
SPRING-CLOUD-EUREKA	n/a (2)	(2)	UP (2) - eureka02-azure-java--0f6qzo0-6b8c54dbcc-7hlvk:sample-cloud-eureka:8761 , eureka01-azure-java--aij7vjl-58cd769475-frr72:sample-cloud-eureka:8761

## General Info

Name	Value
total-avail-memory	71mb
num-of-cpus	1
current-memory-usage	51mb (71%)
server-uptime	00:01
registered-replicas	<a href="https://eureka02-azure-java.internal.wittymushroom-d9378d11.azurecontainerapps-test.io/eureka/">https://eureka02-azure-java.internal.wittymushroom-d9378d11.azurecontainerapps-test.io/eureka/</a>
unavailable-replicas	
available-replicas	<a href="https://eureka02-azure-java.internal.wittymushroom-d9378d11.azurecontainerapps-test.io/eureka/">https://eureka02-azure-java.internal.wittymushroom-d9378d11.azurecontainerapps-test.io/eureka/</a> ,

# Clean up resources

The resources created in this tutorial have an effect on your Azure bill. If you aren't going to use these services long-term, run the following command to remove everything created in this tutorial.

## Azure CLI

```
az group delete \
--resource-group $RESOURCE_GROUP
```

# Next steps

Configure Eureka Server for Spring settings

# Feedback

Was this page helpful?

 Yes

 No

Provide product feedback  | Get help at Microsoft Q&A

# Billing in Azure Container Apps

Article • 10/04/2024

Billing in Azure Container Apps is based on your [plan type](#).

[+] [Expand table](#)

Plan type	Description
Consumption plan	Serverless compute option where you're only billed for the resources your apps use as they're running.
Dedicated plan	Customized compute options where you're billed for instances allocated to each <a href="#">workload profile</a> .

- Your plan selection determines billing calculations.
- Different applications in an environment can use different plans.

This article describes how to calculate the cost of running your container app. For pricing details in your account's currency, see [Azure Container Apps Pricing](#).

## Consumption plan

Billing for apps running in the Consumption plan consists of two types of charges:

- **Resource consumption:** The amount of resources allocated to your container app on a per-second basis, billed in vCPU-seconds and GiB-seconds.
- **HTTP requests:** The number of HTTP requests your container app receives.

The following resources are free during each calendar month, per subscription:

- The first 180,000 vCPU-seconds
- The first 360,000 GiB-seconds
- The first 2 million HTTP requests

Free usage doesn't appear on your bill. You're only charged as your resource usage exceeds the monthly free grants amounts.

### Note

If you use Container Apps with [your own virtual network](#) or your apps utilize other Azure resources, additional charges may apply.

# Resource consumption charges

Azure Container Apps runs replicas of your application based on the [scaling rules and replica count limits](#) you configure for each revision. [Azure Container Apps jobs](#) run replicas when job executions are triggered. You're charged for the amount of resources allocated to each replica while it's running.

There are 2 meters for resource consumption:

- **vCPU-seconds:** The number of vCPU cores allocated to your container app on a per-second basis.
- **GiB-seconds:** The amount of memory allocated to your container app on a per-second basis.

The first 180,000 vCPU-seconds and 360,000 GiB-seconds in each subscription per calendar month are free.

## Container apps

The rate you pay for resource consumption depends on the state of your container app's revisions and replicas. By default, replicas are charged at an *active* rate. However, in certain conditions, a replica can enter an *idle* state. While in an *idle* state, resources are billed at a reduced rate.

### No replicas are running

When a revision is scaled to zero replicas, no resource consumption charges are incurred.

### Minimum number of replicas are running

Idle usage charges might apply when a container app's revision is running under a specific set of circumstances. To be eligible for idle charges, a revision must be:

- Configured with a [minimum replica count](#) greater than zero
- Scaled to the minimum replica count

Usage charges are calculated individually for each replica. A replica is considered idle when *all* of the following conditions are true:

- The replica is running in a revision that is currently eligible for idle charges.
- All of the containers in the replica have started and are running.
- The replica isn't processing any HTTP requests.

- The replica is using less than 0.01 vCPU cores.
- The replica is receiving less than 1,000 bytes per second of network traffic.

When a replica is idle, resource consumption charges are calculated at the reduced idle rates. When a replica isn't idle, the active rates apply.

## More than the minimum number of replicas are running

When a revision is scaled above the [minimum replica count](#), all of its running replicas are charged for resource consumption at the active rate.

## Jobs

In the Consumption plan, resources consumed by Azure Container Apps jobs are charged the active rate. Idle charges don't apply to jobs because executions stop consuming resources once the job completes.

## Request charges

In addition to resource consumption, Azure Container Apps also charges based on the number of HTTP requests received by your container app. Only requests that come from outside a Container Apps environment are billable.

- The first 2 million requests in each subscription per calendar month are free.
- [Health probe](#) requests aren't billable.

Request charges don't apply to Azure Container Apps jobs because they don't support ingress.

## Dedicated plan

You're billed based on workload profile instances, not by individual applications.

Billing for apps and jobs running in the Dedicated plan is based on workload profile instances, not by individual applications. The charges are as follows:

[ ] [Expand table](#)

Fixed management costs	Variable costs
If you have one or more dedicated workload profiles in your environment, you're charged a Dedicated plan management fee.	As profiles scale out, extra costs apply for the extra

Fixed management costs	Variable costs
You aren't billed any plan management charges unless you use a Dedicated workload profile in your environment.	instances; as profiles scale in, billing is reduced.

Make sure to optimize the applications you deploy to a dedicated workload profile. Evaluate the needs of your applications so that they can use the most amount of resources available to the profile.

## Dynamic sessions

Dynamic sessions has two types of session pools: code interpreter and custom container. Each session type has its own billing model.

### Code interpreter

Code interpreter sessions are billed based on running duration for the number allocated sessions. For each allocated session, you're billed from the time it's allocated until it's deallocated in increments of one hour.

### Custom container

Custom container sessions are billed using the [Dedicated plan](#), based on the amount of compute resources used to run the session pool and active sessions.

Each custom container session pool runs on dedicated *E16* compute instances. The number of instances allocated to the session pool is based on the number of active and ready sessions in the pool. To view the number of instances currently allocated to a session pool, use the following Azure CLI command to retrieve the pool's `nodeCount` property. Replace the `<PLACEHOLDERS>` with your values.

```
Bash
```

```
az containerapp sessionpool show --resource-group <RESOURCE_GROUP> --name <POOL_NAME> --query "properties.nodeCount"
```

## General terms

- For pricing details in your account's currency, see [Azure Container Apps Pricing ↗](#).

# Feedback

Was this page helpful?

 Yes

 No

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

# Quotas for Azure Container Apps

Article • 07/05/2024

The following quotas are on a per subscription basis for Azure Container Apps.

You can [request a quota increase in the Azure portal](#). Any time when the maximum quota is larger than the default quota you can request a quota increase. When requesting a quota increase make sure to pick type *Container Apps*. For more information, see [how to request a limit increase](#).

[+] Expand table

Feature	Scope	Default Quota	Maximum Quota	Remarks
Environments	Region	15	Unlimited	Up to 15 environments per subscription, per region. Quota name: Managed Environment Count
Environments	Global	20	Unlimited	Up to 20 environments per subscription, across all regions. Adjusted through Managed Environment Count quota (usually 20% more than Managed Environment Count)
Container Apps	Environment	Unlimited	Unlimited	
Revisions	Container app	Up to 100	Unlimited	
Replicas	Revision	Unlimited	Unlimited	Maximum replicas configurable are 300 in Azure portal and 1000 in Azure CLI. There must also be enough cores quota available.
Session pools	Global	Up to 6	10,000	Maximum number of dynamic session pools per subscription. No official Azure quota yet, please raise support case.

## Workload Profiles Environments

### Consumption workload profile

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

Feature	Scope	Default Quota	Maximum Quota	Remarks
Cores	Replica	4	4	Maximum number of cores available to a revision replica.
Cores	Environment	100	5,000	Maximum number of cores the Consumption workload profile in a Dedicated plan environment can accommodate. Calculated by the sum of cores requested by each active replica of all revisions in an environment. Quota name: Managed Environment General Purpose Cores

## Dedicated workload profiles

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

Feature	Scope	Default Quota	Maximum Quota	Remarks
Cores	Subscription	2,000	Unlimited	Maximum number of dedicated workload profile cores within one subscription
Cores	Replica	Maximum cores a workload profile supports	Same as default quota	Maximum number of cores available to a revision replica.
Cores	Environment	100	5,000	The total cores available to all general purpose (D-series) profiles within an environment. Maximum assumes appropriate network size. Quota name: Managed Environment General Purpose Cores
Cores	Environment	50	5,000	The total cores available to all memory optimized (E-series) profiles within an environment. Maximum assumes appropriate network size. Quota name: Managed Environment Memory Optimized Cores

### ⓘ Note

For GPU enabled workload profiles, you need to request capacity via a [request for a quota increase in the Azure portal](#).

### ⓘ Note

[Free trial](#) and [Azure for Students](#) subscriptions are limited to one environment per subscription globally and ten (10) cores per environment.

## Consumption plan

All new environments use the Consumption workload profile architecture listed above. Only environments created before January 2024 use the consumption plan below.

[+] [Expand table](#)

Feature	Scope	Default Quota	Maximum Quota	Remarks
Cores	Replica	2	2	Maximum number of cores available to a revision replica.
Cores	Environment	100	1,500	Maximum number of cores an environment can accommodate. Calculated by the sum of cores requested by each active replica of all revisions in an environment. Quota name: Managed Environment Consumption Cores

## Considerations

- If an environment runs out of allowed cores:
  - Provisioning times out with a failure
  - The app may be restricted from scaling out
- If you encounter unexpected capacity limits, open a support ticket

---

## Feedback

Was this page helpful?

 Yes

 No

Provide product feedback ↗

# What's new in Azure Container Apps

Article • 06/04/2024

This article lists significant updates and new features available in Azure Container Apps.

## May 2024

[+] Expand table

Feature	Description
Generally Available: Azure Functions on Azure Container Apps	Azure Function's host, runtime, extensions and Azure Function apps can be deployed as containers into the same compute environment. You can use centralized networking, observability, and configuration boundary for multi-type application development like microservices.
Public preview: Dynamic sessions	This fast, sandboxed, ephemeral compute is suitable for running untrusted code at scale in hostile multi-tenancy scenarios. Each session has full compute isolation using Hyper-V.
Public preview: Aspire dashboard support	Access live data about your .NET project and containers in the cloud to evaluate the performance of your applications and debug errors with comprehensive logs, metrics, traces, and more.
Public Preview: NFS Azure Files volume mount support	You can use NFS Azure Files volumes to share data between multiple containers in your application, or to persist data across container restarts.
Public Preview: Monitor apps with Java metrics	You can now monitor the performance and health of your apps with Java metrics such as garbage collection and memory usage.
Public Preview: Set Java log levels	You can now set Java application log levels in Azure Container Apps without redeploying or restarting your apps.

## March 2024

[+] Expand table

Feature	Description
Generally Available: Free managed certificates	Managed certificates are free and enable you to automatically provision and renew TLS certificates for any custom domain you add to your container app.

Feature	Description
<a href="#">Public Preview: OpenTelemetry Agent support</a>	Allows you to use open-source standards to send your app's data without setting up an OTLP collector yourself. You can use the managed agent to choose where to send logs, metrics, and traces.
<a href="#">Public preview: Support for Key Vault Certificates ↗</a>	Use Azure Key Vault to store and manage your own TLS/SSL certificates for use with Azure Container Apps at the app and environment level.
<a href="#">Public Preview: Tomcat support</a>	Azure Container Apps now supports Apache Tomcat in the code-to-cloud build process. This means that you can use your existing code, and configuration, to create a cloud-native container app without the hassle.
<a href="#">Public Preview: JVM memory fit</a>	All Java apps are now calibrated with JVM memory defaults for better performance and reliability in container environment.
<a href="#">Public Preview: Managed Java components: Eureka Server</a> and <a href="#">Public Preview: Managed Java components: Config Server</a>	You can use managed Java components to access platform features for your apps that you would otherwise have to manage yourself.

## January 2024

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

Feature	Description
<a href="#">Generally Available: additional TCP ports</a>	Azure Container Apps now support additional TCP ports, enabling applications to accept TCP connections on multiple ports. This feature is in preview.

## December 2023

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

Feature	Description
<a href="#">Retirement: ACA preview API versions 2022-06-01-preview and 2022-11-01-preview ↗</a>	Starting on March 6, 2024, Azure Container Apps control plane API versions 2023-04-01-preview will be retired. Before that date, migrate to the latest stable API version (2023-05-01) or latest preview API version (2023-08-01-preview).

# November 2023

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

Feature	Description
Generally Available: Landing zone accelerators ↗	Landing zone accelerators provide architectural guidance, reference architecture, reference implementations and automation packaged to deploy workload platforms on Azure at scale.
Public Preview: Dedicated GPU workload profiles	Azure Container Apps support GPU compute in their dedicated workload profiles to unlock machine learning computing for event driven workloads.
Public preview: Vector database add-ons	Azure Container Apps now provides add-ons for three open source vector database variants: Qdrant, Milvus and Weaviate.
Public preview: Policy-driven resiliency	The new resiliency feature enables you to seamlessly recover from service-to-service request and outbound dependency failures just by adding simple policies.
Public preview: Code to cloud ↗	Azure Container Apps now automatically builds and packages application code for deployment.

# September 2023

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

Feature	Description
Generally Available: Azure Container Apps in China Cloud ↗	Azure Container Apps is now available in China North 3.
ACA eligible for savings plans ↗	Azure Container Apps is eligible for Azure savings plan for compute.

# August 2023

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

Feature	Description
Generally Available: Dedicated plan	Azure Container Apps dedicated plan is now generally available in the new workload profiles environment type. When using

Feature	Description
	dedicated workload profiles you're billed per compute instance, compared to consumption where you're billed per app.
<a href="#">Generally Available: UDR, NAT Gateway, and smaller subnets</a>	Improved networking features now allow you to have greater control of egress and support smaller subnets in workload profiles environments.
<a href="#">Generally Available: Azure Container Apps jobs</a>	In addition to continuously running services that can scale to zero, Azure Container Apps now supports jobs. Jobs enable you to run serverless containers that perform tasks that run to completion.
<a href="#">Generally Available: Cross Origin Resource Sharing (CORS)</a>	The CORS feature allows specific origins to make calls on their app through the browser. Azure Container Apps customers can now easily set up Cross Origin Resource Sharing from the portal or through the CLI.
<a href="#">Generally Available: Init containers</a>	Init containers are specialized containers that run to completion before application containers are started in a replica. They can contain utilities or setup scripts not present in your container app image.
<a href="#">Generally Available: Secrets volume mounts</a>	In addition to referencing secrets as environment variables, you can now mount secrets as volumes in your container apps. Your apps can access all or selected secrets as files in a mounted volume.
<a href="#">Generally Available: Session affinity</a>	Session affinity enables you to route all requests from a single client to the same Container Apps replica. This is useful for stateful workloads that require session affinity.
<a href="#">Generally Available: Azure Key Vault references for secrets ↗</a>	Azure Key Vault references enable you to source a container app's secrets from secrets stored in Azure Key Vault. Using the container app's managed identity, the platform automatically retrieves the secret values from Azure Key Vault and injects it into your application's secrets.
<a href="#">Public preview: additional TCP ports</a>	Azure Container Apps now support additional TCP ports, enabling applications to accept TCP connections on multiple ports. This feature is in preview.
<a href="#">Public preview: environment level mTLS encryption</a>	When end-to-end encryption is required, mTLS will encrypt data transmitted between applications within an environment.
<a href="#">Retirement: ACA preview API versions 2022-06-01-preview and 2022-11-01-preview ↗</a>	Starting on November 16, 2023, Azure Container Apps control plane API versions 2022-06-01-preview and 2022-11-01-preview will be retired. Before that date, migrate to the latest stable API version (2023-05-01) or latest preview API version (2023-04-01-preview).

Feature	Description
<a href="#">Dapr: Stable Configuration API ↗</a>	Dapr's Configuration API is now stable and supported in Azure Container Apps. Learn how to do <a href="#">Dapr integration with Azure Container Apps</a>

## June 2023

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

Feature	Description
<a href="#">Generally Available: Running status</a>	The running status helps monitor a container app's health and functionality.
<a href="#">Public Preview: Azure Functions for cloud-native microservices ↗</a>	Azure Function's host, runtime, extensions and Azure Function apps can be deployed as containers into the same compute environment. You can use centralized networking, observability, and configuration boundary for multi-type application development like microservices.
<a href="#">Public Preview: Azure Spring Apps on Azure Container Apps ↗</a>	Azure Spring apps can be deployed as containers to your Azure Container Apps within the same compute environment, so you can use centralized networking, observability, and configuration boundary for multitype application development like microservices.
<a href="#">Public Preview: Azure Container Apps add-ons</a>	As you develop applications in Azure Container Apps, you often need to connect to different services. Rather than creating services ahead of time and manually connecting them to your container app, you can quickly create instances of development-grade services that are designed for nonproduction environments known as "add-ons."
<a href="#">Public Preview: Free and managed TLS certificates</a>	Managed certificates are free and enable you to automatically provision and renew TLS certificates for any custom domain you add to your container app.
<a href="#">Dapr: Multi-app Run improved ↗</a>	Use <code>dapr run -f .</code> to run multiple Dapr apps and see the app logs written to the console <i>and</i> a local log file. Learn how to use <a href="#">multi-app Run logs ↗</a> .

## May 2023

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

Feature	Description
<a href="#">Generally Available: Inbound IP restrictions</a>	Enables container apps to restrict inbound HTTP or TCP traffic by allowing or denying access to a specific list of IP address ranges.
<a href="#">Generally Available: TCP support</a>	Azure Container Apps now supports using TCP-based protocols other than HTTP or HTTPS for ingress.
<a href="#">Generally Available: GitHub Actions for Azure Container Apps</a>	Azure Container Apps allows you to use GitHub Actions to publish revisions to your container app.
<a href="#">Generally Available: Azure Pipelines for Azure Container Apps</a>	Azure Container Apps allows you to use Azure Pipelines to publish revisions to your container app.
<a href="#">Dapr: Easy component creation</a>	You can now configure and secure dependent Azure services to use Dapr APIs in the portal using the Service Connector feature. Learn how to <a href="#">connect to Azure services via Dapr components in the Azure portal</a> .

# Azure Container Apps ARM and YAML template specifications

Article • 09/11/2024

Azure Container Apps deployments are powered by an Azure Resource Manager (ARM) template. Some Container Apps CLI commands also support using a YAML template to specify a resource.

This article includes examples of the ARM and YAML configurations for frequently used Container Apps resources. For a complete list of Container Apps resources see [Azure Resource Manager templates for Container Apps](#). The code listed in this article is for example purposes only. For full schema and type information, see the JSON definitions for your required API version.

## API versions

The latest management API versions for Azure Container Apps are:

- [2023-05-01](#) (stable)
- [2023-08-01-preview](#) (preview)

To learn more about the differences between API versions, see [Microsoft.App change log](#).

## Updating API versions

To use a specific API version in ARM or Bicep, update the version referenced in your templates. To use the latest API version in Azure CLI or Azure PowerShell, update them to the latest version.

Update Azure CLI and the Azure Container Apps extension by running the following commands:

Bash

```
az upgrade
az extension add -n containerapp --upgrade
```

To update Azure PowerShell, see [How to install Azure PowerShell](#).

To programmatically manage Azure Container Apps with the latest API version, use the latest versions of the management SDK:

- .NET
- Go ↗
- Java
- Node.js
- Python

## Container Apps environment

The following tables describe commonly used properties available in the Container Apps environment resource. For a complete list of properties, see [Azure Container Apps REST API reference](#).

### Resource

A Container Apps environment resource includes the following properties:

[+] Expand table

Property	Description	Data type	Read only
daprAIInstrumentationKey	The Application Insights instrumentation key used by Dapr.	string	No
appLogsConfiguration	The environment's logging configuration.	Object	No
peerAuthentication	How to enable mTLS encryption.	Object	No

### Examples

The following example ARM template snippet deploys a Container Apps environment.

#### ⓘ Note

The commands to create container app environments don't support YAML configuration input.

JSON

```
{
 "location": "East US",
 "properties": {
 "appLogsConfiguration": {
 "logAnalyticsConfiguration": {
 "customerId": "string",
 "sharedKey": "string"
 }
 },
 "zoneRedundant": true,
 "vnetConfiguration": {
 "infrastructureSubnetId": "/subscriptions/<subscription_id>/resourceGroups/RGName/providers/Microsoft.Network/virtualNetworks/VNetName/subnets/subnetName1"
 },
 "customDomainConfiguration": {
 "dnsSuffix": "www.my-name.com",
 "certificateValue": "Y2VydA==",
 "certificatePassword": "1234"
 },
 "workloadProfiles": [
 {
 "name": "My-GP-01",
 "workloadProfileType": "GeneralPurpose",
 "minimumCount": 3,
 "maximumCount": 12
 },
 {
 "name": "My-MO-01",
 "workloadProfileType": "MemoryOptimized",
 "minimumCount": 3,
 "maximumCount": 6
 },
 {
 "name": "My-CO-01",
 "workloadProfileType": "ComputeOptimized",
 "minimumCount": 3,
 "maximumCount": 6
 },
 {
 "name": "My-consumption-01",
 "workloadProfileType": "Consumption"
 }
],
 "infrastructureResourceGroup": "myInfrastructureRgName"
 }
}
```

## Container app

The following tables describe the commonly used properties in the container app resource. For a complete list of properties, see [Azure Container Apps REST API reference](#).

## Resource

A container app resource's `properties` object includes the following properties:

[+] Expand table

Property	Description	Data type	Read only
<code>provisioningState</code>	The state of a long running operation, for example when new container revision is created. Possible values include: provisioning, provisioned, failed. Check if app is up and running.	string	Yes
<code>environmentId</code>	The environment ID for your container app. <b>This is a required property to create a container app.</b> If you're using YAML, you can specify the environment ID using the <code>--environment</code> option in the Azure CLI instead.	string	No
<code>latestRevisionName</code>	The name of the latest revision.	string	Yes
<code>latestRevisionFqdn</code>	The latest revision's URL.	string	Yes

The `environmentId` value takes the following form:

Console

```
/subscriptions/<SUBSCRIPTION_ID>/resourcegroups/<RESOURCE_GROUP_NAME>/providers/Microsoft.App/environmentId/<ENVIRONMENT_NAME>
```

In this example, you put your values in place of the placeholder tokens surrounded by `<>` brackets.

### `properties.configuration`

A resource's `properties.configuration` object includes the following properties:

[+] Expand table

Property	Description	Data type
<code>activeRevisionsMode</code>	Setting to <code>single</code> automatically deactivates old revisions, and only keeps the latest revision active. Setting to <code>multiple</code> allows you to maintain multiple revisions.	string
<code>secrets</code>	Defines secret values in your container app.	object
<code>ingress</code>	Object that defines public accessibility configuration of a container app.	object
<code>registries</code>	Configuration object that references credentials for private container registries. Entries defined with <code>secretref</code> reference the secrets configuration object.	object
<code>dapr</code>	Configuration object that defines the Dapr settings for the container app.	object

Changes made to the `configuration` section are [application-scope changes](#), which doesn't trigger a new revision.

## properties.template

A resource's `properties.template` object includes the following properties:

[+] [Expand table](#)

Property	Description	Data type
<code>revisionSuffix</code>	A friendly name for a revision. This value must be unique as the runtime rejects any conflicts with existing revision name suffix values.	string
<code>containers</code>	Configuration object that defines what container images are included in the container app.	object
<code>scale</code>	Configuration object that defines scale rules for the container app.	object

Changes made to the `template` section are [revision-scope changes](#), which triggers a new revision.

## Examples

For details on health probes, refer to [Health probes in Azure Container Apps](#).

## ARM template

The following example ARM template snippet deploys a container app.

JSON

```
{
 "identity": {
 "userAssignedIdentities": [
 "/subscriptions/<subscription_id>/resourcegroups/my-rg/providers/Microsoft.ManagedIdentity/userAssignedIdentities/my-user"
],
 "type": "UserAssigned"
 },
 "properties": {
 "environmentId": "/subscriptions/<subscription_id>/resourceGroups/rg/providers/Microsoft.App/managedEnvironments/demokube",
 "workloadProfileName": "My-GP-01",
 "configuration": {
 "ingress": {
 "external": true,
 "targetPort": 3000,
 "customDomains": [
 {
 "name": "www.my-name.com",
 "bindingType": "SniEnabled",
 "certificateId": "/subscriptions/<subscription_id>/resourceGroups/rg/providers/Microsoft.App/managedEnvironments/demokube/certificates/my-certificate-for-my-name-dot-com"
 },
 {
 "name": "www.my-other-name.com",
 "bindingType": "SniEnabled",
 "certificateId": "/subscriptions/<subscription_id>/resourceGroups/rg/providers/Microsoft.App/managedEnvironments/demokube/certificates/my-certificate-for-my-other-name-dot-com"
 }
],
 "traffic": [
 {
 "weight": 100,
 "revisionName": "testcontainerApp0-ab1234",
 "label": "production"
 }
],
 "ipSecurityRestrictions": [
 {
 "name": "Allow work IP A subnet",
 "ipAddress": "192.168.1.0/24"
 }
]
 }
 }
 }
}
```

```
 "description": "Allowing all IP's within the subnet below to
access containerapp",
 "ipAddressRange": "192.168.1.1/32",
 "action": "Allow"
 },
 {
 "name": "Allow work IP B subnet",
 "description": "Allowing all IP's within the subnet below to
access containerapp",
 "ipAddressRange": "192.168.1.1/8",
 "action": "Allow"
 }
],
"stickySessions": {
 "affinity": "sticky"
},
"clientCertificateMode": "accept",
"corsPolicy": {
 "allowedOrigins": [
 "https://a.test.com",
 "https://b.test.com"
],
 "allowedMethods": [
 "GET",
 "POST"
],
 "allowedHeaders": [
 "HEADER1",
 "HEADER2"
],
 "exposeHeaders": [
 "HEADER3",
 "HEADER4"
],
 "maxAge": 1234,
 "allowCredentials": true
},
},
"dapr": {
 "enabled": true,
 "appPort": 3000,
 "appProtocol": "http",
 "httpReadBufferSize": 30,
 "httpMaxRequestSize": 10,
 "logLevel": "debug",
 "enableApiLogging": true
},
"maxInactiveRevisions": 10,
"service": {
 "type": "redis"
},
},
"template": {
 "containers": [
 {
 "image": "mcr.microsoft.com/redis:alpine3.12",
 "name": "redis",
 "ports": [
 {
 "containerPort": 6379,
 "protocol": "TCP"
 }
],
 "resources": {
 "limits": {
 "cpu": "0.5",
 "memory": "512Mi"
 },
 "requests": {
 "cpu": "0.5",
 "memory": "512Mi"
 }
 }
 }
]
}
```

```
"image": "repo/testcontainerApp0:v1",
"name": "testcontainerApp0",
"probes": [
 {
 "type": "Liveness",
 "httpGet": {
 "path": "/health",
 "port": 8080,
 "httpHeaders": [
 {
 "name": "Custom-Header",
 "value": "Awesome"
 }
]
 },
 "initialDelaySeconds": 3,
 "periodSeconds": 3
 }
],
"volumeMounts": [
 {
 "mountPath": "/myempty",
 "volumeName": "myempty"
 },
 {
 "mountPath": "/myfiles",
 "volumeName": "azure-files-volume"
 },
 {
 "mountPath": "/mysecrets",
 "volumeName": "mysecrets"
 }
],
"initContainers": [
 {
 "image": "repo/testcontainerApp0:v4",
 "name": "testinitcontainerApp0",
 "resources": {
 "cpu": 0.2,
 "memory": "100Mi"
 },
 "command": [
 "/bin/sh"
],
 "args": [
 "-c",
 "while true; do echo hello; sleep 10;done"
]
 }
],
"scale": {
 "minReplicas": 1,
 "maxReplicas": 5,
```

```

"rules": [
 {
 "name": "httpscalingrule",
 "custom": {
 "type": "http",
 "metadata": {
 "concurrentRequests": "50"
 }
 }
 }
],
},
"volumes": [
{
 "name": "myempty",
 "storageType": "EmptyDir"
},
{
 "name": "azure-files-volume",
 "storageType": "AzureFile",
 "storageName": "myazurefiles"
},
{
 "name": "mysecrets",
 "storageType": "Secret",
 "secrets": [
 {
 "secretRef": "mysecret",
 "path": "mysecret.txt"
 }
]
},
],
"serviceBinds": [
{
 "serviceId":
 "/subscriptions/<subscription_id>/resourceGroups/rg/providers/Microsoft.
 App/containerApps/redisService",
 "name": "redisService"
}
]
}
}

```

## Container Apps job

The following tables describe the commonly used properties in the Container Apps job resource. For a complete list of properties, see [Azure Container Apps REST API reference](#).

