

Azure Container Instances documentation

Run Docker containers on-demand in a managed, serverless Azure environment. Azure Container Instances is a solution for any scenario that can operate in isolated containers, without orchestration. Run event-driven applications, quickly deploy from your container development pipelines, and run data processing and build jobs.

About Azure Container Instances

OVERVIEW

[What is Azure Container Instances?](#)

[Container groups](#)

[FAQ](#)

Get started

QUICKSTART

[Deploy container instance - CLI](#)

[Deploy container instance - Portal](#)

[Deploy container instance - ARM template](#)

[Deploy container instance - Docker CLI](#)

TRAINING

[Run Docker containers with Azure Container Instances](#)

Create multi-container groups

TUTORIAL

[Deploy container group - YAML](#)

[Deploy container group - ARM template](#)

Deployment scenarios

HOW-TO GUIDE

[Connect to a virtual network](#)

[Configure restart policy](#)

[Set environment variables](#)

Manage container apps

HOW-TO GUIDE

[Stop and start containers](#)

[Update running containers](#)

[Share data between containers](#)

Integrate with Azure

HOW-TO GUIDE

[Deploy image from Azure Container Registry](#)

[Use managed identity](#)

[Mount an Azure file share](#)

Deploy confidential container groups

TUTORIAL

[Deploy a confidential container group - Resource Manager](#)

[Deploy a confidential container group - portal](#)

What is Azure Container Instances?

Article • 11/14/2024

Containers are becoming the preferred way to package, deploy, and manage cloud applications. Azure Container Instances offers the fastest and simplest way to run Linux or Windows containers in Azure, without having to manage any virtual machines and without having to adopt a higher-level service.

ACI supports [regular](#), [confidential](#), and [Spot](#) containers. ACI can be used as [single-instance](#) or multi-instance via [NGroups](#), or you can get orchestration capabilities by deploying pods in your Azure Kubernetes Service (AKS) cluster via [virtual nodes on ACI](#). For even faster startup times, ACI supports [standby pools](#).

Fast startup times

Containers offer significant startup benefits over virtual machines (VMs). Azure Container Instances can start containers in Azure in seconds, without the need to provision and manage VMs.

Bring Linux or Windows container images from Docker Hub, a private [Azure container registry](#), or another cloud-based Docker registry. Visit the [FAQ](#) to learn which registries ACI supports. Azure Container Instances caches several common base OS images, helping speed deployment of your custom application images.

For even faster startup times, ACI supports [standby pools](#).

Container access

Azure Container Instances enables exposing your container groups directly to the internet with an IP address and a fully qualified domain name (FQDN). When you create a container instance, you can specify a custom DNS name label so your application is reachable at *customlabel.azureregion.azurecontainer.io*.

Azure Container Instances also supports executing a command in a running container by providing an interactive shell to help with application development and troubleshooting. Access takes place over HTTPS, using TLS to secure client connections.

 **Important**

Azure Container Instances requires all secure connections from servers and applications to use TLS 1.2. Support for TLS 1.0 and 1.1 has been retired.

Compliant deployments

Hypervisor-level security

Historically, containers offered application dependency isolation and resource governance but were insufficiently hardened for hostile multitenant usage. Azure Container Instances guarantees your application is as isolated in a container as it would be in a VM.

Customer data

The Azure Container Instances service doesn't store customer data. It does, however, store the subscription IDs of the Azure subscription used to create resources. Storing subscription IDs is required to ensure your container groups continue running as expected.

Custom sizes

Containers are typically optimized to run just a single application, but the exact needs of those applications can differ greatly. Azure Container Instances provides optimum utilization by allowing exact specifications of CPU cores and memory. You pay based on what you need and get billed by the second, so you can fine-tune your spending based on actual need.

For compute-intensive jobs such as machine learning, Azure Container Instances can schedule Linux containers to use NVIDIA Tesla [GPU resources](#) (preview).

Persistent storage

To retrieve and persist state with Azure Container Instances, we offer direct [mounting of Azure Files shares](#) backed by Azure Storage.

Linux and Windows containers

Azure Container Instances can schedule both Windows and Linux containers with the same API. You can specify your OS type preference when you create your [container groups](#).

Some features are currently restricted to Linux containers:

- Multiple containers per container group
- Volume mounting ([Azure Files](#), [emptyDir](#), [GitRepo](#), [secret](#))
- [Resource usage metrics](#) with Azure Monitor
- [GPU resources](#) (preview)

For Windows container deployments, use images based on common [Windows base images](#).

Run multiple containers in a single container group

Azure Container Instances supports scheduling of [multiple containers within a single container group](#) that share the same container host, local network, storage, and lifecycle. This enables you to combine your main application container with other supporting role containers, such as logging sidecars.

Virtual network deployment

Azure Container Instances enables [deployment of container instances into an Azure virtual network](#). When deployed into a subnet within your virtual network, container instances can communicate securely with other resources in the virtual network, including those that are on premises (through [VPN gateway](#) or [ExpressRoute](#)).

Availability zones support

Azure Container Instances supports [zonal container group deployments](#), meaning the instance is pinned to a specific, self-selected availability zone. The availability zone can be specified per container group.

Managed identity

Azure Container Instances supports using [managed identity with your container group](#), which enables your container group to authenticate to any service that supports Microsoft Entra authentication without managing credentials in your container code.

Managed identity authenticated image pull

Azure Container Instances can authenticate with an Azure Container Registry (ACR) instance [using a managed identity](#), allowing you to pull the image without having to include a username and password directly in your container group definition.

Confidential container deployment

Confidential containers on ACI enable you to run containers in a trusted execution environment (TEE) that provides hardware-based confidentiality and integrity protections for your container workloads. Confidential containers on ACI can protect data-in-use and encrypts data being processed in memory. Confidential containers on ACI are supported as a SKU that you can select when deploying your workload. For more information, see [confidential container groups](#).

Spot container deployment

ACI Spot containers allow customers to run interruptible, containerized workloads on unused Azure capacity at discounted prices of up to 70% compared to regular-priority ACI containers. ACI spot containers may be preempted when Azure encounters a shortage of surplus capacity, and they're suitable for workloads without strict availability requirements. Customers are billed for per-second memory and core usage. To utilize ACI Spot containers, you can deploy your workload with a specific property flag indicating that you want to use Spot container groups and take advantage of the discounted pricing model. For more information, see [Spot container groups](#).

NGroups

NGroups provides advanced capabilities for managing multiple related container groups. NGroups provides support for maintaining a specified number of container groups, performing rolling upgrades, deploying across multiple availability zones, using load balancers for ingress, and deploying confidential containers. For more information, see [About NGroups](#).

Virtual nodes on Azure Container Instances

[Virtual nodes on Azure Container Instances](#) allow you to deploy pods in your Azure Kubernetes Service (AKS) cluster that run as container groups in ACI. This allows you to orchestrate your container groups using familiar Kubernetes constructs. Since virtual

nodes are backed by ACI's serverless infrastructure, you can quickly scale up your workload without needing to wait for the Kubernetes cluster autoscaler to deploy VM compute nodes.

Considerations

User's credentials passed via command line interface (CLI) are stored as plain text in the backend. Storing credentials in plain text is a security risk; Microsoft advises customers to store user credentials in CLI environment variables to ensure they're encrypted/transformed when stored in the backend.

If your container group stops working, we suggest trying to restart your container, checking your application code, or your local network configuration before opening a [support request](#).

Container Images can't be larger than 15 GB, any images above this size may cause unexpected behavior: [How large can my container image be?](#)

Some Windows Server base images are no longer compatible with Azure Container Instances:

[What Windows base OS images are supported?](#)

If a container group restarts, the container group's IP may change. We advise against using a hard coded IP address in your scenario. If you need a static public IP address, use Application Gateway: [Static IP address for container group - Azure Container Instances | Microsoft Learn](#)

There are ports that are reserved for service functionality. We advise you not to use these ports because using them leads to unexpected behavior: [Does the ACI service reserve ports for service functionality?](#)

If you're having trouble deploying or running your container, first check the [Troubleshooting Guide](#) for common mistakes and issues

Your container groups may restart due to platform maintenance events. These maintenance events are done to ensure the continuous improvement of the underlying infrastructure: [Container had an isolated restart without explicit user input](#)

ACI doesn't allow [privileged container operations](#). We advise you to not depend on using the root directory for your scenario.

Next steps

Try deploying a container to Azure with a single command using our quickstart guide:

[Azure Container Instances Quickstart](#)

Feedback

Was this page helpful?

 Yes

 No

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

Resource availability & quota limits for ACI

Article • 03/27/2025

This article details the availability and quota limits of Azure Container Instances compute, memory, and storage resources in Azure regions and by target operating system. For a general list of available regions for Azure Container Instances, see [available regions](#).

Values presented are the maximum resources available per deployment of a [container group](#). Values are current at time of publication.

Note

Container groups created within these resource limits are subject to availability within the deployment region. When a region is under heavy load, you may experience a failure when deploying instances. To mitigate such a deployment failure, try deploying instances with lower resource settings, or try your deployment at a later time or in a different region with available resources.

Default Quota Limits

All Azure services include certain default limits and quotas for resources and features. This section details the default quotas and limits for Azure Container Instances.

Use the [List Usage API](#) to review current quota usage in a region for a subscription.

Certain default limits and quotas can be increased. To request an increase of one or more resources that support such an increase, submit an [Azure support request](#) (select "Quota" for **Issue type**).

Important

Not all limit increase requests are guaranteed to be approved. Deployments with GPU resources aren't supported in an Azure virtual network deployment and are only available on Linux container groups. Using GPU resources (preview) isn't fully supported yet and any support is provided on a best-effort basis.

Unchangeable (Hard) Limits

The following limits are default limits that can't be increased through a quota request. Any quota increase requests for these limits won't be approved.

[+] Expand table

Resource	Actual Limit
Number of containers per container group	60
Number of volumes per container group	20
Ports per IP	5
Container instance log size - running instance	4 MB
Container instance log size - stopped instance	16 KB or 1,000 lines

Changeable Limits (Eligible for Quota Increases)

[+] Expand table

Resource	Actual Limit
Standard sku container groups per region per subscription	100
Standard sku cores (CPUs) per region per subscription	100
Standard sku cores (CPUs) for V100 GPU per region per subscription	0
Container group creates per hour	300 ¹
Container group creates per 5 minutes	100 ¹
Container group deletes per hour	300 ¹
Container group deletes per 5 minutes	100 ¹

Note

1: Indicates that the feature maximum is configurable and may be increased through a support request. For more information on how to request a quota increase, see the [How to request a quota increase section of Increase VM-family vCPU quotes](#).

You can also create a support ticket if you'd like to discuss your specific needs with the support team.

Standard Container Resources

By default, the following resources are available general purpose (standard core SKU) containers in general deployments and [Azure virtual network](#) deployments) for Linux & Windows containers. These maximums are hard limits and can't be increased.

Generally Available Limits:

[+] Expand table

Max CPU	Max Memory (GB)	Virtual network Max CPU	Virtual network Max Memory (GB)	Storage (GB)
4	16	4	16	50

For a general list of available regions for Azure Container Instances, see [available regions](#).

Preview Limits:

Note

Big Container SKUs are in preview, and you may experience errors in some regions that you deploy. Preview features aren't ideal for critical workloads. If you're experiencing errors when deploying containers with greater than 4 vCPU and 16 GB in a certain region, please submit an [Azure support request](#) (select "Quota" for Support type).

[+] Expand table

Max CPU	Max Memory (GB)	Virtual network Max CPU	Virtual network Max Memory (GB)	Storage (GB)
32	256	32	256	50

The following resources are available in all Azure Regions supported by Azure Container Instances. For a general list of available regions for Azure Container Instances, see

available regions [↗](#).

Confidential Container Resources

The following maximum resources are available to a container group deployed using [Confidential Containers](#). These maximums are hard limits and can't be increased.

Generally Available Limits:

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

Max CPU	Max Memory (GB)	Virtual network Max CPU	Virtual network Max Memory (GB)	Storage (GB)
4	16	4	256	50

Preview Limits:

Note

Big Container SKUs are in preview, and you may experience errors in some regions that you deploy. Preview features aren't ideal for critical workloads. If you're experiencing errors when deploying containers with greater than 4 vCPU and 16 GB in a certain region, please submit an [Azure support request](#) [↗](#) (select "Quota" for **Support type**).

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

Max CPU	Max Memory (GB)	Virtual network Max CPU	Virtual network Max Memory (GB)	Storage (GB)
32	192	32	192	50

Spot Container Resources (Preview)

The following maximum resources are available to a container group deployed using [Spot Containers](#) (preview). These maximums are hard limits and can't be increased.

Note

Spot Containers are only available in the following regions at this time: East US 2, West Europe, and West US.

[+] Expand table

Max CPU	Max Memory (GB)	Virtual network Max CPU	Virtual network Max Memory (GB)	Storage (GB)
4	16	N/A	N/A	50

GPU Container Resources (Preview)

ⓘ Important

K80 and P100 GPU SKUs were retired on August 31st, 2023. This is due to the retirement of the underlying VMs used: [NC Series](#) and [NCv2 Series](#). Although V100 SKUs will be available, it's recommended to use Azure Kubernetes Service instead. GPU resources aren't fully supported and shouldn't be used for production workloads. Use the following resources to migrate to AKS today: [How to Migrate to AKS](#).

! Note

Not all limit increase requests are guaranteed to be approved. Deployments with GPU resources aren't supported in an Azure virtual network deployment and are only available on Linux container groups. Using GPU resources (preview) isn't fully supported yet and any support is provided on a best-effort basis.

The following maximum resources are available to a container group deployed with [GPU resources](#) (preview). These maximums are hard limits and can't be increased.

[+] Expand table

GPU SKUs	GPU count	Max CPU	Max Memory (GB)	Storage (GB)
V100	1	6	112	50
V100	2	12	224	50
V100	4	24	448	50

Next steps

Certain default limits and quotas can be increased. To request an increase of one or more resources that support such an increase, submit an [Azure support request](#) (select "Quota" for **Issue type**).

Let the team know if you'd like to see other regions or increased resource availability at aka.ms/aci/feedback.

For information on troubleshooting container instance deployment, see [Troubleshoot deployment issues with Azure Container Instances](#).

Feedback

Was this page helpful?

 Yes

 No

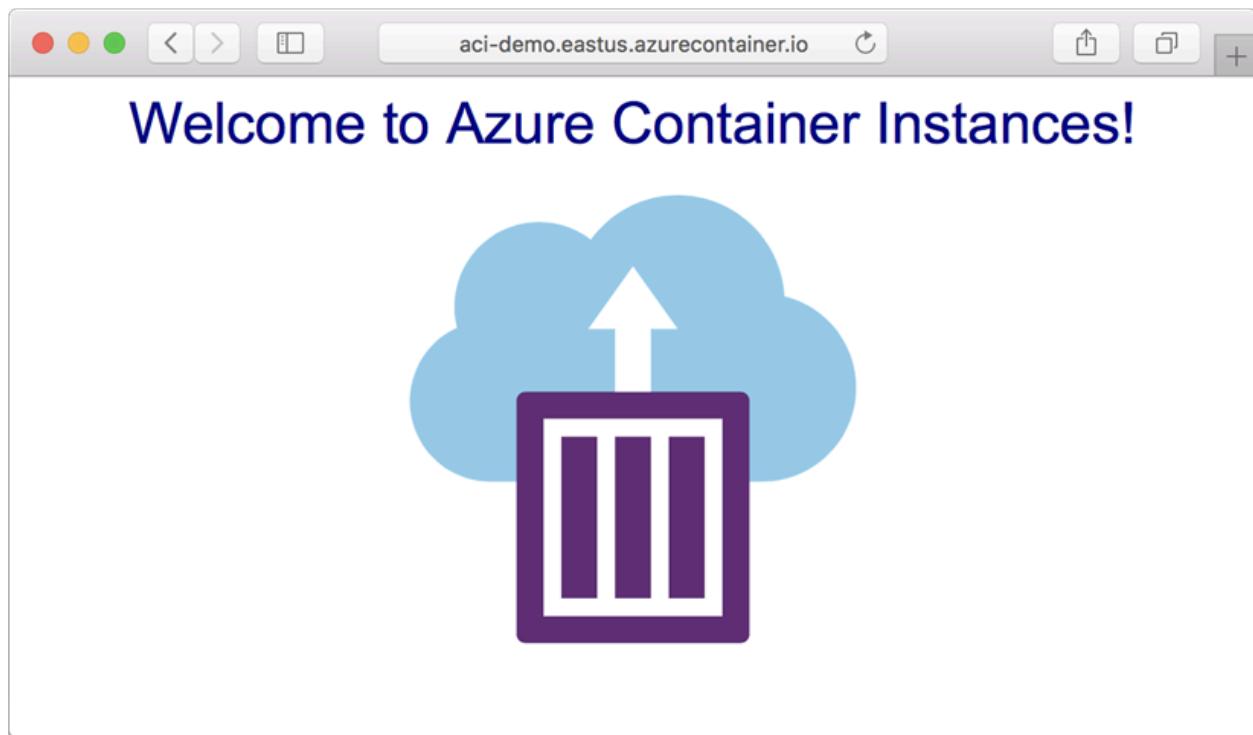
[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

Quickstart: Deploy a container instance in Azure using the Azure CLI

Article • 08/29/2024

Use Azure Container Instances to run serverless Docker containers in Azure with simplicity and speed. Deploy an application to a container instance on-demand when you don't need a full container orchestration platform like Azure Kubernetes Service.

In this quickstart, you use the Azure CLI to deploy an isolated Docker container and make its application available with a fully qualified domain name (FQDN). A few seconds after you execute a single deployment command, you can browse to the application running in the container:



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

Prerequisites

- Use the Bash environment in [Azure Cloud Shell](#). For more information, see [Quickstart for Bash in Azure Cloud Shell](#).
- If you prefer to run CLI reference commands locally, [install](#) the Azure CLI. If you're running on Windows or macOS, consider running Azure CLI in a Docker container.

[A Launch Cloud Shell](#) ↗

For more information, see [How to run the Azure CLI in a Docker container](#).

- If you're using a local installation, sign in to the Azure CLI by using the [az login](#) command. To finish the authentication process, follow the steps displayed in your terminal. For other sign-in options, see [Sign in with the Azure CLI](#).
- When you're prompted, install the Azure CLI extension on first use. For more information about extensions, see [Use extensions with the Azure CLI](#).
- Run [az version](#) to find the version and dependent libraries that are installed. To upgrade to the latest version, run [az upgrade](#).
- This quickstart requires version 2.0.55 or later of the Azure CLI. If using Azure Cloud Shell, the latest version is already installed.

⚠ Warning

Best practice: User's credentials passed via command line interface (CLI) are stored as plain text in the backend. Storing credentials in plain text is a security risk; Microsoft advises customers to store user credentials in CLI environment variables to ensure they are encrypted/transformed when stored in the backend.

Create a resource group

Azure container instances, like all Azure resources, must be deployed into a resource group. Resource groups allow you to organize and manage related Azure resources.

First, create a resource group named *myResourceGroup* in the *eastus* location with the following [az group create](#) command:

Azure CLI

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

Create a container

Now that you have a resource group, you can run a container in Azure. To create a container instance with the Azure CLI, provide a resource group name, container instance name, and Docker container image to the [az container create](#) command. In this

quickstart, you use the public `mcr.microsoft.com/azuredocs/aci-helloworld` image. This image packages a small web app written in Node.js that serves a static HTML page.

You can expose your containers to the internet by specifying one or more ports to open, a DNS name label, or both. In this quickstart, you deploy a container with a DNS name label so that the web app is publicly reachable.

Execute a command similar to the following to start a container instance. Set a `--dns-name-label` value that's unique within the Azure region where you create the instance. If you receive a "DNS name label not available" error message, try a different DNS name label.

Azure CLI

```
az container create --resource-group myResourceGroup --name mycontainer --image mcr.microsoft.com/azuredocs/aci-helloworld --dns-name-label aci-demo -ports 80
```

Within a few seconds, you should get a response from the Azure CLI indicating the deployment completed. Check its status with the `az container show` command:

Azure CLI

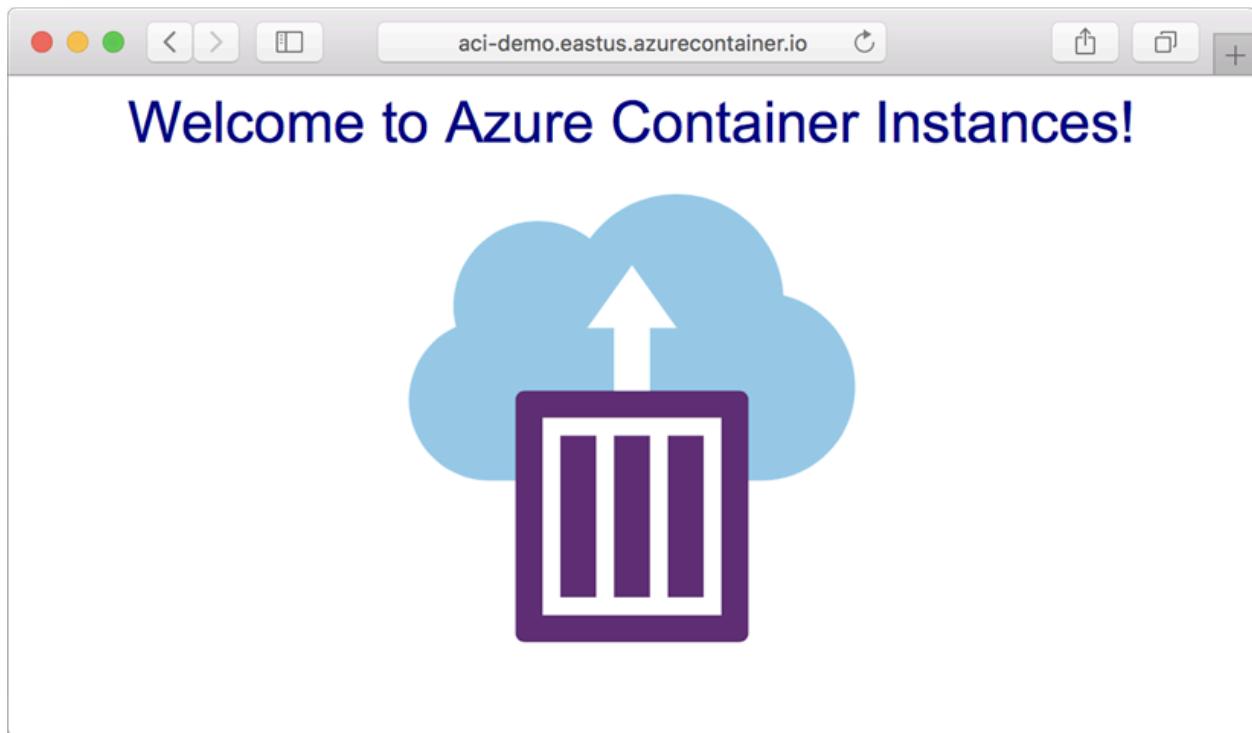
```
az container show --resource-group myResourceGroup --name mycontainer --query "{FQDN:ipAddress.fqdn,ProvisioningState:provisioningState}" --out table
```

When you run the command, the container's fully qualified domain name (FQDN) and its provisioning state are displayed.

Output

FQDN	ProvisioningState
aci-demo.eastus.azurecontainer.io	Succeeded

If the container's `ProvisioningState` is `Succeeded`, go to its FQDN in your browser. If you see a web page similar to the following, congratulations! You successfully deployed an application running in a Docker container to Azure.



If at first the application isn't displayed, you might need to wait a few seconds while DNS propagates, then try refreshing your browser.

Pull the container logs

When you need to troubleshoot a container or the application it runs (or just see its output), start by viewing the container instance's logs.

Pull the container instance logs with the [az container logs](#) command:

```
Azure CLI

az container logs --resource-group myResourceGroup --name mycontainer
```

The output displays the logs for the container, and should show the HTTP GET requests generated when you viewed the application in your browser.

```
Output

listening on port 80
::ffff:10.240.255.55 - - [21/Mar/2019:17:43:53 +0000] "GET / HTTP/1.1" 304 -
"--" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML,
like Gecko) Chrome/72.0.3626.121 Safari/537.36"
::ffff:10.240.255.55 - - [21/Mar/2019:17:44:36 +0000] "GET / HTTP/1.1" 304 -
"--" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML,
like Gecko) Chrome/72.0.3626.121 Safari/537.36"
::ffff:10.240.255.55 - - [21/Mar/2019:17:44:36 +0000] "GET / HTTP/1.1" 304 -
```

```
-- Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/72.0.3626.121 Safari/537.36"
```

Attach output streams

In addition to viewing the logs, you can attach your local standard out and standard error streams to that of the container.

First, execute the [az container attach](#) command to attach your local console to the container's output streams:

Azure CLI

```
az container attach --resource-group myResourceGroup --name mycontainer
```

Once attached, refresh your browser a few times to generate some more output. When you're done, detach your console with [Control+C](#). You should see output similar to the following sample:

Output

```
Container 'mycontainer' is in state 'Running'...
(count: 1) (last timestamp: 2019-03-21 17:27:20+00:00) pulling image
"mcr.microsoft.com/azuredocs/aci-helloworld"
(count: 1) (last timestamp: 2019-03-21 17:27:24+00:00) Successfully pulled
image "mcr.microsoft.com/azuredocs/aci-helloworld"
(count: 1) (last timestamp: 2019-03-21 17:27:27+00:00) Created container
(count: 1) (last timestamp: 2019-03-21 17:27:27+00:00) Started container
```

```
Start streaming logs:
listening on port 80
```

```
::ffff:10.240.255.55 - - [21/Mar/2019:17:43:53 +0000] "GET / HTTP/1.1" 304 -
"-- Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/72.0.3626.121 Safari/537.36"
::ffff:10.240.255.55 - - [21/Mar/2019:17:44:36 +0000] "GET / HTTP/1.1" 304 -
"-- Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/72.0.3626.121 Safari/537.36"
::ffff:10.240.255.55 - - [21/Mar/2019:17:44:36 +0000] "GET / HTTP/1.1" 304 -
"-- Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/72.0.3626.121 Safari/537.36"
::ffff:10.240.255.55 - - [21/Mar/2019:17:44:36 +0000] "GET / HTTP/1.1" 304 -
"-- Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/72.0.3626.121 Safari/537.36"
::ffff:10.240.255.56 - - [21/Mar/2019:17:47:01 +0000] "GET / HTTP/1.1" 304 -
"-- Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/72.0.3626.121 Safari/537.36"
::ffff:10.240.255.56 - - [21/Mar/2019:17:47:12 +0000] "GET / HTTP/1.1" 304 -
"-- Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/72.0.3626.121 Safari/537.36"
```

Clean up resources

When you're done with the container, remove it using the [az container delete](#) command:

```
Azure CLI
```

```
az container delete --resource-group myResourceGroup --name mycontainer
```

To verify that the container deleted, execute the [az container list](#) command:

```
Azure CLI
```

```
az container list --resource-group myResourceGroup --output table
```

The **mycontainer** container shouldn't appear in the command's output. If you have no other containers in the resource group, no output is displayed.

If you're done with the *myResourceGroup* resource group and all the resources it contains, delete it with the [az group delete](#) command:

```
Azure CLI
```

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

Next steps

In this quickstart, you created an Azure container instance by using a public Microsoft image. If you'd like to build a container image and deploy it from a private Azure container registry, continue to the Azure Container Instances tutorial.

[Azure Container Instances tutorial](#)

To try out options for running containers in an orchestration system on Azure, see the [Azure Kubernetes Service \(AKS\)](#) quickstarts.

Feedback

Was this page helpful?

 Yes

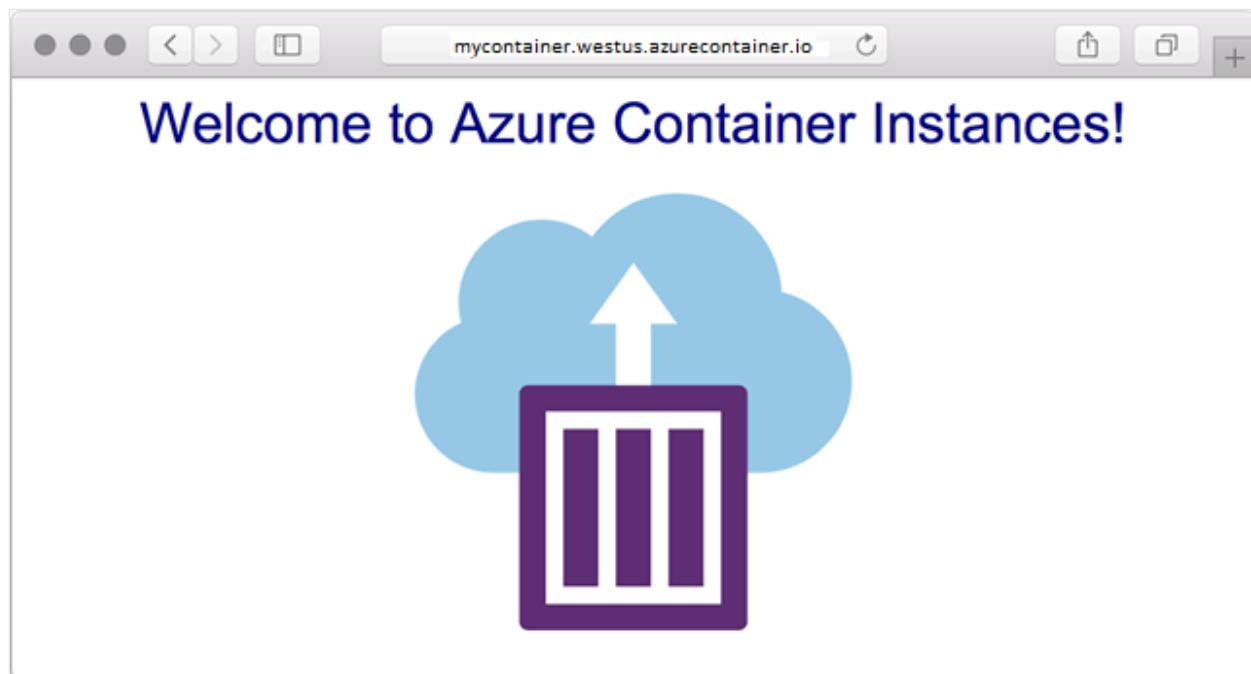
 No

Quickstart: Deploy a container instance in Azure using the Azure portal

Article • 08/29/2024

Use Azure Container Instances to run serverless Docker containers in Azure with simplicity and speed. Deploy an application to a container instance on-demand when you don't need a full container orchestration platform like Azure Kubernetes Service.

In this quickstart, you use the Azure portal to deploy an isolated Docker container and make its application available with a fully qualified domain name (FQDN). After configuring a few settings and deploying the container, you can browse to the running application:



Sign in to Azure

Sign in to the [Azure portal](#).

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

Create a container instance

On the Azure portal homepage, select **Create a resource**.

Microsoft Azure

Search resources, services, and docs (G+)

Azure services

Create a resource Subscriptions Virtual machines All resources Advisor Quickstart Center App Services Storage accounts SQL databases More services

Resources

Recent Favorite

Name	Type	Last Viewed
CLOUDBUILD-ANY	Subscription	a year ago
Modem Device-Modern Mgmt	Subscription	a year ago
IDEAs Reporting	Subscription	a year ago

See all

Navigate

Subscriptions Resource groups All resources Dashboard



Select Containers > Container Instances.

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

Home > Create a resource ...