# Resource

A Container Apps job resource's `properties` object includes the following properties:

[+] Expand table

Property	Description	Data type	Read only
<code>environmentId</code>	The environment ID for your Container Apps job. This <b>property is required to create a Container Apps job</b> . If you're using YAML, you can specify the environment ID using the <code>--environment</code> option in the Azure CLI instead.	string	No

The `environmentId` value takes the following form:

```
Console
/subscriptions/<SUBSCRIPTION_ID>/resourcegroups/<RESOURCE_GROUP_NAME>/providers/Microsoft.App/environmentId/<ENVIRONMENT_NAME>
```

In this example, you put your values in place of the placeholder tokens surrounded by `<>` brackets.

## `properties.configuration`

A resource's `properties.configuration` object includes the following properties:

[+] Expand table

Property	Description	Data type
<code>triggerType</code>	The type of trigger for a Container Apps job. For specific configuration for each trigger type, see <a href="#">Jobs trigger types</a>	string
<code>replicaTimeout</code>	The timeout in seconds for a Container Apps job.	integer
<code>replicaRetryLimit</code>	The number of times to retry a Container Apps job.	integer

## `properties.template`

A resource's `properties.template` object includes the following properties:

Property	Description	Data type
containers	Configuration object that defines what container images are included in the job.	object
scale	Configuration object that defines scale rules for the job.	object

## Examples

ARM template

The following example ARM template snippet deploys a Container Apps job.

JSON

```
{
 "identity": {
 "userAssignedIdentities": {
 "/subscriptions/<subscription_id>/resourcegroups/my-
 rg/providers/Microsoft.ManagedIdentity/userAssignedIdentities/my-user": {
 ...
 },
 "type": "UserAssigned"
 },
 "properties": {
 "environmentId":
 "/subscriptions/<subscription_id>/resourceGroups/rg/providers/Microsoft.
 App/managedEnvironments/demokube",
 "configuration": {
 "replicaTimeout": 10,
 "replicaRetryLimit": 10,
 "manualTriggerConfig": {
 "replicaCompletionCount": 1,
 "parallelism": 4
 },
 "triggerType": "Manual"
 },
 "template": {
 "containers": [
 {
 "image": "repo/testcontainerAppsJob0:v1",
 "name": "testcontainerAppsJob0",
 "probes": [
 {
 "type": "Liveness",
 "httpGet": {
 ...
 }
 }
]
 }
]
 }
 }
 }
}
```

```
 "path": "/health",
 "port": 8080,
 "httpHeaders": [
 {
 "name": "Custom-Header",
 "value": "Awesome"
 }
],
 "initialDelaySeconds": 5,
 "periodSeconds": 3
 }
],
"volumeMounts": [
{
 "mountPath": "/myempty",
 "volumeName": "myempty"
},
{
 "mountPath": "/myfile",
 "volumeName": "azure-files-volume"
},
{
 "mountPath": "/mysecrets",
 "volumeName": "mysecrets"
}
]
],
"initContainers": [
{
 "image": "repo/testcontainerAppsJob0:v4",
 "name": "testinitcontainerAppsJob0",
 "resources": {
 "cpu": 0.2,
 "memory": "100Mi"
 },
 "command": [
 "/bin/sh"
],
 "args": [
 "-c",
 "while true; do echo hello; sleep 10;done"
]
}
],
"volumes": [
{
 "name": "myempty",
 "storageType": "EmptyDir"
},
{
 "name": "azure-files-volume",
 "storageType": "AzureFile",
 "storageName": "myazurefiles"
}
```

```
 },
{
 "name": "mysecrets",
 "storageType": "Secret",
 "secrets": [
 {
 "secretRef": "mysecret",
 "path": "mysecret.txt"
 }
]
}
}
```

## Feedback

Was this page helpful?



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

# az containerapp

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 Azure Container Apps.

## Commands

[Expand table](#)

Name	Description	Type	Status
<a href="#">az containerapp add-on</a>	Commands to manage add-ons available within the environment.	Extension	Preview
<a href="#">az containerapp add-on kafka</a>	Commands to manage the kafka add-on for the Container Apps environment.	Extension	Preview
<a href="#">az containerapp add-on kafka create</a>	Command to create the kafka add-on.	Extension	Preview
<a href="#">az containerapp add-on kafka delete</a>	Command to delete the kafka add-on.	Extension	Preview
<a href="#">az containerapp add-on list</a>	List all add-ons within the environment.	Extension	Preview
<a href="#">az containerapp add-on mariadb</a>	Commands to manage the mariadb add-on for the Container Apps environment.	Extension	Preview
<a href="#">az containerapp add-on mariadb create</a>	Command to create the mariadb add-on.	Extension	Preview
<a href="#">az containerapp add-on mariadb delete</a>	Command to delete the mariadb add-on.	Extension	Preview
<a href="#">az containerapp add-on milvus</a>	Commands to manage the milvus add-on for the Container Apps environment.	Extension	Preview

Name	Description	Type	Status
<a href="#">az containerapp add-on milvus create</a>	Command to create the milvus add-on.	Extension	Preview
<a href="#">az containerapp add-on milvus delete</a>	Command to delete the milvus service.	Extension	Preview
<a href="#">az containerapp add-on postgres</a>	Commands to manage the postgres add-on for the Container Apps environment.	Extension	Preview
<a href="#">az containerapp add-on postgres create</a>	Command to create the postgres add-on.	Extension	Preview
<a href="#">az containerapp add-on postgres delete</a>	Command to delete the postgres add-on.	Extension	Preview
<a href="#">az containerapp add-on qdrant</a>	Commands to manage the qdrant add-on for the Container Apps environment.	Extension	Preview
<a href="#">az containerapp add-on qdrant create</a>	Command to create the qdrant add-on.	Extension	Preview
<a href="#">az containerapp add-on qdrant delete</a>	Command to delete the qdrant add-on.	Extension	Preview
<a href="#">az containerapp add-on redis</a>	Commands to manage the redis add-on for the Container Apps environment.	Extension	Preview
<a href="#">az containerapp add-on redis create</a>	Command to create the redis add-on.	Extension	Preview
<a href="#">az containerapp add-on redis delete</a>	Command to delete the redis add-on.	Extension	Preview
<a href="#">az containerapp add-on weaviate</a>	Commands to manage the weaviate add-on for the Container Apps environment.	Extension	Preview
<a href="#">az containerapp add-on weaviate create</a>	Command to create the weaviate add-on.	Extension	Preview
<a href="#">az containerapp add-on weaviate delete</a>	Command to delete the weaviate service.	Extension	Preview
<a href="#">az containerapp auth</a>	Manage containerapp authentication and authorization.	Core and Extension	GA
<a href="#">az containerapp auth apple</a>	Manage containerapp authentication and authorization of the Apple identity provider.	Core	GA

Name	Description	Type	Status
<a href="#">az containerapp auth apple show</a>	Show the authentication settings for the Apple identity provider.	Core	GA
<a href="#">az containerapp auth apple update</a>	Update the client id and client secret for the Apple identity provider.	Core	GA
<a href="#">az containerapp auth facebook</a>	Manage containerapp authentication and authorization of the Facebook identity provider.	Core	GA
<a href="#">az containerapp auth facebook show</a>	Show the authentication settings for the Facebook identity provider.	Core	GA
<a href="#">az containerapp auth facebook update</a>	Update the app id and app secret for the Facebook identity provider.	Core	GA
<a href="#">az containerapp auth github</a>	Manage containerapp authentication and authorization of the GitHub identity provider.	Core	GA
<a href="#">az containerapp auth github show</a>	Show the authentication settings for the GitHub identity provider.	Core	GA
<a href="#">az containerapp auth github update</a>	Update the client id and client secret for the GitHub identity provider.	Core	GA
<a href="#">az containerapp auth google</a>	Manage containerapp authentication and authorization of the Google identity provider.	Core	GA
<a href="#">az containerapp auth google show</a>	Show the authentication settings for the Google identity provider.	Core	GA
<a href="#">az containerapp auth google update</a>	Update the client id and client secret for the Google identity provider.	Core	GA
<a href="#">az containerapp auth microsoft</a>	Manage containerapp authentication and authorization of the Microsoft identity provider.	Core	GA
<a href="#">az containerapp auth microsoft show</a>	Show the authentication settings for the Azure Active Directory identity provider.	Core	GA
<a href="#">az containerapp auth microsoft update</a>	Update the client id and client secret for the Azure Active Directory identity provider.	Core	GA
<a href="#">az containerapp auth openid-connect</a>	Manage containerapp authentication and authorization of the custom OpenID Connect identity providers.	Core	GA

Name	Description	Type	Status
<a href="#">az containerapp auth openid-connect add</a>	Configure a new custom OpenID Connect identity provider.	Core	GA
<a href="#">az containerapp auth openid-connect remove</a>	Removes an existing custom OpenID Connect identity provider.	Core	GA
<a href="#">az containerapp auth openid-connect show</a>	Show the authentication settings for the custom OpenID Connect identity provider.	Core	GA
<a href="#">az containerapp auth openid-connect update</a>	Update the client id and client secret setting name for an existing custom OpenID Connect identity provider.	Core	GA
<a href="#">az containerapp auth show</a>	Show the authentication settings for the containerapp.	Core	GA
<a href="#">az containerapp auth show (containerapp extension)</a>	Show the authentication settings for the containerapp.	Extension	GA
<a href="#">az containerapp auth twitter</a>	Manage containerapp authentication and authorization of the Twitter identity provider.	Core	GA
<a href="#">az containerapp auth twitter show</a>	Show the authentication settings for the Twitter identity provider.	Core	GA
<a href="#">az containerapp auth twitter update</a>	Update the consumer key and consumer secret for the Twitter identity provider.	Core	GA
<a href="#">az containerapp auth update</a>	Update the authentication settings for the containerapp.	Core	GA
<a href="#">az containerapp auth update (containerapp extension)</a>	Update the authentication settings for the containerapp.	Extension	GA
<a href="#">az containerapp browse</a>	Open a containerapp in the browser, if possible.	Core	GA
<a href="#">az containerapp compose</a>	Commands to create Azure Container Apps from Compose specifications.	Core and Extension	GA
<a href="#">az containerapp compose create</a>	Create one or more Container Apps in a new or existing Container App Environment from a Compose specification.	Core	GA

Name	Description	Type	Status
<a href="#">az containerapp compose create (containerapp extension)</a>	Create one or more Container Apps in a new or existing Container App Environment from a Compose specification.	Extension	GA
<a href="#">az containerapp connected-env</a>	Commands to manage Container Apps Connected environments for use with Arc enabled Container Apps.	Extension	Preview
<a href="#">az containerapp connected-env certificate</a>	Commands to manage certificates for the Container Apps connected environment.	Extension	Preview
<a href="#">az containerapp connected-env certificate delete</a>	Delete a certificate from the Container Apps connected environment.	Extension	Preview
<a href="#">az containerapp connected-env certificate list</a>	List certificates for a connected environment.	Extension	Preview
<a href="#">az containerapp connected-env certificate upload</a>	Add or update a certificate.	Extension	Preview
<a href="#">az containerapp connected-env create</a>	Create a Container Apps connected environment.	Extension	Preview
<a href="#">az containerapp connected-env dapr-component</a>	Commands to manage Dapr components for Container Apps connected environments.	Extension	Preview
<a href="#">az containerapp connected-env dapr-component list</a>	List Dapr components for a connected environment.	Extension	Preview
<a href="#">az containerapp connected-env dapr-component remove</a>	Remove a Dapr component from a connected environment.	Extension	Preview
<a href="#">az containerapp connected-env dapr-component set</a>	Create or update a Dapr component.	Extension	Preview
<a href="#">az containerapp connected-env dapr-component show</a>	Show the details of a Dapr component.	Extension	Preview
<a href="#">az containerapp</a>	Delete a Container Apps connected	Extension	Preview

Name	Description	Type	Status
<a href="#">az containerapp connected-env delete</a>	environment.		
<a href="#">az containerapp connected-env list</a>	List Container Apps connected environments by subscription or resource group.	Extension	Preview
<a href="#">az containerapp connected-env show</a>	Show details of a Container Apps connected environment.	Extension	Preview
<a href="#">az containerapp connected-env storage</a>	Commands to manage storage for the Container Apps connected environment.	Extension	Preview
<a href="#">az containerapp connected-env storage list</a>	List the storages for a connected environment.	Extension	Preview
<a href="#">az containerapp connected-env storage remove</a>	Remove a storage from a connected environment.	Extension	Preview
<a href="#">az containerapp connected-env storage set</a>	Create or update a storage.	Extension	Preview
<a href="#">az containerapp connected-env storage show</a>	Show the details of a storage.	Extension	Preview
<a href="#">az containerapp connection</a>	Commands to manage containerapp connections.	Core and Extension	GA
<a href="#">az containerapp connection create</a>	Create a connection between a containerapp and a target resource.	Core and Extension	GA
<a href="#">az containerapp connection create app-insights</a>	Create a containerapp connection to app-insights.	Core	GA
<a href="#">az containerapp connection create appconfig</a>	Create a containerapp connection to appconfig.	Core	GA
<a href="#">az containerapp connection create cognitiveservices</a>	Create a containerapp connection to cognitiveservices.	Core	GA
<a href="#">az containerapp connection create confluent-cloud</a>	Create a containerapp connection to confluent-cloud.	Core	GA
<a href="#">az containerapp connection create containerapp</a>	Create a containerapp-to-containerapp connection.	Core	GA
<a href="#">az containerapp connection create cosmos-cassandra</a>	Create a containerapp connection to cosmos-cassandra.	Core	GA

Name	Description	Type	Status
<a href="#">az containerapp connection create cosmos-gremlin</a>	Create a containerapp connection to cosmos-gremlin.	Core	GA
<a href="#">az containerapp connection create cosmos-mongo</a>	Create a containerapp connection to cosmos-mongo.	Core	GA
<a href="#">az containerapp connection create cosmos-sql</a>	Create a containerapp connection to cosmos-sql.	Core	GA
<a href="#">az containerapp connection create cosmos-table</a>	Create a containerapp connection to cosmos-table.	Core	GA
<a href="#">az containerapp connection create eventhub</a>	Create a containerapp connection to eventhub.	Core	GA
<a href="#">az containerapp connection create keyvault</a>	Create a containerapp connection to keyvault.	Core	GA
<a href="#">az containerapp connection create mysql</a>	Create a containerapp connection to mysql.	Core	Deprecated
<a href="#">az containerapp connection create mysql-flexible</a>	Create a containerapp connection to mysql-flexible.	Core	GA
<a href="#">az containerapp connection create mysql-flexible (serviceconnector-passwordless extension)</a>	Create a containerapp connection to mysql-flexible.	Extension	GA
<a href="#">az containerapp connection create postgres</a>	Create a containerapp connection to postgres.	Core	Deprecated
<a href="#">az containerapp connection create postgres-flexible</a>	Create a containerapp connection to postgres-flexible.	Core	GA
<a href="#">az containerapp connection create postgres-flexible (serviceconnector-passwordless extension)</a>	Create a containerapp connection to postgres-flexible.	Extension	GA
<a href="#">az containerapp connection create redis</a>	Create a containerapp connection to redis.	Core	GA
<a href="#">az containerapp connection create redis-enterprise</a>	Create a containerapp connection to redis-enterprise.	Core	GA
<a href="#">az containerapp connection create servicebus</a>	Create a containerapp connection to servicebus.	Core	GA

Name	Description	Type	Status
<a href="#">az containerapp connection create signalr</a>	Create a containerapp connection to signalr.	Core	GA
<a href="#">az containerapp connection create sql</a>	Create a containerapp connection to sql.	Core	GA
<a href="#">az containerapp connection create sql (serviceconnector-passwordless extension)</a>	Create a containerapp connection to sql.	Extension	GA
<a href="#">az containerapp connection create storage-blob</a>	Create a containerapp connection to storage-blob.	Core	GA
<a href="#">az containerapp connection create storage-file</a>	Create a containerapp connection to storage-file.	Core	GA
<a href="#">az containerapp connection create storage-queue</a>	Create a containerapp connection to storage-queue.	Core	GA
<a href="#">az containerapp connection create storage-table</a>	Create a containerapp connection to storage-table.	Core	GA
<a href="#">az containerapp connection create webpubsub</a>	Create a containerapp connection to webpubsub.	Core	GA
<a href="#">az containerapp connection delete</a>	Delete a containerapp connection.	Core	GA
<a href="#">az containerapp connection list</a>	List connections of a containerapp.	Core	GA
<a href="#">az containerapp connection list-configuration</a>	List source configurations of a containerapp connection.	Core	GA
<a href="#">az containerapp connection list-support-types</a>	List client types and auth types supported by containerapp connections.	Core	GA
<a href="#">az containerapp connection show</a>	Get the details of a containerapp connection.	Core	GA
<a href="#">az containerapp connection update</a>	Update a containerapp connection.	Core	GA
<a href="#">az containerapp connection update app-insights</a>	Update a containerapp to app-insights connection.	Core	GA
<a href="#">az containerapp connection update appconfig</a>	Update a containerapp to appconfig connection.	Core	GA

Name	Description	Type	Status
<a href="#">az containerapp connection update cognitiveservices</a>	Update a containerapp to cognitiveservices connection.	Core	GA
<a href="#">az containerapp connection update confluent-cloud</a>	Update a containerapp to confluent-cloud connection.	Core	GA
<a href="#">az containerapp connection update containerapp</a>	Update a containerapp-to-containerapp connection.	Core	GA
<a href="#">az containerapp connection update cosmos-cassandra</a>	Update a containerapp to cosmos-cassandra connection.	Core	GA
<a href="#">az containerapp connection update cosmos-gremlin</a>	Update a containerapp to cosmos-gremlin connection.	Core	GA
<a href="#">az containerapp connection update cosmos-mongo</a>	Update a containerapp to cosmos-mongo connection.	Core	GA
<a href="#">az containerapp connection update cosmos-sql</a>	Update a containerapp to cosmos-sql connection.	Core	GA
<a href="#">az containerapp connection update cosmos-table</a>	Update a containerapp to cosmos-table connection.	Core	GA
<a href="#">az containerapp connection update eventhub</a>	Update a containerapp to eventhub connection.	Core	GA
<a href="#">az containerapp connection update keyvault</a>	Update a containerapp to keyvault connection.	Core	GA
<a href="#">az containerapp connection update mysql</a>	Update a containerapp to mysql connection.	Core	Deprecated
<a href="#">az containerapp connection update mysql-flexible</a>	Update a containerapp to mysql-flexible connection.	Core	GA
<a href="#">az containerapp connection update postgres</a>	Update a containerapp to postgres connection.	Core	Deprecated
<a href="#">az containerapp connection update postgres-flexible</a>	Update a containerapp to postgres-flexible connection.	Core	GA
<a href="#">az containerapp connection update redis</a>	Update a containerapp to redis connection.	Core	GA
<a href="#">az containerapp connection update redis-enterprise</a>	Update a containerapp to redis-enterprise connection.	Core	GA
<a href="#">az containerapp connection update servicebus</a>	Update a containerapp to servicebus connection.	Core	GA

Name	Description	Type	Status
<a href="#">az containerapp connection update signalr</a>	Update a containerapp to signalr connection.	Core	GA
<a href="#">az containerapp connection update sql</a>	Update a containerapp to sql connection.	Core	GA
<a href="#">az containerapp connection update storage-blob</a>	Update a containerapp to storage-blob connection.	Core	GA
<a href="#">az containerapp connection update storage-file</a>	Update a containerapp to storage-file connection.	Core	GA
<a href="#">az containerapp connection update storage-queue</a>	Update a containerapp to storage-queue connection.	Core	GA
<a href="#">az containerapp connection update storage-table</a>	Update a containerapp to storage-table connection.	Core	GA
<a href="#">az containerapp connection update webpubsub</a>	Update a containerapp to webpubsub connection.	Core	GA
<a href="#">az containerapp connection validate</a>	Validate a containerapp connection.	Core	GA
<a href="#">az containerapp connection wait</a>	Place the CLI in a waiting state until a condition of the connection is met.	Core	GA
<a href="#">az containerapp create</a>	Create a container app.	Core	GA
<a href="#">az containerapp create (containerapp extension)</a>	Create a container app.	Extension	GA
<a href="#">az containerapp dapr</a>	Commands to manage Dapr. To manage Dapr components, see <a href="#">az containerapp env dapr-component</a> .	Core	GA
<a href="#">az containerapp dapr disable</a>	Disable Dapr for a container app. Removes existing values.	Core	GA
<a href="#">az containerapp dapr enable</a>	Enable Dapr for a container app. Updates existing values.	Core	GA
<a href="#">az containerapp delete</a>	Delete a container app.	Core	GA
<a href="#">az containerapp delete (containerapp extension)</a>	Delete a container app.	Extension	GA
<a href="#">az containerapp env</a>	Commands to manage Container Apps environments.	Core and Extension	GA

Name	Description	Type	Status
<a href="#">az containerapp env certificate</a>	Commands to manage certificates for the Container Apps environment.	Core and Extension	GA
<a href="#">az containerapp env certificate create</a>	Create a managed certificate.	Core	Preview
<a href="#">az containerapp env certificate delete</a>	Delete a certificate from the Container Apps environment.	Core	GA
<a href="#">az containerapp env certificate delete (containerapp extension)</a>	Delete a certificate from the Container Apps environment.	Extension	GA
<a href="#">az containerapp env certificate list</a>	List certificates for an environment.	Core	GA
<a href="#">az containerapp env certificate list (containerapp extension)</a>	List certificates for an environment.	Extension	GA
<a href="#">az containerapp env certificate upload</a>	Add or update a certificate.	Core	GA
<a href="#">az containerapp env certificate upload (containerapp extension)</a>	Add or update a certificate.	Extension	GA
<a href="#">az containerapp env create</a>	Create a Container Apps environment.	Core	GA
<a href="#">az containerapp env create (containerapp extension)</a>	Create a Container Apps environment.	Extension	GA
<a href="#">az containerapp env dapr-component</a>	Commands to manage Dapr components for the Container Apps environment.	Core and Extension	GA
<a href="#">az containerapp env dapr-component init</a>	Initializes Dapr components and dev services for an environment.	Extension	Preview
<a href="#">az containerapp env dapr-component list</a>	List Dapr components for an environment.	Core	GA
<a href="#">az containerapp env dapr-component remove</a>	Remove a Dapr component from an environment.	Core	GA
<a href="#">az containerapp env dapr-component resiliency</a>	Commands to manage resiliency policies for a dapr component.	Extension	Preview
<a href="#">az containerapp env dapr-component resiliency</a>	Create resiliency policies for a dapr component.	Extension	Preview

Name	Description	Type	Status
<a href="#">create</a>			
<a href="#">az containerapp env dapr-component resiliency delete</a>	Delete resiliency policies for a dapr component.	Extension	Preview
<a href="#">az containerapp env dapr-component resiliency list</a>	List resiliency policies for a dapr component.	Extension	Preview
<a href="#">az containerapp env dapr-component resiliency show</a>	Show resiliency policies for a dapr component.	Extension	Preview
<a href="#">az containerapp env dapr-component resiliency update</a>	Update resiliency policies for a dapr component.	Extension	Preview
<a href="#">az containerapp env dapr-component set</a>	Create or update a Dapr component.	Core	GA
<a href="#">az containerapp env dapr-component show</a>	Show the details of a Dapr component.	Core	GA
<a href="#">az containerapp env delete</a>	Delete a Container Apps environment.	Core	GA
<a href="#">az containerapp env delete (containerapp extension)</a>	Delete a Container Apps environment.	Extension	GA
<a href="#">az containerapp env dotnet-component</a>	Commands to manage DotNet components within the environment.	Extension	Preview
<a href="#">az containerapp env dotnet-component create</a>	Command to create DotNet component to enable Aspire Dashboard. Supported DotNet component type is Aspire Dashboard.	Extension	Preview
<a href="#">az containerapp env dotnet-component delete</a>	Command to delete DotNet component to disable Aspire Dashboard.	Extension	Preview
<a href="#">az containerapp env dotnet-component list</a>	Command to list DotNet components within the environment.	Extension	Preview
<a href="#">az containerapp env dotnet-component show</a>	Command to show DotNet component in environment.	Extension	Preview
<a href="#">az containerapp env identity</a>	Commands to manage environment managed identities.	Extension	Preview
<a href="#">az containerapp env identity assign</a>	Assign managed identity to a managed environment.	Extension	Preview

Name	Description	Type	Status
<a href="#">az containerapp env identity remove</a>	Remove a managed identity from a managed environment.	Extension	Preview
<a href="#">az containerapp env identity show</a>	Show managed identities of a managed environment.	Extension	Preview
<a href="#">az containerapp env java-component</a>	Commands to manage Java components within the environment.	Extension	GA
<a href="#">az containerapp env java-component admin-for-spring</a>	Commands to manage the Admin for Spring for the Container Apps environment.	Extension	GA
<a href="#">az containerapp env java-component admin-for-spring create</a>	Command to create the Admin for Spring.	Extension	GA
<a href="#">az containerapp env java-component admin-for-spring delete</a>	Command to delete the Admin for Spring.	Extension	GA
<a href="#">az containerapp env java-component admin-for-spring show</a>	Command to show the Admin for Spring.	Extension	GA
<a href="#">az containerapp env java-component admin-for-spring update</a>	Command to update the Admin for Spring.	Extension	GA
<a href="#">az containerapp env java-component config-server-for-spring</a>	Commands to manage the Config Server for Spring for the Container Apps environment.	Extension	GA
<a href="#">az containerapp env java-component config-server-for-spring create</a>	Command to create the Config Server for Spring.	Extension	GA
<a href="#">az containerapp env java-component config-server-for-spring delete</a>	Command to delete the Config Server for Spring.	Extension	GA
<a href="#">az containerapp env java-component config-server-for-spring show</a>	Command to show the Config Server for Spring.	Extension	GA
<a href="#">az containerapp env java-component config-server-for-spring update</a>	Command to update the Config Server for Spring.	Extension	GA

Name	Description	Type	Status
<a href="#">az containerapp env java-component eureka-server-for-spring</a>	Commands to manage the Eureka Server for Spring for the Container Apps environment.	Extension	GA
<a href="#">az containerapp env java-component eureka-server-for-spring create</a>	Command to create the Eureka Server for Spring.	Extension	GA
<a href="#">az containerapp env java-component eureka-server-for-spring delete</a>	Command to delete the Eureka Server for Spring.	Extension	GA
<a href="#">az containerapp env java-component eureka-server-for-spring show</a>	Command to show the Eureka Server for Spring.	Extension	GA
<a href="#">az containerapp env java-component eureka-server-for-spring update</a>	Command to update the Eureka Server for Spring.	Extension	GA
<a href="#">az containerapp env java-component list</a>	List all Java components within the environment.	Extension	GA
<a href="#">az containerapp env java-component nacos</a>	Commands to manage the Nacos for the Container Apps environment.	Extension	GA
<a href="#">az containerapp env java-component nacos create</a>	Command to create the Nacos.	Extension	GA
<a href="#">az containerapp env java-component nacos delete</a>	Command to delete the Nacos.	Extension	GA
<a href="#">az containerapp env java-component nacos show</a>	Command to show the Nacos.	Extension	GA
<a href="#">az containerapp env java-component nacos update</a>	Command to update the Nacos.	Extension	GA
<a href="#">az containerapp env java-component spring-cloud-config</a>	Commands to manage the Config Server for Spring for the Container Apps environment.	Extension	Deprecated
<a href="#">az containerapp env java-component spring-cloud-config create</a>	Command to create the Spring Cloud Config.	Extension	Deprecated
<a href="#">az containerapp env java-component spring-cloud-config delete</a>	Command to delete the Spring Cloud Config.	Extension	Deprecated

Name	Description	Type	Status
<a href="#">az containerapp env java-component spring-cloud-config show</a>	Command to show the Spring Cloud Config.	Extension	Deprecated
<a href="#">az containerapp env java-component spring-cloud-config update</a>	Command to update the Spring Cloud Config.	Extension	Deprecated
<a href="#">az containerapp env java-component spring-cloud-eureka</a>	Commands to manage the Spring Cloud Eureka for the Container Apps environment.	Extension	Deprecated
<a href="#">az containerapp env java-component spring-cloud-eureka create</a>	Command to create the Spring Cloud Eureka.	Extension	Deprecated
<a href="#">az containerapp env java-component spring-cloud-eureka delete</a>	Command to delete the Spring Cloud Eureka.	Extension	Deprecated
<a href="#">az containerapp env java-component spring-cloud-eureka show</a>	Command to show the Spring Cloud Eureka.	Extension	Deprecated
<a href="#">az containerapp env java-component spring-cloud-eureka update</a>	Command to update the Spring Cloud Eureka.	Extension	Deprecated
<a href="#">az containerapp env list</a>	List Container Apps environments by subscription or resource group.	Core	GA
<a href="#">az containerapp env list (containerapp extension)</a>	List Container Apps environments by subscription or resource group.	Extension	GA
<a href="#">az containerapp env list-usages</a>	List usages of quotas for specific managed environment.	Core	GA
<a href="#">az containerapp env logs</a>	Show container app environment logs.	Core	GA
<a href="#">az containerapp env logs show</a>	Show past environment logs and/or print logs in real time (with the --follow parameter).	Core	GA
<a href="#">az containerapp env show</a>	Show details of a Container Apps environment.	Core	GA
<a href="#">az containerapp env show (containerapp extension)</a>	Show details of a Container Apps environment.	Extension	GA

Name	Description	Type	Status
<a href="#">az containerapp env storage</a>	Commands to manage storage for the Container Apps environment.	Core and Extension	GA
<a href="#">az containerapp env storage list</a>	List the storages for an environment.	Core	GA
<a href="#">az containerapp env storage list (containerapp extension)</a>	List the storages for an environment.	Extension	GA
<a href="#">az containerapp env storage remove</a>	Remove a storage from an environment.	Core	GA
<a href="#">az containerapp env storage remove (containerapp extension)</a>	Remove a storage from an environment.	Extension	GA
<a href="#">az containerapp env storage set</a>	Create or update a storage.	Core	GA
<a href="#">az containerapp env storage set (containerapp extension)</a>	Create or update a storage.	Extension	GA
<a href="#">az containerapp env storage show</a>	Show the details of a storage.	Core	GA
<a href="#">az containerapp env storage show (containerapp extension)</a>	Show the details of a storage.	Extension	GA
<a href="#">az containerapp env telemetry</a>	Commands to manage telemetry settings for the container apps environment.	Extension	Preview
<a href="#">az containerapp env telemetry app-insights</a>	Commands to manage app insights settings for the container apps environment.	Extension	Preview
<a href="#">az containerapp env telemetry app-insights delete</a>	Delete container apps environment telemetry app insights settings.	Extension	Preview
<a href="#">az containerapp env telemetry app-insights set</a>	Create or update container apps environment telemetry app insights settings.	Extension	Preview
<a href="#">az containerapp env telemetry app-insights</a>	Show container apps environment telemetry app insights settings.	Extension	Preview

Name	Description	Type	Status
<a href="#">show</a>			
<a href="#">az containerapp env telemetry data-dog</a>	Commands to manage data dog settings for the container apps environment.	Extension	Preview
<a href="#">az containerapp env telemetry data-dog delete</a>	Delete container apps environment telemetry data dog settings.	Extension	Preview
<a href="#">az containerapp env telemetry data-dog set</a>	Create or update container apps environment telemetry data dog settings.	Extension	Preview
<a href="#">az containerapp env telemetry data-dog show</a>	Show container apps environment telemetry data dog settings.	Extension	Preview
<a href="#">az containerapp env telemetry otlp</a>	Commands to manage otlp settings for the container apps environment.	Extension	Preview
<a href="#">az containerapp env telemetry otlp add</a>	Add container apps environment telemetry otlp settings.	Extension	Preview
<a href="#">az containerapp env telemetry otlp list</a>	List container apps environment telemetry otlp settings.	Extension	Preview
<a href="#">az containerapp env telemetry otlp remove</a>	Remove container apps environment telemetry otlp settings.	Extension	Preview
<a href="#">az containerapp env telemetry otlp show</a>	Show container apps environment telemetry otlp settings.	Extension	Preview
<a href="#">az containerapp env telemetry otlp update</a>	Update container apps environment telemetry otlp settings.	Extension	Preview
<a href="#">az containerapp env update</a>	Update a Container Apps environment.	Core	GA
<a href="#">az containerapp env update (containerapp extension)</a>	Update a Container Apps environment.	Extension	GA
<a href="#">az containerapp env workload-profile</a>	Manage the workload profiles of a Container Apps environment.	Core and Extension	GA
<a href="#">az containerapp env workload-profile add</a>	Create a workload profile in a Container Apps environment.	Core	GA
<a href="#">az containerapp env workload-profile delete</a>	Delete a workload profile from a Container Apps environment.	Core	GA

Name	Description	Type	Status
<a href="#">az containerapp env workload-profile list</a>	List the workload profiles from a Container Apps environment.	Core	GA
<a href="#">az containerapp env workload-profile list-supported</a>	List the supported workload profiles in a region.	Core	GA
<a href="#">az containerapp env workload-profile set</a>	Create or update an existing workload profile in a Container Apps environment.	Extension	Deprecated
<a href="#">az containerapp env workload-profile show</a>	Show a workload profile from a Container Apps environment.	Core	GA
<a href="#">az containerapp env workload-profile update</a>	Update an existing workload profile in a Container Apps environment.	Core	GA
<a href="#">az containerapp exec</a>	Open an SSH-like interactive shell within a container app replica.	Core	GA
<a href="#">az containerapp github-action</a>	Commands to manage GitHub Actions.	Core and Extension	GA
<a href="#">az containerapp github-action add</a>	Add a GitHub Actions workflow to a repository to deploy a container app.	Core	GA
<a href="#">az containerapp github-action add (containerapp extension)</a>	Add a GitHub Actions workflow to a repository to deploy a container app.	Extension	GA
<a href="#">az containerapp github-action delete</a>	Remove a previously configured Container Apps GitHub Actions workflow from a repository.	Core	GA
<a href="#">az containerapp github-action show</a>	Show the GitHub Actions configuration on a container app.	Core	GA
<a href="#">az containerapp hostname</a>	Commands to manage hostnames of a container app.	Core and Extension	GA
<a href="#">az containerapp hostname add</a>	Add the hostname to a container app without binding.	Core	GA
<a href="#">az containerapp hostname bind</a>	Add or update the hostname and binding with a certificate.	Core	GA
<a href="#">az containerapp hostname bind (containerapp extension)</a>	Add or update the hostname and binding with a certificate.	Extension	GA

Name	Description	Type	Status
<a href="#">az containerapp hostname delete</a>	Delete hostnames from a container app.	Core	GA
<a href="#">az containerapp hostname list</a>	List the hostnames of a container app.	Core	GA
<a href="#">az containerapp identity</a>	Commands to manage managed identities.	Core	GA
<a href="#">az containerapp identity assign</a>	Assign managed identity to a container app.	Core	GA
<a href="#">az containerapp identity remove</a>	Remove a managed identity from a container app.	Core	GA
<a href="#">az containerapp identity show</a>	Show managed identities of a container app.	Core	GA
<a href="#">az containerapp ingress</a>	Commands to manage ingress and traffic-splitting.	Core	GA
<a href="#">az containerapp ingress access-restriction</a>	Commands to manage IP access restrictions.	Core	GA
<a href="#">az containerapp ingress access-restriction list</a>	List IP access restrictions for a container app.	Core	GA
<a href="#">az containerapp ingress access-restriction remove</a>	Remove IP access restrictions from a container app.	Core	GA
<a href="#">az containerapp ingress access-restriction set</a>	Configure IP access restrictions for a container app.	Core	GA
<a href="#">az containerapp ingress cors</a>	Commands to manage CORS policy for a container app.	Core	GA
<a href="#">az containerapp ingress cors disable</a>	Disable CORS policy for a container app.	Core	GA
<a href="#">az containerapp ingress cors enable</a>	Enable CORS policy for a container app.	Core	GA
<a href="#">az containerapp ingress cors show</a>	Show CORS policy for a container app.	Core	GA
<a href="#">az containerapp ingress cors update</a>	Update CORS policy for a container app.	Core	GA
<a href="#">az containerapp ingress disable</a>	Disable ingress for a container app.	Core	GA

Name	Description	Type	Status
<a href="#">az containerapp ingress enable</a>	Enable or update ingress for a container app.	Core	GA
<a href="#">az containerapp ingress show</a>	Show details of a container app's ingress.	Core	GA
<a href="#">az containerapp ingress sticky-sessions</a>	Commands to set Sticky session affinity for a container app.	Core	GA
<a href="#">az containerapp ingress sticky-sessions set</a>	Configure Sticky session for a container app.	Core	GA
<a href="#">az containerapp ingress sticky-sessions show</a>	Show the Affinity for a container app.	Core	GA
<a href="#">az containerapp ingress traffic</a>	Commands to manage traffic-splitting.	Core	GA
<a href="#">az containerapp ingress traffic set</a>	Configure traffic-splitting for a container app.	Core	GA
<a href="#">az containerapp ingress traffic show</a>	Show traffic-splitting configuration for a container app.	Core	GA
<a href="#">az containerapp ingress update</a>	Update ingress for a container app.	Core	GA
<a href="#">az containerapp java</a>	Commands to manage Java workloads.	Extension	GA
<a href="#">az containerapp java logger</a>	Dynamically change log level for Java workloads.	Extension	GA
<a href="#">az containerapp java logger delete</a>	Delete logger for Java workloads.	Extension	GA
<a href="#">az containerapp java logger set</a>	Create or update logger for Java workloads.	Extension	GA
<a href="#">az containerapp java logger show</a>	Display logger setting for Java workloads.	Extension	GA
<a href="#">az containerapp job</a>	Commands to manage Container Apps jobs.	Core and Extension	GA
<a href="#">az containerapp job create</a>	Create a container apps job.	Core	GA
<a href="#">az containerapp job create (containerapp extension)</a>	Create a container apps job.	Extension	GA
<a href="#">az containerapp job delete</a>	Delete a Container Apps Job.	Core	GA

Name	Description	Type	Status
<a href="#">az containerapp job delete (containerapp extension)</a>	Delete a Container Apps Job.	Extension	GA
<a href="#">az containerapp job execution</a>	Commands to view executions of a Container App Job.	Core	GA
<a href="#">az containerapp job execution list</a>	Get list of all executions of a Container App Job.	Core	GA
<a href="#">az containerapp job execution show</a>	Get execution of a Container App Job.	Core	GA
<a href="#">az containerapp job identity</a>	Commands to manage managed identities for container app job.	Core	GA
<a href="#">az containerapp job identity assign</a>	Assign managed identity to a container app job.	Core	GA
<a href="#">az containerapp job identity remove</a>	Remove a managed identity from a container app job.	Core	GA
<a href="#">az containerapp job identity show</a>	Show managed identities of a container app job.	Core	GA
<a href="#">az containerapp job list</a>	List Container Apps Job by subscription or resource group.	Core	GA
<a href="#">az containerapp job list (containerapp extension)</a>	List Container Apps Job by subscription or resource group.	Extension	GA
<a href="#">az containerapp job logs</a>	Show container app job logs.	Extension	Preview
<a href="#">az containerapp job logs show</a>	Show past logs and/or print logs in real time (with the --follow parameter). Note that the logs are only taken from one execution, replica, and container.	Extension	Preview
<a href="#">az containerapp job registry</a>	Commands to manage container registry information of a Container App Job.	Core and Extension	Preview
<a href="#">az containerapp job registry list</a>	List container registries configured in a Container App Job.	Core	Preview
<a href="#">az containerapp job registry remove</a>	Remove a container registry's details in a Container App Job.	Core	Preview
<a href="#">az containerapp job registry set</a>	Add or update a container registry's details in a Container App Job.	Core	Preview

Name	Description	Type	Status
<a href="#">az containerapp job registry set (containerapp extension)</a>	Add or update a container registry's details in a Container App Job.	Extension	Preview
<a href="#">az containerapp job registry show</a>	Show details of a container registry from a Container App Job.	Core	Preview
<a href="#">az containerapp job replica</a>	Manage container app replicas.	Extension	Preview
<a href="#">az containerapp job replica list</a>	List a container app job execution's replica.	Extension	Preview
<a href="#">az containerapp job secret</a>	Commands to manage secrets.	Core	GA
<a href="#">az containerapp job secret list</a>	List the secrets of a container app job.	Core	GA
<a href="#">az containerapp job secret remove</a>	Remove secrets from a container app job.	Core	GA
<a href="#">az containerapp job secret set</a>	Create/update secrets.	Core	GA
<a href="#">az containerapp job secret show</a>	Show details of a secret.	Core	GA
<a href="#">az containerapp job show</a>	Show details of a Container Apps Job.	Core	GA
<a href="#">az containerapp job show (containerapp extension)</a>	Show details of a Container Apps Job.	Extension	GA
<a href="#">az containerapp job start</a>	Start a Container Apps Job execution.	Core	GA
<a href="#">az containerapp job stop</a>	Stops a Container Apps Job execution.	Core	GA
<a href="#">az containerapp job update</a>	Update a Container Apps Job.	Core	GA
<a href="#">az containerapp job update (containerapp extension)</a>	Update a Container Apps Job.	Extension	GA
<a href="#">az containerapp list</a>	List container apps.	Core	GA
<a href="#">az containerapp list (containerapp extension)</a>	List container apps.	Extension	GA
<a href="#">az containerapp list-usages</a>	List usages of subscription level quotas in specific region.	Core	GA
<a href="#">az containerapp logs</a>	Show container app logs.	Core	GA

Name	Description	Type	Status
<a href="#">az containerapp logs show</a>	Show past logs and/or print logs in real time (with the --follow parameter). Note that the logs are only taken from one revision, replica, and container (for non-system logs).	Core	GA
<a href="#">az containerapp patch</a>	Patch Azure Container Apps. Patching is only available for the apps built using the source to cloud feature. See <a href="https://aka.ms/aca-local-source-to-cloud">https://aka.ms/aca-local-source-to-cloud</a> .	Extension	Preview
<a href="#">az containerapp patch apply</a>	List and apply container apps to be patched. Patching is only available for the apps built using the source to cloud feature. See <a href="https://aka.ms/aca-local-source-to-cloud">https://aka.ms/aca-local-source-to-cloud</a> .	Extension	Preview
<a href="#">az containerapp patch interactive</a>	List and select container apps to be patched in an interactive way. Patching is only available for the apps built using the source to cloud feature. See <a href="https://aka.ms/aca-local-source-to-cloud">https://aka.ms/aca-local-source-to-cloud</a> .	Extension	Preview
<a href="#">az containerapp patch list</a>	List container apps that can be patched. Patching is only available for the apps built using the source to cloud feature. See <a href="https://aka.ms/aca-local-source-to-cloud">https://aka.ms/aca-local-source-to-cloud</a> .	Extension	Preview
<a href="#">az containerapp registry</a>	Commands to manage container registry information.	Core and Extension	GA
<a href="#">az containerapp registry list</a>	List container registries configured in a container app.	Core	GA
<a href="#">az containerapp registry remove</a>	Remove a container registry's details.	Core	GA
<a href="#">az containerapp registry set</a>	Add or update a container registry's details.	Core	GA
<a href="#">az containerapp registry set (containerapp extension)</a>	Add or update a container registry's details.	Extension	Preview
<a href="#">az containerapp registry show</a>	Show details of a container registry.	Core	GA

Name	Description	Type	Status
<a href="#">az containerapp replica</a>	Manage container app replicas.	Core and Extension	GA
<a href="#">az containerapp replica count</a>	Count of a container app's replica(s).	Extension	Preview
<a href="#">az containerapp replica list</a>	List a container app revision's replica.	Core	GA
<a href="#">az containerapp replica list (containerapp extension)</a>	List a container app revision's replica.	Extension	GA
<a href="#">az containerapp replica show</a>	Show a container app replica.	Core	GA
<a href="#">az containerapp replica show (containerapp extension)</a>	Show a container app replica.	Extension	GA
<a href="#">az containerapp resiliency</a>	Commands to manage resiliency policies for a container app.	Extension	Preview
<a href="#">az containerapp resiliency create</a>	Create resiliency policies for a container app.	Extension	Preview
<a href="#">az containerapp resiliency delete</a>	Delete resiliency policies for a container app.	Extension	Preview
<a href="#">az containerapp resiliency list</a>	List resiliency policies for a container app.	Extension	Preview
<a href="#">az containerapp resiliency show</a>	Show resiliency policies for a container app.	Extension	Preview
<a href="#">az containerapp resiliency update</a>	Update resiliency policies for a container app.	Extension	Preview
<a href="#">az containerapp revision</a>	Commands to manage revisions.	Core	GA
<a href="#">az containerapp revision activate</a>	Activate a revision.	Core	GA
<a href="#">az containerapp revision copy</a>	Create a revision based on a previous revision.	Core	GA
<a href="#">az containerapp revision deactivate</a>	Deactivate a revision.	Core	GA
<a href="#">az containerapp revision label</a>	Manage revision labels assigned to traffic weights.	Core	GA

Name	Description	Type	Status
<a href="#">az containerapp revision label add</a>	Set a revision label to a revision with an associated traffic weight.	Core	GA
<a href="#">az containerapp revision label remove</a>	Remove a revision label from a revision with an associated traffic weight.	Core	GA
<a href="#">az containerapp revision label swap</a>	Swap a revision label between two revisions with associated traffic weights.	Core	GA
<a href="#">az containerapp revision list</a>	List a container app's revisions.	Core	GA
<a href="#">az containerapp revision restart</a>	Restart a revision.	Core	GA
<a href="#">az containerapp revision set-mode</a>	Set the revision mode of a container app.	Core	GA
<a href="#">az containerapp revision show</a>	Show details of a revision.	Core	GA
<a href="#">az containerapp secret</a>	Commands to manage secrets.	Core	GA
<a href="#">az containerapp secret list</a>	List the secrets of a container app.	Core	GA
<a href="#">az containerapp secret remove</a>	Remove secrets from a container app.	Core	GA
<a href="#">az containerapp secret set</a>	Create/update secrets.	Core	GA
<a href="#">az containerapp secret show</a>	Show details of a secret.	Core	GA
<a href="#">az containerapp session</a>	Commands to manage sessions.To learn more about individual commands under each subgroup run <code>containerapp session [subgroup name] --help</code> .	Extension	GA
<a href="#">az containerapp session code-interpreter</a>	Commands to interact with and manage code interpreter sessions.	Extension	Preview
<a href="#">az containerapp session code-interpreter delete-file</a>	Delete a file uploaded to a code interpreter session.	Extension	Preview
<a href="#">az containerapp session code-interpreter execute</a>	Execute code in a code interpreter session.	Extension	Preview
<a href="#">az containerapp session code-interpreter list-files</a>	List files uploaded to a code interpreter session.	Extension	Preview

Name	Description	Type	Status
<a href="#">az containerapp session code-interpreter show-file-content</a>	Show the content a file uploaded to a code interpreter session.	Extension	Preview
<a href="#">az containerapp session code-interpreter show-file-metadata</a>	Shows the meta-data content a file uploaded to a code interpreter session.	Extension	Preview
<a href="#">az containerapp session code-interpreter upload-file</a>	Upload a file to a code interpreter session .	Extension	Preview
<a href="#">az containerapp sessionpool</a>	Commands to manage session pools.	Extension	Preview
<a href="#">az containerapp sessionpool create</a>	Create or update a Session pool.	Extension	Preview
<a href="#">az containerapp sessionpool delete</a>	Delete a session pool.	Extension	Preview
<a href="#">az containerapp sessionpool list</a>	List Session Pools by subscription or resource group.	Extension	Preview
<a href="#">az containerapp sessionpool show</a>	Show details of a Session Pool.	Extension	Preview
<a href="#">az containerapp sessionpool update</a>	Update a Session pool.	Extension	Preview
<a href="#">az containerapp show</a>	Show details of a container app.	Core	GA
<a href="#">az containerapp show (containerapp extension)</a>	Show details of a container app.	Extension	GA
<a href="#">az containerapp show-custom-domain-verification-id</a>	Show the verification id for binding app or environment custom domains.	Core	GA
<a href="#">az containerapp ssl</a>	Upload certificate to a managed environment, add hostname to an app in that environment, and bind the certificate to the hostname.	Core	GA
<a href="#">az containerapp ssl upload</a>	Upload certificate to a managed environment, add hostname to an app in that environment, and bind the certificate to the hostname.	Core	GA

Name	Description	Type	Status
<a href="#">az containerapp up</a>	Create or update a container app as well as any associated resources (ACR, resource group, container apps environment, GitHub Actions, etc.).	Core	GA
<a href="#">az containerapp up (containerapp extension)</a>	Create or update a container app as well as any associated resources (ACR, resource group, container apps environment, GitHub Actions, etc.).	Extension	GA
<a href="#">az containerapp update</a>	Update a container app. In multiple revisions mode, create a new revision based on the latest revision.	Core	GA
<a href="#">az containerapp update (containerapp extension)</a>	Update a container app. In multiple revisions mode, create a new revision based on the latest revision.	Extension	GA

## az containerapp browse

[!\[\]\(be2d4872dcbd8532d182a872dbfc9aee\_img.jpg\) Edit](#)

Open a containerapp in the browser, if possible.

Azure CLI

```
az containerapp browse [--ids]
 [--name]
 [--resource-group]
 [--subscription]
```

## Examples

open a containerapp in the browser

Azure CLI

```
az containerapp browse -n my-containerapp -g MyResourceGroup
```

## Optional Parameters

## --ids

One or more resource IDs (space-delimited). It should be a complete resource ID containing all information of 'Resource Id' arguments. You should provide either --ids or other 'Resource Id' arguments.

## --name -n

The name of the Containerapp. A name must consist of lower case alphanumeric characters or '-', start with a letter, end with an alphanumeric character, cannot have '--', and must be less than 32 characters.

## --resource-group -g

Name of resource group. You can configure the default group using `az configure --defaults group=<name>`.

## --subscription

Name or ID of subscription. You can configure the default subscription using `az account set -s NAME_OR_ID`.

## ▼ Global Parameters

### --debug

Increase logging verbosity to show all debug logs.

### --help -h

Show this help message and exit.

### --only-show-errors

Only show errors, suppressing warnings.

### --output -o

Output format.

Accepted values: json, jsonc, none, table, tsv, yaml, yamlc

Default value: json

### --query

JMESPath query string. See <http://jmespath.org/> for more information and examples.

### --subscription

Name or ID of subscription. You can configure the default subscription using `az account set -s NAME_OR_ID`.

### --verbose

Increase logging verbosity. Use --debug for full debug logs.

## az containerapp create

 Edit

Create a container app.

Azure CLI

```
az containerapp create --name
 --resource-group
 [--allow-insecure {false, true}]
 [--args]
 [--command]
 [--container-name]
 [--cpu]
 [--dal]
 [--dapr-app-id]
 [--dapr-app-port]
 [--dapr-app-protocol {grpc, http}]
 [--dapr-http-max-request-size]
 [--dapr-http-read-buffer-size]
 [--dapr-log-level {debug, error, info, warn}]
 [--enable-dapr {false, true}]
 [--env-vars]
 [--environment]
 [--exposed-port]
 [--image]
 [--ingress {external, internal}]
 [--max-replicas]
 [--memory]
 [--min-replicas]
 [--no-wait]
 [--registry-identity]
 [--registry-password]
 [--registry-server]
 [--registry-username]
 [--revision-suffix]
```

```
[--revisions-mode {multiple, single}]
[--scale-rule-auth]
[--scale-rule-http-concurrency]
[--scale-rule-metadata]
[--scale-rule-name]
[--scale-rule-type]
[--secret-volume-mount]
[--secrets]
[--system-assigned]
[--tags]
[--target-port]
[--termination-grace-period]
[--transport {auto, http, http2, tcp}]
[--user-assigned]
[--workload-profile-name]
[--yaml]
```

## Examples

Create a container app and retrieve its fully qualified domain name.

Azure CLI

```
az containerapp create -n my-containerapp -g MyResourceGroup \
 --image myregistry.azurecr.io/my-app:v1.0 --environment
MyContainerappEnv \
 --ingress external --target-port 80 \
 --registry-server myregistry.azurecr.io --registry-username myregistry \
 --registry-password $REGISTRY_PASSWORD \
 --query properties.configuration.ingress.fqdn
```

Create a container app with resource requirements and replica count limits.

Azure CLI

```
az containerapp create -n my-containerapp -g MyResourceGroup \
 --image nginx --environment MyContainerappEnv \
 --cpu 0.5 --memory 1.0Gi \
 --min-replicas 4 --max-replicas 8
```

Create a container app with secrets and environment variables.

Azure CLI

```
az containerapp create -n my-containerapp -g MyResourceGroup \
 --image my-app:v1.0 --environment MyContainerappEnv \
 --secrets mysecret=secretvalue1 anothersecret="secret value 2" \
 --env-vars GREETING="Hello, world" SECRETENV=secretref:anothersecret
```

Create a container app using a YAML configuration. Example YAML configuration - <https://aka.ms/azure-container-apps-yaml>

Azure CLI

```
az containerapp create -n my-containerapp -g MyResourceGroup \
--environment MyContainerappEnv \
--yaml "path/to/yaml/file.yml"
```

Create a container app with an http scale rule

Azure CLI

```
az containerapp create -n myapp -g mygroup --environment myenv --image nginx \
\
--scale-rule-name my-http-rule \
--scale-rule-http-concurrency 50
```

Create a container app with a custom scale rule

Azure CLI

```
az containerapp create -n my-containerapp -g MyResourceGroup \
--image my-queue-processor --environment MyContainerappEnv \
--min-replicas 4 --max-replicas 8 \
--scale-rule-name queue-based-autoscaling \
--scale-rule-type azure-queue \
--scale-rule-metadata "accountName=mystorageaccountname" \
"cloud=AzurePublicCloud" \
"queueLength": "5" "queueName": "foo" \
--scale-rule-auth "connection=my-connection-string-secret-name"
```

Create a container app with secrets and mounts them in a volume.

Azure CLI

```
az containerapp create -n my-containerapp -g MyResourceGroup \
--image my-app:v1.0 --environment MyContainerappEnv \
--secrets mysecret=secretvalue1 anothersecret="secret value 2" \
--secret-volume-mount "mnt/secrets"
```

## Required Parameters

--name -n

The name of the Containerapp. A name must consist of lower case alphanumeric characters or '-', start with a letter, end with an alphanumeric character, cannot have '--', and must be less than 32 characters.

#### **--resource-group -g**

Name of resource group. You can configure the default group using `az configure --defaults group=<name>`.

## Optional Parameters

#### **--allow-insecure**

Allow insecure connections for ingress traffic.

Accepted values: false, true

Default value: False

#### **--args**

A list of container startup command argument(s). Space-separated values e.g. "-c" "mycommand". Empty string to clear existing values.

#### **--command**

A list of supported commands on the container that will executed during startup. Space-separated values e.g. "/bin/queue" "mycommand". Empty string to clear existing values.

#### **--container-name**

Name of the container.

#### **--cpu**

Required CPU in cores from 0.25 - 2.0, e.g. 0.5.

#### **--dal --dapr-enable-api-logging**

Enable API logging for the Dapr sidecar.

Default value: False

#### **--dapr-app-id**

The Dapr application identifier.

#### **--dapr-app-port**

The port Dapr uses to talk to the application.

#### **--dapr-app-protocol**

The protocol Dapr uses to talk to the application.

Accepted values: grpc, http

#### **--dapr-http-max-request-size --dhmrs**

Increase max size of request body http and grpc servers parameter in MB to handle uploading of big files.

#### **--dapr-http-read-buffer-size --dhrbs**

Dapr max size of http header read buffer in KB to handle when sending multi-KB headers..

#### **--dapr-log-level**

Set the log level for the Dapr sidecar.

Accepted values: debug, error, info, warn

#### **--enable-dapr**

Boolean indicating if the Dapr side car is enabled.

Accepted values: false, true

Default value: False

#### **--env-vars**

A list of environment variable(s) for the container. Space-separated values in 'key=value' format. Empty string to clear existing values. Prefix value with 'secretref:' to reference a secret.

#### **--environment**

Name or resource ID of the container app's environment.