Get Started Search services and marketplace

Recently created

Popular Azure services See more in All services

Categories

- AI + Machine Learning
- Analytics
- Blockchain
- Compute
- Containers** Create | Docs | MS Learn
- Databases
- Developer Tools
- DevOps
- Identity
- Integration
- Internet of Things
- IT & Management Tools
- Media
- Migration
- Mixed Reality
- Monitoring & Diagnostics

Kubernetes Service Create | Docs | MS Learn

Web App for Containers Create | Docs | MS Learn

Batch Service Create | Docs | MS Learn

Kubernetes - Azure Arc Create | Learn more

Container App Create | Learn more

Container Instances Create | Learn more

Container Registry Create | Docs | MS Learn

Service Fabric Cluster Create | Docs

On the **Basics** page, choose a subscription and enter the following values for **Resource group**, **Container name**, **Image source**, and **Container image**.

- Resource group: **Create new** > `myresourcegroup`
- Container name: `mycontainer`
- Image source: **Quickstart images**
- Container image: `mcr.microsoft.com/azuredocs/aci-helloworld:latest` (Linux)

Create container instance

Basics Networking Advanced Tags Review + create

Azure Container Instances (ACI) allows you to quickly and easily run containers on Azure without managing servers or having to learn new tools. ACI offers per-second billing to minimize the cost of running containers on the cloud.

[Learn more about Azure Container Instances](#)

Project details

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

Subscription * ⓘ Visual Studio Enterprise

Resource group * ⓘ (new) myresourcegroup

Create new

Container details

Container name * ⓘ mycontainer

Region * ⓘ (US) East US

Availability zones ⓘ None

Image source * ⓘ Quickstart images Azure Container Registry Other registry

Image * ⓘ mcr.microsoft.com/azuredocs/aci-helloworld:latest (Linux)

Size * ⓘ 1 vcpu, 1.5 GiB memory, 0 gpus
[Change size](#)

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

ⓘ Note

For this quickstart, you use default settings to deploy the public Microsoft aci-helloworld:latest image. This sample Linux image packages a small web app written in Node.js that serves a static HTML page. You can also bring your own container images stored in Azure Container Registry, Docker Hub, or other registries.

Leave the other values as their defaults, then select **Next: Networking**.

On the **Networking** page, specify a **DNS name label** for your container. The name must be unique within the Azure region where you create the container instance. Your container is publicly reachable at `<dns-name-label>.<region>.azurecontainer.io`. If you receive a "DNS name label not available" error message, try a different DNS name label.

An autogenerated hash is added as a DNS name label to your container instance's fully qualified domain name (FQDN), which prevents malicious subdomain takeover. Specify the **DNS name label scope reuse** for the FQDN. You can choose one of these options:

- Tenant
- Subscription
- Resource Group
- No reuse
- Any reuse (This option is the least secure.)

For this example, select **Tenant**.

Home > Create a resource >

Create container instance

Basics Networking Advanced Tags Review + create

Choose between three networking options for your container instance:

- **Public** will create a public IP address for your container instance.
- **Private** will allow you to choose a new or existing virtual network for your container instance. This is not yet available for Windows containers.
- **'None'** will not create either a public IP or virtual network. You will still be able to access your container logs using the command line.

Networking type

Public Private None

DNS name label ⓘ mycontainer

DNS name label scope reuse * ⓘ Tenant

Tenant

Subscription

Resource Group

No reuse

Any reuse (unsecure)

Review + create < Previous Next : Advanced >

Leave all other settings as their defaults, then select **Review + create**.

When the validation completes, you're shown a summary of the container's settings. Select **Create** to submit your container deployment request.

Create container instance

...

Validation passed

[Basics](#) [Networking](#) [Advanced](#) [Tags](#) [Review + create](#)

Basics

Subscription	Visual Studio Enterprise
Resource group	(new) myresourcegroup
Region	East US
Container name	mycontainer
Image type	Public
Image	mcr.microsoft.com/azuredocs/aci-helloworld:latest
OS type	Linux
Memory (GiB)	1.5
Number of CPU cores	1
GPU type (Preview)	None
GPU count	0

Networking

Networking type	Public
Ports	80 (TCP)
DNS name label	mycontainer
DNS name label scope reuse	Tenant

Advanced

Restart policy	On failure
----------------	------------

[Create](#)

< Previous

Next >

[Download a template for automation](#)

When deployment starts, a notification appears that indicates the deployment is in progress. Another notification is displayed when the container group deploys.

Open the overview for the container group by navigating to **Resource Groups** > **myresourcegroup** > **mycontainer**. Make a note of the **FQDN** of the container instance and its **Status**.

Home > Resource groups > myResourceGroup >

mycontainer

Container instances

Search (Ctrl+ /) Start Restart Stop Delete Refresh

View Cost | JSON View

Overview Activity log Access control (IAM) Tags

Containers Identity Properties Locks

Monitoring Metrics Alerts

Automation Tasks (preview) Export template

Support + troubleshooting New Support Request

Essentials

Status **Running**

Resource group (move) myResourceGroup

Location West US 2

Subscription (move) Visual Studio Enterprise

Subscription ID <Subscription ID>

Tags (edit) Click here to add tags

OS type Linux

IP address (Public) 20.252.34.567

FQDN **mycontainer.westus2.azurecontainer.io**

Container count 1

CPU

2.5
2
1.5
1
0.5
0

12:15 PM 12:30 PM 12:45 PM 1 PM

CPU Usage (Avg) mycontainer 0.83

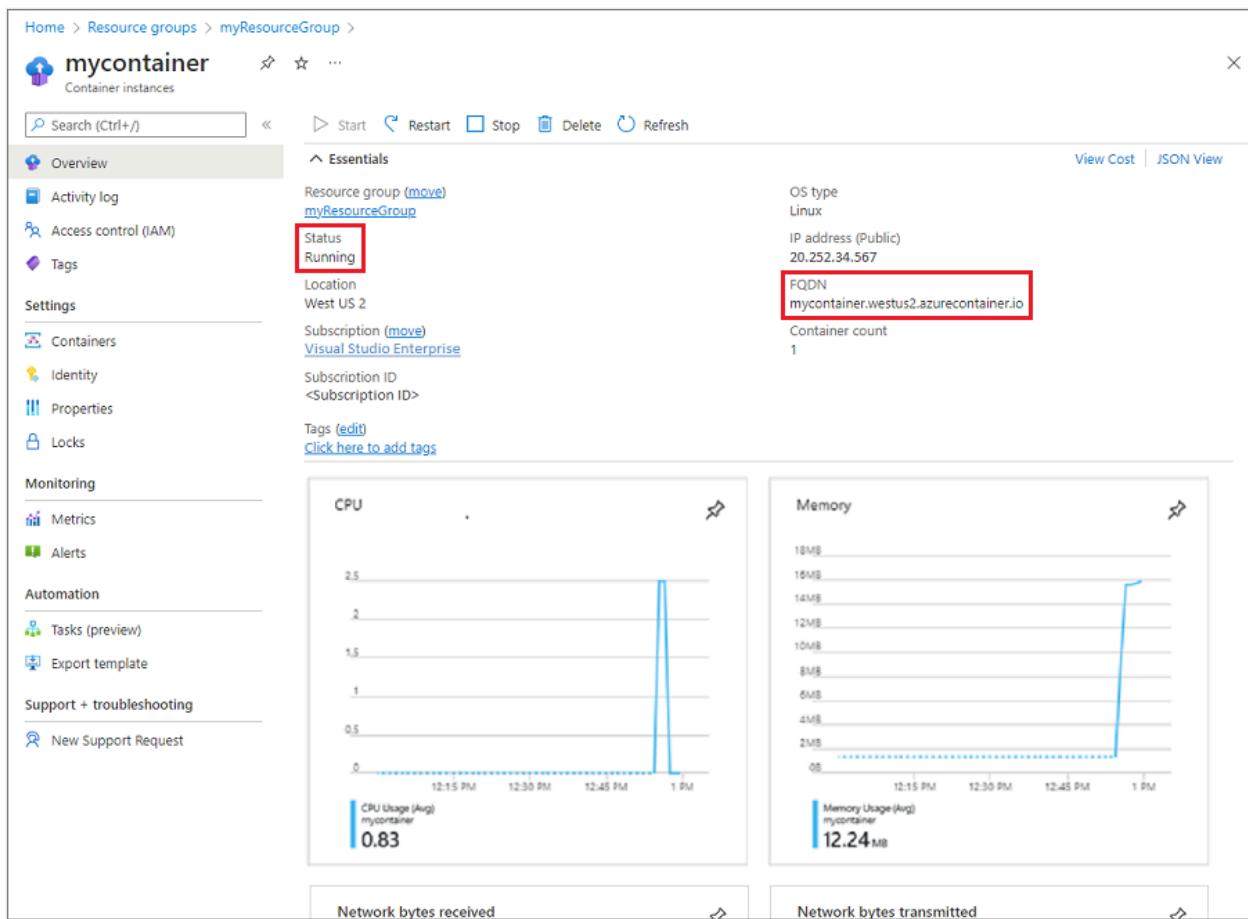
Memory

18MB
16MB
14MB
12MB
10MB
8MB
6MB
4MB
2MB
0B

12:15 PM 12:30 PM 12:45 PM 1 PM

Memory Usage (Avg) mycontainer 12.24 MB

Network bytes received Network bytes transmitted



Once its **Status** is *Running*, navigate to the container's FQDN in your browser.



Congratulations! By configuring just a few settings, you deployed a publicly accessible application in Azure Container Instances.

View container logs

Viewing the logs for a container instance is helpful when troubleshooting issues with your container or the application it runs.

To view the container's logs, under **Settings**, select **Containers > Logs**. You should see the HTTP GET request generated when you viewed the application in your browser.

The screenshot shows the Azure portal interface for a container named 'mycontainer'. On the left, there's a sidebar with options like Overview, Activity log, Access control (IAM), Tags, Settings, Containers (which is highlighted with a red box), Identity, Properties, Locks, Monitoring, Metrics, and Alerts. The main area displays the container details: Name (mycontainer), Image (mcr.microsoft.com/azu...), State (Running), Previous state (-), Start time (2022-07-20T20:20:39...), and Restart count (0). Below this, there are tabs for Events, Properties, Logs (which is highlighted with a red box), and Connect. The Logs tab shows a list of log entries. One entry is expanded to show detailed information:

```
listening on port 80
::ffff:10.92.0.24 - - [20/Jul/2022:20:22:59 +0000] "GET / HTTP/1.1" 200 1663 "-" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/103.0.5066.114 Safari/537.36 Edg/103.0.1264.62"
::ffff:10.92.0.24 - - [20/Jul/2022:20:23:00 +0000] "GET /favicon.ico HTTP/1.1" 404 150 "http://mycontainer.bnhsFbdcrhcu8.westus2.azurecontainer.io/" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/103.0.5066.114 Safari/537.36 Edg/103.0.1264.62"
::ffff:10.92.0.24 - - [20/Jul/2022:20:23:32 +0000] "GET /favicon.ico HTTP/1.1" 404 150 "-" "Mozilla/5.0 (Windows NT 10.0; Win64; x64; Trident/7.0; rv:11.0) like Gecko"
```

Clean up resources

When you're done with the container, select **Overview** for the *mycontainer* container instance, then select **Delete**.

The screenshot shows the Azure portal interface for the 'mycontainer' container instance. On the left, there's a sidebar with options like Overview (which is highlighted with a red box), Activity log, Access control (IAM), Tags, Settings, Containers, Identity, Properties, and Locks. The main area displays the container details: Resource group (myResourceGroup), Status (Running), Location (West US 2), Subscription (Visual Studio Enterprise), and Tags (Click here to add tags). At the top, there are buttons for Start, Restart, Stop, Delete (which is highlighted with a red box), and Refresh. To the right, there are links for View Cost and JSON View.

Select **Yes** when the confirmation dialog appears.

The screenshot shows a confirmation dialog box. It contains the text 'Delete container instances' and 'Do you want to delete all container instances in container group 'mycontainer'?'. At the bottom, there are two buttons: 'Yes' (highlighted with a red box) and 'No'.

Next steps

In this quickstart, you created an Azure container instance from a public Microsoft image. If you'd like to build a container image and deploy it from a private Azure container registry, continue to the Azure Container Instances tutorial.

[Azure Container Instances tutorial](#)

Feedback

Was this page helpful?

 Yes

 No

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

Quickstart: Deploy a container instance in Azure using Azure PowerShell

Article • 08/29/2024

Use Azure Container Instances to run serverless Docker containers in Azure with simplicity and speed. Deploy an application to a container instance on-demand when you don't need a full container orchestration platform like Azure Kubernetes Service.

In this quickstart, you use Azure PowerShell to deploy an isolated Windows container and make its application available with a fully qualified domain name (FQDN) and port. A few seconds after you execute a single deployment command, you can browse to the application running in the container:

![App deployed to Azure Container Instances viewed in browser](./media/container-instances-quickstart/view-an-application-running-in-an-azure-container-instance.png)

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

ⓘ Note

We recommend that you use the Azure Az PowerShell module to interact with Azure. To get started, see [Install Azure PowerShell](#). To learn how to migrate to the Az PowerShell module, see [Migrate Azure PowerShell from AzureRM to Az](#).

Azure Cloud Shell

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

To start Azure Cloud Shell:

ⓘ Expand table

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

Option	Example/Link
Go to https://shell.azure.com , or select the Launch Cloud Shell button to open Cloud Shell in your browser.	
Select the Cloud Shell button on the menu bar at the upper right in the Azure portal .	

To use Azure Cloud Shell:

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

If you choose to install and use the PowerShell locally, this tutorial requires the Azure PowerShell module. Run `Get-Module -ListAvailable Az` to find the version. If you need to upgrade, see [Install Azure PowerShell module](#). If you run PowerShell locally, you also need to run `Connect-AzAccount` to create a connection with Azure.

Create a resource group

Azure container instances, like all Azure resources, must be deployed into a resource group. Resource groups allow you to organize and manage related Azure resources.

First, create a resource group named *myResourceGroup* in the *eastus* location with the following [New-AzResourceGroup](#) command:

```
Azure PowerShell
New-AzResourceGroup -Name myResourceGroup -Location EastUS
```

Create a port for your container instance

You can expose your containers to the internet by specifying one or more ports to open, a DNS name label, or both. In this quickstart, you deploy a container with a DNS name label so it's publicly reachable. In this guide, we do both, but first, you need to create a port object in PowerShell for your container instance to use.

Azure PowerShell

```
$port = New-AzContainerInstancePortObject -Port 80 -Protocol TCP
```

Create a container group

Now that you have a resource group and port, you can run a container exposed to the internet in Azure. To create a container instance with Azure PowerShell, you first need to create a `ContainerInstanceObject` by providing a name, image, and port for the container. In this quickstart, you use the public `mcr.microsoft.com/azuredocs/aci-helloworld` image.

Azure PowerShell

```
New-AzContainerInstanceObject -Name myContainer -Image  
mcr.microsoft.com/azuredocs/aci-helloworld -Port $($port)
```

Next, use the `New-AzContainerGroup` cmdlet. You need to provide a name for the container group, your resource group's name, a location for the container group, the container instance you created, the operating system type, and a unique IP address DNS name label.

Execute a command similar to the following to start a container instance. Set a `-IpAddressDnsNameLabel` value that's unique within the Azure region where you create the instance. If you receive a "DNS name label not available" error message, try a different DNS name label.

Azure PowerShell

```
$containerGroup = New-AzContainerInstanceObject -ResourceGroupName  
myResourceGroup -Name myContainerGroup -Location EastUS -Container  
myContainer -OsType Windows -IpAddressDnsNameLabel aci-quickstart-win -  
IpAddressType Public -IpAddressPort $($port)
```

Within a few seconds, you should receive a response from Azure. The container's `ProvisioningState` is initially `Creating`, but should move to `Succeeded` within a minute or two. Check the deployment state with the `Get-AzContainerGroup` cmdlet:

Azure PowerShell

```
Get-AzContainerGroup -ResourceGroupName myResourceGroup -Name  
myContainerGroup
```

You can also print the \$containerGroup object and filter the table for the container's provisioning state, fully qualified domain name (FQDN), and IP address.

Azure PowerShell

```
$containerGroup | Format-Table InstanceViewState, IPAddressFqdn, IPAddressIP
```

The container's provisioning state, FQDN, and IP address appear in the cmdlet's output:

Console

```
PS Azure:\> Get-AzContainerGroup -ResourceGroupName myResourceGroup -Name myContainerGroup
```

```
ResourceGroupName      : myResourceGroup
Id                   : /subscriptions/<Subscription
ID>/resourceGroups/myResourceGroup/providers/Microsoft.ContainerInstance/containerGroups/myContainerGroup
Name                 : myContainerGroup
Type                : Microsoft.ContainerInstance/containerGroups
Location             : eastus
Tags                :
ProvisioningState    : Creating
Containers           : {myContainer}
ImageRegistryCredentials :
RestartPolicy        : Always
IpAddress            : 52.226.19.87
DnsNameLabel         : aci-demo-win
Fqdn                : aci-demo-win.eastus.azurecontainer.io
Ports               : {80}
OsType              : Windows
Volumes             :
State               : Pending
Events              : {}
```

If the container's `ProvisioningState` is **Succeeded**, go to its FQDN in your browser. If you see a web page similar to the following, congratulations! You successfully deployed an application running in a Docker container to Azure.

![View an app deployed to Azure Container Instances in browser][./media/container-instances-quickstart/view-an-application-running-in-an-azure-container-instance.png]

Clean up resources

When you're done with the container, remove it with the [Remove-AzContainerGroup](#) cmdlet:

Azure PowerShell

```
Remove-AzContainerGroup -ResourceGroupName myResourceGroup -Name  
myContainerGroup
```

Next steps

In this quickstart, you created an Azure container instance from an image in the public Docker Hub registry. If you'd like to build a container image and deploy it from a private Azure container registry, continue to the Azure Container Instances tutorial.

[Azure Container Instances tutorial](#)

Feedback

Was this page helpful?

 Yes

 No

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

Quickstart: Deploy a container instance in Azure using Bicep

Article • 08/29/2024

Use Azure Container Instances to run serverless Docker containers in Azure with simplicity and speed. Deploy an application to a container instance on-demand when you don't need a full container orchestration platform like Azure Kubernetes Service. In this quickstart, you use a Bicep file to deploy an isolated Docker container and make its web application available with a public IP address.

Bicep is a domain-specific language (DSL) that uses declarative syntax to deploy Azure resources. It provides concise syntax, reliable type safety, and support for code reuse. Bicep offers the best authoring experience for your infrastructure-as-code solutions in Azure.

Prerequisites

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

Review the Bicep file

The Bicep file used in this quickstart is from [Azure Quickstart Templates](#).

```
Bicep

@description('Name for the container group')
param name string = 'acilinxpublicipcontainergroup'

@description('Location for all resources.')
param location string = resourceGroup().location

@description('Container image to deploy. Should be of the form
repoName/imagename:tag for images stored in public Docker Hub, or a fully
qualified URI for other registries. Images from private registries require
additional registry credentials.')
param image string = 'mcr.microsoft.com/azuredocs/aci-helloworld'

@description('Port to open on the container and the public IP address.')
param port int = 80

@description('The number of CPU cores to allocate to the container.')
param cpuCores int = 1

@description('The amount of memory to allocate to the container in')
```

```

gigabytes.')
param memoryInGb int = 2

@description('The behavior of Azure runtime if container has stopped.')
@allowed([
    'Always'
    'Never'
    'OnFailure'
])
param restartPolicy string = 'Always'

resource containerGroup 'Microsoft.ContainerInstance/containerGroups@2023-05-01' = {
    name: name
    location: location
    properties: {
        containers: [
            {
                name: name
                properties: {
                    image: image
                    ports: [
                        {
                            port: port
                            protocol: 'TCP'
                        }
                    ]
                    resources: {
                        requests: {
                            cpu: cpuCores
                            memoryInGB: memoryInGb
                        }
                    }
                }
            }
        ]
        osType: 'Linux'
        restartPolicy: restartPolicy
        ipAddress: {
            type: 'Public'
            ports: [
                {
                    port: port
                    protocol: 'TCP'
                }
            ]
        }
    }
}

output name string = containerGroup.name
output resourceGroupName string = resourceGroup().name
output resourceId string = containerGroup.id
output containerIPv4Address string = containerGroup.properties.ipAddress.ip
output location string = location

```

The following resource is defined in the Bicep file:

- **Microsoft.ContainerInstance/containerGroups**: create an Azure container group.
This Bicep file defines a group consisting of a single container instance.

More Azure Container Instances template samples can be found in the [quickstart template gallery](#).

Deploy the Bicep file

1. Save the Bicep file as `main.bicep` to your local computer.
2. Deploy the Bicep file using either Azure CLI or Azure PowerShell.

CLI

```
Azure CLI
az group create --name exampleRG --location eastus
az deployment group create --resource-group exampleRG --template-file main.bicep
```

When the deployment finishes, you should see a message indicating the deployment succeeded.

Review deployed resources

Use the Azure portal, Azure CLI, or Azure PowerShell to list the deployed resources in the resource group.

CLI

```
Azure CLI
az resource list --resource-group exampleRG
```

View container logs

Viewing the logs for a container instance is helpful when troubleshooting issues with your container or the application it runs. Use the Azure portal, Azure CLI, or Azure PowerShell to view the container's logs.

CLI

Azure CLI

```
az container logs --resource-group exampleRG --name  
acilinuxpublicipcontainergroup
```

ⓘ Note

It may take a few minutes for the HTTP GET request to generate.

Clean up resources

When no longer needed, use the Azure portal, Azure CLI, or Azure PowerShell to delete the container and all of the resources in the resource group.

CLI

Azure CLI

```
az group delete --name exampleRG
```

Next steps

In this quickstart, you created an Azure container instance using Bicep. If you'd like to build a container image and deploy it from a private Azure container registry, continue to the Azure Container Instances tutorial.

[Tutorial: Create a container image for deployment to Azure Container Instances](#)

Feedback

Was this page helpful?

 Yes

 No

Provide product feedback  | Get help at Microsoft Q&A

Quickstart: Deploy a container instance in Azure using an ARM template

Article • 08/29/2024

Use Azure Container Instances to run serverless Docker containers in Azure with simplicity and speed. Deploy an application to a container instance on-demand when you don't need a full container orchestration platform like Azure Kubernetes Service. In this quickstart, you use an Azure Resource Manager template (ARM template) to deploy an isolated Docker container and make its web application available with a public IP address.

An [Azure Resource Manager template](#) is a JavaScript Object Notation (JSON) file that defines the infrastructure and configuration for your project. The template uses declarative syntax. You describe your intended deployment without writing the sequence of programming commands to create the deployment.

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



Deploy to Azure



Prerequisites

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

Review the template

The template used in this quickstart is from [Azure Quickstart Templates](#).

JSON

```
{  
  "$schema": "https://schema.management.azure.com/schemas/2019-04-01/deploymentTemplate.json#",  
  "contentVersion": "1.0.0.0",  
  "metadata": {  
    "_generator": {  
      "name": "bicep",  
      "version": "0.26.54.24096",  
      "templateHash": "15999594271314721416"  
    }  
  },
```

```
"parameters": {
    "name": {
        "type": "string",
        "defaultValue": "acilinuxpublicipcontainergroup",
        "metadata": {
            "description": "Name for the container group"
        }
    },
    "location": {
        "type": "string",
        "defaultValue": "[resourceGroup().location]",
        "metadata": {
            "description": "Location for all resources."
        }
    },
    "image": {
        "type": "string",
        "defaultValue": "mcr.microsoft.com/azuredocs/aci-helloworld",
        "metadata": {
            "description": "Container image to deploy. Should be of the form repoName/imagename:tag for images stored in public Docker Hub, or a fully qualified URI for other registries. Images from private registries require additional registry credentials."
        }
    },
    "port": {
        "type": "int",
        "defaultValue": 80,
        "metadata": {
            "description": "Port to open on the container and the public IP address."
        }
    },
    "cpuCores": {
        "type": "int",
        "defaultValue": 1,
        "metadata": {
            "description": "The number of CPU cores to allocate to the container."
        }
    },
    "memoryInGb": {
        "type": "int",
        "defaultValue": 2,
        "metadata": {
            "description": "The amount of memory to allocate to the container in gigabytes."
        }
    },
    "restartPolicy": {
        "type": "string",
        "defaultValue": "Always",
        "allowedValues": [
            "Always",
            "Never",
            "OnFailure"
        ]
    }
}
```

```
        "OnFailure"
    ],
    "metadata": {
        "description": "The behavior of Azure runtime if container has
stopped."
    }
},
"resources": [
{
    "type": "Microsoft.ContainerInstance/containerGroups",
    "apiVersion": "2023-05-01",
    "name": "[parameters('name')]",
    "location": "[parameters('location')]",
    "properties": {
        "containers": [
{
            "name": "[parameters('name')]",
            "properties": {
                "image": "[parameters('image')]",
                "ports": [
{
                    "port": "[parameters('port')]",
                    "protocol": "TCP"
}
],
                "resources": {
                    "requests": {
                        "cpu": "[parameters('cpuCores')]",
                        "memoryInGB": "[parameters('memoryInGb')]"
}
}
}
]
}
],
        "osType": "Linux",
        "restartPolicy": "[parameters('restartPolicy')]",
        "ipAddress": {
            "type": "Public",
            "ports": [
{
            "port": "[parameters('port')]",
            "protocol": "TCP"
}
]
}
}
],
    "outputs": {
        "name": {
            "type": "string",
            "value": "[parameters('name')]"
}
},
    "resourceGroupName": {
```

```

    "type": "string",
    "value": "[resourceGroup().name]"
},
"resourceId": {
    "type": "string",
    "value": "[resourceId('Microsoft.ContainerInstance/containerGroups',
parameters('name'))]"
},
"containerIPv4Address": {
    "type": "string",
    "value": "
[reference(resourceId('Microsoft.ContainerInstance/containerGroups',
parameters('name')), '2023-05-01').ipAddress.ip]"
},
"location": {
    "type": "string",
    "value": "[parameters('location')]"
}
}
}

```

The following resource is defined in the template:

- **Microsoft.ContainerInstance/containerGroups**: create an Azure container group.
This template defines a group consisting of a single container instance.

More Azure Container Instances template samples can be found in the [quickstart template gallery](#).

Deploy the template

1. Select the following image to sign in to Azure and open a template. The template creates a registry and a replica in another location.



2. Select or enter the following values.

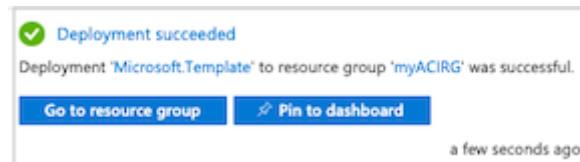
- **Subscription**: select an Azure subscription.
- **Resource group**: select **Create new**, enter a unique name for the resource group, and then select **OK**.
- **Location**: select a location for the resource group. Example: **Central US**.
- **Name**: accept the generated name for the instance, or enter a name.
- **Image**: accept the default image name. This sample Linux image packages a small web app written in Node.js that serves a static HTML page.

Accept default values for the remaining properties.

Review the terms and conditions. If you agree, select I agree to the terms and conditions stated above.

The screenshot shows the Azure Container Instances - Linux container with public IP template creation page. It includes sections for TEMPLATE, BASICS, SETTINGS, and TERMS AND CONDITIONS. In the SETTINGS section, the Name is set to 'acilinuxpublicipcontainergroup', Image to 'mcr.microsoft.com/azuredocs/aci-helloworld', Port to '80', Cpu Cores to '1.0', Memory In Gb to '1.5', and Location to '[resourceGroup().location]'. Under TERMS AND CONDITIONS, there is a checkbox for 'I agree to the terms and conditions stated above' which is unchecked. A 'Purchase' button is at the bottom.

3. After the instance successfully creates, you get a notification:



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

Review deployed resources

Use the Azure portal or a tool such as the [Azure CLI](#) to review the properties of the container instance.

1. In the portal, search for Container Instances, and select the container instance you created.

2. On the Overview page, note the Status of the instance and its IP address.

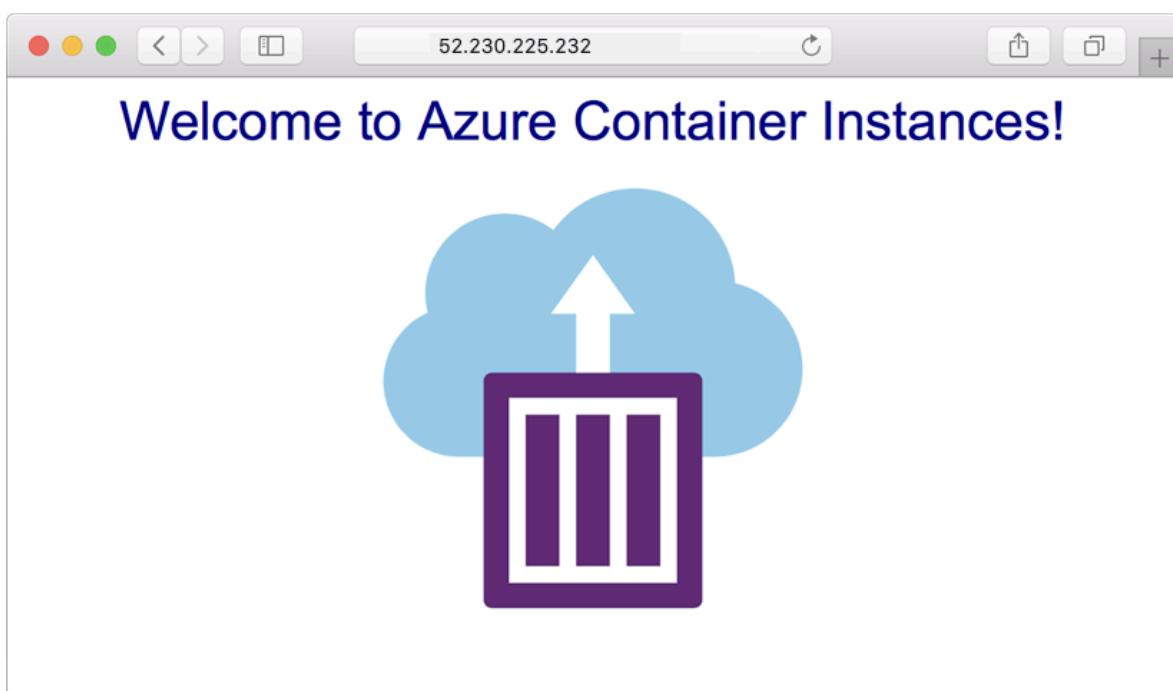
The screenshot shows the Azure Container Instances Overview page for a group named 'acilinuxpublicipcontainergroup'. The left sidebar includes options like Overview (highlighted with a red box), Activity log, Access control (IAM), Tags, Settings, Containers, Identity, Properties, Locks, Export template, Monitoring (Metrics and Alerts), Support + troubleshooting, and New support request. The main content area displays the following details:

- Resource group (change):** myACIRG
- Status:** Running (highlighted with a red box)
- IP address:** 52.230.225.232 (Public)
- OS type:** Linux
- Location:** Central US
- FQDN:** ...
- Subscription:** Visual Studio Enterprise Subscription
- Subscription ID:** e060064b-a789-4ad5-9474-e52523d
- Container count:** 1
- Tags (change):** Click here to add tags

Below these details are two performance charts:

- CPU:** A line chart showing CPU usage over time. The Y-axis ranges from 0 to 100. The X-axis shows times from 1:15 PM to 1:45 PM UTC-07:00. The chart shows a sharp spike starting at 1:30 PM, reaching approximately 85% usage by 1:45 PM.
- Memory:** A line chart showing memory usage over time. The Y-axis ranges from 0B to 22MB. The X-axis shows times from 1:15 PM to 1:45 PM UTC-07:00. The chart shows a sharp spike starting at 1:30 PM, reaching approximately 15.77 MB by 1:45 PM.