#### **--exposed-port**

Additional exposed port. Only supported by tcp transport protocol. Must be unique per environment if the app ingress is external.

#### **--image -i**

Container image, e.g. publisher/image-name:tag.

#### **--ingress**

The ingress type.

Accepted values: external, internal

#### **--max-replicas**

The maximum number of replicas.

#### **--memory**

Required memory from 0.5 - 4.0 ending with "Gi", e.g. 1.0Gi.

#### **--min-replicas**

The minimum number of replicas.

#### **--no-wait**

Do not wait for the long-running operation to finish.

Default value: False

#### **--registry-identity**

A Managed Identity to authenticate with the registry server instead of username/password. Use a resource ID or 'system' for user-defined and system-defined identities, respectively. The registry must be an ACR. If possible, an 'acrpull' role assignment will be created for the identity automatically.

#### **--registry-password**

The password to log in to container registry. If stored as a secret, value must start with 'secretref:' followed by the secret name.

#### **--registry-server**

The container registry server hostname, e.g. myregistry.azurecr.io.

**--registry-username**

The username to log in to container registry.

**--revision-suffix**

User friendly suffix that is appended to the revision name.

**--revisions-mode**

The active revisions mode for the container app.

Accepted values: multiple, single

Default value: single

**--scale-rule-auth --sra**

Scale rule auth parameters. Auth parameters must be in format " = = ...".

**--scale-rule-http-concurrency --scale-rule-tcp-concurrency --srhc --srtc**

The maximum number of concurrent requests before scale out. Only supported for http and tcp scale rules.

**--scale-rule-metadata --srm**

Scale rule metadata. Metadata must be in format " = = ...".

**--scale-rule-name --srn**

The name of the scale rule.

**--scale-rule-type --srt**

The type of the scale rule. Default: http. For more information please visit <https://learn.microsoft.com/azure/container-apps/scale-app#scale-triggers>.

**--secret-volume-mount**

Path to mount all secrets e.g. mnt/secrets.

**--secrets -s**

A list of secret(s) for the container app. Space-separated values in 'key=value' format.

#### --system-assigned

Boolean indicating whether to assign system-assigned identity.

Default value: False

#### --tags

Space-separated tags: key[=value] [key[=value] ...]. Use "" to clear existing tags.

#### --target-port

The application port used for ingress traffic.

#### --termination-grace-period --tgp

Duration in seconds a replica is given to gracefully shut down before it is forcefully terminated. (Default: 30).

#### --transport

The transport protocol used for ingress traffic.

Accepted values: auto, http, http2, tcp

Default value: auto

#### --user-assigned

Space-separated user identities to be assigned.

#### --workload-profile-name -w

Name of the workload profile to run the app on.

#### --yaml

Path to a .yaml file with the configuration of a container app. All other parameters will be ignored. For an example, see <https://docs.microsoft.com/azure/container-apps/azure-resource-manager-api-spec#examples>.

### ▼ Global Parameters

#### --debug

Increase logging verbosity to show all debug logs.

#### --help -h

Show this help message and exit.

#### --only-show-errors

Only show errors, suppressing warnings.

#### --output -o

Output format.

Accepted values: json, jsonc, none, table, tsv, yaml, yamlc

Default value: json

#### --query

JMESPath query string. See <http://jmespath.org/> for more information and examples.

#### --subscription

Name or ID of subscription. You can configure the default subscription using `az account set -s NAME_OR_ID`.

#### --verbose

Increase logging verbosity. Use --debug for full debug logs.

## az containerapp create (containerapp extension)

Create a container app.

Azure CLI

```
az containerapp create --name
 --resource-group
 [--allow-insecure {false, true}]
 [--args]
 [--artifact]
```

```
[--bind]
[--branch]
[--build-env-vars]
[--command]
[--container-name]
[--context-path]
[--cpu]
[--customized-keys]
[--dal]
[--dapr-app-id]
[--dapr-app-port]
[--dapr-app-protocol {grpc, http}]
[--dapr-http-max-request-size]
[--dapr-http-read-buffer-size]
[--dapr-log-level {debug, error, info, warn}]
[--enable-dapr {false, true}]
[--enable-java-agent {false, true}]
[--enable-java-metrics {false, true}]
[--env-vars]
[--environment]
[--environment-type {connected, managed}]
[--exposed-port]
[--image]
[--ingress {external, internal}]
[--max-inactive-revisions]
[--max-replicas]
[--memory]
[--min-replicas]
[--no-wait]
[--registry-identity]
[--registry-password]
[--registry-server]
[--registry-username]
[--repo]
[--revision-suffix]
[--revisions-mode {multiple, single}]
[--runtime {generic, java}]
[--scale-rule-auth]
[--scale-rule-http-concurrency]
[--scale-rule-identity]
[--scale-rule-metadata]
[--scale-rule-name]
[--scale-rule-type]
[--secret-volume-mount]
[--secrets]
[--service-principal-client-id]
[--service-principal-client-secret]
[--service-principal-tenant-id]
[--source]
[--system-assigned]
[--tags]
[--target-port]
[--termination-grace-period]
[--token]
[--transport {auto, http, http2, tcp}]
```

```
[--user-assigned]
[--workload-profile-name]
[--yaml]
```

## Examples

Create a container app and retrieve its fully qualified domain name.

Azure CLI

```
az containerapp create -n my-containerapp -g MyResourceGroup \
 --image myregistry.azurecr.io/my-app:v1.0 --environment
 MyContainerappEnv \
 --ingress external --target-port 80 \
 --registry-server myregistry.azurecr.io --registry-username myregistry -
 --registry-password $REGISTRY_PASSWORD \
 --query properties.configuration.ingress.fqdn
```

Create a container app with resource requirements and replica count limits.

Azure CLI

```
az containerapp create -n my-containerapp -g MyResourceGroup \
 --image nginx --environment MyContainerappEnv \
 --cpu 0.5 --memory 1.0Gi \
 --min-replicas 4 --max-replicas 8
```

Create a container app with secrets and environment variables.

Azure CLI

```
az containerapp create -n my-containerapp -g MyResourceGroup \
 --image my-app:v1.0 --environment MyContainerappEnv \
 --secrets mysecret=secretvalue1 anothersecret="secret value 2" \
 --env-vars GREETING="Hello, world" SECRETENV=secretref:anothersecret
```

Create a container app using a YAML configuration. Example YAML configuration -  
<https://aka.ms/azure-container-apps-yaml>

Azure CLI

```
az containerapp create -n my-containerapp -g MyResourceGroup \
 --environment MyContainerappEnv \
 --yaml "path/to/yaml/file.yml"
```

Create a container app with an http scale rule

#### Azure CLI

```
az containerapp create -n myapp -g mygroup --environment myenv --image nginx \
--scale-rule-name my-http-rule \
--scale-rule-http-concurrency 50
```

Create a container app with a custom scale rule

#### Azure CLI

```
az containerapp create -n my-containerapp -g MyResourceGroup \
--image my-queue-processor --environment MyContainerappEnv \
--min-replicas 4 --max-replicas 8 \
--scale-rule-name queue-based-autoscaling \
--scale-rule-type azure-queue \
--scale-rule-metadata "accountName=mystorageaccountname" \
"cloud=AzurePublicCloud" \
"queueLength=5" "queueName=foo" \
--scale-rule-auth "connection=my-connection-string-secret-name"
```

Create a container app with a custom scale rule using identity to authenticate

#### Azure CLI

```
az containerapp create -n my-containerapp -g MyResourceGroup \
--image my-queue-processor --environment MyContainerappEnv \
--user-assigned myUserIdentityResourceId --min-replicas 4 --max-replicas
8 \
--scale-rule-name queue-based-autoscaling \
--scale-rule-type azure-queue \
--scale-rule-metadata "accountName=mystorageaccountname" \
"cloud=AzurePublicCloud" \
"queueLength=5" "queueName=foo" \
--scale-rule-identity myUserIdentityResourceId
```

Create a container app with secrets and mounts them in a volume.

#### Azure CLI

```
az containerapp create -n my-containerapp -g MyResourceGroup \
--image my-app:v1.0 --environment MyContainerappEnv \
--secrets mysecret=secretvalue1 anothersecret="secret value 2" \
--secret-volume-mount "mnt/secrets"
```

Create a container app hosted on a Connected Environment.

#### Azure CLI

```
az containerapp create -n my-containerapp -g MyResourceGroup \
--image my-app:v1.0 --environment MyContainerappConnectedEnv \
--environment-type connected
```

Create a container app from a new GitHub Actions workflow in the provided GitHub repository

Azure CLI

```
az containerapp create -n my-containerapp -g MyResourceGroup \
--environment MyContainerappEnv --registry-server MyRegistryServer \
--registry-user MyRegistryUser --registry-pass MyRegistryPass \
--repo https://github.com/myAccount/myRepo
```

Create a Container App from the provided application source

Azure CLI

```
az containerapp create -n my-containerapp -g MyResourceGroup \
--environment MyContainerappEnv --registry-server MyRegistryServer \
--registry-user MyRegistryUser --registry-pass MyRegistryPass \
--source .
```

Create a container app with java metrics enabled

Azure CLI

```
az containerapp create -n my-containerapp -g MyResourceGroup \
--image my-app:v1.0 --environment MyContainerappEnv \
--enable-java-metrics
```

Create a container app with java agent enabled

Azure CLI

```
az containerapp create -n my-containerapp -g MyResourceGroup \
--image my-app:v1.0 --environment MyContainerappEnv \
--enable-java-agent
```

## Required Parameters

--name -n

The name of the Containerapp. A name must consist of lower case alphanumeric characters or '-', start with a letter, end with an alphanumeric character, cannot have '--', and must be less than 32 characters.

#### --resource-group -g

Name of resource group. You can configure the default group using `az configure --defaults group=<name>`.

## Optional Parameters

#### --allow-insecure

Allow insecure connections for ingress traffic.

Accepted values: false, true

Default value: False

#### --args

A list of container startup command argument(s). Space-separated values e.g. "-c" "mycommand". Empty string to clear existing values.

#### --artifact

[Preview](#)

Local path to the application artifact for building the container image. See the supported artifacts here: <https://aka.ms/SourceToCloudSupportedArtifacts>.

#### --bind

[Preview](#)

Space separated list of services, bindings or Java components to be connected to this app. e.g. SVC\_NAME1[:BIND\_NAME1] SVC\_NAME2[:BIND\_NAME2]...

#### --branch -b

[Preview](#)

Branch in the provided GitHub repository. Assumed to be the GitHub repository's default branch if not specified.

#### --build-env-vars

[Preview](#)

A list of environment variable(s) for the build. Space-separated values in 'key=value' format.

## --command

A list of supported commands on the container that will be executed during startup.  
Space-separated values e.g. "/bin/queue" "mycommand". Empty string to clear existing values.

## --container-name

Name of the container.

## --context-path

[Preview](#)

Path in the repository to run docker build. Defaults to "./". Dockerfile is assumed to be named "Dockerfile" and in this directory.

## --cpu

Required CPU in cores from 0.25 - 2.0, e.g. 0.5.

## --customized-keys

[Preview](#)

The customized keys used to change default configuration names. Key is the original name, value is the customized name.

## --dal --dapr-enable-api-logging

Enable API logging for the Dapr sidecar.

Default value: False

## --dapr-app-id

The Dapr application identifier.

## --dapr-app-port

The port Dapr uses to talk to the application.

## --dapr-app-protocol

The protocol Dapr uses to talk to the application.

Accepted values: grpc, http

## --dapr-http-max-request-size --dhmrs

Increase max size of request body http and grpc servers parameter in MB to handle uploading of big files.

#### **--dapr-http-read-buffer-size --dhrbs**

Dapr max size of http header read buffer in KB to handle when sending multi-KB headers..

#### **--dapr-log-level**

Set the log level for the Dapr sidecar.

Accepted values: debug, error, info, warn

#### **--enable-dapr**

Boolean indicating if the Dapr side car is enabled.

Accepted values: false, true

Default value: False

#### **--enable-java-agent**

Boolean indicating whether to enable Java agent for the app. Only applicable for Java runtime.

Accepted values: false, true

#### **--enable-java-metrics**

Boolean indicating whether to enable Java metrics for the app. Only applicable for Java runtime.

Accepted values: false, true

#### **--env-vars**

A list of environment variable(s) for the container. Space-separated values in 'key=value' format. Empty string to clear existing values. Prefix value with 'secretref:' to reference a secret.

#### **--environment**

Name or resource ID of the container app's environment.

#### **--environment-type**

Preview

Type of environment.

Accepted values: connected, managed

Default value: managed

### --exposed-port

Additional exposed port. Only supported by tcp transport protocol. Must be unique per environment if the app ingress is external.

### --image -i

Container image, e.g. publisher/image-name:tag.

### --ingress

The ingress type.

Accepted values: external, internal

### --max-inactive-revisions

Preview

Max inactive revisions a Container App can have.

### --max-replicas

The maximum number of replicas.

### --memory

Required memory from 0.5 - 4.0 ending with "Gi", e.g. 1.0Gi.

### --min-replicas

The minimum number of replicas.

### --no-wait

Do not wait for the long-running operation to finish.

Default value: False

### --registry-identity

The managed identity with which to authenticate to the Azure Container Registry (instead of username/password). Use 'system' for a system-defined identity, Use 'system-environment' for an environment level system-defined identity or a resource

id for a user-defined environment/containerapp level identity. The managed identity should have been assigned acrpull permissions on the ACR before deployment (use 'az role assignment create --role acrpull ...').

#### --registry-password

The password to log in to container registry. If stored as a secret, value must start with 'secretref:' followed by the secret name.

#### --registry-server

The container registry server hostname, e.g. myregistry.azurecr.io.

#### --registry-username

The username to log in to container registry.

#### --repo Preview

Create an app via GitHub Actions in the format: <https://github.com/> or /.

#### --revision-suffix

User friendly suffix that is appended to the revision name.

#### --revisions-mode

The active revisions mode for the container app.

Accepted values: multiple, single

Default value: single

#### --runtime

The runtime of the container app.

Accepted values: generic, java

#### --scale-rule-auth --sra

Scale rule auth parameters. Auth parameters must be in format " = = ...".

#### --scale-rule-http-concurrency --scale-rule-tcp-concurrency --srhc --srtc

The maximum number of concurrent requests before scale out. Only supported for http and tcp scale rules.

**--scale-rule-identity --sri** Preview

Resource ID of a managed identity to authenticate with Azure scaler resource(storage account/eventhub or else), or System to use a system-assigned identity.

**--scale-rule-metadata --srm**

Scale rule metadata. Metadata must be in format " = = ...".

**--scale-rule-name --srn**

The name of the scale rule.

**--scale-rule-type --srt**

The type of the scale rule. Default: http. For more information please visit <https://learn.microsoft.com/azure/container-apps/scale-app#scale-triggers>.

**--secret-volume-mount**

Path to mount all secrets e.g. mnt/secrets.

**--secrets -s**

A list of secret(s) for the container app. Space-separated values in 'key=value' format.

**--service-principal-client-id --sp-cid** Preview

The service principal client ID. Used by GitHub Actions to authenticate with Azure.

**--service-principal-client-secret --sp-sec** Preview

The service principal client secret. Used by GitHub Actions to authenticate with Azure.

**--service-principal-tenant-id --sp-tid** Preview

The service principal tenant ID. Used by GitHub Actions to authenticate with Azure.

**--source** Preview

Local directory path containing the application source and Dockerfile for building the container image. Preview: If no Dockerfile is present, a container image is generated using buildpacks. If Docker is not running or buildpacks cannot be used, Oryx will be used to generate the image. See the supported Oryx runtimes here: <https://aka.ms/SourceToCloudSupportedVersions>.

#### --system-assigned

Boolean indicating whether to assign system-assigned identity.

Default value: False

#### --tags

Space-separated tags: key[=value] [key[=value] ...]. Use "" to clear existing tags.

#### --target-port

The application port used for ingress traffic.

#### --termination-grace-period --tgp

Duration in seconds a replica is given to gracefully shut down before it is forcefully terminated. (Default: 30).

#### --token

Preview

A Personal Access Token with write access to the specified repository. For more information: <https://help.github.com/en/github/authenticating-to-github/creating-a-personal-access-token-for-the-command-line>. If not provided or not found in the cache (and using --repo), a browser page will be opened to authenticate with Github.

#### --transport

The transport protocol used for ingress traffic.

Accepted values: auto, http, http2, tcp

Default value: auto

#### --user-assigned

Space-separated user identities to be assigned.

#### --workload-profile-name -w

Name of the workload profile to run the app on.

#### --yaml

Path to a .yaml file with the configuration of a container app. All other parameters will be ignored. For an example, see <https://docs.microsoft.com/azure/container-apps/azure-resource-manager-api-spec#examples>.

### ▼ Global Parameters

#### --debug

Increase logging verbosity to show all debug logs.

#### --help -h

Show this help message and exit.

#### --only-show-errors

Only show errors, suppressing warnings.

#### --output -o

Output format.

Accepted values: json, jsonc, none, table, tsv, yaml, yamlc

Default value: json

#### --query

JMESPath query string. See <http://jmespath.org/> for more information and examples.

#### --subscription

Name or ID of subscription. You can configure the default subscription using `az account set -s NAME_OR_ID`.

#### --verbose

Increase logging verbosity. Use --debug for full debug logs.

# az containerapp delete

 Edit

Delete a container app.

Azure CLI

```
az containerapp delete [--ids]
 [--name]
 [--no-wait]
 [--resource-group]
 [--subscription]
 [--yes]
```

## Examples

Delete a container app.

Azure CLI

```
az containerapp delete -g MyResourceGroup -n MyContainerapp
```

## Optional Parameters

### --ids

One or more resource IDs (space-delimited). It should be a complete resource ID containing all information of 'Resource Id' arguments. You should provide either --ids or other 'Resource Id' arguments.

### --name -n

The name of the Containerapp. A name must consist of lower case alphanumeric characters or '-', start with a letter, end with an alphanumeric character, cannot have '--', and must be less than 32 characters.

### --no-wait

Do not wait for the long-running operation to finish.

Default value: False

## --resource-group -g

Name of resource group. You can configure the default group using `az configure --defaults group=<name>`.

## --subscription

Name or ID of subscription. You can configure the default subscription using `az account set -s NAME_OR_ID`.

## --yes -y

Do not prompt for confirmation.

Default value: False

## ▼ Global Parameters

### --debug

Increase logging verbosity to show all debug logs.

### --help -h

Show this help message and exit.

### --only-show-errors

Only show errors, suppressing warnings.

### --output -o

Output format.

Accepted values: json, jsonc, none, table, tsv, yaml, yamlc

Default value: json

### --query

JMESPath query string. See <http://jmespath.org/> for more information and examples.

### --subscription

Name or ID of subscription. You can configure the default subscription using `az`

```
account set -s NAME_OR_ID.
```

### --verbose

Increase logging verbosity. Use `--debug` for full debug logs.

## az containerapp delete (containerapp extension)

Delete a container app.

Azure CLI

```
az containerapp delete [--ids]
 [--name]
 [--no-wait]
 [--resource-group]
 [--subscription]
 [--yes]
```

## Examples

Delete a container app.

Azure CLI

```
az containerapp delete -g MyResourceGroup -n MyContainerapp
```

## Optional Parameters

### --ids

One or more resource IDs (space-delimited). It should be a complete resource ID containing all information of 'Resource Id' arguments. You should provide either `--ids` or other 'Resource Id' arguments.

### --name -n

The name of the Containerapp. A name must consist of lower case alphanumeric characters or '-', start with a letter, end with an alphanumeric character, cannot have '--', and must be less than 32 characters.

#### **--no-wait**

Do not wait for the long-running operation to finish.

Default value: False

#### **--resource-group -g**

Name of resource group. You can configure the default group using `az configure --defaults group=<name>`.

#### **--subscription**

Name or ID of subscription. You can configure the default subscription using `az account set -s NAME_OR_ID`.

#### **--yes -y**

Do not prompt for confirmation.

Default value: False

### ▼ Global Parameters

#### **--debug**

Increase logging verbosity to show all debug logs.

#### **--help -h**

Show this help message and exit.

#### **--only-show-errors**

Only show errors, suppressing warnings.

#### **--output -o**

Output format.

Accepted values: json, jsonc, none, table, tsv, yaml, yamlc

Default value: json

## --query

JMESPath query string. See <http://jmespath.org/> for more information and examples.

## --subscription

Name or ID of subscription. You can configure the default subscription using `az account set -s NAME_OR_ID`.

## --verbose

Increase logging verbosity. Use `--debug` for full debug logs.

# az containerapp exec

 Edit

Open an SSH-like interactive shell within a container app replica.

Azure CLI

```
az containerapp exec --name
 --resource-group
 [--command]
 [--container]
 [--replica]
 [--revision]
```

## Examples

exec into a container app

Azure CLI

```
az containerapp exec -n my-containerapp -g MyResourceGroup
```

exec into a particular container app replica and revision

Azure CLI

```
az containerapp exec -n my-containerapp -g MyResourceGroup --replica
```

```
MyReplica --revision MyRevision
```

open a bash shell in a containerapp

Azure CLI

```
az containerapp exec -n my-containerapp -g MyResourceGroup --command bash
```

## Required Parameters

### --name -n

The name of the Containerapp.

### --resource-group -g

Name of resource group. You can configure the default group using `az configure --defaults group=<name>`.

## Optional Parameters

### --command

The startup command (bash, zsh, sh, etc.).

Default value: sh

### --container

The name of the container to ssh into.

### --replica

The name of the replica to ssh into. List replicas with 'az containerapp replica list'. A replica may not exist if there is not traffic to your app.

### --revision

The name of the container app revision to ssh into. Defaults to the latest revision.

## ▼ Global Parameters

## --debug

Increase logging verbosity to show all debug logs.

## --help -h

Show this help message and exit.

## --only-show-errors

Only show errors, suppressing warnings.

## --output -o

Output format.

Accepted values: json, jsonc, none, table, tsv, yaml, yamlc

Default value: json

## --query

JMESPath query string. See <http://jmespath.org/> for more information and examples.

## --subscription

Name or ID of subscription. You can configure the default subscription using `az account set -s NAME_OR_ID`.

## --verbose

Increase logging verbosity. Use --debug for full debug logs.

# az containerapp list

 Edit

List container apps.

Azure CLI

```
az containerapp list [--environment]
 [--resource-group]
```

# Examples

List container apps in the current subscription.

```
Azure CLI
```

```
az containerapp list
```

List container apps by resource group.

```
Azure CLI
```

```
az containerapp list -g MyResourceGroup
```

## Optional Parameters

### --environment

Name or resource ID of the container app's environment.

### --resource-group -g

Name of resource group. You can configure the default group using `az configure --defaults group=<name>`.

### ▼ Global Parameters

#### --debug

Increase logging verbosity to show all debug logs.

#### --help -h

Show this help message and exit.

#### --only-show-errors

Only show errors, suppressing warnings.

#### --output -o

Output format.

Accepted values: json, jsonc, none, table, tsv, yaml, yamlc

Default value: json

#### --query

JMESPath query string. See <http://jmespath.org/> for more information and examples.

#### --subscription

Name or ID of subscription. You can configure the default subscription using `az account set -s NAME_OR_ID`.

#### --verbose

Increase logging verbosity. Use --debug for full debug logs.

## az containerapp list (containerapp extension)

List container apps.

Azure CLI

```
az containerapp list [--environment]
 [--environment-type {connected, managed}]
 [--resource-group]
```

## Examples

List container apps in the current subscription.

Azure CLI

```
az containerapp list
```

List container apps by resource group.

Azure CLI

```
az containerapp list -g MyResourceGroup
```

List container apps by environment type.

Azure CLI

```
az containerapp list --environment-type connected
```

## Optional Parameters

### --environment

Name or resource ID of the container app's environment.

### --environment-type Preview

Type of environment.

Accepted values: connected, managed

Default value: all

### --resource-group -g

Name of resource group. You can configure the default group using `az configure --defaults group=<name>`.

### ▼ Global Parameters

### --debug

Increase logging verbosity to show all debug logs.

### --help -h

Show this help message and exit.

### --only-show-errors

Only show errors, suppressing warnings.

### --output -o

Output format.

Accepted values: json, jsonc, none, table, tsv, yaml, yamlc

Default value: json

### --query

JMESPath query string. See <http://jmespath.org/> for more information and examples.

### --subscription

Name or ID of subscription. You can configure the default subscription using `az account set -s NAME_OR_ID`.

### --verbose

Increase logging verbosity. Use --debug for full debug logs.

## az containerapp list-usages

 Edit

List usages of subscription level quotas in specific region.

Azure CLI

```
az containerapp list-usages --location
```

## Examples

List usages of quotas in specific region.

Azure CLI

```
az containerapp list-usages -l eastus
```

## Required Parameters

### --location -l

Location. Values from: `az account list-locations`. You can configure the default location using `az configure --defaults location=<location>`.

## ▼ Global Parameters

### --debug

Increase logging verbosity to show all debug logs.

### --help -h

Show this help message and exit.

### --only-show-errors

Only show errors, suppressing warnings.

### --output -o

Output format.

Accepted values: json, jsonc, none, table, tsv, yaml, yamlc

Default value: json

### --query

JMESPath query string. See <http://jmespath.org/> for more information and examples.

### --subscription

Name or ID of subscription. You can configure the default subscription using `az account set -s NAME_OR_ID`.

### --verbose

Increase logging verbosity. Use --debug for full debug logs.

## az containerapp show

 Edit

Show details of a container app.

Azure CLI

```
az containerapp show [--ids]
 [--name]
```

```
[--resource-group]
[--show-secrets]
[--subscription]
```

## Examples

Show the details of a container app.

Azure CLI

```
az containerapp show -n my-containerapp -g MyResourceGroup
```

## Optional Parameters

### --ids

One or more resource IDs (space-delimited). It should be a complete resource ID containing all information of 'Resource Id' arguments. You should provide either --ids or other 'Resource Id' arguments.

### --name -n

The name of the Containerapp. A name must consist of lower case alphanumeric characters or '-', start with a letter, end with an alphanumeric character, cannot have '--', and must be less than 32 characters.

### --resource-group -g

Name of resource group. You can configure the default group using `az configure --defaults group=<name>`.

### --show-secrets

Show Containerapp secrets.

Default value: False

### --subscription

Name or ID of subscription. You can configure the default subscription using `az account set -s NAME_OR_ID`.

## ▼ Global Parameters

### --debug

Increase logging verbosity to show all debug logs.

### --help -h

Show this help message and exit.

### --only-show-errors

Only show errors, suppressing warnings.

### --output -o

Output format.

Accepted values: json, jsonc, none, table, tsv, yaml, yamlc

Default value: json

### --query

JMESPath query string. See <http://jmespath.org/> for more information and examples.

### --subscription

Name or ID of subscription. You can configure the default subscription using `az account set -s NAME_OR_ID`.

### --verbose

Increase logging verbosity. Use --debug for full debug logs.

## az containerapp show (containerapp extension)

Show details of a container app.

Azure CLI

```
az containerapp show [--ids]
 [--name]
```

```
[--resource-group]
[--show-secrets]
[--subscription]
```

## Examples

Show the details of a container app.

Azure CLI

```
az containerapp show -n my-containerapp -g MyResourceGroup
```

## Optional Parameters

### --ids

One or more resource IDs (space-delimited). It should be a complete resource ID containing all information of 'Resource Id' arguments. You should provide either --ids or other 'Resource Id' arguments.

### --name -n

The name of the Containerapp. A name must consist of lower case alphanumeric characters or '-', start with a letter, end with an alphanumeric character, cannot have '--', and must be less than 32 characters.

### --resource-group -g

Name of resource group. You can configure the default group using `az configure --defaults group=<name>`.

### --show-secrets

Show Containerapp secrets.

Default value: False

### --subscription

Name or ID of subscription. You can configure the default subscription using `az account set -s NAME_OR_ID`.

## ▼ Global Parameters

### --debug

Increase logging verbosity to show all debug logs.

### --help -h

Show this help message and exit.

### --only-show-errors

Only show errors, suppressing warnings.

### --output -o

Output format.

Accepted values: json, jsonc, none, table, tsv, yaml, yamlc

Default value: json

### --query

JMESPath query string. See <http://jmespath.org/> for more information and examples.

### --subscription

Name or ID of subscription. You can configure the default subscription using `az account set -s NAME_OR_ID`.

### --verbose

Increase logging verbosity. Use --debug for full debug logs.

## az containerapp show-custom-domain-verification-id

 Edit

Show the verification id for binding app or environment custom domains.

```
az containerapp show-custom-domain-verification-id
```

## Examples

Get the verification id, which needs to be added as a TXT record for app custom domain to verify domain ownership

Azure CLI

```
az containerapp show-custom-domain-verification-id
```

Get the verification id, which needs to be added as a TXT record for custom environment DNS suffix to verify domain ownership

Azure CLI

```
az containerapp show-custom-domain-verification-id
```

### ▼ Global Parameters

**--debug**

Increase logging verbosity to show all debug logs.

**--help -h**

Show this help message and exit.

**--only-show-errors**

Only show errors, suppressing warnings.

**--output -o**

Output format.

Accepted values: json, jsonc, none, table, tsv, yaml, yamlc

Default value: json

**--query**

JMESPath query string. See <http://jmespath.org/> for more information and examples.

## --subscription

Name or ID of subscription. You can configure the default subscription using [az account set -s NAME\\_OR\\_ID](#).

## --verbose

Increase logging verbosity. Use --debug for full debug logs.

# az containerapp up

 Edit

Create or update a container app as well as any associated resources (ACR, resource group, container apps environment, GitHub Actions, etc.).

Azure CLI

```
az containerapp up --name
 [--branch]
 [--browse]
 [--context-path]
 [--env-vars]
 [--environment]
 [--image]
 [--ingress {external, internal}]
 [--location]
 [--logs-workspace-id]
 [--logs-workspace-key]
 [--registry-password]
 [--registry-server]
 [--registry-username]
 [--repo]
 [--resource-group]
 [--service-principal-client-id]
 [--service-principal-client-secret]
 [--service-principal-tenant-id]
 [--source]
 [--target-port]
 [--token]
 [--workload-profile-name]
```

## Examples

Create a container app from a dockerfile in a GitHub repo (setting up github actions)

Azure CLI

```
az containerapp up -n my-containerapp --repo
https://github.com/myAccount/myRepo
```

Create a container app from a dockerfile in a local directory (or autogenerate a container if no dockerfile is found)

Azure CLI

```
az containerapp up -n my-containerapp --source .
```

Create a container app from an image in a registry

Azure CLI

```
az containerapp up -n my-containerapp --image
myregistry.azurecr.io/myImage:myTag
```

Create a container app from an image in a registry with ingress enabled and a specified environment

Azure CLI

```
az containerapp up -n my-containerapp --image
myregistry.azurecr.io/myImage:myTag --ingress external --target-port 80 --
environment MyEnv
```

## Required Parameters

**--name -n**

The name of the Containerapp. A name must consist of lower case alphanumeric characters or '-', start with a letter, end with an alphanumeric character, cannot have '--', and must be less than 32 characters.

## Optional Parameters

**--branch -b**

The branch of the Github repo. Assumed to be the Github repo's default branch if not specified.

#### **--browse**

Open the app in a web browser after creation and deployment, if possible.

Default value: False

#### **--context-path**

Path in the repo from which to run the docker build. Defaults to "./". Dockerfile is assumed to be named "Dockerfile" and in this directory.

#### **--env-vars**

A list of environment variable(s) for the container. Space-separated values in 'key=value' format. Empty string to clear existing values. Prefix value with 'secretref:' to reference a secret.

#### **--environment**

Name or resource ID of the container app's environment.

#### **--image -i**

Container image, e.g. publisher/image-name:tag.

#### **--ingress**

The ingress type.

Accepted values: external, internal

#### **--location -l**

Location. Values from: `az account list-locations`. You can configure the default location using `az configure --defaults location=<location>`.

#### **--logs-workspace-id**

Workspace ID of the Log Analytics workspace to send diagnostics logs to. You can use "az monitor log-analytics workspace create" to create one. Extra billing may apply.

## **--logs-workspace-key**

Log Analytics workspace key to configure your Log Analytics workspace. You can use "az monitor log-analytics workspace get-shared-keys" to retrieve the key.

## **--registry-password**

The password to log in to container registry. If stored as a secret, value must start with 'secretref:' followed by the secret name.

## **--registry-server**

The container registry server hostname, e.g. myregistry.azurecr.io.

## **--registry-username**

The username to log in to container registry.

## **--repo**

Create an app via Github Actions. In the format: <https://github.com/> or /.

## **--resource-group -g**

Name of resource group. You can configure the default group using `az configure --defaults group=<name>`.

## **--service-principal-client-id --sp-cid**

The service principal client ID. Used by Github Actions to authenticate with Azure.

## **--service-principal-client-secret --sp-sec**

The service principal client secret. Used by Github Actions to authenticate with Azure.

## **--service-principal-tenant-id --sp-tid**

The service principal tenant ID. Used by Github Actions to authenticate with Azure.

## **--source**

Local directory path containing the application source and Dockerfile for building the container image. Preview: If no Dockerfile is present, a container image is generated using buildpacks. If Docker is not running or buildpacks cannot be used, Oryx will be

used to generate the image. See the supported Oryx runtimes here:  
<https://github.com/microsoft/Oryx/blob/main/doc/supportedRuntimeVersions.md>.

### --target-port

The application port used for ingress traffic.

### --token

A Personal Access Token with write access to the specified repository. For more information: <https://help.github.com/en/github/authenticating-to-github/creating-a-personal-access-token-for-the-command-line>. If not provided or not found in the cache (and using --repo), a browser page will be opened to authenticate with Github.

### --workload-profile-name -w

The friendly name for the workload profile.

## ▼ Global Parameters

### --debug

Increase logging verbosity to show all debug logs.

### --help -h

Show this help message and exit.

### --only-show-errors

Only show errors, suppressing warnings.

### --output -o

Output format.

Accepted values: json, jsonc, none, table, tsv, yaml, yamlc

Default value: json

### --query

JMESPath query string. See <http://jmespath.org/> for more information and examples.

## --subscription

Name or ID of subscription. You can configure the default subscription using `az account set -s NAME_OR_ID`.

## --verbose

Increase logging verbosity. Use `--debug` for full debug logs.

# az containerapp up (containerapp extension)

Create or update a container app as well as any associated resources (ACR, resource group, container apps environment, GitHub Actions, etc.).

Azure CLI

```
az containerapp up --name
 [--artifact]
 [--branch]
 [--browse]
 [--build-env-vars]
 [--connected-cluster-id]
 [--context-path]
 [--custom-location]
 [--env-vars]
 [--environment]
 [--image]
 [--ingress {external, internal}]
 [--location]
 [--logs-workspace-id]
 [--logs-workspace-key]
 [--registry-password]
 [--registry-server]
 [--registry-username]
 [--repo]
 [--resource-group]
 [--service-principal-client-id]
 [--service-principal-client-secret]
 [--service-principal-tenant-id]
 [--source]
 [--target-port]
 [--token]
 [--workload-profile-name]
```

## Examples

Create a container app from a dockerfile in a GitHub repo (setting up github actions)

Azure CLI

```
az containerapp up -n my-containerapp --repo
https://github.com/myAccount/myRepo
```

Create a container app from a dockerfile in a local directory (or autogenerate a container if no dockerfile is found)

Azure CLI

```
az containerapp up -n my-containerapp --source .
```

Create a container app from an image in a registry

Azure CLI

```
az containerapp up -n my-containerapp --image
myregistry.azurecr.io/myImage:myTag
```

Create a container app from an image in a registry with ingress enabled and a specified environment

Azure CLI

```
az containerapp up -n my-containerapp --image
myregistry.azurecr.io/myImage:myTag --ingress external --target-port 80 --
environment MyEnv
```

Create a container app from an image in a registry on a Connected cluster

Azure CLI

```
az containerapp up -n my-containerapp --image
myregistry.azurecr.io/myImage:myTag --connected-cluster-id
MyConnectedClusterResourceId
```

Create a container app from an image in a registry on a connected environment

Azure CLI

```
az containerapp up -n my-containerapp --image
myregistry.azurecr.io/myImage:myTag --environment MyConnectedEnvironmentId
```

# Required Parameters

## --name -n

The name of the Containerapp. A name must consist of lower case alphanumeric characters or '-', start with a letter, end with an alphanumeric character, cannot have '--', and must be less than 32 characters.

# Optional Parameters

## --artifact

[Preview](#)

Local path to the application artifact for building the container image. See the supported artifacts here: <https://aka.ms/SourceToCloudSupportedArtifacts>.

## --branch -b

The branch of the Github repo. Assumed to be the Github repo's default branch if not specified.

## --browse

Open the app in a web browser after creation and deployment, if possible.

Default value: False

## --build-env-vars

[Preview](#)

A list of environment variable(s) for the build. Space-separated values in 'key=value' format.

## --connected-cluster-id

[Preview](#)

Resource ID of connected cluster. List using 'az connectedk8s list'.

## --context-path

Path in the repo from which to run the docker build. Defaults to "./". Dockerfile is assumed to be named "Dockerfile" and in this directory.

## --custom-location

[Preview](#)

Resource ID of custom location. List using 'az customlocation list'.

#### --env-vars

A list of environment variable(s) for the container. Space-separated values in 'key=value' format. Empty string to clear existing values. Prefix value with 'secretref:' to reference a secret.

#### --environment

Name or resource ID of the container app's managed environment or connected environment.

#### --image -i

Container image, e.g. publisher/image-name:tag.

#### --ingress

The ingress type.

Accepted values: external, internal

#### --location -l

Location. Values from: `az account list-locations`. You can configure the default location using `az configure --defaults location=<location>`.

#### --logs-workspace-id

Workspace ID of the Log Analytics workspace to send diagnostics logs to. You can use "az monitor log-analytics workspace create" to create one. Extra billing may apply.

#### --logs-workspace-key

Log Analytics workspace key to configure your Log Analytics workspace. You can use "az monitor log-analytics workspace get-shared-keys" to retrieve the key.

#### --registry-password

The password to log in to container registry. If stored as a secret, value must start with 'secretref:' followed by the secret name.

## --registry-server

The container registry server hostname, e.g. myregistry.azurecr.io.

## --registry-username

The username to log in to container registry.

## --repo

Create an app via Github Actions. In the format: <https://github.com/> or /.

## --resource-group -g

Name of resource group. You can configure the default group using `az configure --defaults group=<name>`.

## --service-principal-client-id --sp-cid

The service principal client ID. Used by Github Actions to authenticate with Azure.

## --service-principal-client-secret --sp-sec

The service principal client secret. Used by Github Actions to authenticate with Azure.

## --service-principal-tenant-id --sp-tid

The service principal tenant ID. Used by Github Actions to authenticate with Azure.

## --source

Local directory path containing the application source and Dockerfile for building the container image. Preview: If no Dockerfile is present, a container image is generated using buildpacks. If Docker is not running or buildpacks cannot be used, Oryx will be used to generate the image. See the supported Oryx runtimes here:

<https://github.com/microsoft/Oryx/blob/main/doc/supportedRuntimeVersions.md>.

## --target-port

The application port used for ingress traffic.

## --token

A Personal Access Token with write access to the specified repository. For more information: <https://help.github.com/en/github/authenticating-to-github/creating-a-personal-access-token-for-the-command-line>. If not provided or not found in the cache (and using --repo), a browser page will be opened to authenticate with Github.

### --workload-profile-name -w

The friendly name for the workload profile.

## ▼ Global Parameters

### --debug

Increase logging verbosity to show all debug logs.

### --help -h

Show this help message and exit.

### --only-show-errors

Only show errors, suppressing warnings.

### --output -o

Output format.

Accepted values: json, jsonc, none, table, tsv, yaml, yamlc

Default value: json

### --query

JMESPath query string. See <http://jmespath.org/> for more information and examples.

### --subscription

Name or ID of subscription. You can configure the default subscription using `az account set -s NAME_OR_ID`.

### --verbose

Increase logging verbosity. Use --debug for full debug logs.

# az containerapp update

[!\[\]\(b6612f37f12ae77f15151e51efd18e14\_img.jpg\) Edit](#)

Update a container app. In multiple revisions mode, create a new revision based on the latest revision.

Azure CLI

```
az containerapp update [--args]
 [--command]
 [--container-name]
 [--cpu]
 [--ids]
 [--image]
 [--max-replicas]
 [--memory]
 [--min-replicas]
 [--name]
 [--no-wait]
 [--remove-all-env-vars]
 [--remove-env-vars]
 [--replace-env-vars]
 [--resource-group]
 [--revision-suffix]
 [--scale-rule-auth]
 [--scale-rule-http-concurrency]
 [--scale-rule-metadata]
 [--scale-rule-name]
 [--scale-rule-type]
 [--secret-volume-mount]
 [--set-env-vars]
 [--subscription]
 [--tags]
 [--termination-grace-period]
 [--workload-profile-name]
 [--yaml]
```

## Examples

Update a container app's container image.

Azure CLI

```
az containerapp update -n my-containerapp -g MyResourceGroup \
 --image myregistry.azurecr.io/my-app:v2.0
```

Update a container app's resource requirements and scale limits.

#### Azure CLI

```
az containerapp update -n my-containerapp -g MyResourceGroup \
--cpu 0.5 --memory 1.0Gi \
--min-replicas 4 --max-replicas 8
```

Update a container app with an http scale rule

#### Azure CLI

```
az containerapp update -n myapp -g mygroup \
--scale-rule-name my-http-rule \
--scale-rule-http-concurrency 50
```

Update a container app with a custom scale rule

#### Azure CLI

```
az containerapp update -n myapp -g mygroup \
--scale-rule-name my-custom-rule \
--scale-rule-type my-custom-type \
--scale-rule-metadata key1=value1 key2=value2 \
--scale-rule-auth triggerparam=secretref triggerparam=secretref
```

## Optional Parameters

### --args

A list of container startup command argument(s). Space-separated values e.g. "-c" "mycommand". Empty string to clear existing values.

### --command

A list of supported commands on the container that will executed during startup. Space-separated values e.g. "/bin/queue" "mycommand". Empty string to clear existing values.

### --container-name

Name of the container.

### --cpu

Required CPU in cores from 0.25 - 2.0, e.g. 0.5.

#### **--ids**

One or more resource IDs (space-delimited). It should be a complete resource ID containing all information of 'Resource Id' arguments. You should provide either --ids or other 'Resource Id' arguments.

#### **--image -i**

Container image, e.g. publisher/image-name:tag.

#### **--max-replicas**

The maximum number of replicas.

#### **--memory**

Required memory from 0.5 - 4.0 ending with "Gi", e.g. 1.0Gi.

#### **--min-replicas**

The minimum number of replicas.

#### **--name -n**

The name of the Containerapp. A name must consist of lower case alphanumeric characters or '-', start with a letter, end with an alphanumeric character, cannot have '--', and must be less than 32 characters.

#### **--no-wait**

Do not wait for the long-running operation to finish.

Default value: False

#### **--remove-all-env-vars**

Remove all environment variable(s) from container..

Default value: False

#### **--remove-env-vars**

Remove environment variable(s) from container. Space-separated environment variable names.

## **--replace-env-vars**

Replace environment variable(s) in container. Other existing environment variables are removed. Space-separated values in 'key=value' format. If stored as a secret, value must start with 'secretref:' followed by the secret name.

## **--resource-group -g**

Name of resource group. You can configure the default group using `az configure --defaults group=<name>`.

## **--revision-suffix**

User friendly suffix that is appended to the revision name.

## **--scale-rule-auth --sra**

Scale rule auth parameters. Auth parameters must be in format " = = ...".

## **--scale-rule-http-concurrency --scale-rule-tcp-concurrency --srhc --srtc**

The maximum number of concurrent requests before scale out. Only supported for http and tcp scale rules.

## **--scale-rule-metadata --srm**

Scale rule metadata. Metadata must be in format " = = ...".

## **--scale-rule-name --srn**

The name of the scale rule.

## **--scale-rule-type --srt**

The type of the scale rule. Default: http. For more information please visit <https://learn.microsoft.com/azure/container-apps/scale-app#scale-triggers>.

## **--secret-volume-mount**

Path to mount all secrets e.g. mnt/secrets.

## --set-env-vars

Add or update environment variable(s) in container. Existing environment variables are not modified. Space-separated values in 'key=value' format. If stored as a secret, value must start with 'secretref:' followed by the secret name.

## --subscription

Name or ID of subscription. You can configure the default subscription using `az account set -s NAME_OR_ID`.

## --tags

Space-separated tags: key[=value] [key[=value] ...]. Use "" to clear existing tags.

## --termination-grace-period --tgp

Duration in seconds a replica is given to gracefully shut down before it is forcefully terminated. (Default: 30).

## --workload-profile-name -w

The friendly name for the workload profile.

## --yaml

Path to a .yaml file with the configuration of a container app. All other parameters will be ignored. For an example, see <https://docs.microsoft.com/azure/container-apps/azure-resource-manager-api-spec#examples>.

## ▼ Global Parameters

### --debug

Increase logging verbosity to show all debug logs.

### --help -h

Show this help message and exit.

### --only-show-errors

Only show errors, suppressing warnings.

## --output -o

Output format.

Accepted values: json, jsonc, none, table, tsv, yaml, yamlc

Default value: json

## --query

JMESPath query string. See <http://jmespath.org/> for more information and examples.

## --subscription

Name or ID of subscription. You can configure the default subscription using `az`

`account set -s NAME_OR_ID`.

## --verbose

Increase logging verbosity. Use --debug for full debug logs.

# az containerapp update (containerapp extension)

Update a container app. In multiple revisions mode, create a new revision based on the latest revision.

Azure CLI

```
az containerapp update [--args]
 [--artifact]
 [--bind]
 [--build-env-vars]
 [--command]
 [--container-name]
 [--cpu]
 [--customized-keys]
 [--enable-java-agent {false, true}]
 [--enable-java-metrics {false, true}]
 [--ids]
 [--image]
 [--max-inactive-revisions]
 [--max-replicas]
 [--memory]
 [--min-replicas]
```

```
[--name]
[--no-wait]
[--remove-all-env-vars]
[--remove-env-vars]
[--replace-env-vars]
[--resource-group]
[--revision-suffix]
[--runtime {generic, java}]
[--scale-rule-auth]
[--scale-rule-http-concurrency]
[--scale-rule-identity]
[--scale-rule-metadata]
[--scale-rule-name]
[--scale-rule-type]
[--secret-volume-mount]
[--set-env-vars]
[--source]
[--subscription]
[--tags]
[--termination-grace-period]
[--unbind]
[--workload-profile-name]
[--yaml]
```

## Examples

Update a container app's container image.

Azure CLI

```
az containerapp update -n my-containerapp -g MyResourceGroup \
--image myregistry.azurecr.io/my-app:v2.0
```

Update a container app's resource requirements and scale limits.

Azure CLI

```
az containerapp update -n my-containerapp -g MyResourceGroup \
--cpu 0.5 --memory 1.0Gi \
--min-replicas 4 --max-replicas 8
```

Update a container app with an http scale rule

Azure CLI

```
az containerapp update -n myapp -g mygroup \
--scale-rule-name my-http-rule \
--scale-rule-http-concurrency 50
```

## Update a container app with a custom scale rule

Azure CLI

```
az containerapp update -n myapp -g mygroup \
 --scale-rule-name my-custom-rule \
 --scale-rule-type my-custom-type \
 --scale-rule-metadata key=value key2=value2 \
 --scale-rule-auth triggerparam=secretref triggerparam=secretref
```

## Update a Container App from the provided application source

Azure CLI

```
az containerapp update -n my-containerapp -g MyResourceGroup --source .
```

## Update a container app with java metrics enabled

Azure CLI

```
az containerapp update -n my-containerapp -g MyResourceGroup \
 --enable-java-metrics
```

## Update a container app with java agent enabled

Azure CLI

```
az containerapp update -n my-containerapp -g MyResourceGroup \
 --enable-java-agent
```

## Update a container app to erase java enhancement capabilities, like java metrics, java agent, etc.

Azure CLI

```
az containerapp update -n my-containerapp -g MyResourceGroup \
 --runtime generic
```

## Optional Parameters

--args

A list of container startup command argument(s). Space-separated values e.g. "-c" "mycommand". Empty string to clear existing values.

#### --artifact Preview

Local path to the application artifact for building the container image. See the supported artifacts here: [https://aka.ms/SourceToCloudSupportedArtifacts ↗](https://aka.ms/SourceToCloudSupportedArtifacts).

#### --bind Preview

Space separated list of services, bindings or Java components to be connected to this app. e.g. SVC\_NAME1[:BIND\_NAME1] SVC\_NAME2[:BIND\_NAME2]...

#### --build-env-vars Preview

A list of environment variable(s) for the build. Space-separated values in 'key=value' format.

#### --command

A list of supported commands on the container that will be executed during startup. Space-separated values e.g. "/bin/queue" "mycommand". Empty string to clear existing values.

#### --container-name

Name of the container.

#### --cpu

Required CPU in cores from 0.25 - 2.0, e.g. 0.5.

#### --customized-keys Preview

The customized keys used to change default configuration names. Key is the original name, value is the customized name.

#### --enable-java-agent

Boolean indicating whether to enable Java agent for the app. Only applicable for Java runtime.

Accepted values: false, true

## --enable-java-metrics

Boolean indicating whether to enable Java metrics for the app. Only applicable for Java runtime.

Accepted values: false, true

## --ids

One or more resource IDs (space-delimited). It should be a complete resource ID containing all information of 'Resource Id' arguments. You should provide either --ids or other 'Resource Id' arguments.

## --image -i

Container image, e.g. publisher/image-name:tag.

## --max-inactive-revisions

[Preview](#)

Max inactive revisions a Container App can have.

## --max-replicas

The maximum number of replicas.

## --memory

Required memory from 0.5 - 4.0 ending with "Gi", e.g. 1.0Gi.

## --min-replicas

The minimum number of replicas.

## --name -n

The name of the Containerapp. A name must consist of lower case alphanumeric characters or '-', start with a letter, end with an alphanumeric character, cannot have '--', and must be less than 32 characters.

## --no-wait

Do not wait for the long-running operation to finish.

Default value: False

## --remove-all-env-vars

Remove all environment variable(s) from container..

Default value: False

#### **--remove-env-vars**

Remove environment variable(s) from container. Space-separated environment variable names.

#### **--replace-env-vars**

Replace environment variable(s) in container. Other existing environment variables are removed. Space-separated values in 'key=value' format. If stored as a secret, value must start with 'secretref:' followed by the secret name.

#### **--resource-group -g**

Name of resource group. You can configure the default group using `az configure --defaults group=<name>`.

#### **--revision-suffix**

User friendly suffix that is appended to the revision name.

#### **--runtime**

The runtime of the container app.

Accepted values: generic, java

#### **--scale-rule-auth --sra**

Scale rule auth parameters. Auth parameters must be in format " = = ...".

#### **--scale-rule-http-concurrency --scale-rule-tcp-concurrency --srhc --srtc**

The maximum number of concurrent requests before scale out. Only supported for http and tcp scale rules.

#### **--scale-rule-identity --sri**

[Preview](#)

Resource ID of a managed identity to authenticate with Azure scaler resource(storage account/eventhub or else), or System to use a system-assigned identity.

## --scale-rule-metadata --srm

Scale rule metadata. Metadata must be in format " = = ...".

## --scale-rule-name --srn

The name of the scale rule.

## --scale-rule-type --srt

The type of the scale rule. Default: http. For more information please visit <https://learn.microsoft.com/azure/container-apps/scale-app#scale-triggers>.

## --secret-volume-mount

Path to mount all secrets e.g. mnt/secrets.

## --set-env-vars

Add or update environment variable(s) in container. Existing environment variables are not modified. Space-separated values in 'key=value' format. If stored as a secret, value must start with 'secretref:' followed by the secret name.

## --source Preview

Local directory path containing the application source and Dockerfile for building the container image. Preview: If no Dockerfile is present, a container image is generated using buildpacks. If Docker is not running or buildpacks cannot be used, Oryx will be used to generate the image. See the supported Oryx runtimes here: <https://aka.ms/SourceToCloudSupportedVersions>.

## --subscription

Name or ID of subscription. You can configure the default subscription using `az account set -s NAME_OR_ID`.

## --tags

Space-separated tags: key[=value] [key[=value] ...]. Use "" to clear existing tags.

## --termination-grace-period --tgp

Duration in seconds a replica is given to gracefully shut down before it is forcefully terminated. (Default: 30).

#### --unbind Preview

Space separated list of services, bindings or Java components to be removed from this app. e.g. BIND\_NAME1...

#### --workload-profile-name -w

The friendly name for the workload profile.

#### --yaml

Path to a .yaml file with the configuration of a container app. All other parameters will be ignored. For an example, see <https://docs.microsoft.com/azure/container-apps/azure-resource-manager-api-spec#examples>.

### ▼ Global Parameters

#### --debug

Increase logging verbosity to show all debug logs.

#### --help -h

Show this help message and exit.

#### --only-show-errors

Only show errors, suppressing warnings.

#### --output -o

Output format.

Accepted values: json, jsonc, none, table, tsv, yaml, yamlc

Default value: json

#### --query

JMESPath query string. See <http://jmespath.org/> for more information and examples.

## --subscription

Name or ID of subscription. You can configure the default subscription using `az account set -s NAME_OR_ID`.

## --verbose

Increase logging verbosity. Use `--debug` for full debug logs.

# Az.App

Reference

Microsoft Azure PowerShell: App cmdlets

## Container Apps

[Expand table](#)

<a href="#">Disable-AzContainerAppRevision</a>	Deactivates a revision for a Container App
<a href="#">Enable-AzContainerAppRevision</a>	Activates a revision for a Container App
<a href="#">Get-AzContainerApp</a>	Get the properties of a Container App.
<a href="#">Get-AzContainerAppAuthConfig</a>	Get a AuthConfig of a Container App.
<a href="#">Get-AzContainerAppAuthToken</a>	Get auth token for a container app
<a href="#">Get-AzContainerAppAvailableWorkloadProfile</a>	Get all available workload profiles for a location.
<a href="#">Get-AzContainerAppBillingMeter</a>	Get all billingMeters for a location.
<a href="#">Get-AzContainerAppConnectedEnv</a>	Get the properties of an connectedEnvironment.
<a href="#">Get-AzContainerAppConnectedEnvCert</a>	Get the specified Certificate.
<a href="#">Get-AzContainerAppConnectedEnvDapr</a>	Get a dapr component.
<a href="#">Get-AzContainerAppConnectedEnvDaprSecret</a>	List secrets for a dapr component
<a href="#">Get-AzContainerAppConnectedEnvStorage</a>	Get storage for a connectedEnvironment.
<a href="#">Get-AzContainerAppCustomHostName</a>	Analyzes a custom hostname for a Container App
<a href="#">Get-AzContainerAppDiagnosticDetector</a>	Get a diagnostics result of a Container App.
<a href="#">Get-AzContainerAppDiagnosticRevision</a>	Get a revision of a Container App.
<a href="#">Get-AzContainerAppDiagnosticRoot</a>	Get the properties of a Container App.
<a href="#">Get-AzContainerAppJob</a>	Get the properties of a Container Apps Job.
<a href="#">Get-AzContainerAppJobExecution</a>	Get details of a single job execution

<a href="#">Get-AzContainerAppJobSecret</a>	List secrets for a container apps job
<a href="#">Get-AzContainerAppManagedCert</a>	Get the specified Managed Certificate.
<a href="#">Get-AzContainerAppManagedEnv</a>	Get the properties of a Managed Environment used to host container apps.
<a href="#">Get-AzContainerAppManagedEnvAuthToken</a>	Checks if resource name is available.
<a href="#">Get-AzContainerAppManagedEnvCert</a>	Get the specified Certificate.
<a href="#">Get-AzContainerAppManagedEnvDapr</a>	Get a dapr component.
<a href="#">Get-AzContainerAppManagedEnvDaprSecret</a>	List secrets for a dapr component
<a href="#">Get-AzContainerAppManagedEnvDiagnosticDetector</a>	Get the diagnostics data for a Managed Environment used to host container apps.
<a href="#">Get-AzContainerAppManagedEnvDiagnosticRoot</a>	Get the properties of a Managed Environment used to host container apps.
<a href="#">Get-AzContainerAppManagedEnvStorage</a>	Get storage for a managedEnvironment.
<a href="#">Get-AzContainerAppManagedEnvWorkloadProfileState</a>	Get all workload Profile States for a Managed Environment.
<a href="#">Get-AzContainerAppRevision</a>	Get a revision of a Container App.
<a href="#">Get-AzContainerAppRevisionReplica</a>	Get a replica for a Container App Revision.
<a href="#">Get-AzContainerAppSecret</a>	List secrets for a container app
<a href="#">Get-AzContainerAppSourceControl</a>	Get a SourceControl of a Container App.
<a href="#">New-AzContainerApp</a>	Create a Container App.
<a href="#">New-AzContainerAppAuthConfig</a>	Create the AuthConfig for a Container App.
<a href="#">New-AzContainerAppConfigurationObject</a>	Create an in-memory object for Configuration.
<a href="#">New-AzContainerAppConnectedEnv</a>	Create an connectedEnvironment.
<a href="#">New-AzContainerAppConnectedEnvCert</a>	Create a Certificate.
<a href="#">New-AzContainerAppConnectedEnvDapr</a>	Create a Dapr Component in a connected environment.
<a href="#">New-AzContainerAppConnectedEnvStorage</a>	Create storage for a connectedEnvironment.
<a href="#">New-AzContainerAppCustomDomainObject</a>	Create an in-memory object for CustomDomain.