3. Once its status is *Running*, navigate to the IP address in your browser.



View container logs

Viewing the logs for a container instance is helpful when troubleshooting issues with your container or the application it runs.

To view the container's logs, under **Settings**, select **Containers > Logs**. You should see the HTTP GET request generated when you viewed the application in your browser.

The screenshot shows the Azure portal interface for managing container instances. On the left, there's a sidebar with various navigation options like Overview, Activity log, Access control (IAM), Tags, Settings, and Containers. The 'Containers' option is highlighted with a red box. The main area displays a table of container instances. One instance is listed: 'acilinxpublicipcontain...', which is running and was started on 2020-04-27T20:24:02Z. Below the table, there are tabs for Events, Properties, Logs, and Connect. The 'Logs' tab is currently selected and shows a log stream with several lines of text representing HTTP requests. The first line is 'listening on port 80'. Subsequent lines show requests from different clients, including 'Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_4) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/81.0.4044.122 Safari/537.36' and 'Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/52.0.2743.116 Safari/537.36'.

Clean up resources

When you're done with the container, on the **Overview** page for the container instance, select **Delete**. When prompted, confirm the deletion.

Next steps

In this quickstart, you created an Azure container instance from a public Microsoft image. If you'd like to build a container image and deploy it from a private Azure container registry, continue to the Azure Container Instances tutorial.

[Tutorial: Create a container image for deployment to Azure Container Instances](#)

For a step-by-step tutorial that guides you through the process of creating a template, see:

[Tutorial: Create and deploy your first ARM template](#)

Feedback

Was this page helpful?

Yes

No

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

Quickstart: Create an Azure Container Instance with a public IP address using Terraform

Article • 08/29/2024

Use Azure Container Instances to run serverless Docker containers in Azure with simplicity and speed. Deploy an application to a container instance on-demand when you don't need a full container orchestration platform like Azure Kubernetes Service. In this article, you use [Terraform](#) to deploy an isolated Docker container and make its web application available with a public IP address.

[Terraform](#)  enables the definition, preview, and deployment of cloud infrastructure. Using Terraform, you create configuration files using [HCL syntax](#)  . The HCL syntax allows you to specify the cloud provider - such as Azure - and the elements that make up your cloud infrastructure. After you create your configuration files, you create an *execution plan* that allows you to preview your infrastructure changes before they're deployed. Once you verify the changes, you apply the execution plan to deploy the infrastructure.

In this article, you learn how to:

- ✓ Create a random value for the Azure resource group name using [random_pet](#) 
- ✓ Create an Azure resource group using [azurerm_resource_group](#) 
- ✓ Create a random value for the container name using [random_string](#) 
- ✓ Create an Azure container group using [azurerm_container_group](#) 

Prerequisites

- [Install and configure Terraform](#)

Implement the Terraform code

Note

The sample code for this article is located in the [Azure Terraform GitHub repo](#)  . You can view the log file containing the [test results from current and previous versions of Terraform](#)  .

See more [articles](#) and [sample code showing how to use Terraform to manage Azure resources](#)

1. Create a directory in which to test and run the sample Terraform code and make it the current directory.
2. Create a file named `main.tf` and insert the following code:

Terraform

```
resource "random_pet" "rg_name" {
    prefix = var.resource_group_name_prefix
}

resource "azurerm_resource_group" "rg" {
    name      = random_pet.rg_name.id
    location  = var.resource_group_location
}

resource "random_string" "container_name" {
    length   = 25
    lower    = true
    upper    = false
    special  = false
}

resource "azurerm_container_group" "container" {
    name          =
"${var.container_group_name_prefix}-${random_string.container_name.result}"
    location      = azurerm_resource_group.rg.location
    resource_group_name = azurerm_resource_group.rg.name
    ip_address_type = "Public"
    os_type        = "Linux"
    restart_policy = var.restart_policy

    container {
        name      =
"${var.container_name_prefix}-${random_string.container_name.result}"
        image    = var.image
        cpu      = var.cpu_cores
        memory  = var.memory_in_gb

        ports {
            port    = var.port
            protocol = "TCP"
        }
    }
}
```

3. Create a file named `outputs.tf` and insert the following code:

Terraform

```
output "container_ipv4_address" {
    value = azurerm_container_group.container.ip_address
}
```

4. Create a file named `providers.tf` and insert the following code:

Terraform

```
terraform {
  required_version = ">=1.0"
  required_providers {
    azurerm = {
      source  = "hashicorp/azurerm"
      version = "~>3.0"
    }
    random = {
      source  = "hashicorp/random"
      version = "~>3.0"
    }
  }
  provider "azurerm" {
    features {}
  }
}
```

5. Create a file named `variables.tf` and insert the following code:

Terraform

```
variable "resource_group_location" {
  type     = string
  default  = "eastus"
  description = "Location for all resources."
}

variable "resource_group_name_prefix" {
  type     = string
  default  = "rg"
  description = "Prefix of the resource group name that's combined with
a random value so name is unique in your Azure subscription."
}

variable "container_group_name_prefix" {
  type     = string
  description = "Prefix of the container group name that's combined
with a random value so name is unique in your Azure subscription."
  default  = "acigroup"
}
```

```

variable "container_name_prefix" {
  type      = string
  description = "Prefix of the container name that's combined with a
random value so name is unique in your Azure subscription."
  default    = "aci"
}

variable "image" {
  type      = string
  description = "Container image to deploy. Should be of the form
repoName/imagename:tag for images stored in public Docker Hub, or a
fully qualified URI for other registries. Images from private
registries require additional registry credentials."
  default    = "mcr.microsoft.com/azuredocs/aci-helloworld"
}

variable "port" {
  type      = number
  description = "Port to open on the container and the public IP
address."
  default    = 80
}

variable "cpu_cores" {
  type      = number
  description = "The number of CPU cores to allocate to the container."
  default    = 1
}

variable "memory_in_gb" {
  type      = number
  description = "The amount of memory to allocate to the container in
gigabytes."
  default    = 2
}

variable "restart_policy" {
  type      = string
  description = "The behavior of Azure runtime if container has
stopped."
  default    = "Always"
  validation {
    condition    = contains(["Always", "Never", "OnFailure"],
var.restart_policy)
    error_message = "The restart_policy must be one of the following:
Always, Never, OnFailure."
  }
}

```

Initialize Terraform

Run [terraform init](#) to initialize the Terraform deployment. This command downloads the Azure provider required to manage your Azure resources.

Console

```
terraform init -upgrade
```

Key points:

- The `-upgrade` parameter upgrades the necessary provider plugins to the newest version that complies with the configuration's version constraints.

Create a Terraform execution plan

Run [terraform plan](#) to create an execution plan.

Console

```
terraform plan -out main.tfplan
```

Key points:

- The `terraform plan` command creates an execution plan, but doesn't execute it. Instead, it determines what actions are necessary to create the configuration specified in your configuration files. This pattern allows you to verify whether the execution plan matches your expectations before making any changes to actual resources.
- The optional `-out` parameter allows you to specify an output file for the plan. Using the `-out` parameter ensures that the plan you reviewed is exactly what is applied.

Apply a Terraform execution plan

Run [terraform apply](#) to apply the execution plan to your cloud infrastructure.

Console

```
terraform apply main.tfplan
```

Key points:

- The example `terraform apply` command assumes you previously ran `terraform plan -out main.tfplan`.
- If you specified a different filename for the `-out` parameter, use that same filename in the call to `terraform apply`.
- If you didn't use the `-out` parameter, call `terraform apply` without any parameters.

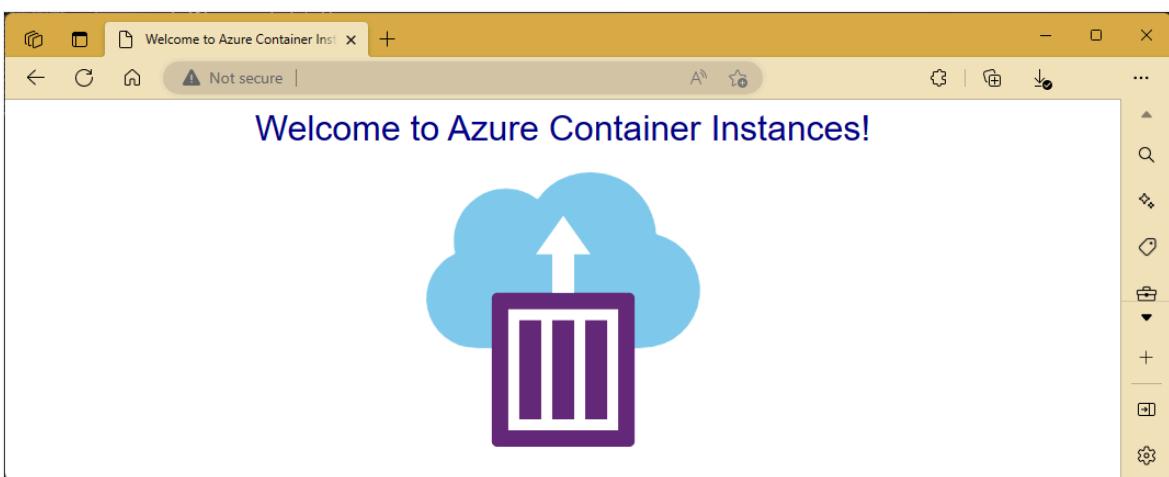
Verify the results

1. When you apply the execution plan, Terraform outputs the public IP address. To display the IP address again, run `terraform output`.

```
Console

terraform output -raw container_ipv4_address
```

2. Enter the sample's public IP address in your browser's address bar.



Clean up resources

When you no longer need the resources created via Terraform, do the following steps:

1. Run `terraform plan` and specify the `destroy` flag.

```
Console

terraform plan -destroy -out main.destroy.tfplan
```

Key points:

- The `terraform plan` command creates an execution plan, but doesn't execute it. Instead, it determines what actions are necessary to create the

configuration specified in your configuration files. This pattern allows you to verify whether the execution plan matches your expectations before making any changes to actual resources.

- The optional `-out` parameter allows you to specify an output file for the plan. Using the `-out` parameter ensures that the plan you reviewed is exactly what is applied.

2. Run [terraform apply ↗](#) to apply the execution plan.

Console

```
terraform apply main.destroy.tfplan
```

Troubleshoot Terraform on Azure

[Troubleshoot common problems when using Terraform on Azure](#)

Next steps

[Tutorial: Create a container image for deployment to Azure Container Instances](#)

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

Feedback

Was this page helpful?

 Yes

 No

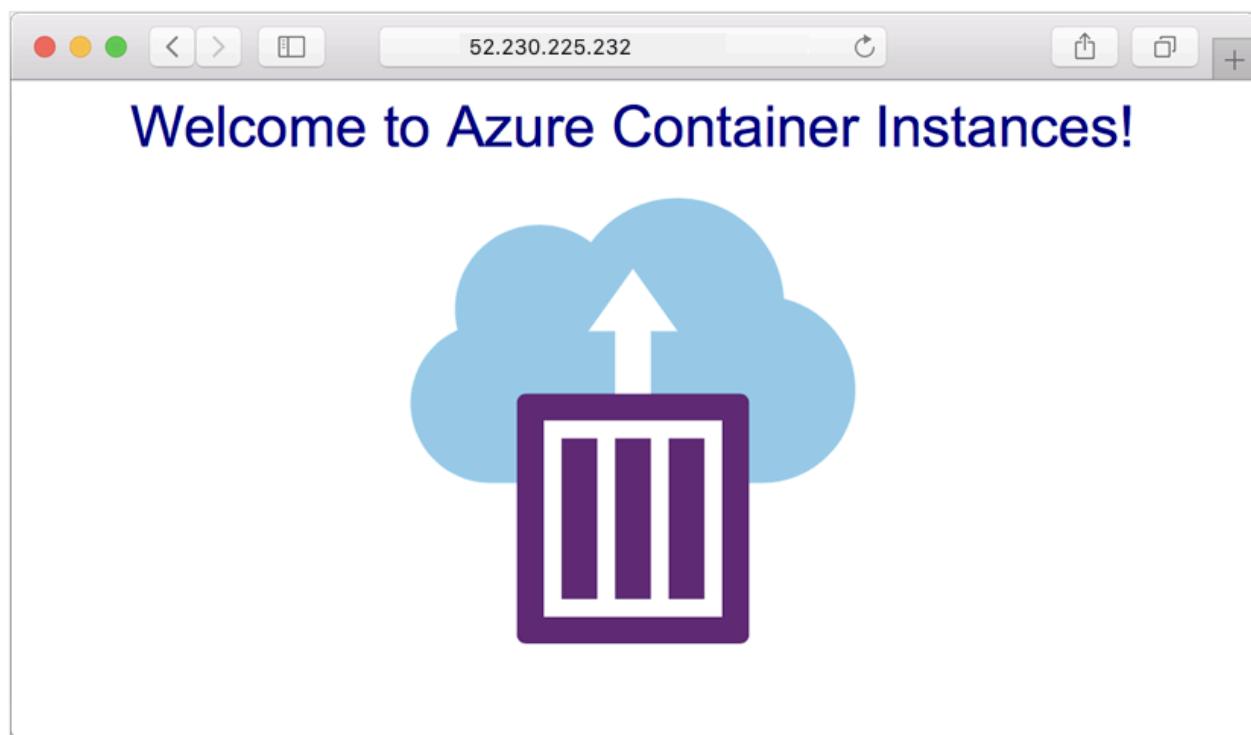
[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

Quickstart: Deploy a container instance in Azure using the Docker CLI

Article • 08/29/2024

Use Azure Container Instances to run serverless Docker containers in Azure with simplicity and speed. Deploy to a container instance on-demand when you develop cloud-native apps and you want to switch seamlessly from local development to cloud deployment.

In this quickstart, you use native Docker CLI commands to deploy a Docker container and make its application available in Azure Container Instances. The [integration between Docker and Azure](#) enables this capability. A few seconds after you execute a `docker run` command, you can browse to the application running in the container:



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

For this quickstart, you need Docker Desktop version 2.3.0.5 or later, available for [Windows](#) or [macOS](#). Or install the [Docker ACI Integration CLI for Linux](#).

ⓘ Important

Not all features of Azure Container Instances are supported. Provide feedback about the Docker-Azure integration by creating an issue in the [aci-integration-beta](#) GitHub repository.

Create Azure context

To use Docker commands to run containers in Azure Container Instances, first log into Azure:

```
Bash
```

```
docker login azure --tenant-id "[tenant ID]"
```

To find your tenant ID, browse to the Microsoft Entra ID properties.

When prompted, enter or select your Azure credentials.

Create an ACI context by running `docker context create aci`. This context associates Docker with an Azure subscription and resource group so you can create and manage container instances. For example, to create a context called *myacicontext*:

```
docker context create aci myacicontext
```

When prompted, select your Azure subscription ID, then select an existing resource group or **create a new resource group**. If you choose a new resource group, it has a system-generated name on creation. Azure container instances, like all Azure resources, must be deployed into a resource group. Resource groups allow you to organize and manage related Azure resources.

Run `docker context ls` to confirm that you added the ACI context to your Docker contexts:

```
docker context ls
```

Create a container

After creating a Docker context, you can create a container in Azure. In this quickstart, you use the public `mcr.microsoft.com/azuredocs/aci-helloworld` image. This image packages a small web app written in Node.js that serves a static HTML page.

First, change to the ACI context. All subsequent Docker commands run in this context.

```
docker context use myacicontext
```

Run the following `docker run` command to create the Azure container instance with port 80 exposed to the internet:

```
docker run -p 80:80 mcr.microsoft.com/azuredocs/aci-helloworld
```

Sample output for a successful deployment:

```
[+] Running 2/2
  :: hungry-kirch          Created
  5.1s
  :: single--container--aci Done
  11.3s
hungry-kirch
```

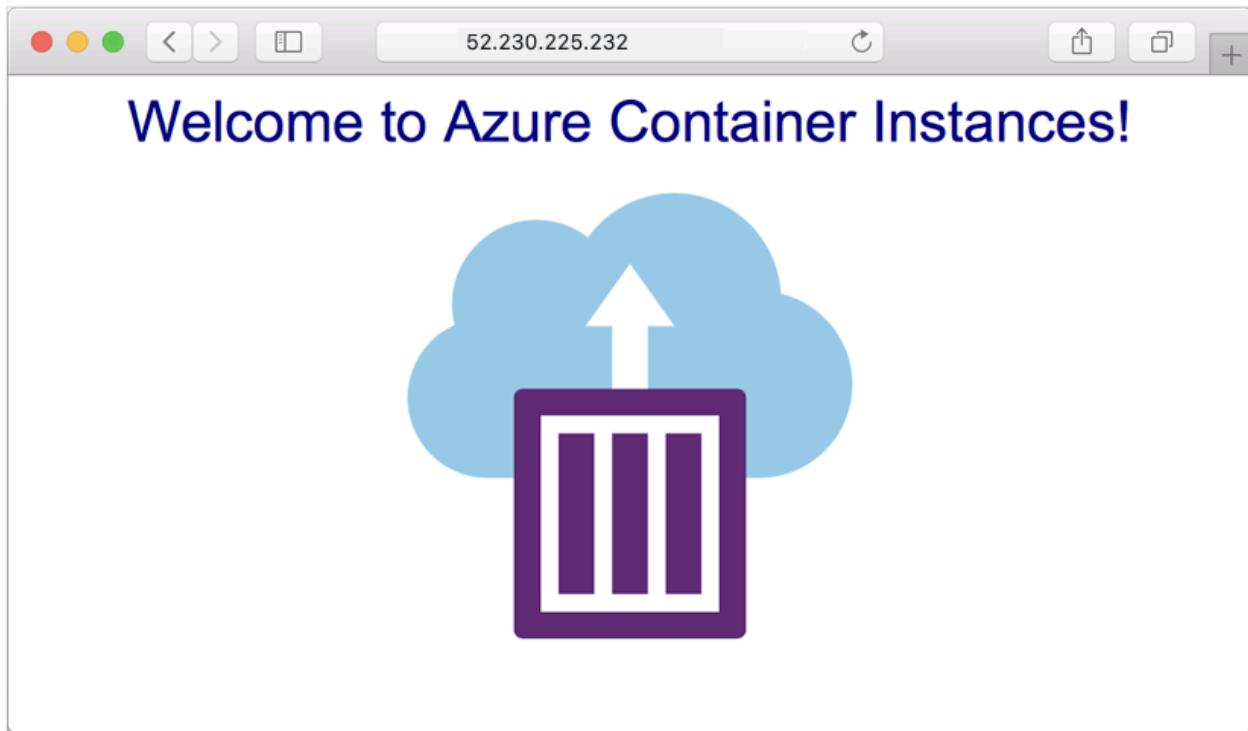
Run `docker ps` to get details about the running container, including the public IP address:

```
docker ps
```

Sample output shows a public IP address, in this case 52.230.225.232:

CONTAINER ID	IMAGE	COMMAND
STATUS	PORTS	
hungry-kirch	mcr.microsoft.com/azuredocs/aci-helloworld	
Running	52.230.225.232:80->80/tcp	

Now go to the IP address in your browser. If you see a web page similar to the following, congratulations! You successfully deployed an application running in a Docker container to Azure.



Pull the container logs

When you need to troubleshoot a container or the application it runs (or just see its output), start by viewing the container instance's logs.

For example, run the `docker logs` command to see the logs of the *hungry-kirch* container in the ACI context:

```
Azure CLI  
docker logs hungry-kirch
```

The output displays the logs for the container, and should show the HTTP GET requests generated when you viewed the application in your browser.

```
Output  
  
listening on port 80  
::ffff:10.240.255.55 - - [07/Jul/2020:17:43:53 +0000] "GET / HTTP/1.1" 304 -  
"-- Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML,  
like Gecko) Chrome/72.0.3626.121 Safari/537.36"  
::ffff:10.240.255.55 - - [07/Jul/2020:17:44:36 +0000] "GET / HTTP/1.1" 304 -  
"-- Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML,  
like Gecko) Chrome/72.0.3626.121 Safari/537.36"  
::ffff:10.240.255.55 - - [07/Jul/2020:17:44:36 +0000] "GET / HTTP/1.1" 304 -  
"-- Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML,  
like Gecko) Chrome/72.0.3626.121 Safari/537.36"
```

Clean up resources

When you're done with the container, run `docker rm` to remove it. This command stops and deletes the Azure container instance.

```
docker rm hungry-kirch
```

Next steps

In this quickstart, you created an Azure container instance from a public image by using integration between Docker and Azure. Learn more about integration scenarios in the [Docker documentation](#).

You can also use the [Docker extension for Visual Studio Code](#) for an integrated experience to develop, run, and manage containers, images, and contexts.

To use Azure tools to create and manage container instances, see other quickstarts using the [Azure CLI](#), [Azure PowerShell](#), [Azure portal](#), and [Azure Resource Manager template](#).

If you'd like to use Docker Compose to define and run a multi-container application locally and then switch to Azure Container Instances, continue to the tutorial.

[Docker Compose tutorial](#)

Feedback

Was this page helpful?

 Yes

 No

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

Tutorial: Create a container image for deployment to Azure Container Instances

Article • 08/29/2024

Azure Container Instances enables deployment of Docker containers onto Azure infrastructure without provisioning any virtual machines or adopting a higher-level service. In this tutorial, you package a small Node.js web application into a container image that can be run using Azure Container Instances.

In this article, part one of the series, you:

- ✓ Clone application source code from GitHub
- ✓ Create a container image from application source
- ✓ Test the image in a local Docker environment

In tutorial parts two and three, you upload your image to Azure Container Registry, and then deploy it to Azure Container Instances.

Before you begin

You must satisfy the following requirements to complete this tutorial:

Azure CLI: You must have Azure CLI version 2.0.29 or later installed on your local computer. To find the version, run `az --version`. If you need to install or upgrade, see [Install the Azure CLI](#).

Docker: This tutorial assumes a basic understanding of core Docker concepts like containers, container images, and basic `docker` commands. For a primer on Docker and container basics, see the [Docker overview](#).

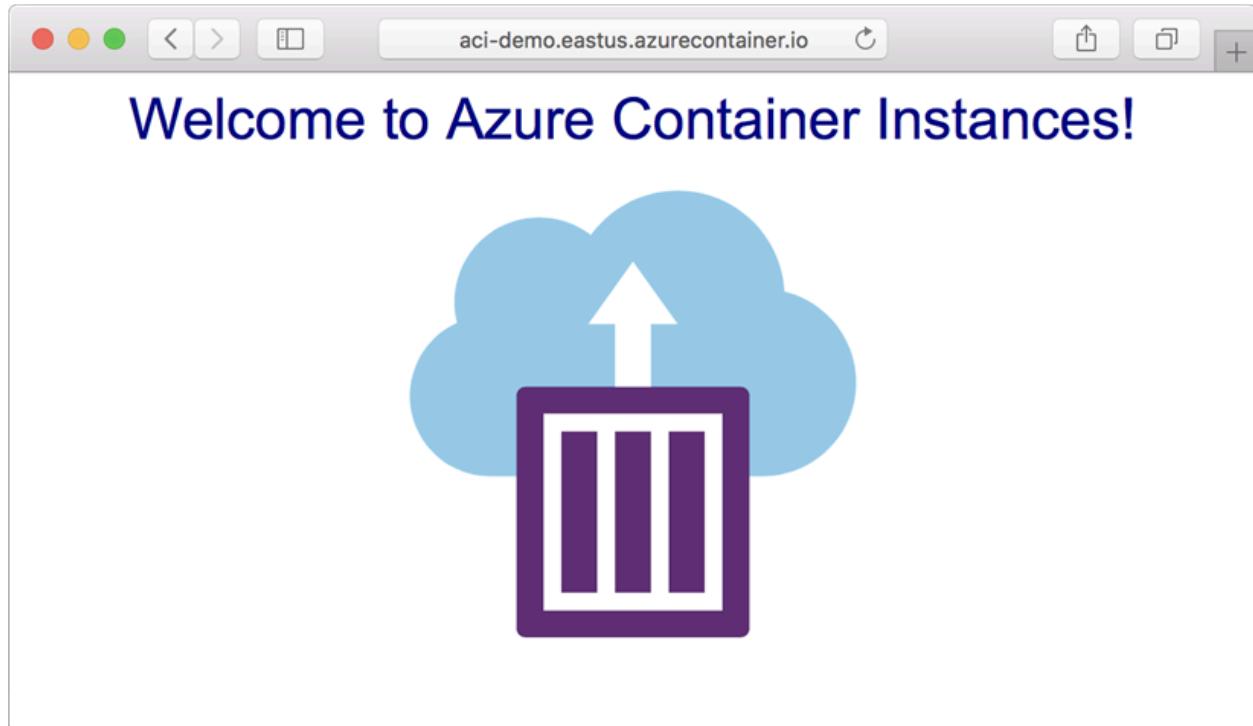
Docker: To complete this tutorial, you need Docker installed locally. Docker provides packages that configure the Docker environment on [macOS](#), [Windows](#), and [Linux](#).

Important

Because the Azure Cloud shell does not include the Docker daemon, you *must* install both the Azure CLI and Docker Engine on your *local computer* to complete this tutorial. You cannot use the Azure Cloud Shell for this tutorial.

Get application code

The sample application in this tutorial is a simple web app built in [Node.js](#). The application serves a static HTML page, and looks similar to the following screenshot:



Use Git to clone the sample application's repository:

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

You can also [download the ZIP archive](#) from GitHub directly.

Build the container image

The Dockerfile in the sample application shows how the container is built. It starts from an [official Node.js image](#) based on [Alpine Linux](#), a small distribution that is well suited for use with containers. It then copies the application files into the container, installs dependencies using the Node Package Manager, and finally, starts the application.

```
Dockerfile
FROM node:8.9.3-alpine
RUN mkdir -p /usr/src/app
COPY ./app/* /usr/src/app/
WORKDIR /usr/src/app
```

```
RUN npm install  
CMD node /usr/src/app/index.js
```

Use the [docker build](#) command to create the container image and tag it as *aci-tutorial-app*:

Bash

```
docker build ./aci-helloworld -t aci-tutorial-app
```

Output from the [docker build](#) command is similar to the following (truncated for readability):

Bash

```
docker build ./aci-helloworld -t aci-tutorial-app
```

Output

```
Sending build context to Docker daemon 119.3kB  
Step 1/6 : FROM node:8.9.3-alpine  
8.9.3-alpine: Pulling from library/node  
88286f41530e: Pull complete  
84f3a4bf8410: Pull complete  
d0d9b2214720: Pull complete  
Digest:  
sha256:c73277ccc763752b42bb2400d1aaecb4e3d32e3a9dbedd0e49885c71bea07354  
Status: Downloaded newer image for node:8.9.3-alpine  
--> 90f5ee24bee2  
...  
Step 6/6 : CMD node /usr/src/app/index.js  
--> Running in f4a1ea099eec  
--> 6edad76d09e9  
Removing intermediate container f4a1ea099eec  
Successfully built 6edad76d09e9  
Successfully tagged aci-tutorial-app:latest
```

Use the [docker images](#) command to see the built image:

Bash

```
docker images
```

Your newly built image should appear in the list:

Bash

```
docker images
```

Output

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
aci-tutorial-app	latest	5c745774dfa9	39 seconds ago	68.1 MB

Run the container locally

Before you deploy the container to Azure Container Instances, use [docker run](#) to run it locally and confirm that it works. The `-d` switch lets the container run in the background, while `-p` allows you to map an arbitrary port on your computer to port 80 in the container.