<a href="#">New-AzContainerAppDaprMetadataObject</a>	Create an in-memory object for DaprMetadata.
<a href="#">New-AzContainerAppEnvironmentVarObject</a>	Create an in-memory object for EnvironmentVar.
<a href="#">New-AzContainerAppIdentityProviderObject</a>	Create an in-memory object for IdentityProviders.
<a href="#">New-AzContainerAppInitContainerTemplateObject</a>	Create an in-memory object for InitContainer.
<a href="#">New-AzContainerAppIPSecurityRestrictionRuleObject</a>	Create an in-memory object for IPSecurityRestrictionRule.
<a href="#">New-AzContainerAppJob</a>	Create a Container Apps Job.
<a href="#">New-AzContainerAppJobExecutionContainerObject</a>	Create an in-memory object for JobExecutionContainer.
<a href="#">New-AzContainerAppJobScaleRuleObject</a>	Create an in-memory object for JobScaleRule.
<a href="#">New-AzContainerAppManagedCert</a>	Create a Managed Certificate.
<a href="#">New-AzContainerAppManagedEnv</a>	Create a Managed Environment used to host container apps.
<a href="#">New-AzContainerAppManagedEnvCert</a>	Create a Certificate.
<a href="#">New-AzContainerAppManagedEnvDapr</a>	Create a Dapr Component in a Managed Environment.
<a href="#">New-AzContainerAppManagedEnvStorage</a>	Create storage for a managedEnvironment.
<a href="#">New-AzContainerAppProbeHeaderObject</a>	Create an in-memory object for ContainerAppProbeHttpGetHttpHeadersItem.
<a href="#">New-AzContainerAppProbeObject</a>	Create an in-memory object for ContainerAppProbe.
<a href="#">New-AzContainerAppRegistryCredentialObject</a>	Create an in-memory object for RegistryCredentials.
<a href="#">New-AzContainerAppScaleRuleAuthObject</a>	Create an in-memory object for ScaleRuleAuth.
<a href="#">New-AzContainerAppScaleRuleObject</a>	Create an in-memory object for ScaleRule.
<a href="#">New-AzContainerAppSecretObject</a>	Create an in-memory object for Secret.
<a href="#">New-AzContainerAppSecretVolumeItemObject</a>	Create an in-memory object for

	SecretVolumeItem.
New-AzContainerAppServiceBindObject	Create an in-memory object for ServiceBind.
New-AzContainerAppSourceControl	Create the SourceControl for a Container App.
New-AzContainerAppTemplateObject	Create an in-memory object for Container.
New-AzContainerAppTrafficWeightObject	Create an in-memory object for TrafficWeight.
New-AzContainerAppVolumeMountObject	Create an in-memory object for VolumeMount.
New-AzContainerAppVolumeObject	Create an in-memory object for Volume.
New-AzContainerAppWorkloadProfileObject	Create an in-memory object for WorkloadProfile.
Remove-AzContainerApp	Delete a Container App.
Remove-AzContainerAppAuthConfig	Delete a Container App AuthConfig.
Remove-AzContainerAppConnectedEnv	Delete an connectedEnvironment.
Remove-AzContainerAppConnectedEnvCert	Deletes the specified Certificate.
Remove-AzContainerAppConnectedEnvDapr	Delete a Dapr Component from a connected environment.
Remove-AzContainerAppConnectedEnvStorage	Delete storage for a connectedEnvironment.
Remove-AzContainerAppJob	Delete a Container Apps Job.
Remove-AzContainerAppManagedCert	Deletes the specified Managed Certificate.
Remove-AzContainerAppManagedEnv	Delete a Managed Environment if it does not have any container apps.
Remove-AzContainerAppManagedEnvCert	Deletes the specified Certificate.
Remove-AzContainerAppManagedEnvDapr	Delete a Dapr Component from a Managed Environment.
Remove-AzContainerAppManagedEnvStorage	Delete storage for a managedEnvironment.
Restart-AzContainerAppRevision	Restarts a revision for a Container App
Start-AzContainerApp	Start a container app
Start-AzContainerAppJob	Start a Container Apps Job

<a href="#">Stop-AzContainerApp</a>	Stop a container app
<a href="#">Stop-AzContainerAppJobExecution</a>	Terminates execution of a running container apps job
<a href="#">Test-AzContainerAppConnectedEnvNameAvailability</a>	Checks if resource connectedEnvironmentName is available.
<a href="#">Test-AzContainerAppNamespaceAvailability</a>	Checks if resource name is available.
<a href="#">Update-AzContainerApp</a>	Patches a Container App using JSON Merge Patch
<a href="#">Update-AzContainerAppAuthConfig</a>	Update the AuthConfig for a Container App.
<a href="#">Update-AzContainerAppConnectedEnvCert</a>	Patches a certificate. Currently only patching of tags is supported
<a href="#">Update-AzContainerAppConnectedEnvDapr</a>	Update a Dapr Component in a connected environment.
<a href="#">Update-AzContainerAppConnectedEnvStorage</a>	Update storage for a connectedEnvironment.
<a href="#">Update-AzContainerAppJob</a>	Patches a Container Apps Job using JSON Merge Patch
<a href="#">Update-AzContainerAppManagedCert</a>	Patches a managed certificate. Only patching of tags is supported
<a href="#">Update-AzContainerAppManagedEnv</a>	Patches a Managed Environment using JSON Merge Patch
<a href="#">Update-AzContainerAppManagedEnvCert</a>	Patches a certificate. Currently only patching of tags is supported
<a href="#">Update-AzContainerAppManagedEnvDapr</a>	Update a Dapr Component in a Managed Environment.
<a href="#">Update-AzContainerAppManagedEnvStorage</a>	Update storage for a managedEnvironment.
<a href="#">Update-AzContainerAppSourceControl</a>	Update the SourceControl for a Container App.

# Azure Policy built-in definitions for Azure Container Apps

Article • 02/06/2024

This page is an index of [Azure Policy](#) built-in policy definitions for Azure Container Apps. 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](#).

## Policy definitions

[ ] Expand table

Name (Azure portal)	Description	Effect(s)	Version (GitHub)
<a href="#">Authentication should be enabled on Container Apps</a>	Container Apps Authentication is a feature that can prevent anonymous HTTP requests from reaching the Container App, or authenticate those that have tokens before they reach the Container App	AuditIfNotExists, Disabled	<a href="#">1.0.1</a>
<a href="#">Container App environments should use network injection</a>	Container Apps environments should use virtual network injection to: 1. Isolate Container Apps from the public internet 2. Enable network integration with resources on-premises or in other Azure virtual networks 3. Achieve more granular control over network traffic flowing to and from the environment.	Audit, Disabled, Deny	<a href="#">1.0.2</a>
<a href="#">Container App should configure with volume mount</a>	Enforce the use of volume mounts for Container Apps to ensure availability of persistent storage capacity.	Audit, Deny, Disabled	<a href="#">1.0.1</a>
<a href="#">Container Apps environment should disable public network access</a>	Disable public network access to improve security by exposing the Container Apps environment through an internal load balancer. This removes the need for a public IP address and prevents internet access to all Container Apps within the environment.	Audit, Deny, Disabled	<a href="#">1.0.1</a>

Name (Azure portal)	Description	Effect(s)	Version (GitHub)
<a href="#">Container Apps should disable external network access ↗</a>	Disable external network access to your Container Apps by enforcing internal-only ingress. This will ensure inbound communication for Container Apps is limited to callers within the Container Apps environment.	Audit, Deny, Disabled	<a href="#">1.0.1 ↗</a>
<a href="#">Container Apps should only be accessible over HTTPS ↗</a>	Use of HTTPS ensures server/service authentication and protects data in transit from network layer eavesdropping attacks. Disabling 'allowInsecure' will result in the automatic redirection of requests from HTTP to HTTPS connections for container apps.	Audit, Deny, Disabled	<a href="#">1.0.1 ↗</a>
<a href="#">Managed Identity should be enabled for Container Apps ↗</a>	Enforcing managed identity ensures Container Apps can securely authenticate to any resource that supports Azure AD authentication	Audit, Deny, Disabled	<a href="#">1.0.1 ↗</a>

## Next steps

- See the built-ins on the [Azure Policy GitHub repo ↗](#).
- Review the [Azure Policy definition structure](#).
- Review [Understanding policy effects](#).

# Azure Container Apps

Article • 05/07/2024

Azure Container Apps allows you to run containerized applications without worrying about orchestration or infrastructure. For a more detailed overview, see the [Azure Container Apps product page](#).

# Azure.ResourceManager.AppContainers Namespace

Reference

## Classes

[ ] [Expand table](#)

<a href="#">AppContainersExtensions</a>	A class to add extension methods to Azure.ResourceManager.AppContainers.
<a href="#">ContainerAppAuthConfigCollection</a>	A class representing a collection of <a href="#">ContainerAppAuthConfigResource</a> and their operations. Each <a href="#">ContainerAppAuthConfigResource</a> in the collection will belong to the same instance of <a href="#">ContainerAppResource</a> . To get a <a href="#">ContainerAppAuthConfigCollection</a> instance call the GetContainerAppAuthConfigs method from an instance of <a href="#">ContainerAppResource</a> .
<a href="#">ContainerAppAuthConfigData</a>	A class representing the ContainerAppAuthConfig data model. Configuration settings for the Azure ContainerApp Service Authentication / Authorization feature.
<a href="#">ContainerAppAuthConfigResource</a>	A Class representing a ContainerAppAuthConfig along with the instance operations that can be performed on it. If you have a <a href="#">ResourceIdentifier</a> you can construct a <a href="#">ContainerAppAuthConfigResource</a> from an instance of <a href="#">ArmClient</a> using the GetContainerAppAuthConfigResource method. Otherwise you can get one from its parent resource <a href="#">ContainerAppResource</a> using the GetContainerAppAuthConfig method.
<a href="#">ContainerAppCertificateData</a>	A class representing the ContainerAppCertificate data model. Certificate used for Custom Domain bindings of Container Apps in a Managed Environment
<a href="#">ContainerAppCollection</a>	A class representing a collection of <a href="#">ContainerAppResource</a> and their operations. Each <a href="#">ContainerAppResource</a> in the collection will belong to the same instance of <a href="#">ResourceGroupResource</a> . To get a <a href="#">ContainerAppCollection</a> instance call the GetContainerApps method from an instance of <a href="#">ResourceGroupResource</a> .
<a href="#">ContainerAppConnectedEnvironmentCertificateCollection</a>	A class representing a collection of <a href="#">ContainerAppConnectedEnvironmentCertificateResource</a> and their operations. Each <a href="#">ContainerAppConnectedEnvironmentCertificateResource</a> in the collection will belong to the same instance of <a href="#">ContainerAppConnectedEnvironmentResource</a> . To get a <a href="#">ContainerAppConnectedEnvironmentCertificateCollection</a> instance call the

	<p>GetContainerAppConnectedEnvironmentCertificates method from an instance of <a href="#">ContainerAppConnectedEnvironmentResource</a>.</p>
<a href="#">ContainerAppConnectedEnvironmentCertificateResource</a>	<p>A Class representing a ContainerAppConnectedEnvironmentCertificate along with the instance operations that can be performed on it. If you have a <a href="#">ResourceIdentifier</a> you can construct a <a href="#">ContainerAppConnectedEnvironmentCertificateResource</a> from an instance of <a href="#">ArmClient</a> using the GetContainerAppConnectedEnvironmentCertificateResource method. Otherwise you can get one from its parent resource <a href="#">ContainerAppConnectedEnvironmentResource</a> using the GetContainerAppConnectedEnvironmentCertificate method.</p>
<a href="#">ContainerAppConnectedEnvironmentCollection</a>	<p>A class representing a collection of <a href="#">ContainerAppConnectedEnvironmentResource</a> and their operations. Each <a href="#">ContainerAppConnectedEnvironmentResource</a> in the collection will belong to the same instance of <a href="#">ResourceGroupResource</a>. To get a <a href="#">ContainerAppConnectedEnvironmentCollection</a> instance call the GetContainerAppConnectedEnvironments method from an instance of <a href="#">ResourceGroupResource</a>.</p>
<a href="#">ContainerAppConnectedEnvironmentDaprComponentCollection</a>	<p>A class representing a collection of <a href="#">ContainerAppConnectedEnvironmentDaprComponentResource</a> and their operations. Each <a href="#">ContainerAppConnectedEnvironmentDaprComponentResource</a> in the collection will belong to the same instance of <a href="#">ContainerAppConnectedEnvironmentResource</a>. To get a <a href="#">ContainerAppConnectedEnvironmentDaprComponentCollection</a> instance call the GetContainerAppConnectedEnvironmentDaprComponents method from an instance of <a href="#">ContainerAppConnectedEnvironmentResource</a>.</p>
<a href="#">ContainerAppConnectedEnvironmentDaprComponentResource</a>	<p>A Class representing a ContainerAppConnectedEnvironmentDaprComponent along with the instance operations that can be performed on it. If you have a <a href="#">ResourceIdentifier</a> you can construct a <a href="#">ContainerAppConnectedEnvironmentDaprComponentResource</a> from an instance of <a href="#">ArmClient</a> using the GetContainerAppConnectedEnvironmentDaprComponentResource method. Otherwise you can get one from its parent resource <a href="#">ContainerAppConnectedEnvironmentResource</a> using the GetContainerAppConnectedEnvironmentDaprComponent method.</p>
<a href="#">ContainerAppConnectedEnvironmentData</a>	<p>A class representing the ContainerAppConnectedEnvironment data model. An environment for Kubernetes cluster specialized for web workloads by Azure App Service</p>
<a href="#">ContainerAppConnectedEnvironmentResource</a>	<p>A Class representing a ContainerAppConnectedEnvironment along with the instance operations that can be performed on it. If you have a <a href="#">ResourceIdentifier</a> you can construct a <a href="#">ContainerAppConnectedEnvironmentResource</a> from an instance of <a href="#">ArmClient</a> using the GetContainerAppConnectedEnvironmentResource</p>

	<p>method. Otherwise you can get one from its parent resource <a href="#">ResourceGroupResource</a> using the <a href="#">GetContainerAppConnectedEnvironment</a> method.</p>
<a href="#">ContainerAppConnectedEnvironmentStorageCollection</a>	<p>A class representing a collection of <a href="#">ContainerAppConnectedEnvironmentStorageResource</a> and their operations. Each <a href="#">ContainerAppConnectedEnvironmentStorageResource</a> in the collection will belong to the same instance of <a href="#">ContainerAppConnectedEnvironmentResource</a>. To get a <a href="#">ContainerAppConnectedEnvironmentStorageCollection</a> instance call the <a href="#">GetContainerAppConnectedEnvironmentStorages</a> method from an instance of <a href="#">ContainerAppConnectedEnvironmentResource</a>.</p>
<a href="#">ContainerAppConnectedEnvironmentStorageData</a>	<p>A class representing the ContainerAppConnectedEnvironmentStorage data model. Storage resource for connectedEnvironment.</p>
<a href="#">ContainerAppConnectedEnvironmentStorageResource</a>	<p>A Class representing a ContainerAppConnectedEnvironmentStorage along with the instance operations that can be performed on it. If you have a <a href="#">ResourceIdentifier</a> you can construct a <a href="#">ContainerAppConnectedEnvironmentStorageResource</a> from an instance of <a href="#">ArmClient</a> using the <a href="#">GetContainerAppConnectedEnvironmentStorageResource</a> method. Otherwise you can get one from its parent resource <a href="#">ContainerAppConnectedEnvironmentResource</a> using the <a href="#">GetContainerAppConnectedEnvironmentStorage</a> method.</p>
<a href="#">ContainerAppDaprComponentData</a>	<p>A class representing the ContainerAppDaprComponent data model. Dapr Component.</p>
<a href="#">ContainerAppData</a>	<p>A class representing the ContainerApp data model.</p>
<a href="#">ContainerAppDetectorCollection</a>	<p>A class representing a collection of <a href="#">ContainerAppDetectorResource</a> and their operations. Each <a href="#">ContainerAppDetectorResource</a> in the collection will belong to the same instance of <a href="#">ContainerAppResource</a>. To get a <a href="#">ContainerAppDetectorCollection</a> instance call the <a href="#">GetContainerAppDetectors</a> method from an instance of <a href="#">ContainerAppResource</a>.</p>
<a href="#">ContainerAppDetectorPropertyResource</a>	<p>A Class representing a ContainerAppDetectorProperty along with the instance operations that can be performed on it. If you have a <a href="#">ResourceIdentifier</a> you can construct a <a href="#">ContainerAppDetectorPropertyResource</a> from an instance of <a href="#">ArmClient</a> using the <a href="#">GetContainerAppDetectorPropertyResource</a> method. Otherwise you can get one from its parent resource <a href="#">ContainerAppResource</a> using the <a href="#">GetContainerAppDetectorProperty</a> method.</p>
<a href="#">ContainerAppDetectorPropertyRevisionCollection</a>	<p>A class representing a collection of <a href="#">ContainerAppDetectorPropertyRevisionResource</a> and their operations. Each <a href="#">ContainerAppDetectorPropertyRevisionResource</a> in the collection will</p>

belong to the same instance of [ContainerAppResource](#). To get a [ContainerAppDetectorPropertyRevisionCollection](#) instance call the [GetContainerAppDetectorPropertyRevisions](#) method from an instance of [ContainerAppResource](#).

<a href="#">ContainerAppDetectorPropertyRevisionResource</a>	A Class representing a ContainerAppDetectorPropertyRevision along with the instance operations that can be performed on it. If you have a <a href="#">ResourceIdentifier</a> you can construct a <a href="#">ContainerAppDetectorPropertyRevisionResource</a> from an instance of <a href="#">ArmClient</a> using the <a href="#">GetContainerAppDetectorPropertyRevisionResource</a> method. Otherwise you can get one from its parent resource <a href="#">ContainerAppResource</a> using the <a href="#">GetContainerAppDetectorPropertyRevision</a> method.
<a href="#">ContainerAppDetectorResource</a>	A Class representing a ContainerAppDetector along with the instance operations that can be performed on it. If you have a <a href="#">ResourceIdentifier</a> you can construct a <a href="#">ContainerAppDetectorResource</a> from an instance of <a href="#">ArmClient</a> using the <a href="#">GetContainerAppDetectorResource</a> method. Otherwise you can get one from its parent resource <a href="#">ContainerAppResource</a> using the <a href="#">GetContainerAppDetector</a> method.
<a href="#">ContainerAppDiagnosticData</a>	A class representing the ContainerAppDiagnostic data model. Diagnostics data for a resource.
<a href="#">ContainerAppJobCollection</a>	A class representing a collection of <a href="#">ContainerAppJobResource</a> and their operations. Each <a href="#">ContainerAppJobResource</a> in the collection will belong to the same instance of <a href="#">ResourceGroupResource</a> . To get a <a href="#">ContainerAppJobCollection</a> instance call the <a href="#">GetContainerAppJobs</a> method from an instance of <a href="#">ResourceGroupResource</a> .
<a href="#">ContainerAppJobData</a>	A class representing the ContainerAppJob data model. Container App Job
<a href="#">ContainerAppJobDetectorCollection</a>	A class representing a collection of <a href="#">ContainerAppJobDetectorResource</a> and their operations. Each <a href="#">ContainerAppJobDetectorResource</a> in the collection will belong to the same instance of <a href="#">ContainerAppJobResource</a> . To get a <a href="#">ContainerAppJobDetectorCollection</a> instance call the <a href="#">GetContainerAppJobDetectors</a> method from an instance of <a href="#">ContainerAppJobResource</a> .
<a href="#">ContainerAppJobDetectorPropertyCollection</a>	A class representing a collection of <a href="#">ContainerAppJobDetectorPropertyResource</a> and their operations. Each <a href="#">ContainerAppJobDetectorPropertyResource</a> in the collection will belong to the same instance of <a href="#">ContainerAppJobResource</a> . To get a <a href="#">ContainerAppJobDetectorPropertyCollection</a> instance call the <a href="#">GetContainerAppJobDetectorProperties</a> method from an instance of <a href="#">ContainerAppJobResource</a> .
<a href="#">ContainerAppJobDetectorPropertyResource</a>	A Class representing a ContainerAppJobDetectorProperty along with the instance operations that can be performed on it. If you have a <a href="#">ResourceIdentifier</a> you can construct a

	<p><code>ContainerAppJobDetectorPropertyResource</code> from an instance of <code>ArmClient</code> using the <code>GetContainerAppJobDetectorPropertyResource</code> method.</p> <p>Otherwise you can get one from its parent resource <code>ContainerAppJobResource</code> using the <code>GetContainerAppJobDetectorProperty</code> method.</p>
<code>ContainerAppJobDetectorResource</code>	<p>A Class representing a ContainerAppJobDetector along with the instance operations that can be performed on it. If you have a <code>ResourceIdentifier</code> you can construct a <code>ContainerAppJobDetectorResource</code> from an instance of <code>ArmClient</code> using the <code>GetContainerAppJobDetectorResource</code> method.</p> <p>Otherwise you can get one from its parent resource <code>ContainerAppJobResource</code> using the <code>GetContainerAppJobDetector</code> method.</p>
<code>ContainerAppJobExecutionCollection</code>	<p>A class representing a collection of <code>ContainerAppJobExecutionResource</code> and their operations. Each <code>ContainerAppJobExecutionResource</code> in the collection will belong to the same instance of <code>ContainerAppJobResource</code>. To get a <code>ContainerAppJobExecutionCollection</code> instance call the <code>GetContainerAppJobExecutions</code> method from an instance of <code>ContainerAppJobResource</code>.</p>
<code>ContainerAppJobExecutionData</code>	<p>A class representing the ContainerAppJobExecution data model. Container Apps Job execution.</p>
<code>ContainerAppJobExecutionResource</code>	<p>A Class representing a ContainerAppJobExecution along with the instance operations that can be performed on it. If you have a <code>ResourceIdentifier</code> you can construct a <code>ContainerAppJobExecutionResource</code> from an instance of <code>ArmClient</code> using the <code>GetContainerAppJobExecutionResource</code> method.</p> <p>Otherwise you can get one from its parent resource <code>ContainerAppJobResource</code> using the <code>GetContainerAppJobExecution</code> method.</p>
<code>ContainerAppJobResource</code>	<p>A Class representing a ContainerAppJob along with the instance operations that can be performed on it. If you have a <code>ResourceIdentifier</code> you can construct a <code>ContainerAppJobResource</code> from an instance of <code>ArmClient</code> using the <code>GetContainerAppJobResource</code> method. Otherwise you can get one from its parent resource <code>ResourceGroupResource</code> using the <code>GetContainerAppJob</code> method.</p>
<code>ContainerAppManagedCertificateCollection</code>	<p>A class representing a collection of <code>ContainerAppManagedCertificateResource</code> and their operations. Each <code>ContainerAppManagedCertificateResource</code> in the collection will belong to the same instance of <code>ContainerAppManagedEnvironmentResource</code>. To get a <code>ContainerAppManagedCertificateCollection</code> instance call the <code>GetContainerAppManagedCertificates</code> method from an instance of <code>ContainerAppManagedEnvironmentResource</code>.</p>
<code>ContainerAppManagedCertificateData</code>	<p>A class representing the ContainerAppManagedCertificate data model. Managed certificates used for Custom Domain bindings of Container Apps in a Managed Environment</p>

<a href="#">ContainerAppManagedCertificateResource</a>	A Class representing a ContainerAppManagedCertificate along with the instance operations that can be performed on it. If you have a <a href="#">ResourceIdentifier</a> you can construct a <a href="#">ContainerAppManagedCertificateResource</a> from an instance of <a href="#">ArmClient</a> using the GetContainerAppManagedCertificateResource method. Otherwise you can get one from its parent resource <a href="#">ContainerAppManagedEnvironmentResource</a> using the GetContainerAppManagedCertificate method.
<a href="#">ContainerAppManagedEnvironmentCertificateCollection</a>	A class representing a collection of <a href="#">ContainerAppManagedEnvironmentCertificateResource</a> and their operations. Each <a href="#">ContainerAppManagedEnvironmentCertificateResource</a> in the collection will belong to the same instance of <a href="#">ContainerAppManagedEnvironmentResource</a> . To get a <a href="#">ContainerAppManagedEnvironmentCertificateCollection</a> instance call the GetContainerAppManagedEnvironmentCertificates method from an instance of <a href="#">ContainerAppManagedEnvironmentResource</a> .
<a href="#">ContainerAppManagedEnvironmentCertificateResource</a>	A Class representing a ContainerAppManagedEnvironmentCertificate along with the instance operations that can be performed on it. If you have a <a href="#">ResourceIdentifier</a> you can construct a <a href="#">ContainerAppManagedEnvironmentCertificateResource</a> from an instance of <a href="#">ArmClient</a> using the GetContainerAppManagedEnvironmentCertificateResource method. Otherwise you can get one from its parent resource <a href="#">ContainerAppManagedEnvironmentResource</a> using the GetContainerAppManagedEnvironmentCertificate method.
<a href="#">ContainerAppManagedEnvironmentCollection</a>	A class representing a collection of <a href="#">ContainerAppManagedEnvironmentResource</a> and their operations. Each <a href="#">ContainerAppManagedEnvironmentResource</a> in the collection will belong to the same instance of <a href="#">ResourceGroupResource</a> . To get a <a href="#">ContainerAppManagedEnvironmentCollection</a> instance call the GetContainerAppManagedEnvironments method from an instance of <a href="#">ResourceGroupResource</a> .
<a href="#">ContainerAppManagedEnvironmentDaprComponentCollection</a>	A class representing a collection of <a href="#">ContainerAppManagedEnvironmentDaprComponentResource</a> and their operations. Each <a href="#">ContainerAppManagedEnvironmentDaprComponentResource</a> in the collection will belong to the same instance of <a href="#">ContainerAppManagedEnvironmentResource</a> . To get a <a href="#">ContainerAppManagedEnvironmentDaprComponentCollection</a> instance call the GetContainerAppManagedEnvironmentDaprComponents method from an instance of <a href="#">ContainerAppManagedEnvironmentResource</a> .
<a href="#">ContainerAppManagedEnvironmentDapr</a>	A Class representing a <a href="#">ContainerAppManagedEnvironmentDaprComponent</a> along with the instance operations that can be performed on it. If you have a <a href="#">ResourceIdentifier</a> you can construct a

Component Resource	<p>ContainerAppManagedEnvironmentDaprComponentResource from an instance of <a href="#">ArmClient</a> using the GetContainerAppManagedEnvironmentDaprComponentResource method. Otherwise you can get one from its parent resource ContainerAppManagedEnvironmentResource using the GetContainerAppManagedEnvironmentDaprComponent method.</p>
ContainerApp Managed EnvironmentData	A class representing the ContainerAppManagedEnvironmentData data model.
ContainerApp Managed Environment DetectorCollection	A class representing a collection of ContainerAppManagedEnvironmentDetectorResource and their operations. Each ContainerAppManagedEnvironmentDetectorResource in the collection will belong to the same instance of ContainerAppManagedEnvironmentResource. To get a ContainerAppManagedEnvironmentDetectorCollection instance call the GetContainerAppManagedEnvironmentDetectors method from an instance of ContainerAppManagedEnvironmentResource.
ContainerApp Managed Environment DetectorResource	A Class representing a ContainerAppManagedEnvironmentDetector along with the instance operations that can be performed on it. If you have a ResourceIdentifier you can construct a ContainerAppManagedEnvironmentDetectorResource from an instance of ArmClient using the GetContainerAppManagedEnvironmentDetectorResource method. Otherwise you can get one from its parent resource ContainerAppManagedEnvironmentResource using the GetContainerAppManagedEnvironmentDetector method.
ContainerApp Managed Environment DetectorResource PropertyResource	A Class representing a ContainerAppManagedEnvironmentDetectorResourceProperty along with the instance operations that can be performed on it. If you have a ResourceIdentifier you can construct a ContainerAppManagedEnvironmentDetectorResourcePropertyResource from an instance of ArmClient using the GetContainerAppManagedEnvironmentDetectorResourcePropertyResource method. Otherwise you can get one from its parent resource ContainerAppManagedEnvironmentResource using the GetContainerAppManagedEnvironmentDetectorResourceProperty method.
ContainerApp Managed Environment Resource	A Class representing a ContainerAppManagedEnvironment along with the instance operations that can be performed on it. If you have a ResourceIdentifier you can construct a ContainerAppManagedEnvironmentResource from an instance of ArmClient using the GetContainerAppManagedEnvironmentResource method. Otherwise you can get one from its parent resource ResourceGroupResource using the GetContainerAppManagedEnvironment method.

<a href="#">ContainerAppManagedEnvironmentStorageCollection</a>	A class representing a collection of <a href="#">ContainerAppManagedEnvironmentStorageResource</a> and their operations. Each <a href="#">ContainerAppManagedEnvironmentStorageResource</a> in the collection will belong to the same instance of <a href="#">ContainerAppManagedEnvironmentResource</a> . To get a <a href="#">ContainerAppManagedEnvironmentStorageCollection</a> instance call the GetContainerAppManagedEnvironmentStorages method from an instance of <a href="#">ContainerAppManagedEnvironmentResource</a> .
<a href="#">ContainerAppManagedEnvironmentStorageData</a>	A class representing the ContainerAppManagedEnvironmentStorage data model. Storage resource for managedEnvironment.
<a href="#">ContainerAppManagedEnvironmentStorageResource</a>	A Class representing a ContainerAppManagedEnvironmentStorage along with the instance operations that can be performed on it. If you have a <a href="#">ResourceIdentifier</a> you can construct a <a href="#">ContainerAppManagedEnvironmentStorageResource</a> from an instance of <a href="#">ArmClient</a> using the GetContainerAppManagedEnvironmentStorageResource method. Otherwise you can get one from its parent resource <a href="#">ContainerAppManagedEnvironmentResource</a> using the GetContainerAppManagedEnvironmentStorage method.
<a href="#">ContainerAppReplicaCollection</a>	A class representing a collection of <a href="#">ContainerAppReplicaResource</a> and their operations. Each <a href="#">ContainerAppReplicaResource</a> in the collection will belong to the same instance of <a href="#">ContainerAppRevisionResource</a> . To get a <a href="#">ContainerAppReplicaCollection</a> instance call the GetContainerAppReplicas method from an instance of <a href="#">ContainerAppRevisionResource</a> .
<a href="#">ContainerAppReplicaData</a>	A class representing the ContainerAppReplica data model. Container App Revision Replica.
<a href="#">ContainerAppReplicaResource</a>	A Class representing a ContainerAppReplica along with the instance operations that can be performed on it. If you have a <a href="#">ResourceIdentifier</a> you can construct a <a href="#">ContainerAppReplicaResource</a> from an instance of <a href="#">ArmClient</a> using the GetContainerAppReplicaResource method. Otherwise you can get one from its parent resource <a href="#">ContainerAppRevisionResource</a> using the GetContainerAppReplica method.
<a href="#">ContainerAppResource</a>	A Class representing a ContainerApp along with the instance operations that can be performed on it. If you have a <a href="#">ResourceIdentifier</a> you can construct a <a href="#">ContainerAppResource</a> from an instance of <a href="#">ArmClient</a> using the GetContainerAppResource method. Otherwise you can get one from its parent resource <a href="#">ResourceGroupResource</a> using the GetContainerApp method.
<a href="#">ContainerAppRevisionCollection</a>	A class representing a collection of <a href="#">ContainerAppRevisionResource</a> and their operations. Each <a href="#">ContainerAppRevisionResource</a> in the collection will belong to the same instance of <a href="#">ContainerAppResource</a> . To get a

	<p><a href="#">ContainerAppRevisionCollection</a> instance call the GetContainerAppRevisions method from an instance of <a href="#">ContainerAppResource</a>.</p>
<a href="#">ContainerAppRevisionData</a>	A class representing the ContainerAppRevision data model. Container App Revision.
<a href="#">ContainerAppRevisionResource</a>	A Class representing a ContainerAppRevision along with the instance operations that can be performed on it. If you have a <a href="#">ResourceIdentifier</a> you can construct a <a href="#">ContainerAppRevisionResource</a> from an instance of <a href="#">ArmClient</a> using the GetContainerAppRevisionResource method. Otherwise you can get one from its parent resource <a href="#">ContainerAppResource</a> using the GetContainerAppRevision method.
<a href="#">ContainerAppSourceControlCollection</a>	A class representing a collection of <a href="#">ContainerAppSourceControlResource</a> and their operations. Each <a href="#">ContainerAppSourceControlResource</a> in the collection will belong to the same instance of <a href="#">ContainerAppResource</a> . To get a <a href="#">ContainerAppSourceControlCollection</a> instance call the GetContainerAppSourceControls method from an instance of <a href="#">ContainerAppResource</a> .
<a href="#">ContainerAppSourceControlData</a>	A class representing the ContainerAppSourceControl data model. Container App SourceControl.
<a href="#">ContainerAppSourceControlResource</a>	A Class representing a ContainerAppSourceControl along with the instance operations that can be performed on it. If you have a <a href="#">ResourceIdentifier</a> you can construct a <a href="#">ContainerAppSourceControlResource</a> from an instance of <a href="#">ArmClient</a> using the GetContainerAppSourceControlResource method. Otherwise you can get one from its parent resource <a href="#">ContainerAppResource</a> using the GetContainerAppSourceControl method.

# Azure Resource Manager ContainerAppsApi client library for Java - version 1.0.0

Article • 08/07/2024

Azure Resource Manager ContainerAppsApi client library for Java.

This package contains Microsoft Azure SDK for ContainerAppsApi Management SDK. Package tag package-2024-03. For documentation on how to use this package, please see [Azure Management Libraries for Java ↗](#).

## We'd love to hear your feedback

We're always working on improving our products and the way we communicate with our users. So we'd love to learn what's working and how we can do better.

If you haven't already, please take a few minutes to [complete this short survey ↗](#) we have put together.

Thank you in advance for your collaboration. We really appreciate your time!

## Documentation

Various documentation is available to help you get started

- [API reference documentation ↗](#)

## Getting started

### Prerequisites

- [Java Development Kit \(JDK\)](#) with version 8 or above
- [Azure Subscription ↗](#)

## Adding the package to your product

XML

```
<dependency>
 <groupId>com.azure.resourcemanager</groupId>
 <artifactId>azure-resourcemanager-appcontainers</artifactId>
 <version>1.0.0</version>
</dependency>
```

## Include the recommended packages

Azure Management Libraries require a `TokenCredential` implementation for authentication and an `HttpClient` implementation for HTTP client.

[Azure Identity](#) and [Azure Core Netty HTTP](#) packages provide the default implementation.

## Authentication

Microsoft Entra ID token authentication relies on the [credential class](#) from [Azure Identity](#) package.

Azure subscription ID can be configured via `AZURE_SUBSCRIPTION_ID` environment variable.

Assuming the use of the `DefaultAzureCredential` credential class, the client can be authenticated using the following code:

Java

```
AzureProfile profile = new AzureProfile(AzureEnvironment.AZURE);
TokenCredential credential = new DefaultAzureCredentialBuilder()
 .authorityHost(profile.getEnvironment().getActiveDirectoryEndpoint())
 .build();
ContainerAppsApiManager manager = ContainerAppsApiManager
 .authenticate(credential, profile);
```

The sample code assumes global Azure. Please change `AzureEnvironment.AZURE` variable if otherwise.

See [Authentication](#) for more options.

## Key concepts

See [API design](#) for general introduction on design and key concepts on Azure Management Libraries.

# Examples

[Code snippets and samples ↗](#)

# Troubleshooting

## Next steps

## Contributing

For details on contributing to this repository, see the [contributing guide ↗](#).

This project welcomes contributions and suggestions. Most contributions require you to agree to a Contributor License Agreement (CLA) declaring that you have the right to, and actually do, grant us the rights to use your contribution. For details, visit <https://cla.microsoft.com> ↗.

When you submit a pull request, a CLA-bot will automatically determine whether you need to provide a CLA and decorate the PR appropriately (e.g., label, comment). Simply follow the instructions provided by the bot. You will only need to do this once across all repositories using our CLA.

This project has adopted the [Microsoft Open Source Code of Conduct](#) ↗. For more information see the [Code of Conduct FAQ](#) ↗ or contact [opencode@microsoft.com](mailto:opencode@microsoft.com) with any additional questions or comments.

### Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see [our contributor guide](#).



### Azure SDK for Java feedback

Azure SDK for Java is an open source project. Select a link to provide feedback:

 [Open a documentation issue](#)

 [Provide product feedback](#)

# Azure ContainerApps API client library for JavaScript - version 2.1.0

Article • 08/08/2024

This package contains an isomorphic SDK (runs both in Node.js and in browsers) for Azure ContainerApps API client.

[Source code ↗](#) | [Package \(NPM\) ↗](#) | [API reference documentation](#) | [Samples ↗](#)

## Getting started

### Currently supported environments

- [LTS versions of Node.js ↗](#)
- Latest versions of Safari, Chrome, Edge and Firefox.

See our [support policy ↗](#) for more details.

### Prerequisites

- An [Azure subscription ↗](#).

### Install the `@azure/arm-appcontainers` package

Install the Azure ContainerApps API client library for JavaScript with `npm`:

Bash

```
npm install @azure/arm-appcontainers
```

### Create and authenticate a `ContainerAppsAPIClient`

To create a client object to access the Azure ContainerApps API API, you will need the `endpoint` of your Azure ContainerApps API resource and a `credential`. The Azure ContainerApps API client can use Azure Active Directory credentials to authenticate. You can find the endpoint for your Azure ContainerApps API resource in the [Azure Portal ↗](#).

You can authenticate with Azure Active Directory using a credential from the [@azure/identity ↗](#) library or [an existing AAD Token ↗](#).

To use the [DefaultAzureCredential](#) provider shown below, or other credential providers provided with the Azure SDK, please install the `@azure/identity` package:

Bash

```
npm install @azure/identity
```

You will also need to **register a new AAD application and grant access to Azure ContainerApps API** by assigning the suitable role to your service principal (note: roles such as "Owner" will not grant the necessary permissions). Set the values of the client ID, tenant ID, and client secret of the AAD application as environment variables:

`AZURE_CLIENT_ID`, `AZURE_TENANT_ID`, `AZURE_CLIENT_SECRET`.

For more information about how to create an Azure AD Application check out [this guide](#).

JavaScript

```
const { ContainerAppsAPIClient } = require("@azure/arm-appcontainers");
const { DefaultAzureCredential } = require("@azure/identity");
// For client-side applications running in the browser, use
InteractiveBrowserCredential instead of DefaultAzureCredential. See
https://aka.ms/azsdk/js/identity/examples for more details.

const subscriptionId = "00000000-0000-0000-0000-000000000000";
const client = new ContainerAppsAPIClient(new DefaultAzureCredential(),
subscriptionId);

// For client-side applications running in the browser, use this code
instead:
// const credential = new InteractiveBrowserCredential({
// tenantId: "<YOUR_TENANT_ID>",
// clientId: "<YOUR_CLIENT_ID>"
// });
// const client = new ContainerAppsAPIClient(credential, subscriptionId);
```

## JavaScript Bundle

To use this client library in the browser, first you need to use a bundler. For details on how to do this, please refer to our [bundling documentation](#).

## Key concepts

### ContainerAppsAPIClient

`ContainerAppsAPIClient` is the primary interface for developers using the Azure ContainerApps API client library. Explore the methods on this client object to understand the different features of the Azure ContainerApps API service that you can access.

## Troubleshooting

### Logging

Enabling logging may help uncover useful information about failures. In order to see a log of HTTP requests and responses, set the `AZURE_LOG_LEVEL` environment variable to `info`. Alternatively, logging can be enabled at runtime by calling `setLogLevel` in the `@azure/logger`:

JavaScript

```
const { setLogLevel } = require("@azure/logger");
setLogLevel("info");
```

For more detailed instructions on how to enable logs, you can look at the [@azure/logger package docs](#).

## Next steps

Please take a look at the [samples](#) directory for detailed examples on how to use this library.

## Contributing

If you'd like to contribute to this library, please read the [contributing guide](#) to learn more about how to build and test the code.

## Related projects

- [Microsoft Azure SDK for JavaScript](#)

 Collaborate with us on  
GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see [our contributor guide](#).



## Azure SDK for JavaScript feedback

Azure SDK for JavaScript is an open source project. Select a link to provide feedback:

 [Open a documentation issue](#)

 [Provide product feedback](#)

# appcontainers Package

Reference

## Packages

[\[\] Expand table](#)

[aio](#)

[models](#)

[operations](#)

## Classes

[\[\] Expand table](#)

[ContainerAppsAPIClient](#)

ContainerAppsAPIClient.

# Azure Container Apps on Azure Arc (Preview)

Article • 09/26/2024

You can run Container Apps on an Azure Arc-enabled AKS or AKS-HCI cluster.

Running in an Azure Arc-enabled Kubernetes cluster allows:

- Developers to take advantage of Container Apps' features
- IT administrators to maintain corporate compliance by hosting Container Apps on internal infrastructure.

Learn to set up your Kubernetes cluster for Container Apps, via [Set up an Azure Arc-enabled Kubernetes cluster to run Azure Container Apps](#)

As you configure your cluster, you carry out these actions:

- **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 Container Apps extension [installs the required resources 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](#).
- **A Container Apps connected 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.

## Public preview limitations

The following public preview limitations apply to Azure Container Apps on Azure Arc enabled Kubernetes.

[+] [Expand table](#)

Limitation	Details
Supported Azure regions	East US, West Europe, East Asia
Cluster networking requirement	Must support <a href="#">LoadBalancer</a> service type
Node OS requirement	Linux only.
Feature: Managed identities	<a href="#">Not available</a>
Feature: Pull images from ACR with managed identity	Not available (depends on managed identities)
Logs	Log Analytics must be configured with cluster extension; not per-application

i **Important**

If deploying onto AKS-HCI ensure that you have [setup HAProxy as your load balancer](#) before attempting to install the extension.

## Resources created by the Container Apps extension

When the Container Apps extension is installed on the Azure Arc-enabled Kubernetes cluster, several resources are created in the specified release namespace. These resources enable your cluster to be an extension of the `Microsoft.App` resource provider to support the management and operation of your apps.

Optionally, you can choose to have the extension install [KEDA](#) for event-driven scaling. However, only one KEDA installation is allowed on the cluster. If you have an existing installation, disable the KEDA installation as you install the cluster extension.

The following table describes the role of each revision created for you:

[ ] [Expand table](#)

Pod	Description	Number of Instances	CPU	Memory	Type
<code>&lt;extensionName&gt;-k8se-activator</code>	Used as part of the scaling pipeline	2	100 millicpu	500 MB	ReplicaSet

Pod	Description	Number of Instances	CPU	Memory	Type
<extensionName>-k8se-billing	Billing record generation - Azure Container Apps on Azure Arc enabled Kubernetes is Free of Charge during preview	3	100 millicpu	100 MB	ReplicaSet
<extensionName>-k8se-containerapp-controller	The core operator pod that creates resources on the cluster and maintains the state of components.	2	100 millicpu	1 GB	ReplicaSet
<extensionName>-k8se-envoy	A front-end proxy layer for all data-plane http requests. It routes the inbound traffic to the correct apps.	3	1 Core	1,536 MB	ReplicaSet
<extensionName>-k8se-envoy-controller	Operator, which generates Envoy configuration	2	200 millicpu	500 MB	ReplicaSet
<extensionName>-k8se-event-processor	An alternative routing destination to help with apps that have scaled to zero while the system gets the first instance available.	2	100 millicpu	500 MB	ReplicaSet
<extensionName>-k8se-http-scaler	Monitors inbound request volume in order to provide scaling information to KEDA <a href="#">↗</a> .	1	100 millicpu	500 MB	ReplicaSet
<extensionName>-k8se-keda-cosmosdb-scaler	KEDA Cosmos DB Scaler	1	10 m	128 MB	ReplicaSet
<extensionName>-k8se-keda-metrics-apiserver	KEDA Metrics Server	1	1 Core	1,000 MB	ReplicaSet
<extensionName>-k8se-keda-operator	Scales workloads in and out from 0/1 to N instances	1	100 millicpu	500 MB	ReplicaSet

<b>Pod</b>	<b>Description</b>	<b>Number of Instances</b>	<b>CPU</b>	<b>Memory</b>	<b>Type</b>
<extensionName>-k8se-log-processor	Gathers logs from apps and other components and sends them to Log Analytics.	2	200 millicpu	500 MB	DaemonSet
<extensionName>-k8se-mdm	Metrics and Logs Agent	2	500 millicpu	500 MB	ReplicaSet
dapr-metrics	Dapr metrics pod	1	100 millicpu	500 MB	ReplicaSet
dapr-operator	Manages component updates and service endpoints for Dapr	1	100 millicpu	500 MB	ReplicaSet
dapr-placement-server	Used for Actors only - creates mapping tables that map actor instances to pods	1	100 millicpu	500 MB	StatefulSet
dapr-sentry	Manages mTLS between services and acts as a CA	2	800 millicpu	200 MB	ReplicaSet

## FAQ for Azure Container Apps on Azure Arc (Preview)

- How much does it cost?
- Which Container Apps features are 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 the extension be installed on Windows nodes?
- Can I deploy the Container Apps extension on an Arm64 based cluster?

### How much does it cost?

Azure Container Apps on Azure Arc-enabled Kubernetes is free during the public preview.

## Which Container Apps features are supported?

During the preview period, certain Azure Container App 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 managed identities supported?

Managed Identities aren't supported. Apps can't 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 Container Apps on Azure Arc-enabled Kubernetes are able to scale within the limits of the underlying Kubernetes cluster. If the cluster runs out of available compute resources (CPU and memory primarily), then applications 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 application environment 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 disables forwarding of application to your Log Analytics workspace. Disabling the log processor might affect the time needed for any support cases, and you'll be asked to collect logs from standard output through some other means.

## What do I do if I see a provider registration error?

As you create an Azure Container Apps connected environment resource, some subscriptions might see the "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.App` provider. Re-registering the provider has no effect on existing applications or APIs. To re-register, use the Azure CLI to run `az provider register --namespace Microsoft.App --wait`. Then reattempt the connected environment command.

## Can the extension be installed on Windows nodes?

No, the extension cannot be installed on Windows nodes. The extension supports installation on **Linux nodes only**.

## Can I deploy the Container Apps extension on an Arm64 based cluster?

Arm64 based clusters aren't supported at this time.

## Extension Release Notes

### Container Apps extension v1.0.46 (December 2022)

- Initial public preview release of Container apps extension

### Container Apps extension v1.0.47 (January 2023)

- Upgrade of Envoy to 1.0.24

### Container Apps extension v1.0.48 (February 2023)

- Add probes to EasyAuth container(s)
- Increased memory limit for dapr-operator
- Added prevention of platform header overwriting

### Container Apps extension v1.0.49 (February 2023)

- Upgrade of KEDA to 2.9.1 and Dapr to 1.9.5
- Increase Envoy Controller resource limits to 200 m CPU
- Increase Container App Controller resource limits to 1-GB memory
- Reduce EasyAuth sidecar resource limits to 50 m CPU
- Resolve KEDA error logging for missing metric values

## **Container Apps extension v1.0.50 (March 2023)**

- Updated logging images in sync with Public Cloud

## **Container Apps extension v1.5.1 (April 2023)**

- New versioning number format
- Upgrade of Dapr to 1.10.4
- Maintain scale of Envoy after deployments of new revisions
- Change to when default startup probes are added to a container, if developer doesn't define both startup and readiness probes, then default startup probes are added
- Adds CONTAINER\_APP\_REPLICA\_NAME environment variable to custom containers
- Improvement in performance when multiple revisions are stopped

## **Container Apps extension v1.12.8 (June 2023)**

- Update OSS Fluent Bit to 2.1.2 and Dapr to 1.10.6
- Support for container registries exposed on custom port
- Enable activate/deactivate revision when a container app is stopped
- Fix Revisions List not returning init containers
- Default allow headers added for cors policy

## **Container Apps extension v1.12.9 (July 2023)**

- Minor updates to EasyAuth sidecar containers
- Update of Extension Monitoring Agents

## **Container Apps extension v1.17.8 (August 2023)**

- Update EasyAuth to 1.6.16, Dapr to 1.10.8, and Envoy to 1.25.6
- Add volume mount support for Azure Container App jobs
- Added IP Restrictions for applications with TCP Ingress type
- Added support for Container Apps with multiple exposed ports

## **Container Apps extension v1.23.5 (December 2023)**

- Update Envoy to 1.27.2, KEDA to v2.10.0, EasyAuth to 1.6.20, and Dapr to 1.11
- Set Envoy to max TLS 1.3
- Fix to resolve crashes in Log Processor pods

- Fix to image pull secret retrieval issues
- Update placement of Envoy to distribute across available nodes where possible
- When container apps fail to provision as a result of revision conflicts, set the provisioning state to failed

## Container Apps extension v1.30.6 (January 2024)

- Update KEDA to v2.12, Envoy SC image to v1.0.4, and Dapr image to v1.11.6
- Added default response timeout for Envoy routes to 1,800 seconds
- Changed Fluent bit default log level to warn
- Delay deletion of job pods to ensure log emission
- Fixed issue for job pod deletion for failed job executions
- Ensure jobs in suspended state have failed pods deleted
- Update to not resolve HTTPOptions for TCP applications
- Allow applications to listen on HTTP or HTTPS
- Add ability to suspend jobs
- Fixed issue where KEDA scaler was failing to create job after stopped job execution
- Add startingDeadlineSeconds to Container App Job if there's a cluster reboot
- Removed heavy logging in Envoy access log server
- Updated Monitoring Configuration version for Azure Container Apps on Azure Arc enabled Kubernetes

## Container Apps extension v1.36.15 (April 2024)

- Update Dapr to v1.12 and Dapr Metrics to v0.6
- Allow customers to enable Azure SDK debug logging in Dapr
- Scale Envoy in response to memory usage
- Change of Envoy log format to Json
- Export additional Envoy metrics
- Truncate Envoy log to first 1,024 characters when log content failed to parse
- Handle SIGTERM gracefully in local proxy
- Allow ability to use different namespaces with KEDA
- Validation added for scale rule name
- Enabled revision GC by default
- Enabled emission of metrics for sidecars
- Added volumeMounts to job executions
- Added validation to webhook endpoints for jobs

## Container Apps extension v1.37.1 (July 2024)

- Update EasyAuth to support MISE

## Container Apps extension v1.37.2 (September 2024)

- Updated Dapr-Metrics image to v0.6.8 to resolve network timeout issue
- Resolved issue in Log Processor which prevented MDSD container from starting when cluster is connected behind a Proxy

## Next steps

[Create a Container Apps connected environment \(Preview\)](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# Tutorial: Enable Azure Container Apps on Azure Arc-enabled Kubernetes (Preview)

Article • 09/24/2024

With [Azure Arc-enabled Kubernetes clusters](#), you can create a [Container Apps enabled custom location](#) in your on-premises or cloud Kubernetes cluster to deploy your Azure Container Apps applications as you would any other region.

This tutorial will show you how to enable Azure Container Apps on your Arc-enabled Kubernetes cluster. In this tutorial you will:

- ✓ Create a connected cluster.
- ✓ Create a Log Analytics workspace.
- ✓ Install the Container Apps extension.
- ✓ Create a custom location.
- ✓ Create the Azure Container Apps connected environment.

## ⓘ Note

During the preview, Azure Container Apps on Arc are not supported in production configurations. This article provides an example configuration for evaluation purposes only.

This tutorial uses [Azure Kubernetes Service \(AKS\)](#) to provide concrete instructions for setting up an environment from scratch. However, for a production workload, you may not want to enable Azure Arc on an AKS cluster as it is already managed in Azure.

## Prerequisites

- An Azure account with an active subscription.
  - If you don't have one, you [can create one for free ↗](#).
- Install the [Azure CLI](#).
- Access to a public or private container registry, such as the [Azure Container Registry](#).

- Review the [requirements and limitations](#) of the public preview. Of particular importance are the cluster requirements.

## Setup

Install the following Azure CLI extensions.

```
Azure CLI

az extension add --name connectedk8s --upgrade --yes
az extension add --name k8s-extension --upgrade --yes
az extension add --name customlocation --upgrade --yes
az extension add --name containerapp --upgrade --yes
```

Register the required namespaces.

```
Azure CLI

az provider register --namespace Microsoft.ExtendedLocation --wait
az provider register --namespace Microsoft.KubernetesConfiguration --wait
az provider register --namespace Microsoft.App --wait
az provider register --namespace Microsoft.OperationalInsights --wait
```

Set environment variables based on your Kubernetes cluster deployment.

```
Azure CLI

Bash

GROUP_NAME="my-arc-cluster-group"
AKS_CLUSTER_GROUP_NAME="my-aks-cluster-group"
AKS_NAME="my-aks-cluster"
LOCATION="eastus"
```

## Create a connected cluster

The following steps 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.

```
Azure CLI

az group create --name $AKS_CLUSTER_GROUP_NAME --location $LOCATION
az aks create \
 --resource-group $AKS_CLUSTER_GROUP_NAME \
 --name $AKS_NAME \
 --enable-aad \
 --generate-ssh-keys
```

2. Get the [kubeconfig](#) file and test your connection to the cluster. By default, the kubeconfig file is saved to `~/.kube/config`.

```
Azure CLI

az aks get-credentials --resource-group $AKS_CLUSTER_GROUP_NAME --name
$AKS_NAME --admin

kubectl get ns
```

3. Create a resource group to contain your Azure Arc resources.

```
Azure CLI

az group create --name $GROUP_NAME --location $LOCATION
```

4. Connect the cluster you created to Azure Arc.

```
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

A [Log Analytics workspace](#) provides access to logs for Container Apps applications running in the Azure Arc-enabled Kubernetes cluster. A Log Analytics workspace is optional, but recommended.

1. Create a Log Analytics workspace.

Azure CLI

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.

Azure CLI

Azure CLI

```
LOG_ANALYTICS_WORKSPACE_ID=$(az monitor log-analytics workspace
show \
--name $WORKSPACE_NAME --query id --output tsv)
```

```

--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 Container Apps extension

i **Important**

If deploying onto AKS-HCI ensure that you have [setup HAProxy or a custom load balancer](#) before attempting to install the extension.

1. Set the following environment variables to the desired name of the [Container Apps extension](#), the cluster namespace in which resources should be provisioned, and the name for the Azure Container Apps connected environment. Choose a unique name for <connected-environment-name>. The connected environment name will be part of the domain name for app you'll create in the Azure Container Apps connected environment.

Azure CLI

Bash

```

EXTENSION_NAME="appenv-ext"
NAMESPACE="appplat-ns"
CONNECTED_ENVIRONMENT_NAME=""
```

2. Install the Container Apps extension to your Azure Arc-connected cluster with Log Analytics enabled. Log Analytics can't be added to the extension later.

Azure CLI

## Azure CLI

```
az k8s-extension create \
 --resource-group $GROUP_NAME \
 --name $EXTENSION_NAME \
 --cluster-type connectedClusters \
 --cluster-name $CLUSTER_NAME \
 --extension-type 'Microsoft.App.Environment' \
 --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=${CONNECTED_ENVIRONMENT_NAME}" \
 --configuration-settings
"envoy.annotations.service.beta.kubernetes.io/azure-load-balancer-
resource-group=${AKS_CLUSTER_GROUP_NAME}" \
 --configuration-settings "logProcessor.appLogs.destination=log-
analytics" \
 --configuration-protected-settings
"logProcessor.appLogs.logAnalyticsConfig.customerId=${LOG_ANALYTICS
_WORKSPACE_ID_ENC}" \
 --configuration-protected-settings
"logProcessor.appLogs.logAnalyticsConfig.sharedKey=${LOG_ANALYTICS_
KEY_ENC}"
```

### ⚠ Note

To install the extension without Log Analytics integration, remove the last three `--configuration-settings` parameters from the command.