Bash

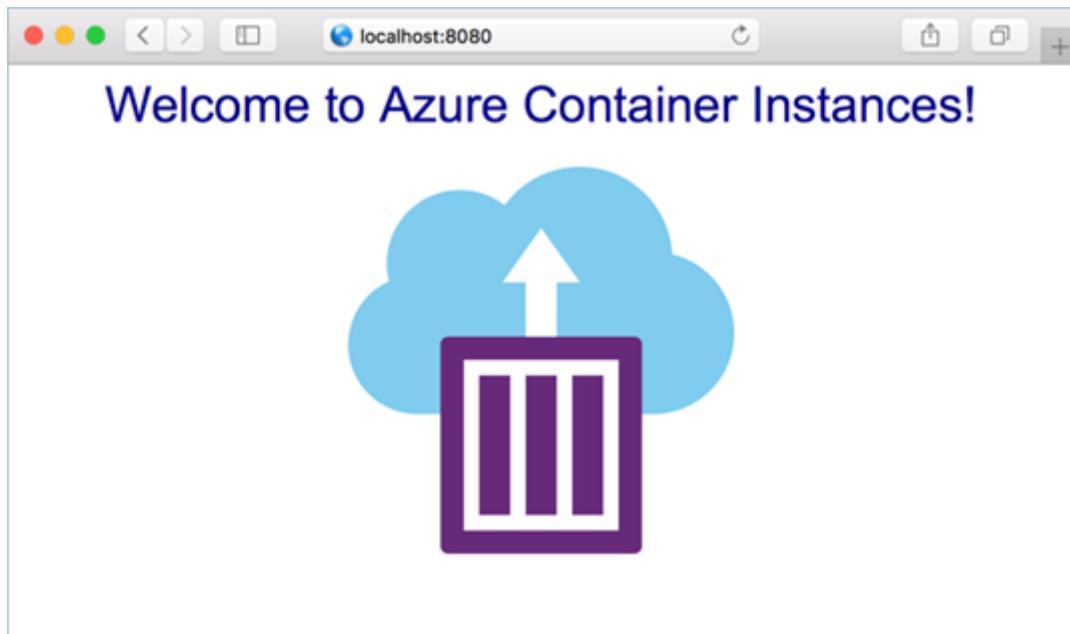
```
docker run -d -p 8080:80 aci-tutorial-app
```

Output from the `docker run` command displays the running container's ID if the command was successful:

Bash

```
docker run -d -p 8080:80 aci-tutorial-app
```
a2e3e4435db58ab0c664ce521854c2e1a1bda88c9cf2fcff46aedf48df86cccf
```

Now, navigate to `http://localhost:8080` in your browser to confirm that the container is running. You should see a web page similar to the following screenshot:



## Next steps

In this tutorial, you created a container image that can be deployed in Azure Container Instances, and verified that it runs locally. So far, you completed the following steps:

- ✓ Cloned the application source from GitHub
- ✓ Created a container image from the application source
- ✓ Tested the container locally

Advance to the next tutorial in the series to learn about storing your container image in Azure Container Registry:

[Push image to Azure Container Registry](#)

---

## Feedback

Was this page helpful?

Yes

No

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

# Tutorial: Create an Azure container registry and push a container image

Article • 08/29/2024

This article is part two of a three-part tutorial. [Part one](#) of the tutorial created a Docker container image for a Node.js web application. In this tutorial, you push the image to Azure Container Registry. If you need to create the container image, return to [Tutorial 1 – Create container image](#).

Azure Container Registry is your private Docker registry in Azure. In this tutorial, part two of the series, you:

- ✓ Create an Azure Container Registry instance with the Azure CLI
- ✓ Tag a container image for your Azure container registry
- ✓ Upload the image to your registry

In the next article, the last in the series, you deploy the container from your private registry to Azure Container Instances.

## Before you begin

You must satisfy the following requirements to complete this tutorial:

**Azure CLI:** You must have Azure CLI version 2.0.29 or later installed on your local computer. To find the version, run `az --version`. If you need to install or upgrade, see [Install the Azure CLI](#).

**Docker:** This tutorial assumes a basic understanding of core Docker concepts like containers, container images, and basic `docker` commands. For a primer on Docker and container basics, see the [Docker overview](#).

**Docker:** To complete this tutorial, you need Docker installed locally. Docker provides packages that configure the Docker environment on [macOS](#), [Windows](#), and [Linux](#).

### Important

Because the Azure Cloud shell does not include the Docker daemon, you *must* install both the Azure CLI and Docker Engine on your *local computer* to complete this tutorial. You cannot use the Azure Cloud Shell for this tutorial.

# Create Azure container registry

Before you create your container registry, you need a *resource group* to deploy it to. A resource group is a logical collection into which all Azure resources are deployed and managed.

Create a resource group with the [az group create](#) command. In the following example, a resource group named *myResourceGroup* is created in the *eastus* region:

Azure CLI

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

Once you create the resource group, create an Azure container registry with the [az acr create](#) command. The container registry name must be unique within Azure, and contain 5-50 alphanumeric characters. Replace `<acrName>` with a unique name for your registry:

Azure CLI

```
az acr create --resource-group myResourceGroup --name <acrName> --sku Basic
```

Here's partial output for a new Azure container registry named *mycontainerregistry082*:

Output

```
{
 "creationDate": "2020-07-16T21:54:47.297875+00:00",
 "id": "/subscriptions/<Subscription
ID>/resourceGroups/myResourceGroup/providers/Microsoft.ContainerRegistry/reg
istries/mycontainerregistry082",
 "location": "eastus",
 "loginServer": "mycontainerregistry082.azurecr.io",
 "name": "mycontainerregistry082",
 "provisioningState": "Succeeded",
 "resourceGroup": "myResourceGroup",
 "sku": {
 "name": "Basic",
 "tier": "Basic"
 },
 "status": null,
 "storageAccount": null,
 "tags": {},
 "type": "Microsoft.ContainerRegistry/registries"
}
```

The rest of the tutorial refers to `<acrName>` as a placeholder for the container registry name that you chose in this step.

## Sign in to container registry

You must sign in to your Azure Container Registry instance before pushing images to it. Use the [az acr login](#) command to complete the operation. You must provide the unique name you chose for the container registry when you created it.

Azure CLI

```
az acr login --name <acrName>
```

For example:

Azure CLI

```
az acr login --name mycontainerregistry082
```

The command returns `Login Succeeded` once completed:

Output

```
Login Succeeded
```

## Tag container image

To push a container image to a private registry like Azure Container Registry, you must first tag the image with the full name of the registry's login server.

First, get the full login server name for your Azure container registry. Run the following [az acr show](#) command, and replace `<acrName>` with the name of registry you created:

Azure CLI

```
az acr show --name <acrName> --query loginServer --output table
```

For example, if your registry is named *mycontainerregistry082*:

Azure CLI

```
az acr show --name mycontainerregistry082 --query loginServer --output table
```

Output

Result

```

mycontainerregistry082.azurecr.io
```

Now, display the list of your local images with the [docker images](#) command:

Bash

```
docker images
```

Along with any other images you have on your machine, you should see the *aci-tutorial-app* image you built in the [previous tutorial](#):

Bash

```
docker images
```

Output

| REPOSITORY       | TAG    | IMAGE ID     | CREATED        | SIZE    |
|------------------|--------|--------------|----------------|---------|
| aci-tutorial-app | latest | 5c745774dfa9 | 39 minutes ago | 68.1 MB |

Tag the *aci-tutorial-app* image with the login server of your container registry. Also, add the `:v1` tag to the end of the image name to indicate the image version number.

Replace `<acrLoginServer>` with the result of the `az acr show` command you executed earlier.

Bash

```
docker tag aci-tutorial-app <acrLoginServer>/aci-tutorial-app:v1
```

Run `docker images` again to verify the tagging operation:

Bash

```
docker images
```

Output

| REPOSITORY                                         |         | TAG    | IMAGE ID     |
|----------------------------------------------------|---------|--------|--------------|
| CREATED                                            | SIZE    |        |              |
| aci-tutorial-app                                   |         | latest | 5c745774dfa9 |
| 39 minutes ago                                     | 68.1 MB |        |              |
| mycontainerregistry082.azurecr.io/aci-tutorial-app |         | v1     | 5c745774dfa9 |
| 7 minutes ago                                      | 68.1 MB |        |              |

## Push image to Azure Container Registry

Now that you tagged the *aci-tutorial-app* image with the full login server name of your private registry, you can push the image to the registry with the [docker push ↗](#) command. Replace `<acrLoginServer>` with the full login server name you obtained in the earlier step.

Bash

```
docker push <acrLoginServer>/aci-tutorial-app:v1
```

The `push` operation should take a few seconds to a few minutes depending on your internet connection, and output is similar to the following example:

Bash

```
docker push mycontainerregistry082.azurecr.io/aci-tutorial-app:v1
```

Output

```
The push refers to a repository [mycontainerregistry082.azurecr.io/aci-tutorial-app]
3db9cac20d49: Pushed
13f653351004: Pushed
4cd158165f4d: Pushed
d8fb47558a8: Pushed
44ab46125c35: Pushed
5bef08742407: Pushed
v1: digest:
sha256:ed67ffff971da47175856505585dcd92d1270c3b37543e8af46014d328f05715
size: 1576
```

## List images in Azure Container Registry

To verify that the image you just pushed is indeed in your Azure container registry, list the images in your registry with the [az acr repository list](#) command. Replace `<acrName>`

with the name of your container registry.

Azure CLI

```
az acr repository list --name <acrName> --output table
```

For example:

Azure CLI

```
az acr repository list --name mycontainerregistry082 --output table
```

Output

Result

```

aci-tutorial-app
```

To see the *tags* for a specific image, use the [az acr repository show-tags](#) command.

Azure CLI

```
az acr repository show-tags --name <acrName> --repository aci-tutorial-app -
-output table
```

You should see output similar to the following example:

Output

```

v1
```

## Next steps

In this tutorial, you prepared an Azure container registry for use with Azure Container Instances, and pushed a container image to the registry. The following steps were completed:

- ✓ Created an Azure Container Registry instance with the Azure CLI
- ✓ Tagged a container image for Azure Container Registry
- ✓ Uploaded an image to Azure Container Registry

Advance to the next tutorial to learn how to deploy the container to Azure using Azure Container Instances:

[Deploy container to Azure Container Instances](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# Tutorial: Deploy a container application to Azure Container Instances

Article • 08/29/2024

This is the final tutorial in a three-part series. Earlier in the series, [a container image was created](#) and [pushed to Azure Container Registry](#). This article completes the series by deploying the container to Azure Container Instances.

In this tutorial, you:

- ✓ Deploy the container from Azure Container Registry to Azure Container Instances
- ✓ View the running application in the browser
- ✓ Display the container's logs

## Before you begin

You must satisfy the following requirements to complete this tutorial:

**Azure CLI:** You must have Azure CLI version 2.0.29 or later installed on your local computer. To find the version, run `az --version`. If you need to install or upgrade, see [Install the Azure CLI](#).

**Docker:** This tutorial assumes a basic understanding of core Docker concepts like containers, container images, and basic `docker` commands. For a primer on Docker and container basics, see the [Docker overview](#).

**Docker:** To complete this tutorial, you need Docker installed locally. Docker provides packages that configure the Docker environment on [macOS](#), [Windows](#), and [Linux](#).

### Important

Because the Azure Cloud shell does not include the Docker daemon, you *must* install both the Azure CLI and Docker Engine on your *local computer* to complete this tutorial. You cannot use the Azure Cloud Shell for this tutorial.

## Deploy the container using the Azure CLI

In this section, you use the Azure CLI to deploy the image built in the [first tutorial](#) and pushed to Azure Container Registry in the [second tutorial](#). Be sure you complete those

tutorials before proceeding.

## Get registry credentials

When you deploy an image that's hosted in a private Azure container registry like the one created in the [second tutorial](#), you must supply credentials to access the registry.

A best practice for many scenarios is to create and configure a Microsoft Entra service principal with *pull* permissions to your registry. See [Authenticate with Azure Container Registry from Azure Container Instances](#) for sample scripts to create a service principal with the necessary permissions. Take note of the *service principal ID* and *service principal password*. You use these credentials to access the registry when you deploy the container.

You also need the full name of the container registry sign-in server (replace `<acrName>` with the name of your registry):

Azure CLI

```
az acr show --name <acrName> --query loginServer
```

## Deploy container

Now, use the [az container create](#) command to deploy the container. Replace `<acrLoginServer>` with the value you obtained from the previous command. Replace `<service-principal-ID>` and `<service-principal-password>` with the service principal ID and password that you created to access the registry. Replace `<aciDnsLabel>` with a desired DNS name.

Azure CLI

```
az container create --resource-group myResourceGroup --name aci-tutorial-app
--image <acrLoginServer>/aci-tutorial-app:v1 --cpu 1 --memory 1 --registry-
login-server <acrLoginServer> --registry-username <service-principal-ID> --
registry-password <service-principal-password> --ip-address Public --dns-
name-label <aciDnsLabel> --ports 80
```

Within a few seconds, you should receive an initial response from Azure. The `--dns-name-label` value must be unique within the Azure region you create the container instance. Modify the value in the preceding command if you receive a **DNS name label** error message when you execute the command.

## Verify deployment progress

To view the state of the deployment, use [az container show](#):

Azure CLI

```
az container show --resource-group myResourceGroup --name aci-tutorial-app -
-query instanceView.state
```

Repeat the [az container show](#) command until the state changes from *Pending* to *Running*, which should take under a minute. When the container is *Running*, proceed to the next step.

## View the application and container logs

Once the deployment succeeds, display the container's fully qualified domain name (FQDN) with the [az container show](#) command:

Azure CLI

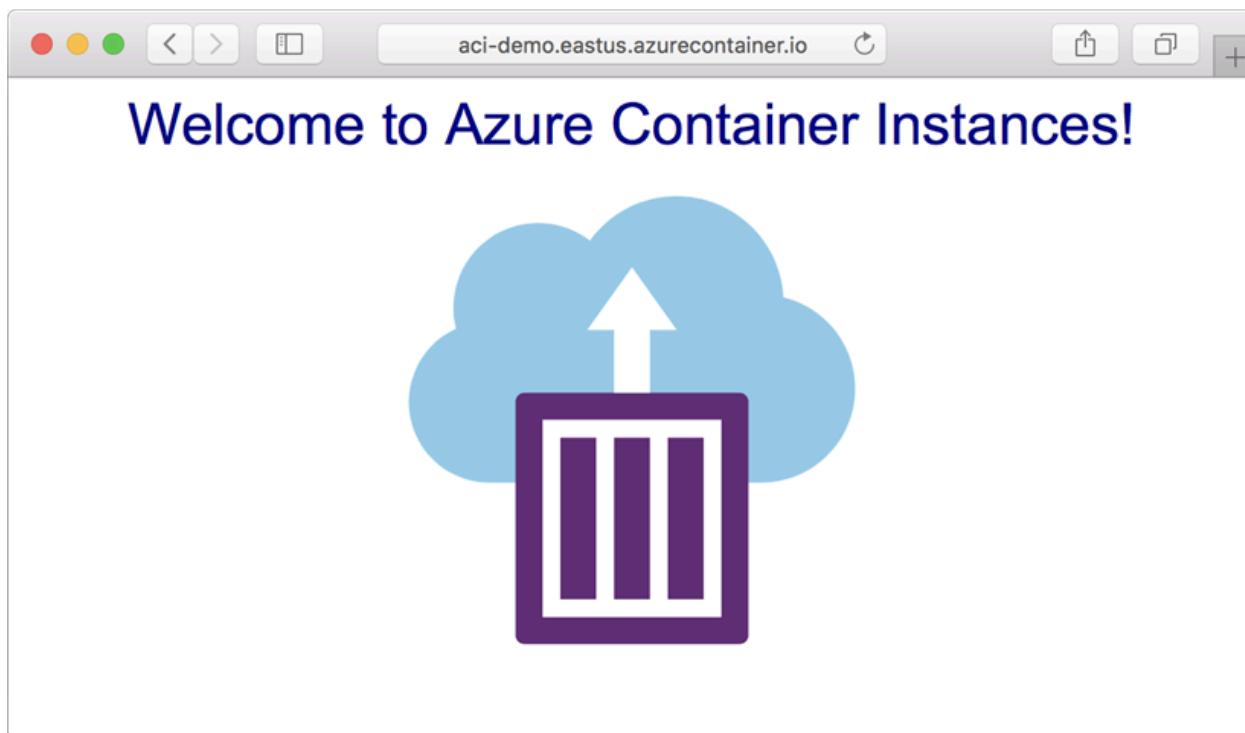
```
az container show --resource-group myResourceGroup --name aci-tutorial-app -
-query ipAddress.fqdn
```

For example:

Output

```
"aci-demo.eastus.azurecontainer.io"
```

To see the running application, navigate to the displayed DNS name in your favorite browser:



You can also view the log output of the container:

```
Azure CLI

az container logs --resource-group myResourceGroup --name aci-tutorial-app
```

Example output:

```
Output

listening on port 80
::ffff:10.240.0.4 - - [21/Jul/2017:06:00:02 +0000] "GET / HTTP/1.1" 200 1663
"-- Mozilla/5.0 (Macintosh; Intel Mac OS X 10_12_5) AppleWebKit/537.36
(KHTML, like Gecko) Chrome/59.0.3071.115 Safari/537.36"
::ffff:10.240.0.4 - - [21/Jul/2017:06:00:02 +0000] "GET /favicon.ico
HTTP/1.1" 404 150 "http://aci-demo.eastus.azurecontainer.io/" "Mozilla/5.0
(Macintosh; Intel Mac OS X 10_12_5) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/59.0.3071.115 Safari/537.36"
```

## Clean up resources

If you no longer need any of the resources you created in this tutorial series, you can execute the [az group delete](#) command to remove the resource group and all resources it contains. This command deletes the container registry you created, the running container, and all related resources.

```
Azure CLI
```

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

## Next steps

In this tutorial, you completed the process of deploying your container to Azure Container Instances. The following steps were completed:

- ✓ Deployed the container from Azure Container Registry using the Azure CLI
- ✓ Viewed the application in the browser
- ✓ Viewed the container logs

Now that you have the basics down, move on to learning more about Azure Container Instances, such as how container groups work:

[Container groups in Azure Container Instances](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# Tutorial: Deploy a multi-container group using a YAML file

Article • 08/29/2024

Azure Container Instances supports the deployment of multiple containers onto a single host using a [container group](#). A container group is useful when building an application sidecar for logging, monitoring, or any other configuration where a service needs a second attached process.

In this tutorial, you follow steps to run a two-container sidecar configuration by deploying a [YAML file](#) using the Azure CLI. A YAML file provides a concise format for specifying the instance settings. You learn how to:

- ✓ Configure a YAML file
- ✓ Deploy the container group
- ✓ View the logs of the containers

## ⓘ Note

Multi-container groups are currently restricted to Linux containers.

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

## Prerequisites

- Use the Bash environment in [Azure Cloud Shell](#). For more information, see [Quickstart for Bash in Azure Cloud Shell](#).  
[A Launch Cloud Shell](#)
- If you prefer to run CLI reference commands locally, [install](#) the Azure CLI. If you're running on Windows or macOS, consider running Azure CLI in a Docker container. For more information, see [How to run the Azure CLI in a Docker container](#).
  - If you're using a local installation, sign in to the Azure CLI by using the [az login](#) command. To finish the authentication process, follow the steps displayed in your terminal. For other sign-in options, see [Sign in with the Azure CLI](#).

- When you're prompted, install the Azure CLI extension on first use. For more information about extensions, see [Use extensions with the Azure CLI](#).
- Run `az version` to find the version and dependent libraries that are installed. To upgrade to the latest version, run `az upgrade`.

## Configure a YAML file

To deploy a multi-container group with the `az container create` command in the Azure CLI, you must specify the container group configuration in a YAML file. Then pass the YAML file as a parameter to the command.

Start by copying the following YAML into a new file named `deploy-aci.yaml`. In Azure Cloud Shell, you can use Visual Studio Code to create the file in your working directory:

```
code deploy-aci.yaml
```

This YAML file defines a container group named "myContainerGroup" with two containers, a public IP address, and two exposed ports. The containers are deployed from public Microsoft images. The first container in the group runs an internet-facing web application. The second container, the sidecar, periodically makes HTTP requests to the web application running in the first container via the container group's local network.

YAML

```
apiVersion: 2019-12-01
location: eastus
name: myContainerGroup
properties:
 containers:
 - name: aci-tutorial-app
 properties:
 image: mcr.microsoft.com/azuredocs/aci-helloworld:latest
 resources:
 requests:
 cpu: 1
 memoryInGb: 1.5
 ports:
 - port: 80
 - port: 8080
 - name: aci-tutorial-sidecar
 properties:
 image: mcr.microsoft.com/azuredocs/aci-tutorial-sidecar
 resources:
```

```
 requests:
 cpu: 1
 memoryInGb: 1.5
 osType: Linux
 ipAddress:
 type: Public
 ports:
 - protocol: tcp
 port: 80
 - protocol: tcp
 port: 8080
 tags: {exampleTag: tutorial}
 type: Microsoft.ContainerInstance/containerGroups
```

To use a private container image registry, add the `imageRegistryCredentials` property to the container group, with values modified for your environment:

YAML

```
imageRegistryCredentials:
- server: imageRegistryLoginServer
 username: imageRegistryUsername
 password: imageRegistryPassword
```

## Deploy the container group

Create a resource group with the `az group create` command:

Azure CLI

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

Deploy the container group with the `az container create` command, passing the YAML file as an argument:

Azure CLI

```
az container create --resource-group myResourceGroup --file deploy-aci.yaml
```

Within a few seconds, you should receive an initial response from Azure.

## View deployment state

To view the state of the deployment, use the following `az container show` command:

## Azure CLI

```
az container show --resource-group myResourceGroup --name myContainerGroup -
-output table
```

If you'd like to view the running application, navigate to its IP address in your browser. For example, the IP is 52.168.26.124 in this example output:

### Console

| Name                 | ResourceGroup | Status          | Image                                                     |          |
|----------------------|---------------|-----------------|-----------------------------------------------------------|----------|
| IP:ports             | Network       | CPU/Memory      | OsType                                                    | Location |
| myContainerGroup     | danlep0318r   | Running         | mcr.microsoft.com/azuredocs/aci-tutorial-                 |          |
|                      |               |                 | sidecar,mcr.microsoft.com/azuredocs/aci-helloworld:latest |          |
| 20.42.26.114:80,8080 | Public        | 1.0 core/1.5 gb | Linux                                                     | eastus   |

## View container logs

View the log output of a container using the `az container logs` command. The `--container-name` argument specifies the container from which to pull logs. In this example, the `aci-tutorial-app` container is specified.

## Azure CLI

```
az container logs --resource-group myResourceGroup --name myContainerGroup -
-container-name aci-tutorial-app
```

Output:

### Console

```
listening on port 80
::1 - - [02/Jul/2020:23:17:48 +0000] "HEAD / HTTP/1.1" 200 1663 "-"
"curl/7.54.0"
::1 - - [02/Jul/2020:23:17:51 +0000] "HEAD / HTTP/1.1" 200 1663 "-"
"curl/7.54.0"
::1 - - [02/Jul/2020:23:17:54 +0000] "HEAD / HTTP/1.1" 200 1663 "-"
"curl/7.54.0"
```

To see the logs for the sidecar container, run a similar command specifying the `aci-tutorial-sidecar` container.

## Azure CLI

```
az container logs --resource-group myResourceGroup --name myContainerGroup -
-container-name aci-tutorial-sidecar
```

Output:

### Console

```
Every 3s: curl -I http://localhost
20:36:41 2020-07-02

% Total % Received % Xferd Average Speed Time Time Time
Current Dload Upload Total Spent Left
Speed
0 1663 0 0 0 0 0 --::-- --::-- --::--
0
HTTP/1.1 200 OK
X-Powered-By: Express
Accept-Ranges: bytes
Cache-Control: public, max-age=0
Last-Modified: Wed, 29 Nov 2017 06:40:40 GMT
ETag: W/"67f-16006818640"
Content-Type: text/html; charset=UTF-8
Content-Length: 1663
Date: Thu, 02 Jul 2020 20:36:41 GMT
Connection: keep-alive
```

As you can see, the sidecar is periodically making an HTTP request to the main web application via the group's local network to ensure it runs. This sidecar example could be expanded to trigger an alert if it received an HTTP response code other than `200 OK`.

## Next steps

In this tutorial, you used a YAML file to deploy a multi-container group in Azure Container Instances. You learned how to:

- ✓ Configure a YAML file for a multi-container group
- ✓ Deploy the container group
- ✓ View the logs of the containers

You can also specify a multi-container group using a [Resource Manager template](#). A Resource Manager template can be readily adapted for scenarios when you need to deploy more Azure service resources with the container group.

# Feedback

Was this page helpful?

 Yes

 No

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

# Tutorial: Deploy a multi-container group using a Resource Manager template

Article • 08/29/2024

Azure Container Instances supports the deployment of multiple containers onto a single host using a [container group](#). A container group is useful when building an application sidecar for logging, monitoring, or any other configuration where a service needs a second attached process.

In this tutorial, you follow steps to run a two-container sidecar configuration by deploying an Azure Resource Manager template using the Azure CLI. You learn how to:

- ✓ Configure a multi-container group template
- ✓ Deploy the container group
- ✓ View the logs of the containers

A Resource Manager template can be readily adapted for scenarios when you need to deploy more Azure service resources (for example, an Azure Files share or a virtual network) with the container group.

## ⓘ Note

Multi-container groups are currently restricted to Linux containers.

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

## Prerequisites

- Use the Bash environment in [Azure Cloud Shell](#). For more information, see [Quickstart for Bash in Azure Cloud Shell](#).  
A blue rectangular button with a white 'A' icon and the text 'Launch Cloud Shell'. To the right of the text is a small blue square with a white right-pointing arrow.
- If you prefer to run CLI reference commands locally, [install](#) the Azure CLI. If you're running on Windows or macOS, consider running Azure CLI in a Docker container. For more information, see [How to run the Azure CLI in a Docker container](#).
  - If you're using a local installation, sign in to the Azure CLI by using the [az login](#) command. To finish the authentication process, follow the steps displayed in

your terminal. For other sign-in options, see [Sign in with the Azure CLI](#).

- When you're prompted, install the Azure CLI extension on first use. For more information about extensions, see [Use extensions with the Azure CLI](#).
- Run `az version` to find the version and dependent libraries that are installed. To upgrade to the latest version, run `az upgrade`.

## Configure a template

Start by copying the following JSON into a new file named `azuredeploy.json`. In Azure Cloud Shell, you can use Visual Studio Code to create the file in your working directory:

```
code azuredeploy.json
```

This Resource Manager template defines a container group with two containers, a public IP address, and two exposed ports. The first container in the group runs an internet-facing web application. The second container, the sidecar, makes an HTTP request to the main web application via the group's local network.

JSON

```
{
 "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",
 "contentVersion": "1.0.0.0",
 "parameters": {
 "containerGroupName": {
 "type": "string",
 "defaultValue": "myContainerGroup",
 "metadata": {
 "description": "Container Group name."
 }
 }
 },
 "variables": {
 "container1name": "aci-tutorial-app",
 "container1image": "mcr.microsoft.com/azuredocs/aci-helloworld:latest",
 "container2name": "aci-tutorial-sidecar",
 "container2image": "mcr.microsoft.com/azuredocs/aci-tutorial-sidecar"
 },
 "resources": [
 {
 "name": "[parameters('containerGroupName')]",
 "type": "Microsoft.ContainerInstance/containerGroups",
 "apiVersion": "2019-12-01",
 "properties": {
 "containers": [
 {
 "name": "aci-tutorial-app",
 "image": "mcr.microsoft.com/azuredocs/aci-helloworld:latest",
 "ports": [
 {
 "port": 80
 }
],
 "resources": {
 "requests": {
 "cpu": 1,
 "memory": 1
 }
 }
 },
 {
 "name": "aci-tutorial-sidecar",
 "image": "mcr.microsoft.com/azuredocs/aci-tutorial-sidecar",
 "ports": [
 {
 "port": 80
 }
],
 "resources": {
 "requests": {
 "cpu": 1,
 "memory": 1
 }
 }
 }
],
 "osType": "Linux"
 }
 }
]
}
```

```
"location": "[resourceGroup().location]",
"properties": {
 "containers": [
 {
 "name": "[variables('container1name')]",
 "properties": {
 "image": "[variables('container1image')]",
 "resources": {
 "requests": {
 "cpu": 1,
 "memoryInGb": 1.5
 }
 },
 "ports": [
 {
 "port": 80
 },
 {
 "port": 8080
 }
]
 }
 },
 {
 "name": "[variables('container2name')]",
 "properties": {
 "image": "[variables('container2image')]",
 "resources": {
 "requests": {
 "cpu": 1,
 "memoryInGb": 1.5
 }
 }
 }
 }
],
 "osType": "Linux",
 "ipAddress": {
 "type": "Public",
 "ports": [
 {
 "protocol": "tcp",
 "port": 80
 },
 {
 "protocol": "tcp",
 "port": 8080
 }
]
 }
},
"outputs": {
 "containerIPv4Address": {

```

```
 "type": "string",
 "value": "
[reference(resourceId('Microsoft.ContainerInstance/containerGroups/',
parameters('containerGroupName'))).ipAddress.ip]"
 }
}
}
```

To use a private container image registry, add an object to the JSON document with the following format. For an example implementation of this configuration, see the [ACI Resource Manager template reference](#) documentation.

JSON

```
"imageRegistryCredentials": [
{
 "server": "[parameters('imageRegistryLoginServer')]",
 "username": "[parameters('imageRegistryUsername')]",
 "password": "[parameters('imageRegistryPassword')]"
}
]
```

## Deploy the template

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

Azure CLI

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

Deploy the template with the [az deployment group create](#) command.

Azure CLI

```
az deployment group create --resource-group myResourceGroup --template-file
azuredeploy.json
```

Within a few seconds, you should receive an initial response from Azure.

## View deployment state

To view the state of the deployment, use the following [az container show](#) command:

Azure CLI

```
az container show --resource-group myResourceGroup --name myContainerGroup --output table
```

If you'd like to view the running application, navigate to its IP address in your browser. For example, the IP is 52.168.26.124 in this example output:

| Name                                                                                               | ResourceGroup | Status          | Image  |          |
|----------------------------------------------------------------------------------------------------|---------------|-----------------|--------|----------|
| IP:ports                                                                                           | Network       | CPU/Memory      | OsType | Location |
| myContainerGroup                                                                                   | danlep0318r   | Running         |        |          |
| mcr.microsoft.com/azuredocs/aci-tutorial-sidecar,mcr.microsoft.com/azuredocs/aci-helloworld:latest |               |                 |        |          |
| 20.42.26.114:80,8080                                                                               | Public        | 1.0 core/1.5 gb | Linux  | eastus   |

## View container logs

View the log output of a container using the `az container logs` command. The `--container-name` argument specifies the container from which to pull logs. In this example, the `aci-tutorial-app` container is specified.

| Azure CLI                                                                                                               |
|-------------------------------------------------------------------------------------------------------------------------|
| <pre>az container logs --resource-group myResourceGroup --name myContainerGroup --container-name aci-tutorial-app</pre> |

Output:

| Console                                                                                                                                                                                                                                                                               |
|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <pre>listening on port 80 ::1 - - [02/Jul/2020:23:17:48 +0000] "HEAD / HTTP/1.1" 200 1663 "-" "curl/7.54.0" ::1 - - [02/Jul/2020:23:17:51 +0000] "HEAD / HTTP/1.1" 200 1663 "-" "curl/7.54.0" ::1 - - [02/Jul/2020:23:17:54 +0000] "HEAD / HTTP/1.1" 200 1663 "-" "curl/7.54.0"</pre> |

To see the logs for the sidecar container, run a similar command specifying the `aci-tutorial-sidecar` container.

## Azure CLI

```
az container logs --resource-group myResourceGroup --name myContainerGroup -
-container-name aci-tutorial-sidecar
```

Output:

### Console

```
Every 3s: curl -I http://localhost 2020-07-02
20:36:41

% Total % Received % Xferd Average Speed Time Time Time
Current Dload Upload Total Spent Left
Speed
0 1663 0 0 0 0 0 --:--:-- --:--:-- --:--:--
0
HTTP/1.1 200 OK
X-Powered-By: Express
Accept-Ranges: bytes
Cache-Control: public, max-age=0
Last-Modified: Wed, 29 Nov 2017 06:40:40 GMT
ETag: W/"67f-16006818640"
Content-Type: text/html; charset=UTF-8
Content-Length: 1663
Date: Thu, 02 Jul 2020 20:36:41 GMT
Connection: keep-alive
```

As you can see, the sidecar is periodically making an HTTP request to the main web application via the group's local network to ensure it runs. This sidecar example could be expanded to trigger an alert if it received an HTTP response code other than `200 OK`.

## Next steps

In this tutorial, you used an Azure Resource Manager template to deploy a multi-container group in Azure Container Instances. You learned how to:

- ✓ Configure a multi-container group template
- ✓ Deploy the container group
- ✓ View the logs of the containers

For more template samples, see [Azure Resource Manager templates for Azure Container Instances](#).

You can also specify a multi-container group using a [YAML file](#). Due to the YAML format's more concise nature, deployment with a YAML file is a good choice when your

deployment includes only container instances.

---

## Feedback

Was this page helpful?

 Yes

 No

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

# Tutorial: Use an HTTP-triggered Azure function to create a container group

Article • 08/29/2024

Azure Functions is a serverless compute service that can run scripts or code in response to various events, such as an HTTP request, a timer, or a message in an Azure Storage queue.

In this tutorial, you create an Azure function that takes an HTTP request and triggers deployment of a [container group](#). This example shows the basics of using Azure Functions to automatically create resources in Azure Container Instances. Modify or extend the example for more complex scenarios or other event triggers.

You learn how to:

- ✓ Use Visual Studio Code with the [Azure Functions extension](#) to create a basic HTTP-triggered PowerShell function.
- ✓ Enable an identity in the function app and give it permissions to create Azure resources.
- ✓ Modify and republish the PowerShell function to automate deployment of a single-container container group.
- ✓ Verify the HTTP-triggered deployment of the container.

## Prerequisites

See [Create your first function in Azure using Visual Studio Code](#) for prerequisites to install and use Visual Studio Code with the Azure Functions extension on your OS.

Additional steps in this article use Azure PowerShell. If you need to install or upgrade, see [Install Azure PowerShell](#) and [Sign into Azure](#).

## Create a basic PowerShell function

Follow steps in [Create your first PowerShell function in Azure](#) to create a PowerShell function using the HTTP Trigger template. Use the default Azure function name `HttpTrigger`. As shown in the quickstart, test the function locally, and publish the project to a function app in Azure. This example is a basic HTTP-triggered function that returns a text string. In later steps in this article, you modify the function to create a container group.

This article assumes you publish the project using the name *myfunctionapp*, in an Azure resource group automatically named according to the function app name (also *myfunctionapp*). Substitute your unique function app name and resource group name in later steps.

## Enable an Azure-managed identity in the function app

The following commands enable a system-assigned [managed identity](#) in your function app. The PowerShell host running the app can automatically authenticate to Azure using this identity, enabling functions to take actions on Azure services to which the identity is granted access. In this tutorial, you grant the managed identity permissions to create resources in the function app's resource group.

Add an identity to the function app:

```
Azure PowerShell

Update-AzFunctionApp -Name myfunctionapp `

 -ResourceGroupName myfunctionapp `

 -IdentityType SystemAssigned
```

Assign the identity the contributor role scoped to the resource group:

```
Azure PowerShell

$SP=(Get-AzADServicePrincipal -DisplayName myfunctionapp).Id
$RG=(Get-AzResourceGroup -Name myfunctionapp).ResourceId
New-AzRoleAssignment -ObjectId $SP -RoleDefinitionName "Contributor" -Scope
$RG
```

## Modify HttpTrigger function

Modify the PowerShell code for the **HttpTrigger** function to create a container group. In file `run.ps1` for the function, find the following code block. This code displays a name value, if one is passed as a query string in the function URL:

```
Azure PowerShell

[...]
if ($name) {
 $body = "Hello, $name. This HTTP triggered function executed
successfully."
```

```
}
```

```
[...]
```

Replace this code with the following example block. Here, if a name value is passed in the query string, it's used to name and create a container group using the [New-AzContainerGroup](#) cmdlet. Make sure to replace the resource group name `myfunctionapp` with the name of the resource group for your function app:

```
Azure PowerShell
```

```
[...]
if ($name) {
 New-AzContainerGroup -ResourceGroupName myfunctionapp -Name $name `
 -Image alpine -OsType Linux `
 -Command "echo 'Hello from an Azure container instance triggered by
an Azure function'" `
 -RestartPolicy Never
 if ($?) {
 $body = "This HTTP triggered function executed successfully. Started
container group $name"
 }
 else {
 $body = "There was a problem starting the container group."
 }
}
[...]
```

This example creates a container group consisting of a single container instance running the `alpine` image. The container runs a single `echo` command and then terminates. In a real-world example, you might trigger creation of one or more container groups for running a batch job.

## Test function app locally

Ensure the function runs locally before republishing the function app project to Azure. When run locally, the function doesn't create Azure resources. However, you can test the function flow with and without passing a name value in a query string. To debug the function, see [Debug PowerShell Azure Functions locally](#).

## Republish Azure function app

After you verify the function runs locally, republish the project to the existing function app in Azure.

1. In Visual Studio Code, open the Command Palette. Search for and select `Azure Functions: Deploy to Function App...`.
2. Select the current working folder to zip and deploy.
3. Select the subscription and then the name of the existing function app (`myfunctionapp`). Confirm that you want to overwrite the previous deployment.

A notification is displayed after your function app is created and the deployment package is applied. Select **View Output** in this notification to view the creation and deployment results, including the Azure resources that you updated.

## Run the function in Azure

After the deployment completes successfully, get the function URL. For example, use the **Azure: Functions** area in Visual Studio Code to copy the **HttpTrigger** function URL, or get the function URL in the [Azure portal](#).

The function URL is of the form:

```
config
https://myfunctionapp.azurewebsites.net/api/HttpTrigger
```

## Run function without passing a name

As a first test, run the `curl` command and pass the function URL without appending a `name` query string.