The following table describes the various `--configuration-settings` parameters when running the command:

[+] Expand table

Parameter	Description
<code>Microsoft.CustomLocation.ServiceAccount</code>	The service account created for the custom location. It's recommended that it's set to the value <code>default</code> .

Parameter	Description
<code>appsNamespace</code>	The namespace used to create the app definitions and revisions. It <b>must</b> match that of the extension release namespace.
<code>clusterName</code>	The name of the Container Apps extension Kubernetes environment that will be created against this extension.
<code>logProcessor.appLogs.destination</code>	Optional. Destination for application logs. Accepts <code>log-analytics</code> or <code>none</code> , choosing none disables platform logs.
<code>logProcessor.appLogs.logAnalyticsConfig.customerId</code>	Required only when <code>logProcessor.appLogs.destination</code> is set to <code>log-analytics</code> . The base64-encoded Log analytics workspace ID. This parameter should be configured as a protected setting.
<code>logProcessor.appLogs.logAnalyticsConfig.sharedKey</code>	Required only when <code>logProcessor.appLogs.destination</code> is set to <code>log-analytics</code> . The base64-encoded Log analytics workspace shared key. This parameter should be configured as a protected setting.
<code>envoy.annotations.service.beta.kubernetes.io/azure-load-balancer-resource-group</code>	The name of the resource group in which the Azure Kubernetes Service cluster resides. Valid and required only when the underlying cluster is Azure Kubernetes Service.

- Save the `id` property of the Container Apps extension for later.

Azure CLI

Azure CLI

```
EXTENSION_ID=$(az k8s-extension show \
--cluster-type connectedClusters \
--cluster-name $CLUSTER_NAME \
```

```
--resource-group $GROUP_NAME \
--name $EXTENSION_NAME \
--query id \
--output tsv)
```

4. Wait for the extension to fully install before proceeding. You can have your terminal session wait until it completes by running the following command:

Azure CLI

```
az resource wait --ids $EXTENSION_ID --custom
"properties.provisioningState!='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
```

To learn more about these pods and their role in the system, see [Azure Arc overview](#).

## Create a custom location

The [custom location](#) is an Azure location that you assign to the Azure Container Apps connected environment.

1. Set the following environment variables to the desired name of the custom location and for the ID of the Azure Arc-connected cluster.

Azure CLI

Azure CLI

```
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:

Azure CLI

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 a Microsoft Entra user with restricted permissions on the cluster resource.

3. Validate that the custom location is successfully created with the following command. The output should show the `provisioningState` property as `Succeeded`. If not, rerun the command 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.

Azure CLI

Azure CLI

```
CUSTOM_LOCATION_ID=$(az customlocation show \
--resource-group $GROUP_NAME \
--name $CUSTOM_LOCATION_NAME \
--query id \
--output tsv)
```

## Create the Azure Container Apps connected environment

Before you can start creating apps in the custom location, you need an [Azure Container Apps connected environment](#).

1. Create the Container Apps connected environment:

Azure CLI

```
az containerapp connected-env create \
--resource-group $GROUP_NAME \
--name $CONNECTED_ENVIRONMENT_NAME \
--custom-location $CUSTOM_LOCATION_ID \
--location $LOCATION
```

2. Validate that the Container Apps connected 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 containerapp connected-env show --resource-group $GROUP_NAME --name
$CONNECTED_ENVIRONMENT_NAME
```

## Next steps

[Create a container app on Azure Arc](#)

## Feedback

Was this page helpful?

 Yes

 No

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

# Tutorial: Create an Azure Container App on Azure Arc-enabled Kubernetes (Preview)

Article • 03/20/2023

In this tutorial, you create a Container app to an Azure Arc-enabled Kubernetes cluster (Preview) and learn to:

- ✓ Create a container app on Azure Arc
- ✓ View your application's diagnostics

## Prerequisites

Before you proceed to create a container app, you first need to set up an [Azure Arc-enabled Kubernetes cluster](#) to run [Azure Container Apps](#).

## Add Azure CLI extensions

Launch the Bash environment in [Azure Cloud Shell](#).



Next, add the required Azure CLI extensions.

### ⚠️ Warning

The following command installs a custom Container Apps extension that can't be used with the public cloud service. You need to uninstall the extension if you switch back to the Azure public cloud.

Azure CLI

```
az extension add --upgrade --yes --name customlocation
az extension remove --name containerapp
az extension add -s https://aka.ms/acaarccli/containerapp-1latest-py2.py3-
none-any.whl --yes
```

## Create a resource group

Create a resource group for the services created in this tutorial.

Azure CLI

```
myResourceGroup="my-container-apps-resource-group"
az group create --name $myResourceGroup --location eastus
```

## Get custom location information

Get the following location group, name, and ID from your cluster administrator. See [Create a custom location](#) for details.

Azure CLI

```
customLocationGroup="<RESOURCE_GROUP_CONTAINING_CUSTOM_LOCATION>"
```

Azure CLI

```
customLocationName="<NAME_OF_CUSTOM_LOCATION>"
```

Get the custom location ID.

Azure CLI

```
customLocationId=$(az customlocation show \
 --resource-group $customLocationGroup \
 --name $customLocationName \
 --query id \
 --output tsv)
```

## Retrieve connected environment ID

Now that you have the custom location ID, you can query for the connected environment.

A connected environment is largely the same as a standard Container Apps environment, but network restrictions are controlled by the underlying Arc-enabled Kubernetes cluster.

azure

```
myContainerApp="my-container-app"
myConnectedEnvironment=$(az containerapp connected-env list --custom-
```

```
location $customLocationId -o tsv --query '[] .id')
```

## Create an app

The following example creates a Node.js app.

Azure CLI

```
az containerapp create \
 --resource-group $myResourceGroup \
 --name $myContainerApp \
 --environment $myConnectedEnvironment \
 --environment-type connected \
 --image mcr.microsoft.com/azuredocs/containerapps-helloworld:latest \
 --target-port 80 \
 --ingress 'external'

az containerapp browse --resource-group $myResourceGroup --name
$myContainerApp
```

## Get diagnostic logs using Log Analytics

### ⓘ Note

A Log Analytics configuration is required as you [install the Container Apps extension](#) to view diagnostic information. If you installed the extension without Log Analytics, skip this step.

Navigate to the [Log Analytics workspace that's configured with your Container Apps extension](#), then select **Logs** in the left navigation.

Run the following sample query to show logs over the past 72 hours.

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();
ContainerAppConsoleLogs_CL
| where TimeGenerated between (StartTime .. EndTime)
| where ContainerAppName_s =~ "my-container-app"
```

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 `ContainerAppConsoleLogs_CL`.

- `Log_s` contains application logs for a given Container Apps extension
- `AppName_s` contains the Container App app name. In addition to logs you write via your application code, the `Log_s` column also contains logs on container startup and shutdown.

You can learn more about log queries in [getting started with Kusto](#).

## Next steps

- [Communication between microservices](#)

# Azure, Dynamics 365, Microsoft 365, and Power Platform compliance offerings

Article • 04/05/2023

You're wholly responsible for ensuring your own compliance with all applicable laws and regulations. Information provided in Microsoft online documentation doesn't constitute legal advice, and you should consult your legal advisor for any questions regarding regulatory compliance.

## Overview

Azure is a multi-tenant hyperscale cloud platform that is available in more than 60 [regions](#) worldwide. Most Azure services enable you to specify the region where your [customer data](#) will be [located](#). Microsoft may [replicate](#) your customer data to other regions within the same geography for data resiliency but Microsoft won't replicate your customer data outside the chosen geography (for example, United States).

Microsoft makes the following Azure cloud environments available:

- **Azure** is available globally. It is sometimes referred to as Azure commercial, Azure public, or Azure global.
- **Azure China** is a physically separated instance of cloud services located in China. It's independently operated and transacted by 21Vianet, one of the country's largest Internet providers.
- **Azure Government** is available from five regions in the United States to US government agencies and their partners. Two regions (US DoD Central and US DoD East) are reserved for exclusive use by the US Department of Defense.
- **Azure Government Secret** is available from three regions exclusively for the needs of US Government and designed to accommodate classified Secret workloads and native connectivity to classified networks.
- **Azure Government Top Secret** serves the national security mission and empowers leaders across the Intelligence Community (IC), Department of Defense (DoD), and Federal Civilian agencies to process national security workloads classified at the US Top Secret level.

To help you meet your own compliance obligations across regulated industries and markets worldwide, Azure maintains the largest compliance portfolio in the industry both in terms of breadth (total number of offerings), as well as depth (number of

[customer-facing services](#) in assessment scope). For service availability, see [Products available by region](#).

Compliance offerings are grouped into four segments: **globally applicable, US government, industry specific, and region/country specific**. Compliance offerings are based on various types of assurances, including formal certifications, attestations, validations, authorizations, and assessments produced by independent third-party auditing firms, as well as contractual amendments, self-assessments, and customer guidance documents produced by Microsoft. Each offering description provides links to downloadable resources to assist you with your own compliance obligations.

## Services in audit scope

Azure compliance certificates and audit reports state clearly which cloud services are in scope for independent third-party audits. Different audits may have different online services in audit scope. The following Azure, Dynamics 365, Microsoft 365, and Power Platform online services are covered in various Azure audit documents:

- Azure (for detailed insight, see Azure certificates and audit reports or [Cloud services in audit scope](#))
- Azure DevOps (see separate Azure DevOps certificates and audit reports)
- Dynamics 365 (for detailed insight, see Azure certificates and audit reports or [Cloud services in audit scope](#))
- Intelligent Recommendations
- Microsoft 365 Defender (formerly Microsoft Threat Protection)
- Microsoft AppSource
- Microsoft Bing for Commerce
- Microsoft Cloud for Financial Services
- Microsoft Defender for Cloud (formerly Azure Security Center)
- Microsoft Defender for Cloud Apps (formerly Microsoft Cloud App Security)
- Microsoft Defender for Endpoint (formerly Microsoft Defender Advanced Threat Protection)
- Microsoft Defender for Identity (formerly Azure Advanced Threat Protection)
- Microsoft Defender for IoT (formerly Azure Defender for IoT)
- Microsoft Graph
- Microsoft Intune
- Microsoft Managed Desktop
- Microsoft Sentinel (formerly Azure Sentinel)
- Microsoft Stream
- Microsoft Threat Experts
- Nomination Portal

- Power Apps
- Power Automate (formerly Microsoft Flow)
- Power BI
- Power BI Embedded
- Power Virtual Agents
- Universal Print
- Update Compliance

Office 365 services are covered in separate compliance certificates and audit reports maintained by Office 365. For more information, see [Microsoft 365 compliance documentation](#).

## Audit documentation

You must have an existing subscription or free trial account in [Azure](#) or [Azure Government](#) to download audit documents.

You can access Azure, Dynamics 365, Power Platform, and other Microsoft cloud services audit documentation via the [Service Trust Portal](#) (STP). You must sign in to access audit documentation on the STP. For more information, see [Get started with Microsoft Service Trust Portal](#). You can then download audit certificates, assessment reports, and other applicable documents to help you with your own regulatory requirements.

You can find audit documentation in the following STP folders:

- [ISO](#) for ISO certificates and assessment reports.
- [SOC](#) for SOC 1, SOC 2, and SOC 3 attestation reports.
- [PCI](#) for PCI DSS and PCI 3DS attestations of compliance and related documents.
- [Healthcare and Life Sciences](#) for HITRUST certification letters and related documents.
- [FedRAMP](#) for Azure Commercial FedRAMP System Security Plan and penetration test reports.
- [United States Government](#) for various attestation letters applicable to Azure and Azure Government, including DFARS, CNSSI 1253, MARS-E, NIST SP 800-161, NIST SP 800-171, IRS 1075, and others.
- And other STP folders applicable to industry and regional compliance offerings.

For access to Azure Government Secret or Azure Government Top Secret documentation, contact your Microsoft account team.

## Improve your regulatory compliance

Microsoft Defender for Cloud helps streamline the process for meeting regulatory compliance requirements, using the **regulatory compliance dashboard**. Defender for Cloud continuously assesses your hybrid cloud environment to analyze the risk factors according to the controls and best practices in the standards that you've applied to your subscriptions. The dashboard reflects the status of your compliance with these standards. For more information, see [Tutorial: Improve your regulatory compliance](#).

## Resources

- [Azure compliance documentation](#)
- [Microsoft 365 compliance offerings](#)
- [Compliance on the Microsoft Trust Center](#) ↗

# Azure security baseline for Azure Container Apps

Article • 09/20/2023

This security baseline applies guidance from the [Microsoft cloud security benchmark version 1.0](#) to Azure Container Apps. 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 Azure Container Apps.

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 Azure Container Apps have been excluded. To see how Azure Container Apps completely maps to the Microsoft cloud security benchmark, see the [full Azure Container Apps security baseline mapping file](#).

## Security profile

The security profile summarizes high-impact behaviors of Azure Container Apps, which may result in increased security considerations.

[+] Expand table

Service Behavior Attribute	Value
Product Category	Containers
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](#).

[+] Expand table

Supported	Enabled By Default	Configuration Responsibility
True	False	Customer

**Configuration Guidance:** Deploy the service into a virtual network. Assign private IPs to the resource (where applicable) unless there is a strong reason to assign public IPs directly to the resource.

**Reference:** [Azure Container Apps Virtual Network Integration](#)

#### Network Security Group Support

**Description:** Service network traffic respects Network Security Groups rule assignment on its subnets. [Learn more](#).

[+] Expand table

Supported	Enabled By Default	Configuration Responsibility
True	False	Customer

**Configuration Guidance:** Use network security groups (NSG) to restrict or monitor traffic by port, protocol, source IP address, or destination IP address. Create NSG rules to restrict your service's open ports (such as preventing management ports from being accessed from untrusted networks). Be aware that by default, NSGs deny all inbound traffic but allow traffic from virtual network and Azure Load Balancers.

## NS-2: Secure cloud services with network controls

### Features

#### 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.](#)

[+] [Expand table](#)

Supported	Enabled By Default	Configuration Responsibility
True	False	Customer

**Configuration Guidance:** Disable public network access by deploying an internal-only container apps environment configuration.

Reference: [Provide a virtual network to an internal Azure Container Apps environment](#)

## 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.](#)

[+] [Expand table](#)

<b>Supported</b>	<b>Enabled By Default</b>	<b>Configuration Responsibility</b>
True	False	Customer

**Configuration Guidance:** Use Azure Active Directory (Azure AD) as the default authentication method to control your data plane access.

**Reference:** [Enable authentication and authorization in Azure Container Apps with Azure Active Directory](#)

## 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](#).

[ ] Expand table

<b>Supported</b>	<b>Enabled By Default</b>	<b>Configuration Responsibility</b>
False	Not Applicable	Not Applicable

**Configuration Guidance:** This feature is not supported to secure this service.

## IM-3: Manage application identities securely and automatically

### Features

#### Managed Identities

**Description:** Data plane actions support authentication using managed identities. [Learn more](#).

[ ] Expand table

<b>Supported</b>	<b>Enabled By Default</b>	<b>Configuration Responsibility</b>
True	False	Customer

**Feature notes:** Managed Identity is supported for Container Apps and Dapr components but not yet for scale rules on a Container App

**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.

**Reference:** [Using Managed Identity in Azure Container Apps](#)

## Service Principals

**Description:** Data plane supports authentication using service principals. [Learn more](#).

[+] Expand table

Supported	Enabled By Default	Configuration Responsibility
True	False	Customer

**Configuration Guidance:** There is no current Microsoft guidance for this feature configuration. Please review and determine if your organization wants to configure this security feature.

## 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](#).

[+] Expand table

Supported	Enabled By Default	Configuration Responsibility
False	Not Applicable	Not Applicable

**Configuration Guidance:** This feature is not supported to secure this service.

## 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](#).

[+] [Expand table](#)

Supported	Enabled By Default	Configuration Responsibility
True	Applicable	Applicable

**Feature notes:** For Dapr-enabled Container Apps, customers can leverage Azure Key Vault for secret references.

**Configuration Guidance:** This feature is not supported to secure this service.

## Privileged access

For more information, see the [Microsoft cloud security benchmark: Privileged access](#).

### PA-1: Separate and limit highly privileged/administrative users

## Features

### Local Admin Accounts

**Description:** Service has the concept of a local administrative account. [Learn more](#).

[+] [Expand table](#)

Supported	Enabled By Default	Configuration Responsibility
False	Not Applicable	Not Applicable

**Configuration Guidance:** This feature is not supported to secure this service.

## PA-7: Follow just enough administration (least privilege) principle

### Features

#### Azure RBAC for Data Plane

Description: Azure Role-Based Access Control (Azure RBAC) can be used to manage access to service's data plane actions. [Learn more](#).

[+] [Expand table](#)

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](#).

[+] [Expand table](#)

Supported	Enabled By Default	Configuration Responsibility
False	Not Applicable	Not Applicable

Configuration Guidance: This feature is not supported to secure this service.

## 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.](#)

[+] [Expand table](#)

Supported	Enabled By Default	Configuration Responsibility
False	Not Applicable	Not Applicable

**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.](#)

[+] [Expand table](#)

Supported	Enabled By Default	Configuration Responsibility
False	Not Applicable	Not Applicable

**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.](#)

[+] Expand table

Supported	Enabled By Default	Configuration Responsibility
True	False	Customer

**Configuration Guidance:** Enable secure transfer in services where there is a native data in transit encryption feature built in. Enforce HTTPS on any web applications and services and ensure TLS v1.2 or later is used. Legacy versions such as SSL 3.0, TLS v1.0 should be disabled. For remote management of Virtual Machines, use SSH (for Linux) or RDP/TLS (for Windows) instead of an unencrypted protocol.

Reference: [Set up HTTPS or TCP ingress in Azure Container Apps](#)

## 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](#).

[+] Expand table

Supported	Enabled By Default	Configuration Responsibility
True	True	Microsoft

**Feature notes:** Azure Container Apps leverages Microsoft's default encryption for data at rest.

**Configuration Guidance:** No additional configurations are required as this is enabled on a default deployment.

Reference: [Double encryption](#)

## 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.](#)

[+] [Expand table](#)

Supported	Enabled By Default	Configuration Responsibility
False	Not Applicable	Not Applicable

**Configuration Guidance:** This feature is not supported to secure this service.

## 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.](#)

[+] [Expand table](#)

Supported	Enabled By Default	Configuration Responsibility
False	Not Applicable	Not Applicable

**Configuration Guidance:** This feature is not supported to secure this service.

## 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.](#)

[+] [Expand table](#)

Supported	Enabled By Default	Configuration Responsibility
False	Not Applicable	Not Applicable

**Configuration Guidance:** This feature is not supported to secure this 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.](#)

[ ] Expand table

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.

**Reference:** [Azure Policy built-in definitions for Azure Container Apps](#)

## 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.](#)

[+] [Expand table](#)

Supported	Enabled By Default	Configuration Responsibility
False	Not Applicable	Not Applicable

**Configuration Guidance:** This feature is not supported to secure this service.

## 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.](#)

[+] [Expand table](#)

Supported	Enabled By Default	Configuration Responsibility
True	True	Microsoft

**Configuration Guidance:** No additional configurations are required as this is enabled on a default deployment.

**Reference:** [Log storage and monitoring options in Azure Container Apps](#)

## Posture and Vulnerability Management

*For more information, see the [Microsoft cloud security benchmark: Posture and vulnerability management](#).*

## PV-3: Establish secure configurations for compute resources

### Features

#### Custom Containers Images

**Description:** Service supports using user-supplied container images or pre-built images from the marketplace with certain baseline configurations pre-applied. [Learn more](#)

[ ] [Expand table](#)

Supported	Enabled By Default	Configuration Responsibility
True	False	Customer

**Configuration Guidance:** You can pull images from private repositories in Microsoft Azure Container Registry using managed identities for authentication to avoid the use of administrative credentials. You can use a system-assigned or user-assigned managed identity to authenticate with Azure Container Registry.

**Reference:** [Azure Container Apps image pull with managed identity](#)

## PV-5: Perform vulnerability assessments

### Features

#### Vulnerability Assessment using Microsoft Defender

**Description:** Service can be scanned for vulnerability scan using Microsoft Defender for Cloud or other Microsoft Defender services embedded vulnerability assessment capability (including Microsoft Defender for server, container registry, App Service, SQL, and DNS). [Learn more](#)

[ ] [Expand table](#)

Supported	Enabled By Default	Configuration Responsibility
False	Not applicable	Not applicable

**Configuration Guidance:** Though Container Apps does not support vulnerability assessment performed by Defender for Containers, the Azure Container Registry that may be integrated with Container Apps does support vulnerability assessment.

**Reference:** [Use Defender for Containers to scan your Azure Container Registry images for vulnerabilities](#)

## 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](#).

[+] Expand table

Supported	Enabled By Default	Configuration Responsibility
False	Not Applicable	Not Applicable

**Configuration Guidance:** This feature is not supported to secure this service.

##### Service Native Backup Capability

**Description:** Service supports its own native backup capability (if not using Azure Backup). [Learn more](#).

[+] Expand table

Supported	Enabled By Default	Configuration Responsibility
False	Not Applicable	Not Applicable

**Configuration Guidance:** This feature is not supported to secure this service.

## Next steps

- See the [Microsoft cloud security benchmark overview](#)
- Learn more about [Azure security baselines](#)

# Azure Container Apps samples

Article • 08/30/2024

Refer to the following samples to learn how to use Azure Container Apps in different contexts and paired with different technologies.

[+] Expand table

Name	Description
<a href="#">A/B Testing your ASP.NET Core apps using Azure Container Apps ↗</a>	Shows how to use Azure App Configuration, ASP.NET Core Feature Flags, and Azure Container Apps revisions together to gradually release features or perform A/B tests.
<a href="#">gRPC with ASP.NET Core on Azure Container Apps ↗</a>	This repository contains a simple scenario that demonstrates how an ASP.NET Core 6.0 app is built as a cloud-native application. The application is hosted in Azure Container Apps that uses gRPC request/response transmission from Worker microservices. The gRPC service simultaneously streams sensor data to a Blazor server frontend, so you can see the data charted in real-time.
<a href="#">Deploy an Orleans Cluster to Container Apps ↗</a>	An end-to-end sample and tutorial for getting a Microsoft Orleans cluster running on Azure Container Apps. Worker microservices rapidly transmit data to a back-end Orleans cluster for monitoring and storage, emulating thousands of physical devices in the field.
<a href="#">Deploy a shopping cart Orleans app to Container Apps ↗</a>	An end-to-end example shopping cart app built in ASP.NET Core Blazor Server with Orleans deployed to Azure Container Apps.
<a href="#">ASP.NET Core front-end with two back-end APIs on Azure Container Apps ↗</a>	This sample demonstrates ASP.NET Core 6.0 can be used to build a cloud-native application hosted in Azure Container Apps.
<a href="#">ASP.NET Core front-end with two back-end APIs on Azure Container Apps (with Dapr) ↗</a>	Demonstrates how ASP.NET Core 6.0 is used to build a cloud-native application hosted in Azure Container Apps using Dapr.
<a href="#">Deploy Drupal on Azure Container Apps ↗</a>	Demonstrates how to deploy a Drupal site to Azure Container Apps, with Azure Database for MariaDB, and Azure Files to store static assets.
<a href="#">Launch Your First Java app ↗</a>	A monolithic Java application called PetClinic built with Spring Framework. PetClinic is a well-known sample application provided by the Spring Framework community.

---

# Feedback

Was this page helpful?

 Yes

 No

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

# Azure Container Apps frequently asked questions (FAQs)

FAQ

This article lists commonly asked questions about Azure Container Apps together with related answers.

## APIs

**Does Azure Container Apps provide direct access to the underlying Kubernetes API?**

No, there's no access to the Kubernetes API.

**Can I import my Azure Container Apps API from the context of API Management?**

Yes.

## Billing

**How is Azure Container Apps billed?**

Refer to the [billing](#) page for details.

## Configuration

**Can I set up GitHub Actions to automatically build and deploy my code to Azure Container Apps?**

Yes. Using Azure CLI, run `az containerapp github-action -h` to see the options. Using Azure portal, go to the "Continuous deployment" window under your container app.

# Why is the URL my app receives different than the URL specified in the request?

Azure Container Apps decodes the URL to protect your app against [URL confusion attacks](#). A request URL that has encoded portions, such as

`http://mysite.com/archive/http%3A%2F%2Fmysite.com%2Farchive%2F123`, is sent to your app as `http://mysite.com/archive/http%3A/mysite.com/archive/123`.

# Do Consumption only environments support custom user-defined routes?

For Consumption only environments, express routes are unsupported, and limited UDR when configured as follows is supported. The UDR configuration must have a route configured for `Azure.<REGION_NAME>` Service tag with Next Hop = "Internet". In addition, the rules in the [NSG documentation](#) must be configured for the Consumption only environment to be operational. These limitations do not apply for workload profiles, and for full featured UDR and Express Route support, use workload profile environments.

# Data management

## Where does Azure Container Apps store customer data?

Azure Container Apps doesn't move or store customer data out of the deployed region.

# Quotas

## How can I request a quota increase?

[Request a quota increase in the Azure portal](#) with Azure Container Apps selected as the provider.

Keep in mind the following when it comes to quota increase requests:

- **Scaling apps vs environments:** There are many different quotas available to increase. Use these descriptions to help identify your needs:
  - **Increase apps and cores per environment:** Allows you to run more apps within an environment and/or more intensive apps. Recommended if your workloads

can deploy within the same network and security boundaries.

- **Increasing environments:** Recommended if your workloads need network or security boundaries. Note: A detailed business context might be required if your request involves increasing environment-level quotas. When you request a change to your regional environment quota, you should request a corresponding change to your global environment quota.
- **Regions:** Approvals for increase requests vary based on compute capacity available in Azure regions.
- **Specific compute requirements:** The platform supports 4 GB per container app. Memory limits overrides are evaluated on a per-case basis.
- **Business reasoning for scaling:** You might be eligible for a quota increase request if the platform limits are blocking your workload demands. Scale limits overrides are evaluated on a per-case basis.

## Microservice APIs powered by Dapr

### What Dapr features and APIs are available in Azure Container Apps?

Each Dapr capability undergoes thorough evaluation to ensure it positively impacts customers running microservices in the Azure Container Apps environment, while providing the best possible experience.

### Are alpha Dapr APIs and Tier 2 components supported or available in Azure Container Apps?

The availability of Dapr's alpha APIs is not guaranteed or Microsoft-supported.

While Tier 1 components are fully supported, Tier 2 components are supported with best effort. [Learn more.](#)

### What is the Dapr version release cadence in Azure Container Apps?

Dapr's typical release timeline is up to six weeks after [the Dapr OSS release ↗](#). The latest

Dapr version is made available in Azure Container Apps only after rigorous testing.

Rolling out to all regions can take up to two weeks or longer.

# How can I request a Dapr feature enhancement for Azure Container Apps?

You can submit a feature request via the [Azure Container Apps GitHub repository](#). Make sure to include "Dapr" in the feature request title.

## Dockerless deployments

### What is a Docker-less deployment?

A Docker-less deployment allows you to deploy your application without defining a Dockerfile in your code. Instead, the Container Apps cloud build functionality uses Buildpacks to turn source code on your local machine into a container image. This option uses the Azure Container Apps default registry.

### During the deployment of my Docker-less application, messages about "ImagePullBackOff on legion", "Kubernetes error" or "Gateway error" appear and my application doesn't deploy successfully.

You are experiencing a known issue with Docker-less deployments. Retrying might resolve this for you. If you run into this issue, open a [GitHub issue](#) so our team can investigate.

## Deploy .NET applications

### What if my .NET application fails to scale?

You need to enable data protection for all .NET apps on Azure Container Apps. See [Deploying and scaling an ASP.NET Core app on Azure Container Apps](#) for details.

## Deploy Java applications

### Which JDK versions are supported and how can I configure the JDK version?

Container Apps supports four LTS JDK versions: JDK 8, JDK 11, JDK 17 and JDK 21. For source code build, the default version is JDK 17. For a JAR file build, the JDK version is read from the file location *META-INF\MANIFEST.MF* in the JAR, but uses the default JDK version 17 if the specified version isn't available.

You can configure JDK version to override the default version via [build environment variables](#).

## Which Java build tools are supported?

Maven

## How can I customize a Java image build from source code?

You can customize a Java image build via [build environment variables](#).

## How do I ensure the build and image of my Docker-less build are available in the same region as my app?

When using `containerapp up` in combination with a Docker-less code base, use the `--location` parameter so that application runs in a location other than US East.

---

## Feedback

Was this page helpful?

 Yes

 No

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