```
Bash
curl --verbose "https://myfunctionapp.azurewebsites.net/api/HttpTrigger"
```

The function returns status code 200 and the text `This HTTP triggered function executed successfully. Pass a name in the query string or in the request body for a personalized response:`

```
[...]
> GET /api/HttpTrigger? HTTP/1.1
> Host: myfunctionapp.azurewebsites.net
> User-Agent: curl/7.64.1
> Accept: */*
>
```

```
* Connection state changed (MAX_CONCURRENT_STREAMS updated)!
< HTTP/1.1 200 OK
< Content-Length: 135
< Content-Type: text/plain; charset=utf-8
< Request-Context: appId=cid-v1:d0bd0123-f713-4579-8990-bb368a229c38
< Date: Wed, 10 Jun 2020 17:50:27 GMT
<
* Connection #0 to host myfunctionapp.azurewebsites.net left intact
This HTTP triggered function executed successfully. Pass a name in the query
string or in the request body for a personalized response.* Closing
connection 0
```

## Run function and pass the name of a container group

Now run the `curl` command and append the name of a container group (`mycontainergroup`) as a query string `?name=mycontainergroup`:

Bash

```
curl --verbose "https://myfunctionapp.azurewebsites.net/api/HttpTrigger?
name=mycontainergroup"
```

The function returns status code 200 and triggers the creation of the container group:

```
[...]
> GET /api/HttpTrigger1?name=mycontainergroup HTTP/1.1
> Host: myfunctionapp.azurewebsites.net
> User-Agent: curl/7.64.1
> Accept: */*
>
< HTTP/1.1 200 OK
< Content-Length: 92
< Content-Type: text/plain; charset=utf-8
< Request-Context: appId=cid-v1:d0bd0123-f713-4579-8990-bb368a229c38
< Date: Wed, 10 Jun 2020 17:54:31 GMT
<
* Connection #0 to host myfunctionapp.azurewebsites.net left intact
This HTTP triggered function executed successfully. Started container group
mycontainergroup* Closing connection 0
```

Verify that the container ran with the [Get-AzContainerInstanceLog](#) command:

Azure CLI

```
Get-AzContainerInstanceLog -ResourceGroupName myfunctionapp `
```

```
-ContainerGroupName mycontainergroup
```

Sample output:

Output

```
Hello from an Azure container instance triggered by an Azure function
```

## Clean up resources

If you no longer need any of the resources you created in this tutorial, you can execute the [az group delete][az-group-delete] command to remove the resource group and all resources it contains. This command deletes the function app you created, the running container, and all related resources.

Azure CLI

```
az group delete --name myfunctionapp
```

## Next steps

In this tutorial, you created an Azure function that takes an HTTP request and triggers deployment of a container group. You learned how to:

- ✓ Use Visual Studio Code with the Azure Functions extension to create a basic HTTP-triggered PowerShell function.
- ✓ Enable an identity in the function app and give it permissions to create Azure resources.
- ✓ Modify the PowerShell function code to automate deployment of a single-container container group.
- ✓ Verify the HTTP-triggered deployment of the container.

For a detailed example to launch and monitor a containerized job, see the blog post [Event-Driven Serverless Containers with PowerShell Azure Functions and Azure Container Instances](#) and accompanying [code sample](#).

See the [Azure Functions documentation](#) for detailed guidance on creating Azure functions and publishing a functions project.

# Feedback

Was this page helpful?

 Yes

 No

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

# Tutorial: Prepare a deployment for a confidential container on Azure Container Instances

Article • 08/29/2024

In Azure Container Instances, you can use confidential containers on the serverless platform to run container applications in a hardware-based and attested trusted execution environment (TEE). This capability can help protect data in use and provides in-memory encryption via Secure Nested Paging.

In this tutorial, you learn how to:

- ✓ Create an Azure Resource Manager template (ARM template) for a confidential container group.
- ✓ Generate a confidential computing enforcement (CCE) policy.
- ✓ Deploy the confidential container group to Azure.

## Prerequisites

To complete this tutorial, you must satisfy the following requirements:

- **Azure CLI:** You must have Azure CLI version 2.44.1 or later installed on your local computer. To find your version, run `az --version`. If you need to install or upgrade, see [Install the Azure CLI](#).
- **Azure CLI confcom extension:** You must have Azure CLI confcom extension version 0.30+ installed to generate confidential computing enforcement policies.

Bash

```
az extension add -n confcom
```

- **Docker:** You need Docker installed locally. Docker provides packages that configure the Docker environment on [macOS](#), [Windows](#), and [Linux](#).

This tutorial assumes a basic understanding of core Docker concepts like containers, container images, and basic `docker` commands. For a primer on Docker and container basics, see the [Docker overview](#).

 **Important**

Because Azure Cloud Shell doesn't include the Docker daemon, you must install both the Azure CLI and Docker Engine on your *local computer* to complete this tutorial. You can't use Azure Cloud Shell for this tutorial.

# Create an ARM template for a Container Instances container group

In this tutorial, you deploy a Hello World application that generates a hardware attestation report. You start by creating an ARM template with a container group resource to define the properties of this application. You then use this ARM template with the Azure CLI confcom tooling to generate a CCE policy for attestation.

This tutorial uses [this ARM template](#) as an example. To view the source code for this application, see [Azure Confidential Container Instances Hello World](#).

The example template adds two properties to the Container Instances resource definition to make the container group confidential:

- `sku`: Enables you to select between confidential and standard container group deployments. If you don't add this property to the resource, the container group is a standard deployment.
- `confidentialComputeProperties`: Enables you to pass in a custom CCE policy for attestation of your container group. If you don't add this object to the resource, the software components that run within the container group won't validate.

## ⓘ Note

The `ccePolicy` parameter under `confidentialComputeProperties` is blank. You'll fill it in when you generate the policy later in the tutorial.

Use your preferred text editor to save this ARM template on your local machine as `template.json`.

ARM

```
{
 "$schema": "https://schema.management.azure.com/schemas/2019-04-
01/deploymentTemplate.json#",
 "contentVersion": "1.0.0.0",
 "parameters": {
 "name": {
 "type": "string",
```

```
 "defaultValue": "helloworld",
 "metadata": {
 "description": "Name for the container group"
 }
 },
 "location": {
 "type": "string",
 "defaultValue": "North Europe",
 "metadata": {
 "description": "Location for all resources."
 }
 },
 "image": {
 "type": "string",
 "defaultValue":
"mcr.microsoft.com/public/acc/samples/aci/helloworld:2.7",
 "metadata": {
 "description": "Container image to deploy. Should be of the form repoName/imagename:tag for images stored in public Docker Hub, or a fully qualified URI for other registries. Images from private registries require additional registry credentials."
 }
 },
 "port": {
 "type": "int",
 "defaultValue": 80,
 "metadata": {
 "description": "Port to open on the container and the public IP address."
 }
 },
 "cpuCores": {
 "type": "int",
 "defaultValue": 1,
 "metadata": {
 "description": "The number of CPU cores to allocate to the container."
 }
 },
 "memoryInGb": {
 "type": "int",
 "defaultValue": 1,
 "metadata": {
 "description": "The amount of memory to allocate to the container in gigabytes."
 }
 },
 "restartPolicy": {
 "type": "string",
 "defaultValue": "Never",
 "allowedValues": [
 "Always",
 "Never",
 "OnFailure"
],
 }
},
```

```
 "metadata": {
 "description": "The behavior of Azure runtime if container has stopped."
 }
},
"resources": [
{
 "type": "Microsoft.ContainerInstance/containerGroups",
 "apiVersion": "2023-05-01",
 "name": "[parameters('name')]",
 "location": "[parameters('location')]",
 "properties": {
 "confidentialComputeProperties": {
 "ccePolicy": ""
 },
 "containers": [
{
 "name": "[parameters('name')]",
 "properties": {
 "image": "[parameters('image')]",
 "ports": [
{
 "port": "[parameters('port')]",
 "protocol": "TCP"
}
],
 "resources": {
 "requests": {
 "cpu": "[parameters('cpuCores')]",
 "memoryInGB": "[parameters('memoryInGb')]"
}
}
}
]
}
],
 "sku": "Confidential",
 "osType": "Linux",
 "restartPolicy": "[parameters('restartPolicy')]",
 "ipAddress": {
 "type": "Public",
 "ports": [
{
 "port": "[parameters('port')]",
 "protocol": "TCP"
}
]
}
}
],
 "outputs": {
 "containerIPv4Address": {
 "type": "string",
 "value": "

```

```
[reference(resourceId('Microsoft.ContainerInstance/containerGroups',
parameters('name'))).ipAddress.ip]"
}
}
}
```

## Create a custom CCE policy

With the ARM template that you crafted and the Azure CLI confcom extension, you can generate a custom CCE policy. The CCE policy is used for attestation. The tool takes the ARM template as an input to generate the policy. The policy enforces the specific container images, environment variables, mounts, and commands, which can then be validated when the container group starts up. For more information on the Azure CLI confcom extension, see the [documentation on GitHub](#).

1. To generate the CCE policy, run the following command by using the ARM template as input:

```
Azure CLI

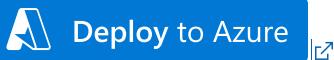
az confcom acipolicygen -a .\template.json
```

When this command finishes, a Base64 string generated as output will automatically appear in the `ccePolicy` property of the ARM template.

## Deploy the template

In the following steps, you use the Azure portal to deploy the template. You can also use Azure PowerShell, the Azure CLI, or the REST API. To learn about other deployment methods, see [Deploy templates](#).

1. Select the **Deploy to Azure** button to sign in to Azure and begin a Container Instances deployment.



2. Select **Build your own template in the editor**.

## Custom deployment ...

Deploy from a custom template

### Select a template

Basics

Review + create

Automate deploying resources with Azure Resource Manager templates in a single, coordinated operation. Create or select a template below to get started. [Learn more about template deployment](#)

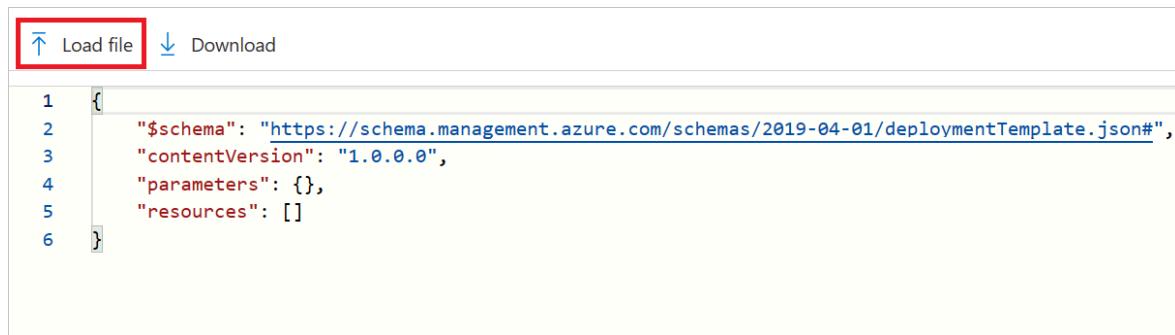


Build your own template in the editor

### Common templates

The template JSON that appears is mostly blank.

3. Select **Load file** and upload *template.json*, which you modified by adding the CCE policy in the previous steps.



```
1 [
2 "$schema": "https://schema.management.azure.com/schemas/2019-04-01/deploymentTemplate.json#",
3 "contentVersion": "1.0.0.0",
4 "parameters": {},
5 "resources": []
6]
```

4. Select **Save**.

5. Select or enter the following values:

- **Subscription:** Select an Azure subscription.
- **Resource group:** Select **Create new**, enter a unique name for the resource group, and then select **OK**.
- **Name:** Accept the generated name for the instance, or enter a name.
- **Location:** Select a location for the resource group. Choose a region where **confidential containers are supported**. Example: **North Europe**.
- **Image:** Accept the default image name. This sample Linux image displays a hardware attestation.

Accept default values for the remaining properties, and then select **Review + create**.

# Custom deployment

Deploy from a custom template



Customized template

1 resource



Edit template



Edit parameters



Visualize

## Project details

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

Subscription \* ⓘ

Windows Fabric - Internal Consumption (PM Team)

Resource group \* ⓘ

(New) myACIRG

Create new

## Instance details

Region \* ⓘ

North Europe

Name ⓘ

helloworld

Location ⓘ

North Europe

Image ⓘ

mcr.microsoft.com/aci/aci-confidential-helloworld:v1

Port ⓘ

80

Cpu Cores ⓘ

1

Memory In Gb ⓘ

1

Restart Policy ⓘ

Never

**Review + create**

< Previous

Next : Review + create >

6. Review the terms and conditions. If you agree, select **I agree to the terms and conditions stated above**.

7. Wait until the **Deployment succeeded** notification appears. It confirms that you successfully created the instance.

**Deployment succeeded**

Deployment 'Microsoft.Template-20230217150208' to resource group 'myACIRG' was successful.

**Pin to dashboard**

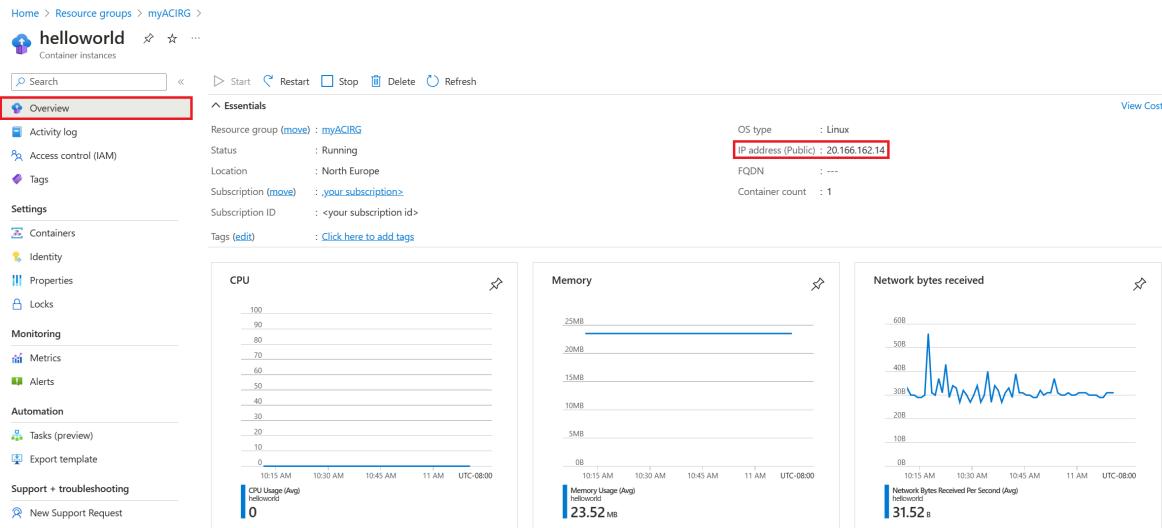
**Go to resource group**

a few seconds ago

# Review deployed resources

In the following steps, you use the Azure portal to review the properties of the container instance. You can also use a tool such as the [Azure CLI](#).

1. In the portal, search for **Container Instances**, and then select the container instance that you created.
2. On the **Overview** page, note the status of the instance and its IP address.



3. When the status of the instance is **Running**, go to the IP address in your browser.

**Welcome to Confidential containers on Azure Container Instances!**



**AMD SEV-SNP Attestation Report:**

```
version: 00000002
guest_svn: 00000002
policy: 000000000003001f
family_id: 00000000000000000000000000000001
image_id: 00000000000000000000000000000002
vmp1: 00000000
signature_algo: 00000001
platform_version: 03000000000000873
platform_info: 0100000000000000
author_key_en: 00000000
reserved1: 00000000
report_data: 00000000000000000000000000000000 00000000000000000000000000000000 00000000000000000000000000000000
measurement: 00000000000000000000000000000000 00000000000000000000000000000000 00000000000000000000000000000000
 2021-01-15T00:00:00Z 21-17:00:11.17149-0-000001-007000
```

The presence of the attestation report below the Azure Container Instances logo confirms that the container is running on hardware that supports a TEE.

If you deploy to hardware that doesn't support a TEE (for example, by choosing a region where Confidential Container Instances isn't available), no attestation report appears.

## Related content

Now that you deployed a confidential container group on Container Instances, you can learn more about how policies are enforced:

- [Confidential containers on Azure Container Instances](#)
  - [Azure CLI confcom extension examples ↗](#)
  - [Confidential Hello World application ↗](#)
- 

## Feedback

Was this page helpful?



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

# Tutorial: Deploy a confidential container to Azure Container Instances via Azure portal

Article • 08/29/2024

In this tutorial, you use Azure portal to deploy a confidential container to Azure Container Instances with a development confidential computing enforcement policy. After deploying the container, you can browse to the running application.

## ⓘ Note

When deploying confidential containers on Azure Container Instances via Portal you will only be able to deploy with a development confidential computing enforcement policy. This policy is only recommended for development and test workloads. Logging, and exec functionality are still available in the container group when using this policy and software components are not validated. To full attest your container group while running production workloads, it is recommended that you deploy with a custom confidential computing enforcement policy via an Azure Resource Manager template. For more information, see the [tutorial](#).

Welcome to Confidential containers on Azure Container Instances!



AMD SEV-SNP Attestation Report:

```
version:
0000002
guest_svn:
0000002
policy:
000000000003001f
family_id:
00000000000000000000000000000001
image_id:
00000000000000000000000000000002
vmpk:
00000000
signature_algo:
00000001
platform_version:
030000000000873
platform_info:
0100000000000000
author_key_en:
00000000
reserved1:
00000000
report_data:
00000000000000000000000000000000 00000000000000000000000000000000
00000000000000000000000000000000 00000000000000000000000000000000
measurement:
00000000000000000000000000000000
```

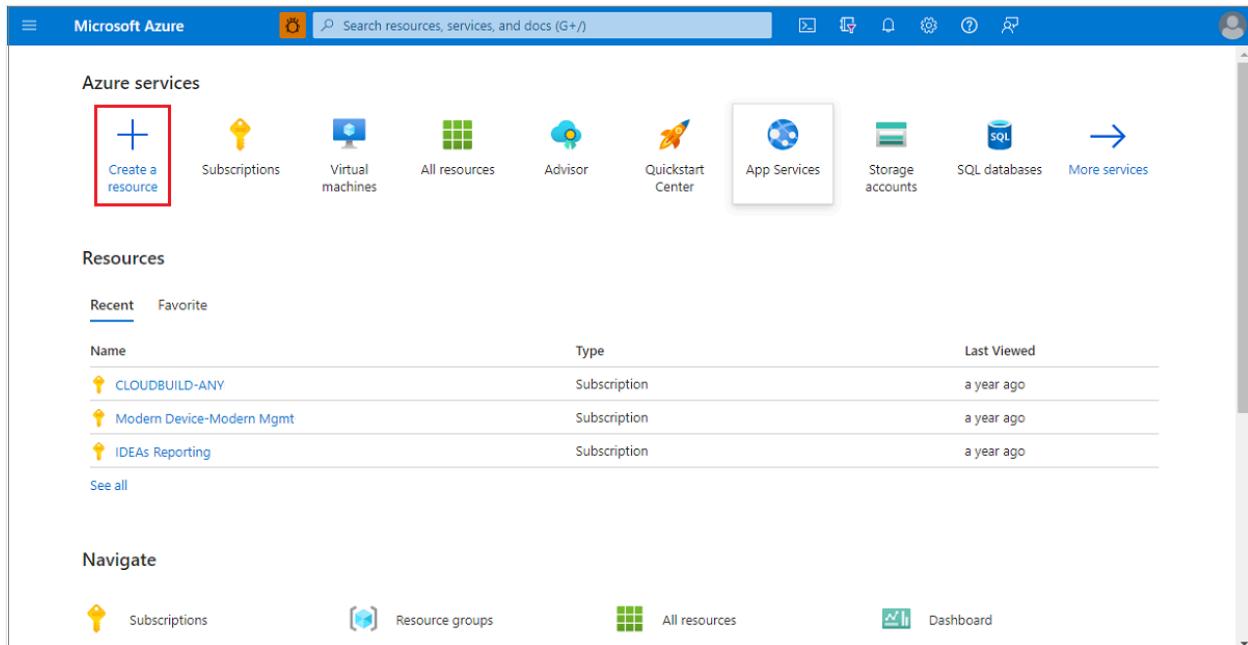
# Sign in to Azure

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

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

## Create a confidential container on Azure Container Instances

On the Azure portal homepage, select **Create a resource**.



The screenshot shows the Microsoft Azure portal interface. At the top, there's a navigation bar with the Microsoft Azure logo, a search bar, and various icons for account management and help. Below the navigation bar, the main content area is titled "Azure services". It features a grid of service icons: "Create a resource" (highlighted with a red box), Subscriptions, Virtual machines, All resources, Advisor, Quickstart Center, App Services, Storage accounts, SQL databases, and More services. Under the "Resources" section, there are tabs for "Recent" and "Favorite", followed by a table listing three recent subscriptions. The table has columns for "Name", "Type", and "Last Viewed". At the bottom, there are links for "See all" and "Navigate" with options like "Subscriptions", "Resource groups", "All resources", and "Dashboard".

| Name                      | Type         | Last Viewed |
|---------------------------|--------------|-------------|
| CLOUDBUILD-ANY            | Subscription | a year ago  |
| Modern Device-Modern Mgmt | Subscription | a year ago  |
| IDEAs Reporting           | Subscription | a year ago  |

Select **Containers > Container Instances**.

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

Home > Create a resource ...

Get Started Search services and marketplace

Recently created

Popular Azure services See more in All services

Categories

- AI + Machine Learning
- Analytics
- Blockchain
- Compute
- Containers** Create | Docs | MS Learn
- Databases
- Developer Tools
- DevOps
- Identity
- Integration
- Internet of Things
- IT & Management Tools
- Media
- Migration
- Mixed Reality
- Monitoring & Diagnostics

**Kubernetes Service** Create | Docs | MS Learn

**Web App for Containers** Create | Docs | MS Learn

**Batch Service** Create | Docs | MS Learn

**Kubernetes - Azure Arc** Create | Learn more

**Container App** Create | Learn more

**Container Instances** Create | Learn more

**Container Registry** Create | Docs | MS Learn

**Service Fabric Cluster** Create | Docs

On the **Basics** page, choose a subscription and enter the following values for **Resource group**, **Container name**, **Image source**, and **Container image**.

- Resource group: **Create new** > `myresourcegroup`
- Container name: `helloworld`
- Region: One of `West Europe / North Europe / East US`
- SKU: `Confidential (development policy)`
- Image source: **Quickstart images**
- Container image: `mcr.microsoft.com/aci/aci-confidential-helloworld:v1 (Linux)`

## Create container instance

**Project details**

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

Subscription \* ⓘ Subscription Name

Resource group \* ⓘ (New) myACIRG    
[Create new](#)

**Container details**

Container name \* ⓘ helloworld

Region \* ⓘ (Europe) North Europe

Availability zones ⓘ None

i No availability zones are available for the SKU you have selected.

SKU Confidential (development policy)

! Container groups deployed from the portal use development policies and are recommended for dev/test workloads. Logging, and exec functionality are still available in the container group when using this policy. For production workloads, it is highly recommended you generate a more restrictive custom policy.  
[Learn more ↗](#)

Image source \* ⓘ  Quickstart images  Container image  Custom image

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

### i Note

When deploying confidential containers on Azure Container Instances via Portal you will only be able to deploy with a development confidential computing enforcement policy. This policy is only recommended for development and test workloads. Logging, and exec functionality are still available in the container group when using this policy and software components are not validated. To fully attest your container group while running production workloads, it is recommended that you deploy with a custom confidential computing enforcement policy via an Azure Resource Manager template. For more information, see the [tutorial](#).

Leave all other settings as their defaults, then select **Review + create**.

When the validation completes, you're shown a summary of the container's settings. Select **Create** to submit your container deployment request.

## Create container instance

...

Validation passed

Basics Networking Advanced Tags **Review + create**

### Basics

|                     |                                                   |
|---------------------|---------------------------------------------------|
| Subscription        | Subscription Name                                 |
| Resource group      | (new) myACIRG                                     |
| Region              | North Europe                                      |
| Container name      | helloworld                                        |
| SKU                 | Confidential (development policy)                 |
| Image type          | Public                                            |
| Image               | mcr.microsoft.com/azuredocs/aci-helloworld:latest |
| OS type             | Linux                                             |
| Memory (GiB)        | 1.5                                               |
| Number of CPU cores | 1                                                 |
| GPU type (preview)  | None                                              |
| GPU count           | 0                                                 |

### Networking

|                            |                      |
|----------------------------|----------------------|
| Networking type            | Public               |
| Ports                      | 80 (TCP)             |
| DNS name label scope reuse | Any reuse (unsecure) |

**Create**

< Previous

Next >

Download a template for automation

When deployment starts, a notification appears that indicates the deployment is in progress. Another notification is displayed when the container group deployed.

Open the overview for the container group by navigating to **Resource Groups > myACIRG > helloworld**. Make a note of the **IP** of the container instance and its **Status**.

1. On the **Overview** page, note the **Status** of the instance and its **IP address**.

Home > Resource groups > myACRG >

**helloworld** Container instances

Search <> Start Restart Stop Delete Refresh

**Overview**

Resource group (move) : myACRG  
Status : Running  
Location : North Europe  
Subscription (move) : <your subscription>  
Subscription ID : <your subscription id>  
Tags (edit) : Click here to add tags

OS type : Linux  
IP address (Public) : 20.166.162.14  
FQDN : ---  
Container count : 1

**CPU**  
CPU Usage (Avg)  
helloworld 0

**Memory**  
Memory Usage (Avg)  
helloworld 23.52 MB

**Network bytes received**  
Network Bytes Received Per Second (Avg)  
helloworld 31.52 B

View Cost

Activity log  
Access control (IAM)  
Tags  
Containers  
Identity  
Properties  
Locks  
Monitoring  
Metrics  
Alerts  
Tasks (preview)  
Export template  
Support + troubleshooting  
New Support Request

2. Once its status is *Running*, navigate to the IP address in your browser.

Welcome to Confidential containers on Azure Container Instances!

AMD SEV-SNP Attestation Report:

```

version: 00000002
guest_svn: 00000002
policy: 000000000003001f
family_id: 00000000000000000000000000000001
image_id: 00000000000000000000000000000002
vmp1: 00000000
signature_algo: 00000001
platform_version: 030000000000873
platform_info: 0100000000000000
author_key_en: 00000000
reserved1: 00000000
report_data: 00000000000000000000000000000000 00000000000000000000000000000000 00000000000000000000000000000000
measurement: 2021-01-15T00:00:00Z-2021-01-15T00:00:00Z

```

The presence of the attestation report below the Azure Container Instances logo confirms that the container is running on hardware that supports a hardware-based and attested trusted execution environment (TEE). If you deploy to hardware that doesn't support a TEE, for example by choosing a region where the [ACI Confidential SKU isn't available](#), no attestation report is shown.

Congratulations! You deployed a confidential container on Azure Container Instances, which displays a hardware attestation report in your browser.

## Clean up resources

When you're done with the container, select **Overview** for the *helloworld* container instance, then select **Delete**.

## Next steps

In this tutorial, you created a confidential container on Azure Container instances with a development confidential computing enforcement policy. If you would like to deploy a confidential container group with a custom computing enforcement policy continue to the confidential containers on Azure Container Instances - deploy with Azure Resource Manager template tutorial.

- [Azure Container Instances Azure Resource Manager template tutorial](#)
  - [Confidential computing enforcement policies overview](#)
  - [Azure CLI confcom extension examples ↗](#)
  - [Confidential Hello World application ↗](#)
- 

## Feedback

Was this page helpful?

 Yes

 No

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

# Tutorial: Deploy a Spot container with Azure Container Instances using the Azure CLI (Preview)

Article • 08/29/2024

Spot Containers combine the simplicity of ACI with the low cost of Spot VMs making it easy and affordable for customers to run containerized interruptible workloads at scale. Use Azure Container Instances to run serverless Spot containers. Deploy an application to a Spot container on-demand when you want to run interruptible, containerized workloads on unused Azure capacity at low cost and you don't need a full container orchestration platform like Azure Kubernetes Service.

In this quickstart, you use the Azure CLI to deploy a helloworld container using Spot containers. A few seconds after you execute a single deployment command, you can browse to the container logs:

- This quickstart requires version 2xxx later of the Azure CLI. If using Azure Cloud Shell, the latest version is already installed.

## Create a resource group

Azure container instances, like all Azure resources, must be deployed into a resource group. Resource groups allow you to organize and manage related Azure resources.

First, create a resource group named *myResourceGroup* in the *westus* location with the following [az group create](#) command:

Azure CLI

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

## Create a container

Now that you have a resource group, you can run a Spot container in Azure. To create a Spot container group with the Azure CLI, provide a resource group name, container instance name, container image, and new property called 'priority' with value of 'Spot' to the [az container create](#) command. In this quickstart, you use the public

`mcr.microsoft.com/azuredocs/aci-helloworld` image. This image packages a small web app written in Node.js that serves a static HTML page.

You can't expose your spot containers to the internet by specifying one or more ports to open, a DNS name label, or both. In this quickstart, you deploy a container using `helloworld` image without a DNS name label. It isn't publicly reachable. You can query the container logs to verify container is listening on default port 80.

Execute a command similar to the following to start a container instance.

Azure CLI

```
az container create --resource-group acispotdemo --name acispotclitest --image mcr.microsoft.com/azuredocs/aci-helloworld --priority spot
```

Within a few seconds, you should get a response from the Azure CLI indicating that the deployment completed. Check its status with the `az container show` command:

Azure CLI

```
az container show --resource-group acispotdemo --name acispotclitest --query "{ProvisioningState:provisioningState}" --out table
```

When you run the command, the container's fully qualified domain name (FQDN) and its provisioning state are displayed.

Output

| ContainerGroupName | ProvisioningState |
|--------------------|-------------------|
| acispotclitest     | Succeeded         |

If the container's `ProvisioningState` is `Succeeded`, congratulations! You successfully deployed an application running in a Docker container to Azure.

## Pull the container logs

When you need to troubleshoot a container or the application it runs (or just see its output), start by viewing the container instance's logs.

Pull the container instance logs with the `az container logs` command:

Azure CLI

```
az container logs --resource-group acispotdemo --name acispotclitest
```

The output displays the logs for the container, and should show the following output

Output

```
listening on port 80
```

## Attach output streams

In addition to viewing the logs, you can attach your local standard out and standard error streams to that of the container.

First, execute the [az container attach](#) command to attach your local console to the container's output streams:

Azure CLI

```
az container attach --resource-group acispotdemo --name acispotclitest
```

Once attached, refresh your browser a few times to generate some more output. When you're done, detach your console with [Control+C](#). You should see output similar to the following example:

Output

```
Container 'acispotclitest' is in state 'Running'...
Start streaming logs:
listening on port 80
```

## Clean up resources

When you're done with the container, remove it using the [az container delete](#) command:

Azure CLI

```
az container delete --resource-group acispotdemo --name acispotclitest
```

To verify that the container deleted, execute the [az container list](#) command:

Azure CLI

```
az container list --resource-group acispotdemo --output table
```

The **acispotclitest** container shouldn't appear in the command's output. If you have no other containers in the resource group, no output is displayed.

If you're done with the *acispotdemo* resource group and all the resources it contains, delete it with the [az group delete](#) command:

Azure CLI

```
az group delete --name acispotdemo
```

## Next steps

In this tutorial, you created a Spot container on Azure Container Instances with a default quota and eviction policy using the Azure CLI.

- [Check out the overview for ACI Spot containers](#)
- [Try out Spot containers with Azure Container Instances using the Azure portal](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# Tutorial: Deploy a Spot container with Azure Container Instances using the Azure portal (Preview)

Article • 08/29/2024

In this tutorial, you use Azure portal to deploy a spot container to Azure Container Instances with a default quota. After deploying the container, you can browse to the running application.

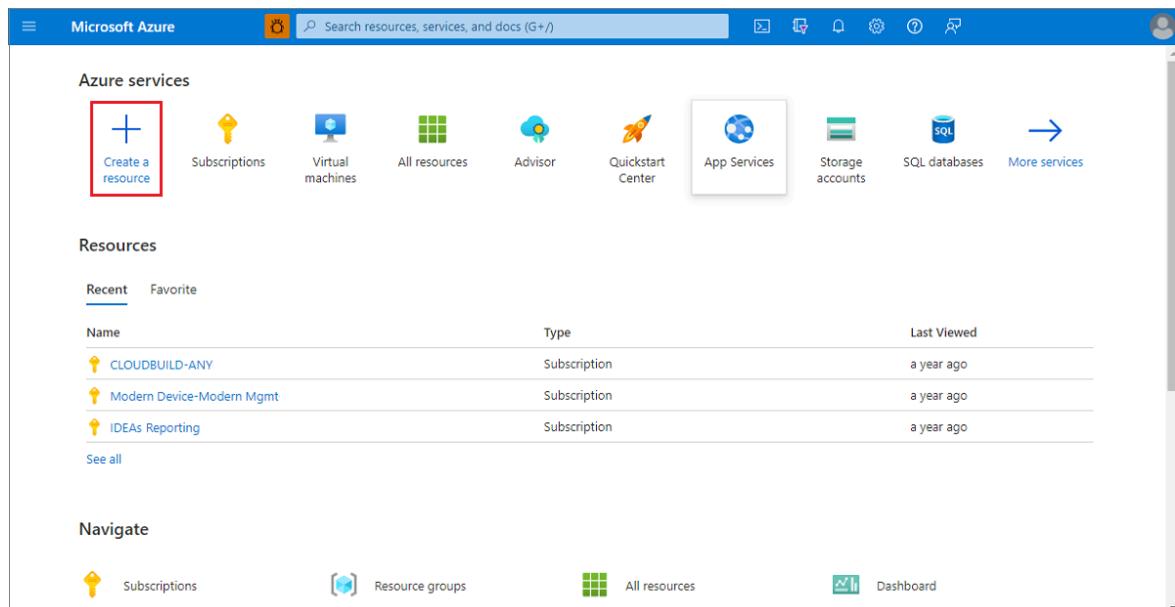
## Sign in to Azure

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

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

## Create a Spot container on Azure Container Instances

1. On the Azure portal homepage, select **Create a resource**.



2. Select **Containers > Container Instances**.

3. On the **Basics** page, choose a subscription and enter the following values for **Resource group**, **Container name**, **Image source**, and **Container image**. Then to deploy ACI Spot container, opt for Spot discount by selecting **Run with Spot**

**discount** field. This enforces limitations for this feature in preview release automatically and allow you to deploy only in supported regions.

- Resource group: **Create new** > `acispotdemo`
- Container name: `acispotportaldemo`
- Region: One of `West Europe / East US2 / West US`
- SKU: `Standard`
- Image source: **Quickstart images**
- Container image: `mcr.microsoft.com/azuredocs/aci-helloworld:v1` (Linux)

**Create container instance** ...

Learn more about Azure Container Instances

**Project details**

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

Subscription \* ⓘ

Resource group \* ⓘ

**Container details**

Container name \* ⓘ

Region \* ⓘ

Availability zones ⓘ

i Spot containers do not support availability zones.

SKU

Image source \* ⓘ  Quickstart images  
 Azure Container Registry  
 Other registry

Run with Azure Spot discount

Image \* ⓘ

Size \* ⓘ

When deploying Spot container on Azure Container Instances, you need to select only regions supported in public preview. You can change the restart policy, region, type of container images and compute resources. If you want more than default quota offered, file a support request.

4. Leave all other settings as their defaults, then select **Review + create**.
5. When the validation completes, you're shown a summary of the container's settings. Select **Create** to submit your container deployment request. When the deployment starts, a notification appears that indicates the deployment is in progress. Another notification is displayed when the container group deploys.
6. Open the overview for the container group by navigating to **Resource Groups > acispotdemo > acispotportaldemo**. Make a note of the **priority** property of the container instance and its **Status**.
7. On the **Overview** page, note the **Status** of the instance.
8. Once its status is *Running*, navigate to the AZ CLI and run the following command to check you're able to listen to container on default port 80.

```
@Azure:~$ az container logs --resource-group acispotdemo --name acispotportaldemo
listening on port 80
```

Congratulations! You deployed a spot container on Azure Container Instances that runs sample hello world container application.

## Clean up resources

When you're done with the container, select **Overview** for the *helloworld* container instance, then select **Delete**.

## Next steps

In this tutorial, you created a Spot container on Azure Container Instances with a default quota and eviction policy using the Azure portal.

- [ACI Spot containers overview](#)
- [Try out Spot containers with Azure Container Instances using the Azure CLI](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# Tutorial: Deploy virtual nodes on Azure Container Instances in your Azure Kubernetes Service cluster

Article • 11/13/2024

In this tutorial, you use Azure portal to deploy an Azure Kubernetes Service (AKS) cluster. After deploying the AKS cluster, you use a Helm chart to deploy virtual nodes on Azure Container Instances.

## Prerequisites

This tutorial assumes you have the following tools installed:

- Azure CLI
- kubectl (version 1.29+)
- Helm
- Git

## Sign in to Azure

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

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

## Create a virtual network for your AKS cluster

On the Azure portal home page, select **Create a resource**.

Microsoft Azure

Search resources, services, and docs (G+)

Azure services

Create a resource

Subscriptions

Virtual machines

All resources

Advisor

Quickstart Center

App Services

Storage accounts

SQL databases

More services

Resources

Recent   Favorite

| Name                     | Type         | Last Viewed |
|--------------------------|--------------|-------------|
| CLOUDBUILD-ANY           | Subscription | a year ago  |
| Modem Device-Modern Mgmt | Subscription | a year ago  |
| IDEAs Reporting          | Subscription | a year ago  |

See all

Navigate

Subscriptions   Resource groups   All resources   Dashboard

The screenshot shows the Microsoft Azure portal interface. At the top, there's a blue header bar with the 'Microsoft Azure' logo, a search bar, and various navigation icons. Below the header, the 'Azure services' section is visible, featuring a 'Create a resource' button (which is highlighted with a red box), 'Subscriptions', 'Virtual machines', 'All resources', 'Advisor', 'Quickstart Center', 'App Services', 'Storage accounts', 'SQL databases', and a 'More services' link. The 'Resources' section follows, with tabs for 'Recent' and 'Favorite'. A table lists three recent subscriptions: 'CLOUDBUILD-ANY', 'Modem Device-Modern Mgmt', and 'IDEAs Reporting', each with a timestamp of 'a year ago'. Below this is a 'See all' link. The 'Navigate' section at the bottom contains links for 'Subscriptions', 'Resource groups', 'All resources', and 'Dashboard', each accompanied by a small icon.

Select **Networking > Virtual network**.

Home >

## Create a resource

...

Get Started



Search services and marketplace

Recently created

Popular Azure services [See more in All services](#)

### Categories

AI + Machine Learning



Public IP address

[Create](#) | [Docs](#)

Analytics



Virtual network

[Create](#) | [Docs](#) | [MS Learn](#)

Blockchain



Local network gateway

[Create](#) | [Docs](#)

Compute



Connection

[Create](#) | [Docs](#)

Databases



Front Door and CDN profiles

[Create](#) | [Docs](#)

Developer Tools



Traffic Manager profile

[Create](#) | [Docs](#) | [MS Learn](#)

DevOps



Network security group

[Create](#) | [Docs](#) | [MS Learn](#)

Identity



Virtual network gateway

[Create](#) | [Docs](#) | [MS Learn](#)

Integration



Application Gateway

[Create](#) | [Docs](#) | [MS Learn](#)

Internet of Things



Load Balancer

[Create](#) | [Docs](#) | [MS Learn](#)

IT & Management Tools

Media

Migration

Mixed Reality

Monitoring & Diagnostics

**Networking**

Security



Load Balancer

[Create](#) | [Docs](#) | [MS Learn](#)

Storage

Web

On the **Basics** page, choose a subscription and enter the following values:

- Resource group: **Create new** > `virtualnodesresourcegroup`
- Virtual network name: `myvirtualnetwork`
- Region: `East US`

# Create virtual network

Basics   Security   IP addresses   Tags   Review + create

Azure Virtual Network (VNet) is the fundamental building block for your private network in Azure. VNet enables many types of Azure resources, such as Azure Virtual Machines (VM), to securely communicate with each other, the internet, and on-premises networks. VNet is similar to a traditional network that you'd operate in your own data center, but brings with it additional benefits of Azure's infrastructure such as scale, availability, and isolation.

[Learn more.](#)

## Project details

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

|                  |                                 |
|------------------|---------------------------------|
| Subscription *   | ACI Development                 |
| Resource group * | (New) virtualnodesresourcegroup |
|                  | <a href="#">Create new</a>      |

## Instance details

|                        |                                                  |
|------------------------|--------------------------------------------------|
| Virtual network name * | myvirtualnetwork                                 |
| Region * ⓘ             | (US) East US                                     |
|                        | <a href="#">Deploy to an Azure Extended Zone</a> |

On the **IP addresses** page, configure the following address spaces and subnets:

- 10.0.0.0/16: `default` subnet with size /24
- 10.1.0.0/16: `aks` subnet with size /16
- 10.2.0.0/16: `cg` subnet with size /16
  - The `cg` subnet must have a subnet delegation to `Microsoft.ContainerInstance/containerGroups`.
  - If your pods running in virtual nodes need outbound networking, you **must** set up a NAT gateway and configure the `cg` subnet to use it. Refer to [Configure a NAT gateway for static IP address for outbound traffic from a container group](#) for instructions on this setup.

## Create virtual network ...

Basics Security IP addresses Tags Review + create

10.0.0.0/16 Delete address space

This address prefix overlaps with virtual network 'ACICloudServiceVnet'. If you intend to peer these virtual networks, change the address space. [Learn more ↗](#)

10.0.0.0 /16  
10.0.0.0 - 10.0.255.255 65,536 addresses

| Subnets | IP address range | Size | NAT gateway |
|---------|------------------|------|-------------|
|---------|------------------|------|-------------|

|         |                       |                     |   |
|---------|-----------------------|---------------------|---|
| default | 10.0.0.0 - 10.0.0.255 | /24 (256 addresses) | - |
|---------|-----------------------|---------------------|---|



10.1.0.0/16 Delete address space

This address prefix overlaps with virtual network 'aksresourcegroup-tscuiu-vnet'. If you intend to peer these virtual networks, change the address space. [Learn more ↗](#)

10.1.0.0 /16  
10.1.0.0 - 10.1.255.255 65,536 addresses

| Subnets | IP address range | Size | NAT gateway |
|---------|------------------|------|-------------|
|---------|------------------|------|-------------|

|     |                         |                        |   |
|-----|-------------------------|------------------------|---|
| aks | 10.1.0.0 - 10.1.255.255 | /16 (65,536 addresses) | - |
|-----|-------------------------|------------------------|---|



10.2.0.0/16 Delete address space

This address prefix overlaps with virtual network 'aks-vnet-28778450'. If you intend to peer these virtual networks, change the address space. [Learn more ↗](#)

10.2.0.0 /16  
10.2.0.0 - 10.2.255.255 65,536 addresses

| Subnets | IP address range | Size | NAT gateway |
|---------|------------------|------|-------------|
|---------|------------------|------|-------------|

|    |                         |                        |   |
|----|-------------------------|------------------------|---|
| cg | 10.2.0.0 - 10.2.255.255 | /16 (65,536 addresses) | - |
|----|-------------------------|------------------------|---|



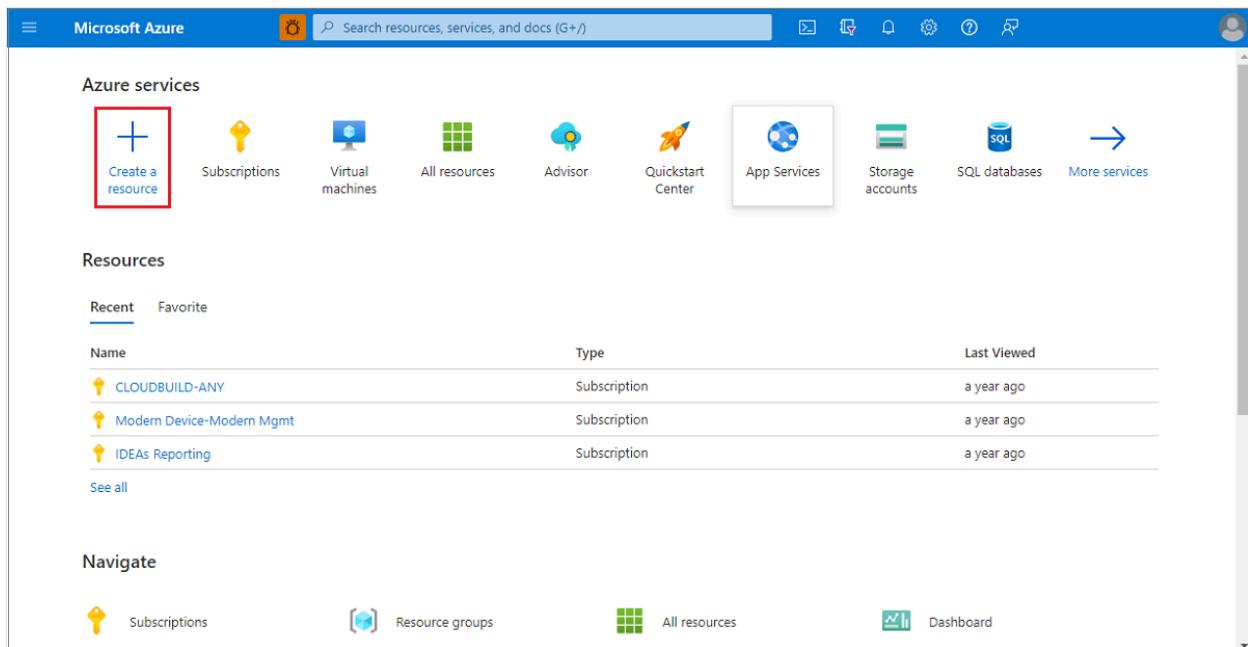
Leave all other settings as their defaults, then select **Review + create**.

When the validation completes, you're shown a summary of the virtual network's settings. Select **Create** to submit your virtual network deployment request.

When deployment starts, a notification appears that indicates the deployment is in progress. Another notification is displayed when the virtual network has deployed.

# Create your AKS cluster

On the Azure portal home page, select **Create a resource**.



The screenshot shows the Microsoft Azure portal interface. At the top, there's a navigation bar with the 'Microsoft Azure' logo, a search bar, and various icons for account management and help. Below the navigation bar, there's a section titled 'Azure services' containing several service icons: 'Create a resource' (highlighted with a red box), 'Subscriptions', 'Virtual machines', 'All resources', 'Advisor', 'Quickstart Center', 'App Services' (highlighted with a white box), 'Storage accounts', 'SQL databases', and 'More services'. Underneath this, there's a 'Resources' section with tabs for 'Recent' (selected) and 'Favorite'. It lists three recent subscriptions: 'CLOUDBUILD-ANY', 'Modern Device-Modern Mgmt', and 'IDEAs Reporting', each with its type (Subscription) and last viewed date (a year ago). Below this is a 'See all' link. At the bottom, there's a 'Navigate' section with links for 'Subscriptions', 'Resource groups', 'All resources' (highlighted with a white box), and 'Dashboard'.

Select **Containers > Azure Kubernetes Service (AKS)**.

# Create a resource

...

Get Started

 Search services and marketplace

Recently created

Popular Azure services [See more in All services](#)

## Categories

AI + Machine Learning

 Azure Kubernetes Service (AKS)  
[Create](#) | [Docs](#) | [MS Learn](#)

Analytics

 Web App for Containers  
[Create](#) | [Docs](#) | [MS Learn](#)

Blockchain

Compute

Containers

 Batch Service  
[Create](#) | [Docs](#) | [MS Learn](#)

Databases

 Kubernetes - Azure Arc  
[Create](#) | [Docs](#) | [MS Learn](#)

Developer Tools

 Container App  
[Create](#) | [Docs](#)

DevOps

Identity

Integration

 Container Instances  
[Create](#) | [MS Learn](#)

Internet of Things

On the **Basics** page, choose the same subscription you used for creating the virtual network and enter the following values:

- Resource group: `virtualnodesresourcegroup`
- Cluster preset configuration: `Dev/Test`
- Kubernetes cluster name: `virtualnodescluster`
- Region: `East US`
- Kubernetes version: Any version starting with `1.29.` - for example, `1.29.10`
- Automatic upgrade: `Enabled with patch`

## Create Kubernetes cluster

...

Basics

Node pools

Networking

Integrations

Monitoring

Security

Advanced

Tags

Review + create

### Project details

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

Subscription \* ⓘ

ACI Development

Resource group \* ⓘ

virtualnodesresourcegroup

[Create new](#)

### Cluster details

Cluster preset configuration \*

Dev/Test

To quickly customize your Kubernetes cluster, choose one of the preset configurations above. You can modify these configurations at any time.  
[Compare presets](#)

Kubernetes cluster name \* ⓘ

virtualnodescluster

Region \* ⓘ

(US) East US

Availability zones ⓘ

None

AKS pricing tier ⓘ

Free

Kubernetes version \* ⓘ

1.29.10 (default)

Automatic upgrade ⓘ

Enabled with patch (recommended)

Automatic upgrade scheduler

**Start on:** Tue Nov 12 2024 00:00 +00:00 (Coordinated Universal Time)

[Edit schedule](#)

Node security channel type ⓘ

Node Image

Security channel scheduler

Every week on Sunday (recommended)

On the **Node pools** page, select any VM SKU with at least 4 vCPUs and 16 GB RAM for the `agentpool` node pool. These nodes will be used to host the virtual nodes infrastructure. Do not select `Enable virtual nodes` - this refers to the previous [virtual nodes](#) offering for AKS.

## Create Kubernetes cluster

...

Basics

**Node pools**

Networking

Integrations

Monitoring

Security

Advanced

Tags

Review + create

### Node pools

In addition to the required primary node pool configured on the Basics tab, you can also add optional node pools to handle a variety of workloads [Learn more](#)

Add node pool Delete

| <input type="checkbox"/> | Name      | Mode   | Node size              | OS SKU | Node count | Availability zone |
|--------------------------|-----------|--------|------------------------|--------|------------|-------------------|
| <input type="checkbox"/> | agentpool | System | Standard_D4s_v3 (c...) | Ubuntu | 2 - 5      | None              |

### Enable virtual nodes

Virtual nodes allow burstable scaling backed by serverless Azure Container Instances. [Learn more](#)

Enable virtual nodes

### Node pool OS disk encryption

By default, all disks in AKS are encrypted at rest with Microsoft-managed keys. For additional control over encryption, you can supply your own keys using a disk encryption set backed by an Azure Key Vault. The disk encryption set will be used to encrypt the OS disks for all node pools in the cluster. [Learn more](#)

Encryption type

(Default) Encryption at-rest with a platform-managed key



On the **Networking** page, enter the following values:

- Network configuration: `Azure CNI Node Subnet`
- Bring your own Azure virtual network: `Enabled`
- Virtual network: `myvirtualnetwork`
- Cluster subnet: `aks`
- Kubernetes service address range: `10.4.0.0/16`
- Kubernetes DNS service IP address: `10.4.0.10`
- Network policy: `Calico`

## Create Kubernetes cluster

...

Basics    Node pools    **Networking**    Integrations    Monitoring    Security    Advanced    Tags    Review + create

**Container networking**

Network configuration ⓘ

Azure CNI Overlay  
Assigns pod IP addresses from a private IP space. Best for scalability

Azure CNI Node Subnet  
Previously named Azure CNI. Assigns pod IP addresses from your host VNet. Best for workloads where pods must be reachable by other VNet resources

kubenet  
Older, route table-based Overlay with limited scalability. Not recommended for most clusters

(i) High pod values may quickly exhaust available IP addresses. [Learn more](#)

Bring your own Azure virtual network ⓘ

Virtual network \* ⓘ  [Create new](#)

Cluster subnet \* ⓘ  [Manage subnet configuration](#)

Kubernetes service address range \* ⓘ

Kubernetes DNS service IP address \* ⓘ

DNS name prefix \* ⓘ

Network policy \* ⓘ

None  
Allow all ingress and egress traffic to the pods

Calico  
Open-source networking solution. Best for large-scale deployments with strict security requirements

Azure  
Native networking solution. Best for simpler deployments with basic security and networking requirements

Leave all other settings as their defaults, then select **Review + create**.

When the validation completes, you're shown a summary of the AKS cluster's settings. Select **Create** to submit your AKS cluster deployment request.

When deployment starts, a notification appears that indicates the deployment is in progress. Another notification is displayed when the AKS cluster has deployed.

## Assign permissions to the AKS cluster managed identity

When you deploy an AKS cluster, AKS creates a [node resource group](#) in the same subscription to host cluster infrastructure. By default, this resource group has a name like `MC_<resource group>_<AKS cluster>_<region>`. For this tutorial, the node resource group should have the name

`MC_virtualnodesresourcegroup_virtualnodescluster_eastus`.

One of the resources created in the node resource group is a managed identity with the name `virtualnodescluster_agentpool`. Navigate to this managed identity and select **Azure role assignments**. Then add the following two role assignments:

- Scope: `Resource group`
- Subscription: `<your subscription name>`
- Resource group: `MC_virtualnodesresourcegroup_virtualnodescluster_eastus`
- Role: `Contributor`

This role assignment allows the creation of ACI container groups in the node resource group.

- Scope: `Resource group`
- Subscription: `<your subscription name>`
- Resource group: `virtualnodesresourcegroup`
- Role: `Contributor`

This role assignment allows ACI container groups to be injected into the virtual network you created in this resource group.

| Role        | Resource Name                                           | Resource Type  |
|-------------|---------------------------------------------------------|----------------|
| Contributor | MC_virtualnodesresourcegroup_virtualnodescluster_eastus | Resource Group |
| Contributor | virtualnodesresourcegroup                               | Resource Group |

## Install virtual nodes on Azure Container Instances using the Helm chart

Use Azure CLI to pull down the configuration and credentials for the AKS cluster you created. This will configure kubectl for your AKS cluster.

```
az login
```

```
az account set --subscription <your subscription ID>
```

```
az aks get-credentials --name virtualnodescluster --resource-group
virtualnodesresourcegroup
```

Additionally, we will register the ACI resource provider on your subscription so that you can deploy container groups.

```
az provider register -n Microsoft.ContainerInstance
```

Clone the [virtualnodesOnAzureContainerInstances](#) GitHub repository. The Helm chart for installing virtual nodes on Azure Container Instances is located in the `Helm/virtualnode` folder.

Install the Helm chart using the following command:

```
helm install virtualnode <cloned repository location>\Helm\virtualnode
```

Within a minute, a new virtual node will be registered and in a Ready state in the AKS cluster. You can check the status of your AKS cluster nodes with kubectl.

```
kubectl get nodes
```

```
C:\>kubectl get nodes
NAME STATUS ROLES AGE VERSION
aks-agentpool-72535977-vmss000000 Ready <none> 6h29m v1.29.10
aks-agentpool-72535977-vmss000001 Ready <none> 6h29m v1.29.10
virtualnode-0 Ready <none> 2s v1.28.9
```

## Deploy your first pod to a virtual node

You can interact with virtual nodes like any other Kubernetes nodes. For example, the sample YAML below will deploy a pod onto the virtual node in your AKS cluster - note the use of node selectors and tolerations to place the pod appropriately. You can deploy the YAML using [kubectl apply](#).

YAML

```
apiVersion: v1
kind: Pod
metadata:
 annotations:
 name: demo-pod
spec:
 containers:
 - command:
 - /bin/bash
 - -c
 - 'counter=1; while true; do echo "Hello, World! Counter: $counter"; counter=$((counter+1)); sleep 1; done'
 image: mcr.microsoft.com/azure-cli
 name: hello-world-counter
 resources:
```

```
limits:
 cpu: 2250m
 memory: 2256Mi
requests:
 cpu: 100m
 memory: 128Mi
nodeSelector:
 virtualization: virtualnode2
tolerations:
- effect: NoSchedule
 key: virtual-kubelet.io/provider
 operator: Exists
```

After your pod is deployed, you can interact with it exactly as you would a "normal" Kubernetes pod.

For example, to see the pod status and events (useful for finding errors!):

```
kubectl describe pods demo-pod
```

To view logs for the pod:

```
kubectl logs demo-pod
```

To get a shell for the pod:

```
kubectl exec demo-pod -it -- /bin/bash
```

## Next steps

In this tutorial, you created an AKS cluster in Azure portal, deployed virtual nodes on Azure Container Instances using a Helm chart, and deployed a pod onto your virtual node. Now you have basic familiarity with how to run Kubernetes pods on your virtual node, and you can utilize most Kubernetes capabilities and constructs on those pods out of the box!

If you are looking to utilize specialized virtual node capabilities or behaviors for your pods, see [Pod Customizations](#).

If you are looking to customize your virtual node installation, see [Node Customizations](#).

If you are planning to deploy high-scale workloads (thousands of pods per minute) on virtual nodes, we have [best practices and recommendations](#).

# Feedback

Was this page helpful?

 Yes

 No

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

# Azure Resource Manager templates for Azure Container Instances

Article • 08/29/2024

The following sample templates deploy container instances in various configurations.

For deployment options, see the [Deployment](#) section. If you'd like to create your own templates, the Azure Container Instances [Resource Manager template reference](#) details template format and available properties.

## Sample templates

### Important

[Network profiles](#) have been retired as of the 2021-07-01 API version. If you're using this or a more recent version, ignore any steps and actions related to network profiles.

 Expand table

| Template                                       | Description                                                                                                                                                                                                                                     |
|------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Applications                                   |                                                                                                                                                                                                                                                 |
| <a href="#">WordPress</a>                      | Creates a WordPress website and its MySQL database in a container group. The WordPress site content and MySQL database are persisted to an Azure Files share. Also creates an application gateway to expose public network access to WordPress. |
| <a href="#">MS NAV with SQL Server and IIS</a> | Deploys a single Windows container with a fully featured self-contained Dynamics NAV / Dynamics 365 Business Central environment.                                                                                                               |
| Volumes                                        |                                                                                                                                                                                                                                                 |
| <a href="#">emptyDir</a>                       | Deploys two Linux containers that share an emptyDir volume.                                                                                                                                                                                     |
| <a href="#">gitRepo</a>                        | Deploys a Linux container that clones a GitHub repo and mounts it as a volume.                                                                                                                                                                  |
| <a href="#">secret</a>                         | Deploys a Linux container with a PFX cert mounted as a secret volume.                                                                                                                                                                           |
| Networking                                     |                                                                                                                                                                                                                                                 |

| Template                                                          | Description                                                                                      |
|-------------------------------------------------------------------|--------------------------------------------------------------------------------------------------|
| <a href="#">UDP-exposed container</a> ↗                           | Deploys a Windows or Linux container that exposes a UDP port.                                    |
| <a href="#">Linux container with public IP</a> ↗                  | Deploys a single Linux container accessible via a public IP.                                     |
| <a href="#">Deploy a container group with a virtual network</a> ↗ | Deploys a new virtual network, subnet, network profile, and container group.                     |
| <b>Azure resources</b>                                            |                                                                                                  |
| <a href="#">Create Azure Storage account and Files share</a> ↗    | Uses the Azure CLI in a container instance to create a storage account and an Azure Files share. |

## Deployment

You have several options for deploying resources with Resource Manager templates:

[Azure CLI](#)

[Azure PowerShell](#)

[Azure portal](#)

[REST API](#)

## Feedback

Was this page helpful?

 Yes

 No

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

# Best practices and considerations for Azure Container Instances

Article • 08/29/2024

Azure Container Instances (ACI) allow you to package, deploy, and manage cloud applications without having to manage the underlying infrastructure. Common scenarios that run on ACI include burst workloads, task automation, and build jobs. You can use ACI by defining the resources they need per container group, including vCPU and memory. ACI is a great solution for any scenario that can operate in isolated container. ACI provides fast start-up times, hyper-visord level security, custom container sizes, and more. The following information helps you determine if Azure Container Instances is best for your scenario.

## What to consider

User's credentials passed via command line interface (CLI) are stored as plain text in the backend. Storing credentials in plain text is a security risk; Microsoft advises customers to store user credentials in CLI environment variables to ensure they're encrypted/transformed when stored in the backend.

There are default limits that may require quota increases. For more details: [Resource availability & quota limits for ACI - Azure Container Instances | Microsoft Learn](#)

Container Images can't be larger than 15 GB, any images above this size may cause unexpected behavior: [How large can my container image be?](#)

If your container image is larger than 15 GB, you can [mount an Azure Fileshare](#) to store the image.

If a container group restarts, the container group's IP may change. We advise against using a hard coded IP address in your scenario. If you need a static public IP address, use Application Gateway: [Static IP address for container group - Azure Container Instances | Microsoft Learn](#).

There are ports that are reserved for service functionality. We advise you not to use these ports, because their use leads to unexpected behavior: [Does the ACI service reserve ports for service functionality?](#)

Your container groups may restart due to platform maintenance events. These maintenance events are done to ensure the continuous improvement of the underlying infrastructure: [Container had an isolated restart without explicit user input.](#)

ACI doesn't allow [privileged container operations](#). We advise you not to depend on using the root directory for your scenario

## Best practices

We advise running container groups in multiple regions so your workloads can continue to run if there's an issue in one region.

We advise against using a hard coded IP address in your scenario since a container group's IP address isn't guaranteed. To mitigate connectivity issues, we recommend configuring a gateway. If your container is behind a public IP address and you need a static public IP address, use [Application Gateway](#). If your container is behind a virtual network and you need a static IP address, we recommend using [NAT Gateway](#).

## Other Azure Container options

### Azure Container Apps

Azure Container Apps enables you to build serverless microservices based on containers. 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 may prefer to start building container microservices with Azure Container Apps.

### Azure App Service

Azure App Service provides fully managed hosting for web applications including websites and web APIs. These web applications may be deployed 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. If you plan to build 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. If you plan to build 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

Red Hat and Microsoft jointly engineers, operates, and supports Azure Red Hat OpenShift to provide 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, or 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

Learn how to deploy a multi-container container group with an Azure Resource Manager template:

[Deploy a container group](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# Container groups in Azure Container Instances

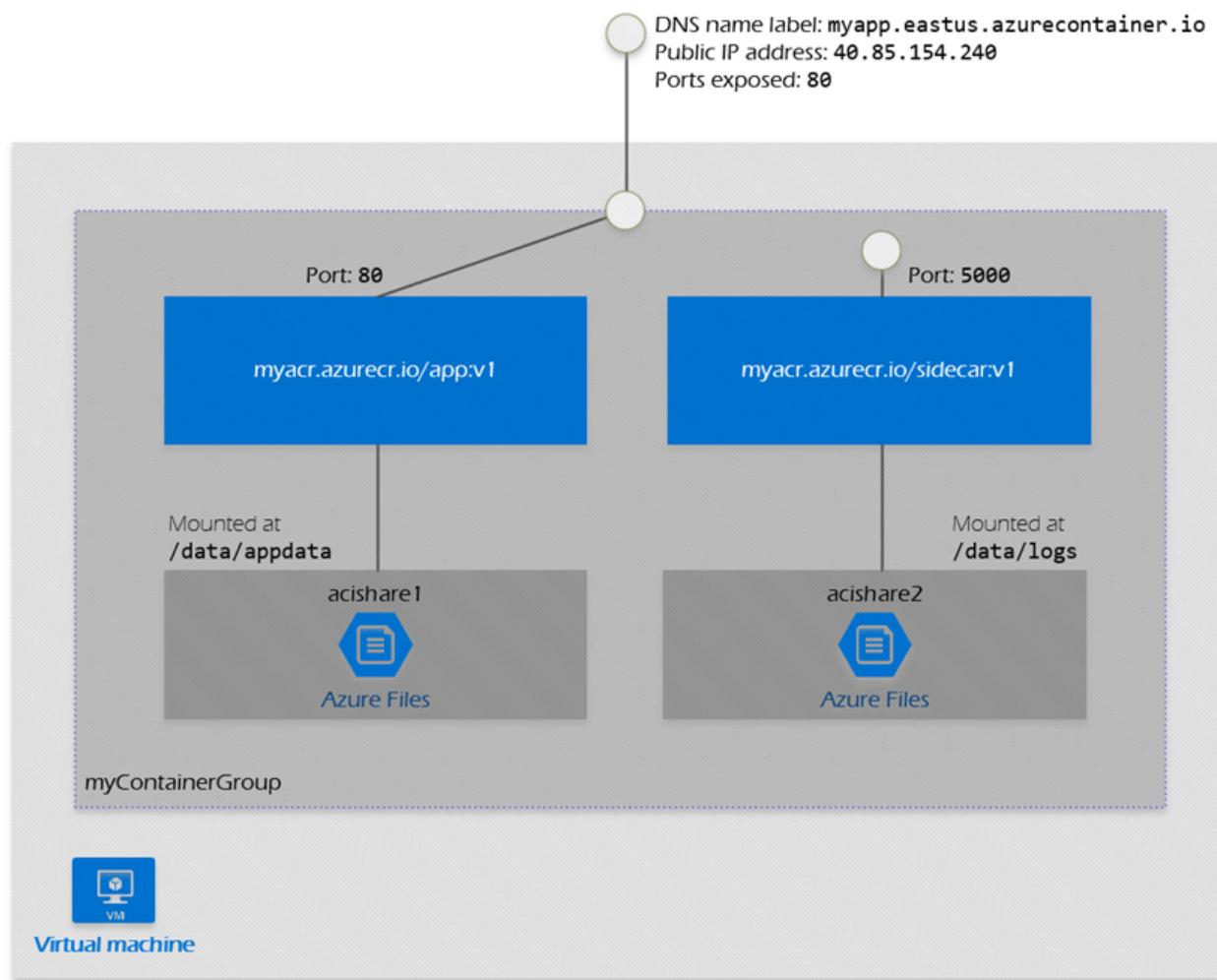
Article • 08/29/2024

The top-level resource in Azure Container Instances is the *container group*. This article describes what container groups are and the types of scenarios they enable.

## What is a container group?

A container group is a collection of containers that get scheduled on the same host machine. The containers in a container group share a lifecycle, resources, local network, and storage volumes. It's similar in concept to a *pod* in [Kubernetes](#).

The following diagram shows an example of a container group that includes multiple containers:



This example container group:

- Is scheduled on a single host machine.

- Is assigned a DNS name label.
- Exposes a single public IP address, with one exposed port.
- Consists of two containers. One container listens on port 80, while the other listens on port 5000.
- Includes two Azure file shares as volume mounts, and each container mounts one of the shares locally.

### Note

Multi-container groups currently support only Linux containers. For Windows containers, Azure Container Instances only supports deployment of a single container instance. While we are working to bring all features to Windows containers, you can find current platform differences in the service [Overview](#).

## Deployment

Here are two common ways to deploy a multi-container group: use a [Resource Manager template](#) or a [YAML file](#). A Resource Manager template is recommended when you need to deploy other Azure service resources (for example, an [Azure Files share](#)) when you deploy the container instances. Due to the YAML format's more concise nature, a YAML file is recommended when your deployment includes only container instances. For details on properties you can set, see the [Resource Manager template reference](#) or [YAML reference](#) documentation.

To preserve a container group's configuration, you can export the configuration to a YAML file by using the Azure CLI command [az container export](#). Export allows you to store your container group configurations in version control for "configuration as code." Or, use the exported file as a starting point when developing a new configuration in YAML.

## Resource allocation

Azure Container Instances allocates resources such as CPUs, memory, and optionally [GPUs](#) (preview) to a multi-container group by adding the [resource requests](#) of the instances in the group. Taking CPU resources as an example, if you create a container group with two container instances, each requesting one CPU, then the container group is allocated 2 CPUs.

## Resource usage by container instances

Each container instance in a group is allocated the resources specified in its resource request. However, the maximum resources used by a container instance in a group could be different if you configure its optional [resource limit](#) property. The resource limit of a container instance must be greater than or equal to the mandatory [resource request](#) property.

- If you don't specify a resource limit, the container instance's maximum resource usage is the same as its resource request.
- If you specify a limit for a container instance, the instance's maximum usage could be greater than the request, up to the limit you set. Correspondingly, resource usage by other container instances in the group could decrease. The maximum resource limit you can set for a container instance is the total resources allocated to the group.

For example, in a group with two container instances each requesting one CPU, one of your containers might run a workload that requires more CPUs to run than the other.

In this scenario, you could set a resource limit of up to 2 CPUs for the container instance. This configuration allows the container instance to use up to 2 CPUs if available.

#### Note

A small amount of a container group's resources is used by the service's underlying infrastructure. Your containers will be able to access most but not all of the resources allocated to the group. For this reason, plan a small resource buffer when requesting resources for containers in the group.

## Minimum and maximum allocation

- Allocate a **minimum** of 1 CPU and 1 GB of memory to a container group. Individual container instances within a group can be provisioned with less than one CPU and 1 GB of memory.
- For the **maximum** resources in a container group, see the [resource availability](#) for Azure Container Instances in the deployment region.

## Networking

Container groups can share an external-facing IP address, one or more ports on that IP address, and a DNS label with a fully qualified domain name (FQDN). To enable external

clients to reach a container within the group, you must expose the port on the IP address and from the container. A container group's IP address and FQDN are released when the container group is deleted.

Within a container group, container instances can reach each other via localhost on any port, even if those ports aren't exposed externally on the group's IP address or from the container.

Optionally deploy container groups into an [Azure virtual network](#) to allow containers to communicate securely with other resources in the virtual network.

## Storage

You can specify external volumes to mount within a container group. Supported volumes include:

- [Azure file share](#)
- [Secret](#)
- [Empty directory](#)
- [Cloned git repo](#)

You can map those volumes into specific paths within the individual containers in a group.

## Common scenarios

Multi-container groups are useful in cases where you want to divide a single functional task into a few container images. These images can have separate resource requirements, and different teams can deliver them.

Example usage could include:

- A container serving a web application and a container pulling the latest content from source control.
- An application container and a logging container. The logging container collects the logs and metrics output by the main application and writes them to long-term storage.
- An application container and a monitoring container. The monitoring container periodically makes a request to the application to ensure that it's running and responding correctly, and raises an alert if it's not.
- A front-end container and a back-end container. The front end might serve a web application, with the back end running a service to retrieve data.

# Next steps

Learn how to deploy a multi-container container group with an Azure Resource Manager template:

[Deploy a container group](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# Azure Container Instances and container orchestrators

Article • 11/13/2024

Because of their small size and application orientation, containers are well suited for agile delivery environments and microservice-based architectures. The task of automating and managing a large number of containers and how they interact is known as *orchestration*. Popular container orchestrators include Kubernetes, DC/OS, and Docker Swarm.

Azure Container Instances provides some of the basic scheduling capabilities of orchestration platforms. And while it doesn't cover the higher-value services that those platforms provide, Azure Container Instances can be complementary to them. This article describes the scope of what Azure Container Instances handles, and how full container orchestrators might interact with it.

## Traditional orchestration

The standard definition of orchestration includes the following tasks:

- **Scheduling:** Given a container image and a resource request, find a suitable machine on which to run the container.
- **Affinity/Anti-affinity:** Specify that a set of containers should run nearby each other (for performance) or sufficiently far apart (for availability).
- **Health monitoring:** Watch for container failures and automatically reschedule them.
- **Failover:** Keep track of what is running on each machine, and reschedule containers from failed machines to healthy nodes.
- **Scaling:** Add or remove container instances to match demand, either manually or automatically.
- **Networking:** Provide an overlay network for coordinating containers to communicate across multiple host machines.
- **Service discovery:** Enable containers to locate each other automatically, even as they move between host machines and change IP addresses.
- **Coordinated application upgrades:** Manage container upgrades to avoid application downtime, and enable rollback if something goes wrong.

# Orchestration with Azure Container Instances: A layered approach

Azure Container Instances enables a layered approach to orchestration, providing all of the scheduling and management capabilities required to run a single container, while allowing orchestrator platforms to manage multi-container tasks on top of it.

Because Azure manages the underlying infrastructure for container instances, an orchestrator platform doesn't need to concern itself with finding an appropriate host machine on which to run a single container. The elasticity of the cloud ensures that one is always available. Instead, the orchestrator can focus on the tasks that simplify the development of multi-container architectures, including scaling and coordinated upgrades.

## Scenarios

While orchestrator integration with Azure Container Instances is still nascent, we anticipate that a few different environments emerge:

### Orchestration of container instances exclusively

Because they start quickly and bill by the second, an environment based exclusively on Azure Container Instances offers the fastest way to get started and to deal with highly variable workloads.

### Combination of container instances and containers in Virtual Machines

For long-running, stable workloads, orchestrating containers in a cluster of dedicated virtual machines is typically cheaper than running the same containers with Azure Container Instances. However, container instances offer a great solution for quickly expanding and contracting your overall capacity to deal with unexpected or short-lived spikes in usage.

Rather than scaling out the number of virtual machines in your cluster, then deploying more containers onto those machines, the orchestrator can schedule the additional containers in Azure Container Instances, and delete them when they're no longer needed.

# Sample implementation: virtual nodes on Azure Container Instances

To rapidly scale application workloads in an [Azure Kubernetes Service \(AKS\)](#) cluster, you can use *virtual nodes* created dynamically in Azure Container Instances. Virtual nodes register as nodes having unlimited capacity with your AKS cluster control plane. When you deploy pods in a virtual node in your AKS cluster, they run as container groups in ACI.

Virtual nodes currently support Linux container instances. See [Virtual nodes on Azure Container Instances](#) to learn more.

## Next steps

Create your first container with Azure Container Instances using the [quickstart guide](#).

---

## Feedback

Was this page helpful?

 Yes

 No

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

# Big Containers on Azure Container Instances (Preview)

Article • 03/27/2025

This article outlines the capabilities and benefits of Big Containers on Azure Container Instances. Customers can now deploy workloads with higher vCPU and memory for standard containers, confidential containers, containers with virtual networks, as well as containers utilizing virtual nodes to connect to AKS. This setup supports vCPU counts greater than 4 and memory capacities of 16 GB, with a maximum of 32 vCPU and 256 GB per standard container group and 32 vCPU and 192 GB per confidential container group. This feature removes limitations for compute and memory intensive workloads!

## Benefits of Big Containers

### Enhanced Performance

More vCPUs mean better processing power, allowing for more efficient handling of complex tasks and applications. The enhanced performance from more vCPUs and larger GB capacity can lead to faster processing times and reduced latency, which can translate to cost savings in terms of time and productivity.

### Increased Memory Capacity

Larger container groups with more GB can handle bigger datasets and more extensive workloads, making them ideal for data-intensive applications.

### Simplified Scalability

Larger container groups provide the flexibility to scale up resources even higher as needed, accommodating growing business demands without compromising performance. Larger container SKUs can simplify the scaling process. Instead of managing many smaller containers, you can scale your applications with fewer, larger ones, potentially reducing the need for frequent scaling adjustments.

## Scenarios for Big Containers

These are a few scenarios that will benefit from Big Containers.

## Data Inferencing

Larger container SKUs are ideal for data inferencing tasks that require robust computational power. Examples include real-time fraud detection in financial transactions, predictive maintenance in manufacturing, and personalized recommendation engines in e-commerce. These containers ensure efficient and secure processing of large datasets for accurate predictions and insights.

## Collaborative Analytics

When multiple parties need to share and analyze data, larger container SKUs provide a secure and efficient solution. For instance, companies in healthcare can collaborate on patient data analytics while maintaining confidentiality. Similarly, research institutions can share large datasets for scientific studies without compromising data privacy.

## Big Data Processing

Organizations dealing with large-scale data processing can benefit from the enhanced capacity of larger container SKUs. Examples include processing customer data for targeted marketing campaigns, analyzing social media trends for sentiment analysis, and conducting large-scale financial modeling for risk assessment. These containers ensure efficient handling of extensive workloads.

## High-Performance Computing

High-performance computing applications, such as climate modeling, genomic research, and computational fluid dynamics, demand substantial computational power. Larger container SKUs provide the necessary resources to support these intensive tasks, enabling precise simulations and faster results.

## Next Steps

To begin using Big Containers, follow these steps.

1. If you plan to run containers larger than 4 vCPU and 16 GB, you must submit an [Azure support request][azure-support] (select "Quota" for **Support type**).
2. Once your quota has been allocated, you can deploy your container groups through Azure portal, Azure CLI, PowerShell, ARM template, or any other medium that allows you to connect to your container groups in Azure.

3. Please note that since Big Containers are in preview, you may receive errors. It is not recommended to run critical workloads on preview features.

To learn more about Azure Container Instances, see [Serverless containers in Azure - Azure Container Instances | Microsoft Learn](#)

---

## Feedback

Was this page helpful?



Yes



No

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

# About NGroups (Preview)

Article • 11/14/2024

Containers became the standard for packaging, deploying, and managing cloud applications, and effectively managing these containers is as crucial as running the apps themselves. [Azure Container Instances](#) (ACI) is a flexible and scalable serverless computing service that allows you to run containerized applications without managing infrastructure.

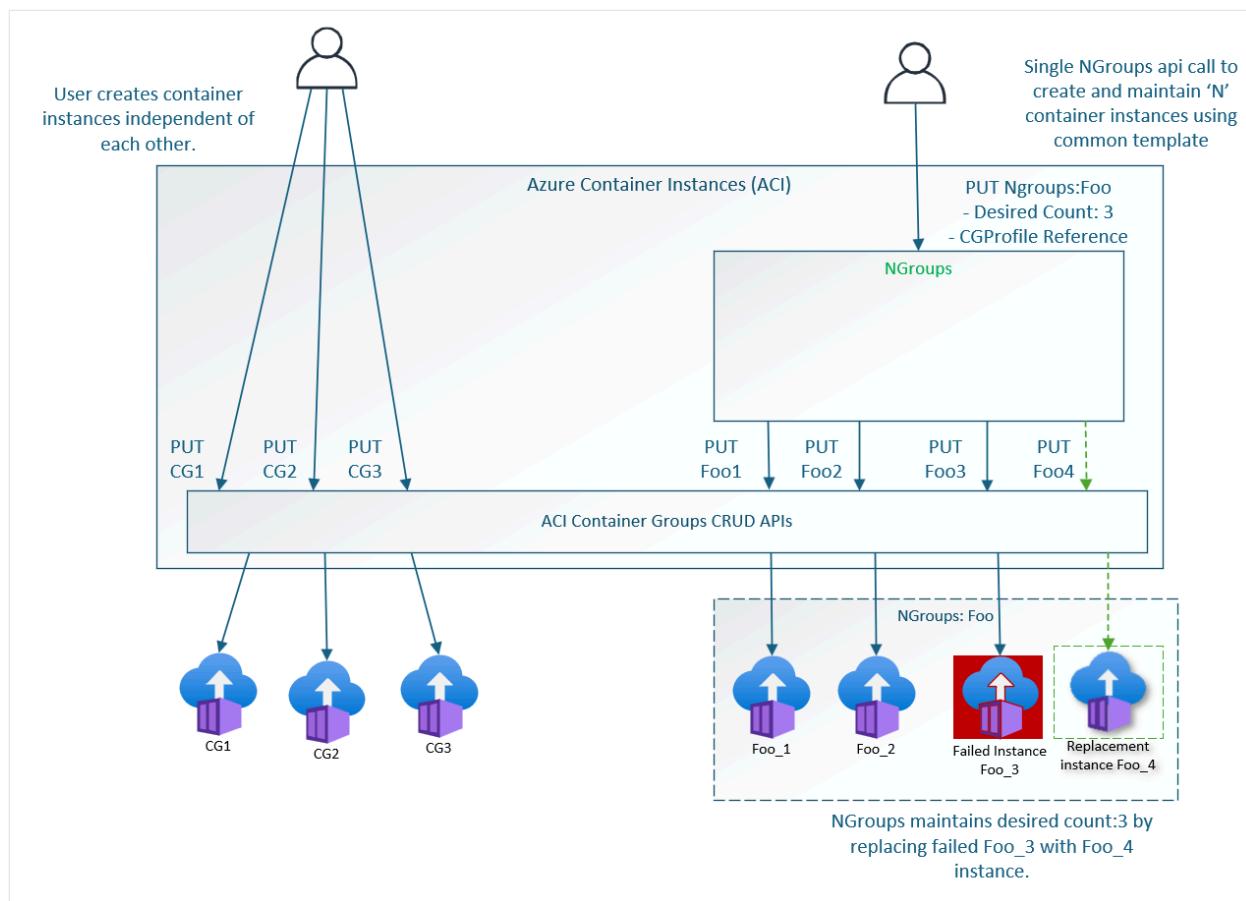
NGroups provides you with advanced capabilities for managing multiple related container groups. Supported features include:

- Maintaining multiple instances
- Rolling upgrades
- High availability through Availability Zones (AZs)
- Managed identity support
- Confidential container support
- Load balancing
- Zone rebalancing (Zone Any)

The NGroups feature builds upon ACI, ensuring container groups are secure, highly available, and supports the feature set of ACI.

For information about Azure Container Instances, see: [What is Azure Container Instances?](#)

## NGroups High Level Architecture



With Azure Container Instances, customers need to create and maintain each individual container group manually. NGroups offers an easier solution to create, update, and manage N container group instances with a single API call.

Creating an NGroups resource is a two step process.

1. Create a [Container Group Profile](#) (CGProfile), which serves as a template. In the CGProfile, a user specifies CG properties that are applied across all CGs created by NGroups.
2. Create an [NGroups](#) resource. You can provide the desired count (number of required CGs) and a reference to the container group profile along with other relevant properties.

NGroups references this Container Group Profile and then calls ACI APIs in order to create/update CGs with the properties mentioned in the CGProfile.

## Concepts

### Container Group Profile (CG Profile)

A large-scale cloud application may require you to manage multiple container groups. As of today, in order to run multiple CGs (Container Groups), customers need to provide

relevant properties such as container images, restart policy, and other properties each time. This can result in throttling, duplicated effort, and management overhead.

To alleviate this concern, NGroups introduced Container Group Profiles. The container group profile (CGProfile) serves as a *template* for creating container groups with same set of properties.

Here are some of the common properties that can be specified in a container group profile:

- osType (Example: Linux, Windows)
- containers. Image name, memory, CPU etc.
- restartPolicy
- ipAddress protocol and internal port
- shutdownGracePeriod
- timeToLive

And here's a sample CG profile:

JSON

```
{
 "location": "{{location}}",
 "properties": {
 "sku": "Standard",
 "containers": [
 {
 "name": "container1",
 "properties": {
 "image": "nginx",
 "ports": [
 {
 "protocol": "TCP",
 "port": 80
 }
],
 "resources": {
 "requests": {
 "memoryInGB": 2.0,
 "cpu": 1.0
 }
 }
 }
 }
],
 "restartPolicy": "Always",
 "shutdownGracePeriod": "PT1H",
 "ipAddress": {
 "ports": [
 {
 "name": "http",
 "port": 80
 }
]
 }
 }
}
```

```

 "protocol": "TCP",
 "port": 80
 }
],
"type": "Public",
},
"timeToLive": "PT1H",
"osType": "Linux"
}
}

```

## NGroups

NGroups resource provides a way to create and manage 'n' container groups with a rich set of operations. An NGroups resource references a container group profile resource and uses that to create N instances of similar looking CGs. Within NGroups resource, customers can also specify other properties including but not limited to number of CGs, update preferences (manual or rolling update), load balancers, subnets, and other relevant properties which they want to associate with CGs under an NGroups resource.

 **Note**

A CG profile needs to be created **before** creating an NGroups resource. Since the CG profile is an ARM resource, it has its own ARM APIs. A CG profile needs to be created **before** creating an NGroups resource.

## Benefits of Referencing Container Group Profile

- Container group profile is a separate resource from NGroups. Customers can create multiple NGroups that can refer to the same container group profile. It also guarantees consistency across all the NGroups that refer to a single container group profile and avoids duplication.
- A single ACI CG could also be created from a CG profile. It allows you to rapidly move from prototype to production mode.

Here's a sample of an NGroups resource with managed identity and zones that refers to a container group profile and creates three container groups:

JSON

```
{
 "location": "{{location}}",
 "properties": {

```

```

 "elasticProfile": {
 "desiredCount": 100 // specifies how many CGs to create
 },
 "containerGroupProfiles": [
 {
 "resource": {
 "id": "/subscriptions/{{subId}}/resourceGroups/{{rgName}}/providers/Microsoft.ContainerInstance/containerGroupProfiles/{{cgProfile1}}"
 }
 }
]
}

```

## NGroups Feature Highlights

- Offers both Rolling and Manual update
- Manage cross zonal container groups
- Supports managed identities
- Add load balancer and application gateway to manage traffic across container groups
- Manage container groups with different container group profiles
- Attach and detach container groups

## NGroups API

NGroups references a CG profile and adds other related properties and capabilities.

Example:

- The desired count of CGs to create or scale out
- The subnet into which CGs are deployed when using a virtual network
- The Load Balancer or Application Gateway to provide network ingress to the CGs

NGroups in turn invokes the ACI ARM APIs to create and manage each CG. Since it uses the same ARM APIs, there's no difference between the CGs created by NGroups and the CGs created directly by the customer. They have the exact same API experience.

## Updating an NGroups Resource

As requirements change, we would need to keep updating our NGroups and its CGs. There are two update modes with which we can update an NGroups – **Manual** (default option) and **Rolling**.

Consider a basic example of updating a CG profile reference from *cgprofile1* to *cgprofile2*:

- In Manual mode, we update the reference to *cgprofile2* and send an UPDATE PUT request to NGroups:

NGroups stores this new CG profile reference. But it doesn't update existing CGs with this reference. Existing CGs are currently running and there's no impact on them. However, when NGroups is scaled out, the CGs are created with *cgprofile2*.

- How do we update existing CGs with *cgprofile2*?

To update existing CGs with new CGProfile, we issue a *manual update* command with an explicit list of CGs we want to update. This command only updates the CGs specified in its list. Updating the CG involves calling the ACI's PUT CG API. The CGs not specified in this list continue to run with *cgprofile1*.

This mode gives us flexibility to update CGs selectively and provides full control over impact on production workloads.

In **Rolling** mode, when we update the reference to *cgprofile2* and issue an UPDATE NGroups command, existing CGs are updated with *cgprofile2*. The update to existing CGs happens in small batches (and not all at once). This ensures that there is a minimal impact on your workload since only a small percentage of CGs may be unavailable during the update.

We can configure the batch size and other related rolling update mode settings with the NGroups API.

## Try NGroups

### Prerequisites for Working on NGroups

The currently supported API version is **2024-09-01-preview**.

1. [Register the feature](#) `Microsoft.ContainerInstace/NGroupsPreview` on your subscriptions.
2. Once the feature flags are applied to the subscription, [register the resource provider](#) `Microsoft.ContainerInstance` on your subscriptions.

#### Note

Use the api-version - 2024-09-01-preview and onwards for preview.

### 💡 Tip

Follow Azure Container Instance Swagger for up to date information on NGroups APIs. [Container Instance NGroups Swagger - 2024-11-01-preview](#)

If these prerequisites aren't met, requests fail, and the NGroups resource type isn't recognized.

## ARM Template Samples

Create CG Profile: [ContainerGroupProfile-Sample.json](#) Create Zonal NGroups with CGProfile: [NGroups-Zonal-Sample.json](#)

Customers can see if a container group is associated to an NGroups resource by checking container group orchestratorId property under the JSON view. The orchestratorId represents the associated NGroups ARM resource ID.

Resource JSON

a0a0a0a0-bbbb-cccc-dddd-e1e1e1e1e1

| Resource ID                                                                                      | API Versions |
|--------------------------------------------------------------------------------------------------|--------------|
| /subscriptions/aaaa0a0a-bb1b-cc2c-dd3d-eeeeee4e4e4e/resourceGroups/sampleRG/providers/Micros ... | 2019-12-01   |

```
531 "state": "Running"
532 },
533 "containerGroupProfile": {
534 "id": "/subscriptions/aaaa0a0a-bb1b-cc2c-dd3d-eeeeee4e4e4e/resourceGroups/sampleRG/provi
535 "revision": 3
536 },
537 "orchestrationProfile": {
538 "id": "/subscriptions/aaaa0a0a-bb1b-cc2c-dd3d-eeeeee4e4e4e/resourceGroups/sampleRG/provi
539 "faultDomainCount": 0
540 }
541 },
```

## How-To Guide

### Perform a Rolling Update

We can use the Rolling Update feature to automatically update all CGs to a newer version without downtime of the NGroups. See rolling update documentation: [NGroups Rolling update](#).

### Create a Regional (zonal/non-zonal) NGroups

First create a CG profile. Here's a sample CG profile. Currently supported API version is 2024-09-01-preview.

JSON

```
{
 "properties": {
 "sku": "Standard",
 "containers": [
 {
 "name": "container1",
 "properties": {
 "image": "nginx",
 "ports": [
 {
 "protocol": "TCP",
 "port": 80
 }],
 "resources": {
 "requests": {
 "memoryInGB": 2.0,
 "cpu": 1.0
 }
 }
 }
 }
],
 "restartPolicy": "Always",
 "shutdownGracePeriod": "PT1H",
 "ipAddress": {
 "ports": [
 {
 "protocol": "TCP",
 "port": 80
 }],
 "type": "Public"
 },
 "timeToLive": "PT1H",
 "osType": "Linux"
 }
}
```

Next, you can create a Zonal/Non-Zonal NGroups by either adding zones outside the properties or leaving the zones array empty.

JSON

```
{
 "properties": {
 "elasticProfile": {
 "desiredCount": 5
 },
 "containerGroupProfiles": [
]
}
```

```

 {
 "resource": {
 "id": "
[resourceId('Microsoft.ContainerInstance/containerGroupProfiles',
parameters('cgProfileName'))]"
 }
 }
},
"zones": ["1", "2", "3"]
}

```

When NGroups is scaled out by setting its desiredCount property, the CGs are distributed evenly across all specified zones. If one zone goes down, the application remains available because the remaining CGs of NGroups continue to run in other zones.

## Can I Update CG Created By an NGroups Resource Directly Through ACI CG APIs?

Yes, customers have the flexibility to update container groups (CGs) directly by using the Azure Container Instances (ACI) APIs. For a deeper understanding of ACI container groups and to explore the related API options, check out this resource: [Container groups in Azure Container Instances](#)

While creating or updating container groups, NGroups relies on the same ACI APIs. This means that customers can use these APIs to update specific container groups as needed, without any extra configurations.

## Technical Capabilities and Constraints

- Once an NGroups resource is created with a set of zones (for example, { "1", "2" }), the zones can't be removed. However, a new zone can be added to the list. For example, { "1", "2", "3" }
- If a specified zone is down, then the overall NGroups operation to create the CGs fails. Retry the request once the zone is back up. Another option is to delete the failed CGs.
- During scale down, NGroups randomly deletes instances, which might not always maintain AZ spread. However, subsequent scale-out operations always try to rebalance the AZ spread.

- AZ spread isn't supported with Spot containers. If you have such a requirement, reach out to the ACI team.
- See also: [impact of availability due to infrastructure/platform updates](#).

## Create NGroups CGs With a Prefix

Customers can create NGroups CGs with a prefix instead of just GUID names:

JSON

```
"properties": {
 "elasticProfile": {
 "desiredCount": 2,
 "containerGroupNamingPolicy": {
 "guidNamingPolicy": {
 "prefix": "cg-"
 }
 }
 },
}
```

This can be useful when you have multiple NGroups in a single resource group and want to differentiate CGs belonging to each NGroup (for example, in the Azure portal view). You can also change it for each scale-out operation to identify CGs that were scaled out together in one operation.

## Create NGroups With Both System-Assigned and User-Assigned Managed Identities

JSON

```
"location": "{{location}}"
"identity": {
 "type": "SystemAssigned, UserAssigned",
 "userAssignedIdentities": {

 "/subscriptions/{{subId}}/resourceGroups/{{rgName}}/providers/Microsoft.ManagedIdentity/userAssignedIdentities/{{userAssignedIdentity1}}": {},
 "/subscriptions/{{subId}}/resourceGroups/{{rgName}}/providers/Microsoft.ManagedIdentity/userAssignedIdentities/{{userAssignedIdentity2}}": {}
 }
}
```

If I delete some CGs from an NGroups, can the NGroups rebuild itself with new CGs to maintain its desired count?

Yes, you can set the properties.elasticProfile.maintainDesiredCount bool property to true.

It creates a new CG for every CG that is being deleted/detached from the NGroups. It tries to maintain the desiredCount property of the NGroups to its set value.

This is useful when you want to use the NGroups as a *pool* which automatically gets replenished when you take away CGs from the pool for your workload scenarios.

It is a nullable bool property. If you omit it for subsequent NGroups PUT/update calls, it doesn't reset to false. To reset, you must explicitly set it to false. When it is null/false, and when a CG is deleted/detached from the NGroups, the desiredCount property for the NGroups reduces accordingly.

## How do I get the CG name, NGroups ID and other metadata propagated down into the container?

Currently, we expose only the CG name and orchestrator ID (the ARM resource ID). In the future, other relevant properties could be considered. These two properties show up as container environment variables.

To get these environment variables on the container, specify these tags *at the NGroups level*:

JSON

```
tags: {
 "metadata.container.environmentVariable.containerGroupName": true,
 "metadata.container.environmentVariable.orchestratorId": true,
 :
 : // other NGroups tags you may have
 :
}
```

NGroups understands these tags as *special* and propagates the required environment variables down to each container as shown here.

The screenshot shows the Microsoft Azure Container Instances dashboard. On the left, a sidebar menu includes 'Overview', 'Activity log', 'Access control (IAM)', 'Tags', 'Resource visualizer', 'Settings' (with 'Containers' selected), 'Identity', 'Properties', 'Locks', 'Monitoring', and 'Automation'. The main content area displays a single container instance named 'container1' with the image 'nginx' and a state of 'Running'. Below the container list is a table titled 'Environment variables' with two entries: 'ContainerGroupName' and 'OrchestratorId'. The 'OrchestratorId' entry is highlighted with a red box.

| Key                | Value                       |
|--------------------|-----------------------------|
| ContainerGroupName | b1b1b1b1-cccc-dddd-eeee-    |
| OrchestratorId     | /subscriptions/aaaa0a0a-bbc |

## What is the impact of availability due to infrastructure/platform updates?

For workloads that offer higher availability (for example, NGroups spread across multiple AZs), there's still a low possibility of CGs in more than one AZ going down at the same time. It can happen when the underlying Azure infrastructure (host machines, Virtual Machine Scale Sets, etc.) goes through an update (called as an infrastructure update or platform update).

This update is done AZ by AZ with not much automated coordination across AZs. Coordination is manually tracked and best-effort.

So, if by chance, a platform update happens simultaneously across 2 or more AZs, then CGs across these AZs can be down simultaneously thus causing unavailability for your NGroups.

## How to use Confidential Containers with NGroups

NGroups supports Confidential ACI container groups. Confidential instances are defined using the following properties within a Container Group Profile.

```
JSON

{
 "location": "{location}",
 ...
}
```

```

"properties": {
 "sku": "Confidential",
 "confidentialComputeProperties": {
 "ccePolicy": "<base 64 encoded policy>"
 },
 "containers": [...],
 "restartPolicy": "Always",
 "shutdownGracePeriod": "PT1H",
 "ipAddress": { ... },
 "timeToLive": "PT1H",
 "osType": "Linux"
}
}

```

Refer to the ACI documentation of confidential containers here: [Tutorial: Prepare a deployment for a confidential container on Azure Container Instances](#)

## Samples

### Container Group Profile Sample

JSON

```
{
 "properties": {
 "sku": "Standard",
 "containers": [
 {
 "name": "web",
 "properties": {
 "image": "mcr.microsoft.com/azuredocs/aci-helloworld",
 "ports": [
 {
 "protocol": "TCP",
 "port": 80
 }
],
 "targetState": "Running",
 "resources": {
 "requests": {
 "memoryInGB": 1,
 "cpu": 1
 }
 }
 }
 }
],
 "restartPolicy": "Always",
 "shutdownGracePeriod": "PT2H",
 "ipAddress": {

```

```

 "ports": [
 {
 "protocol": "TCP",
 "port": 80
 }
],
 "type": "Public"
 },
 "osType": "Linux",
 "revision": 1
},
"id":
"/subscriptions/{{subId}}/resourceGroups/{{rgName}}/providers/Microsoft.ContainerInstance/containerGroupProfiles/{{cgProfile1}}",
"name": "{{cgProfile1}}",
"type": "Microsoft.ContainerInstance/containerGroupProfiles",
"location": "{{location}}"
}

```

## NGroups with Zones Sample

JSON

```
{
 "$schema": "https://schema.management.azure.com/schemas/2019-04-01/deploymentTemplate.json#",
 "contentVersion": "1.0.0.0",
 "parameters": {
 "apiVersion": {
 "type": "string",
 "maxLength": 32
 },
 "NGroupsName": {
 "type": "string",
 "maxLength": 64
 },
 "containerGroupProfileName": {
 "type": "string",
 "maxLength": 64
 },
 "resourceTags": {
 "type": "object"
 },
 "desiredCount": {
 "type": "int"
 }
 },
 "variables": {
 "description": "This ARM template can be parameterized for a basic CRUD scenario for NGroups. It is self contained with cgProfile and NGroups resource",
 "cgProfileName": "[parameters('containerGroupProfileName')]"
 }
}
```

```

 "NGroupsName": "[parameters('NGroupsName')]",
 "resourcePrefix": "[concat('/subscriptions/',
subscription().subscriptionId, '/resourceGroups/', resourceGroup().name,
'/providers/')]"
},
"resources": [
{
 "apiVersion": "[parameters('apiVersion')]",
 "type": "Microsoft.ContainerInstance/containerGroupProfiles",
 "name": "[variables('cgProfileName')]",
 "location": "[resourceGroup().location]",
 "properties": {
 "sku": "Standard",
 "containers": [
 {
 "name": "web",
 "properties": {
 "image": "mcr.microsoft.com/azuredocs/aci-helloworld",
 "ports": [
 {
 "protocol": "TCP",
 "port": 80
 }
],
 "resources": {
 "requests": {
 "memoryInGB": 1.0,
 "cpu": 1.0
 }
 }
 }
 }
],
 "restartPolicy": "Always",
 "ipAddress": {
 "ports": [
 {
 "protocol": "TCP",
 "port": 80
 }
],
 "type": "Public"
 },
 "osType": "Linux"
 }
},
{
 "apiVersion": "[parameters('apiVersion')]",
 "type": "Microsoft.ContainerInstance/NGroups",
 "name": "[variables('NGroupsName')]",
 "tags": "[parameters('resourceTags')]",
 "location": "[resourceGroup().location]",
 "dependsOn": [
 "[concat('Microsoft.ContainerInstance/containerGroupProfiles/',
variables('cgProfileName'))]"
]
}
]

```

```
],
 "identity": {
 "type": "systemAssigned"
 },
 "properties": {
 "elasticProfile": {
 "desiredCount": "[parameters('desiredCount')]",
 "maintainDesiredCount": true
 },
 "containerGroupProfiles": [
 {
 "resource": {
 "id": "[concat(variables('resourcePrefix'),
'Microsoft.ContainerInstance/containerGroupProfiles/',
variables('cgProfileName'))]"
 }
 }
]
 },
 "zones": ["1", "2", "3"]
}
]
```

## Learn Related Topics

- [Azure Container Instances](#)
- [NGroups Rolling Update](#)
- [Virtual nodes on Azure Container Instances](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# NGroups Rolling Update

Article • 11/14/2024

## Introduction

As requirements change, you may need to keep updating your NGroups and its container groups (CGs).

There are two update modes available for updating an NGroups – **Manual** (default option) and **Rolling**.

Within **Rolling Update (RU)**, there are 2 options – *in-place* update and *replace* update. Replace RU is the default option.

This document describes RU in detail. Manual update is covered in the NGroups documentation link [here](#).

## Feature Description

Consider a basic example of updating a CG profile reference from *cgprofile1* to *cgprofile2*.

### In-place Rolling Update

With in-place RU, when we update the reference to *cgprofile2* and issue an UPDATE NGroups command, *existing CGs* are updated with *cgprofile2*. The update to existing CGs happens in small batches (and not all at once). It ensures that there's a minimal impact on your workload since only a small percentage of CGs may be unavailable during the update.

We can configure the batch size and other related rolling update mode settings with the NGroups API.

Because in-place RU updates existing CGs, there are certain limitations to the CG properties that Azure Container Instances (ACI) enforces.

See ACI's [limitations](#) around updating CGs. Also, see [here](#) for properties on CGs that require a delete (versus an update).

### Replace Rolling Update

With replace RU, when we update the reference to *cgprofile2* and issue an UPDATE NGroups command, new CGs are created with *cgprofile2*. Existing CGs with *cgprofile1* are deleted. This creation and deletion happen in small batches (and not all at once). It ensures that there's a minimal impact on your workload since only a small percentage of CGs is impacted during the update.

Like in-place RU, we can configure the batch size and other related rolling update mode settings with the NGroups API.

Because replace RU creates new CGs, there are fewer limitations enforced by ACI. As a result, replace RU is a more powerful option and is the default option when a customer selects RU.

## Usage

### Triggering a Rolling Update

Rolling update is triggered when an NGroups PUT call is made and the CG profile in the PUT call is different from the CG profile currently referenced in the NGroups.

NGroups then automatically group instances into batches and updates one batch at a time. The *maxBatchPercent* parameter determines the size of the batch.

### Updating a Batch

- An **in-place** update invokes a CG PUT call to update each CG of the batch.
- A **replace** update invokes a CG PUT call to create new CGs and delete existing CGs of the batch. There exists a 1:1 correspondence between the CGs being created and the CGs being deleted. However, the CG names will be different.

If a sufficient number of CGs in the batch provide healthy signals after the *pauseTimeBetweenBatches* period, NGroups automatically starts the next batch for the update. Otherwise, it stops the rollout. The *maxUnhealthyPercent* parameter specifies the total number of unhealthy CGs, while the *maxUnhealthyUpdatedPercent* parameter specifies the total number of unhealthy CGs after the update.

Here is an example to issue a rolling update request to NGroups:

JSON

```
{
 "location": "{{location}}",
```

```

"properties": {
 "updateProfile": {
 "updateMode": "Rolling",
 "rollingUpdateProfile": {
 // Maximum percentage of total CGs which can be updated
 // simultaneously by rolling update in one batch.
 "maxBatchPercent": "10", // optional, defaults to 20%
 // Maximum percentage of the total CGs across the whole
 NGroup
 // that can be unhealthy at a time either by rolling update
 or health
 // checks by liveness probes. If there are more unhealthy
 CGs than this,
 // the current rolling update is marked as failed.
 // This check is a prerequisite to start any batch.
 "maxUnhealthyPercent": "10", // optional, defaults to 20%
 // Maximum percentage of the updated CGs which can be in
 unhealthy state
 // after each batch is updated. If there are more unhealthy
 CGs than this,
 // the current rolling update is marked as failed.
 "maxUnhealthyUpdatedPercent": 10, // optional, defaults to
 20%
 // The wait time between batches after completing one batch
 of the rolling
 // update and before starting the next batch. The time
 duration should
 // be specified in ISO 8601 format for duration.
 "pauseTimeBetweenBatches": "PT2M", // optional, defaults to
 PT1M
 // A nullable boolean property. Default is null
 // Sets the mode to either in-place RU (when true) or
 replace (default) RU.
 "inPlaceUpdate": null/false/true
 }
 },
 "containerGroupProfiles": [
 {
 "resource": {
 "id": "/subscriptions/{{subId}}/resourceGroups/{{rgName}}
}providers/Microsoft.ContainerInstance/containerGroupProfiles/cgp1"
 }
]
 }
}

```

If image version is set to the **latest** tag for container images within the CG profile, then NGroups automatically picks up the latest image version during the RU. To prevent unexpected behavior in your application, it is recommended to not use the *latest* tag for images. Instead, use specific versions.

### ⓘ Note

In order to use replace RU, set the tag "rollingupdate.replace.enabled: true" for the NGroups resource. This tag is temporary, and it will not be required in the future.

JSON

```
"tags": {
 "rollingupdate.replace.enabled": true
}
```

## Getting Status of a Running Rolling Update

For getting the latest status of your rolling update, you can use this REST API:

GET

```
/subscriptions/{subscriptionId}/resourceGroups/{rgName}/providers/Microsoft.ContainerInstance/NGroups/{{ngroupsName}}/latestRollingUpdate
```

This returns a response containing relevant information about the RU.

## Cancelling a Rolling Update

To cancel a rolling update, use the following API. Once canceled, the RU cannot be resumed/restarted. A new RU needs to be triggered.

POST

```
/subscriptions/{subscriptionId}/resourceGroups/{rgName}/providers/Microsoft.ContainerInstance/NGroups/{{ngroupsName}}/cancelRollingUpdate
```

You do not need to provide a request body when calling this API.

## Boundary of a Batch in a Rolling Update

The CGs of a specific batch in an RU do not cross a fault model boundary. A fault model represents a zone/fault-domain (FD) combination. For example, zone 1 / FD 0 is a fault

model boundary, zone 1 / FD 1 is another fault model boundary, and zone 2 / FD 0 is yet another fault model boundary.

If a customer has a multi-zonal NGroups set up with three zones, a batch is confined to CGs belonging to only one zone at most. A batch never consists of CGs spread across multiple zones.

If a customer has a multi-zonal and multi-FD NGroups setup, a batch still consists of CGs belonging to only one FD in a single zone at most.

NGroups maintains this fault model boundary in a batch, even when the number of CGs selected for a batch is much less than the maxBatchPercent setting. It reflects that NGroups prefers safe updates over faster (and thus riskier) updates.

The only time a fault model boundary is crossed when the RU selects unhealthy CGs for the first batch. In this batch, the RU attempts to update all unhealthy CGs to improve the overall availability of NGroups. As a result, when updating unhealthy CGs, the RU may exceed the maxBatchPercent setting.

## Learn Related Topics

- [Azure Container Instances](#)
- [About NGroups](#)

---

## Feedback

Was this page helpful?



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

# Azure security baseline for Container Instances

Article • 02/25/2025

This security baseline applies guidance from the [Microsoft cloud security benchmark version 1.0](#) to Container Instances. 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 Container Instances.

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 Container Instances have been excluded. To see how Container Instances completely maps to the Microsoft cloud security benchmark, see the [full Container Instances security baseline mapping file](#).

## Security profile

The security profile summarizes high-impact behaviors of Container Instances, which may result in increased security considerations.

ⓘ [Expand table](#)

| Service Behavior Attribute                              | Value               |
|---------------------------------------------------------|---------------------|
| Product Category                                        | Compute, Containers |
| Customer can access HOST / OS                           | No Access           |
| Service can be deployed into customer's virtual network | True                |
| Stores customer content at rest                         | False               |

# 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:** [Deploy container instances into Virtual Network](#)

#### 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 NSG rules to control container access to subnets or other network resources

**Note:** Placing an Azure Load Balancer in front of container instances in a networked container group is not supported.

## NS-2: Secure cloud services with network controls

### Features

#### Azure Private Link

**Description:** Service native IP filtering capability for filtering network traffic (not to be confused with NSG or Azure Firewall). [Learn more](#).

[+] Expand table

| Supported | Enabled By Default | Configuration Responsibility |
|-----------|--------------------|------------------------------|
| False     | Not Applicable     | Not Applicable               |

**Configuration Guidance:** This feature is not supported to secure this service.

#### 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 |
|-----------|--------------------|------------------------------|
| False     | Not Applicable     | Not Applicable               |

**Configuration Guidance:** This feature is not supported to secure this service.

## 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](#)

| Supported | Enabled By Default | Configuration Responsibility |
|-----------|--------------------|------------------------------|
| False     | Not Applicable     | Not Applicable               |

**Feature notes:** Customer can create use or system assigned Managed identity to securely access resources in ARM

**Configuration Guidance:** This feature is not supported to secure this service.

## 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](#)

| Supported | Enabled By Default | Configuration Responsibility |
|-----------|--------------------|------------------------------|
| 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](#)

| Supported | Enabled By Default | Configuration Responsibility |
|-----------|--------------------|------------------------------|
| True      | False              | Customer                     |

**Configuration Guidance:** Use managed identities for Azure resources to run code in Azure Container Instances that interacts with other Azure services - without maintaining any secrets or credentials in code. The feature provides an Azure Container Instances deployment with an automatically managed identity in Azure Active Directory.

Reference: [How to Use Managed Identitites with Azure Container Instances](#)

## Service Principals

**Description:** Data plane supports authentication using service principals. [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-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 |
|-----------|--------------------|------------------------------|
| 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-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 |
|-----------|--------------------|------------------------------|
| False     | Not Applicable     | Not Applicable               |

**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-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 |
|-----------|--------------------|------------------------------|
| True      | False              | Customer                     |

**Configuration Guidance:** If using a cloud-based private registry like Azure Container Registry with Azure Container Instances, use Azure role-based access control (Azure RBAC) to manage access to data and resources in an Azure container registry.

**Reference:** [Security recommendations for Azure Container Instances](#)

## 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-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      | False              | Microsoft                    |

Configuration Guidance: Enable data at rest encryption using platform managed (Microsoft managed) keys where not automatically configured by the service.

Reference: [Azure Container Instances - encrypt deployment data](#)

## 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 |
|-----------|--------------------|------------------------------|
| True      | False              | Customer                     |

**Configuration Guidance:** If required for regulatory compliance, define the use case and service scope where encryption using customer-managed keys are needed. Enable and implement data at rest encryption using customer-managed key for those services.

**Reference:** [Azure Container Instances - encrypt deployment data](#)

## 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 |
|-----------|--------------------|------------------------------|
| True      | False              | Customer                     |

**Configuration Guidance:** Use Azure Key Vault to create and control the lifecycle of your encryption keys used for the customer-managed key-based encryption for your container instance.

Reference: [Create a Key Vault resource](#)

## 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                     |

Feature notes: Azure Container Instances supports configuring policy for 2 features (CMK and deployments in private virtual network)

**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 Instances](#)

## Logging and threat detection

For more information, see the [Microsoft cloud security benchmark: Logging and threat detection](#).

### 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      | False              | Customer                     |

**Configuration Guidance:** Log Analytics workspaces provide a centralized location for storing and querying log data not only from Azure resources, but also on-premises resources and resources in other clouds. Azure Container Instances includes built-in support for sending logs and event data to Azure Monitor logs.

Reference: [Container group and instance logging with Azure Monitor logs](#)

## 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               |

**Feature notes:** ACI is recommended for stateless applications and we do not offer backup service. Customer can opt for regions which support AZ support to back their deployments in same region. For cross-region, they need to design backup and disaster recovery solution by deploying their apps in ACI in multiple regions and route traffic using traffic manager to handle disaster recovery use cases.

**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](#)

# Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#)

# Security considerations for Azure Container Instances

Article • 08/29/2024

This article introduces security considerations for using Azure Container Instances to run container applications. Topics include:

- ✓ **Security recommendations** for managing images and secrets for Azure Container Instances
- ✓ **Considerations for the container ecosystem** throughout the container lifecycle, for any container platform

For comprehensive recommendations that help you improve the security posture of your deployment, see the [Azure security baseline for Container Instances](#).

## Security recommendations for Azure Container Instances

### Use a private registry

Containers are built from images that are stored in one or more repositories. These repositories can belong to a public registry, like [Docker Hub](#), or to a private registry. An example of a private registry is the [Docker Trusted Registry](#), which can be installed on-premises or in a virtual private cloud. You can also use cloud-based private container registry services, including [Azure Container Registry](#).

A publicly available container image doesn't guarantee security. Container images consist of multiple software layers, and each software layer might have vulnerabilities. To help reduce the threat of attacks, you should store and retrieve images from a private registry, such as Azure Container Registry or Docker Trusted Registry. In addition to providing a managed private registry, Azure Container Registry supports [service principal-based authentication](#) through Microsoft Entra ID for basic authentication flows. This authentication includes role-based access for read-only (pull), write (push), and other permissions.

### Monitor and scan container images

Take advantage of solutions to scan container images in a private registry and identify potential vulnerabilities. It's important to understand the depth of threat detection that

the different solutions provide.

For example, Azure Container Registry optionally [integrates with Microsoft Defender for Cloud](#) to automatically scan all Linux images pushed to a registry. Microsoft Defender for Cloud's integrated Qualys scanner detects image vulnerabilities, classifies them, and provides remediation guidance.

Security monitoring and image scanning solutions such as [Twistlock](#) and [Aqua Security](#) are also available through the Azure Marketplace.

## Protect credentials

Containers can spread across several clusters and Azure regions. So, you must secure credentials required for logins or API access, such as passwords or tokens. Ensure that only privileged users can access those containers in transit and at rest. Inventory all credential secrets, and then require developers to use emerging secrets-management tools that are designed for container platforms. Make sure that your solution includes encrypted databases, TLS encryption for secrets data in transit, and least-privilege [Azure role-based access control \(Azure RBAC\)](#). [Azure Key Vault](#) is a cloud service that safeguards encryption keys and secrets (such as certificates, connection strings, and passwords) for containerized applications. Because this data is sensitive and business critical, secure access to your key vaults so that only authorized applications and users can access them.

## Considerations for the container ecosystem

The following security measures, implemented well and managed effectively, can help you secure and protect your container ecosystem. These measures apply throughout the container lifecycle, from development through production deployment, and to a range of container orchestrators, hosts, and platforms.

## Use vulnerability management as part of your container development lifecycle

By using effective vulnerability management throughout the container development lifecycle, you improve the odds that you identify and resolve security concerns before they become a more serious problem.

## Scan for vulnerabilities

New vulnerabilities are discovered all the time, so scanning for and identifying vulnerabilities is a continuous process. Incorporate vulnerability scanning throughout the container lifecycle:

- As a final check in your development pipeline, you should perform a vulnerability scan on containers before pushing the images to a public or private registry.
- Continue to scan container images in the registry both to identify any flaws that were somehow missed during development and to address any newly discovered vulnerabilities that might exist in the code used in the container images.

## Map image vulnerabilities to running containers

You need to have a means of mapping vulnerabilities identified in container images to running containers, so security issues can be mitigated or resolved.

## Ensure that only approved images are used in your environment

There's enough change and volatility in a container ecosystem without allowing unknown containers as well. Allow only approved container images. Have tools and processes in place to monitor for and prevent the use of unapproved container images.

An effective way of reducing the attack surface and preventing developers from making critical security mistakes is to control the flow of container images into your development environment. For example, you might sanction a single Linux distribution as a base image, preferably one that is lean (Alpine or CoreOS rather than Ubuntu), to minimize the surface for potential attacks.

Image signing or fingerprinting can provide a chain of custody that enables you to verify the integrity of the containers. For example, Azure Container Registry supports Docker's [content trust](#) model, which allows image publishers to sign images that are pushed to a registry, and image consumers to pull only signed images.

## Permit only approved registries

An extension of ensuring that your environment uses only approved images is to permit only the use of approved container registries. Requiring the use of approved container registries reduces your exposure to risk by limiting the potential for the introduction of unknown vulnerabilities or security issues.

## Ensure the integrity of images throughout the lifecycle

Part of managing security throughout the container lifecycle is to ensure the integrity of the container images in the registry and as they're altered or deployed into production.

- Images with vulnerabilities, even minor, shouldn't be allowed to run in a production environment. Ideally, all images deployed in production should be saved in a private registry accessible to a select few. Keep the number of production images small to ensure that they can be managed effectively.
- Because it's hard to pinpoint the origin of software from a publicly available container image, build images from the source to ensure knowledge of the origin of the layer. When a vulnerability surfaces in a self-built container image, customers can find a quicker path to a resolution. With a public image, customers would need to find the root of a public image to fix it or get another secure image from the publisher.
- A thoroughly scanned image deployed in production isn't guaranteed to be up-to-date for the lifetime of the application. Security vulnerabilities might be reported for layers of the image that weren't previously known or were introduced after the production deployment.

Periodically audit images deployed in production to identify images that are out of date or haven't been updated in a while. You might use blue-green deployment methodologies and rolling upgrade mechanisms to update container images without downtime. You can scan images by using tools described in the preceding section.

- Use a continuous integration (CI) pipeline with integrated security scanning to build secure images and push them to your private registry. The vulnerability scanning built into the CI solution ensures that images that pass all the tests are pushed to the private registry from which production workloads are deployed.

A CI pipeline failure ensures that vulnerable images aren't pushed to the private registry that's used for production workload deployments. It also automates image security scanning if there's a significant number of images. Otherwise, manually auditing images for security vulnerabilities can be painstakingly lengthy and error prone.

## Enforce least privileges in runtime

The concept of least privileges is a basic security best practice that also applies to containers. When a vulnerability is exploited, it generally gives the attacker access and privileges equal to those of the compromised application or process. Ensuring that

containers operate with the lowest privileges and access required to get the job done reduces your exposure to risk.

## Reduce the container attack surface by removing unneeded privileges

You can also minimize the potential attack surface by removing any unused or unnecessary processes or privileges from the container runtime. Privileged containers run as root. If a malicious user or workload escapes in a privileged container, the container will then run as root on that system.

## Preapprove files and executables that the container is allowed to access or run

Reducing the number of variables or unknowns helps you maintain a stable, reliable environment. Limiting containers so they can access or run only preapproved or safe listed files and executables is a proven method of limiting exposure to risk.

It's a lot easier to manage a safe list when it's implemented from the beginning. A safe list provides a measure of control and manageability as you learn what files and executables are required for the application to function correctly.

A safe list not only reduces the attack surface but can also provide a baseline for anomalies and prevent the use cases of the "noisy neighbor" and container breakout scenarios.

## Enforce network segmentation on running containers

To help protect containers in one subnet from security risks in another subnet, maintain network segmentation (or nano-segmentation) or segregation between running containers. Maintaining network segmentation may also be necessary to use containers in industries that are required to meet compliance mandates.

For example, the partner tool [Aqua](#) provides an automated approach for nano-segmentation. Aqua monitors container network activities in runtime. It identifies all inbound and outbound network connections to/from other containers, services, IP addresses, and the public internet. Nano-segmentation is automatically created based on monitored traffic.

## Monitor container activity and user access

As with any IT environment, you should consistently monitor activity and user access to your container ecosystem to quickly identify any suspicious or malicious activity. Azure provides container monitoring solutions including:

- [Azure Monitor for containers](#) monitors the performance of your workloads deployed to Kubernetes environments hosted on Azure Kubernetes Service (AKS). Azure Monitor for containers gives you performance visibility by collecting memory and processor metrics from controllers, nodes, and containers that are available in Kubernetes through the Metrics API.
- The [Azure Container Monitoring solution](#) helps you view and manage other Docker and Windows container hosts in a single location. For example:
  - View detailed audit information that shows commands used with containers.
  - Troubleshoot containers by viewing and searching centralized logs without having to remotely view Docker or Windows hosts.
  - Find containers that may be noisy and consume excess resources on a host.
  - View centralized CPU, memory, storage, and network usage and performance information for containers.

The solution supports container orchestrators including Docker Swarm, DC/OS, unmanaged Kubernetes, Service Fabric, and Red Hat OpenShift.

## Monitor container resource activity

Monitor your resource activity, like files, network, and other resources that your containers access. Monitoring resource activity and consumption is useful both for performance monitoring and as a security measure.

[Azure Monitor](#) enables core monitoring for Azure services by allowing the collection of metrics, activity logs, and diagnostic logs. For example, the activity log tells you when new resources are created or modified.

Metrics are available that provide performance statistics for different resources and even the operating system inside a virtual machine. You can view this data with one of the explorers in the Azure portal and [create alerts](#) based on these metrics.

## Log all container administrative user access for auditing

Maintain an accurate audit trail of administrative access to your container ecosystem, including your Kubernetes cluster, container registry, and container images. These logs might be necessary for auditing purposes and will be useful as forensic evidence after any security incident. Azure solutions include:

- [Integration of Azure Kubernetes Service with Microsoft Defender for Cloud](#) to monitor the security configuration of the cluster environment and generate security recommendations
- [Azure Container Monitoring solution](#)
- Resource logs for [Azure Container Instances](#) and [Azure Container Registry](#)

## Next steps

- See the [Azure security baseline for Container Instances](#) for comprehensive recommendations that help you improve the security posture of your deployment.
- Learn more about using [Microsoft Defender for Cloud](#) for real-time threat detection in your containerized environments.
- Learn more about managing container vulnerabilities with solutions from [Twistlock](#) and [Aqua Security](#).

---

## Feedback

Was this page helpful?



Yes



No

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

# Virtual network scenarios and resources

Article • 08/29/2024

Azure Virtual Network provides secure, private networking for your Azure and on-premises resources. By deploying container groups into an Azure virtual network, your containers can communicate securely with other resources in the virtual network.

This article provides background about virtual network scenarios, limitations, and resources. For deployment examples using the Azure CLI, see [Deploy container instances into an Azure virtual network](#).

## Important

Container group deployment to a virtual network is generally available for Linux and Windows containers, in most regions where Azure Container Instances is available. For details, see [Resource availability and quota limits](#).

## Scenarios

Container groups deployed into an Azure virtual network enable scenarios like:

- Direct communication between container groups in the same subnet
- Send [task-based](#) workload output from container instances to a database in the virtual network
- Retrieve content for container instances from a [service endpoint](#) in the virtual network
- Enable container communication with on-premises resources through a [VPN gateway](#) or [ExpressRoute](#)
- Integrate with [Azure Firewall](#) to identify outbound traffic originating from the container
- Resolve names via the internal Azure DNS for communication with Azure resources in the virtual network, such as virtual machines
- Use NSG rules to control container access to subnets or other network resources

## Unsupported networking scenarios

- **Azure Load Balancer** - Placing an Azure Load Balancer in front of container instances in a networked container group isn't supported

- **Global virtual network peering** - Global peering (connecting virtual networks across Azure regions) isn't supported
- **Public IP or DNS label** - Container groups deployed to a virtual network don't currently support exposing containers directly to the internet with a public IP address or a fully qualified domain name
- **Managed Identity with Virtual Network in Azure Government Regions** - Managed Identity with virtual networking capabilities isn't supported in Azure Government Regions

## Other limitations

- To deploy container groups to a subnet, the subnet can't contain other resource types. Remove all existing resources from an existing subnet before deploying container groups to it, or create a new subnet.
- To deploy container groups to a subnet, the subnet and the container group must be on the same Azure subscription.
- Due to the additional networking resources involved, deployments to a virtual network are typically slower than deploying a standard container instance.
- Outbound connections to port 25 and 19390 aren't supported at this time. Port 19390 needs to be opened in your Firewall for connecting to ACI from Azure portal when container groups are deployed in virtual networks.
- For inbound connections, the firewall should also allow all ip addresses within the virtual network.
- If you're connecting your container group to an Azure Storage Account, you must add a [service endpoint](#) to that resource.
- [IPv6 addresses](#) aren't supported at this time.
- Depending on your subscription type, [certain ports could be blocked](#).
- Container instances don't read or inherit DNS settings from an associated virtual network. DNS settings must be explicitly set for container instances.

## Required network resources

There are three Azure Virtual Network resources required for deploying container groups to a virtual network: the [virtual network](#) itself, a [delegated subnet](#) within the virtual network, and a [network profile](#).

### Virtual network

A virtual network defines the address space in which you create one or more subnets. You then deploy Azure resources (like container groups) into the subnets in your virtual

network.

## Subnet (delegated)

Subnets segment the virtual network into separate address spaces usable by the Azure resources you place in them. You create one or several subnets within a virtual network.

The subnet that you use for container groups can contain only container groups. Before you deploy a container group to a subnet, you must explicitly delegate the subnet before provisioning. Once delegated, the subnet can be used only for container groups. If you attempt to deploy resources other than container groups to a delegated subnet, the operation fails.

## Network profile

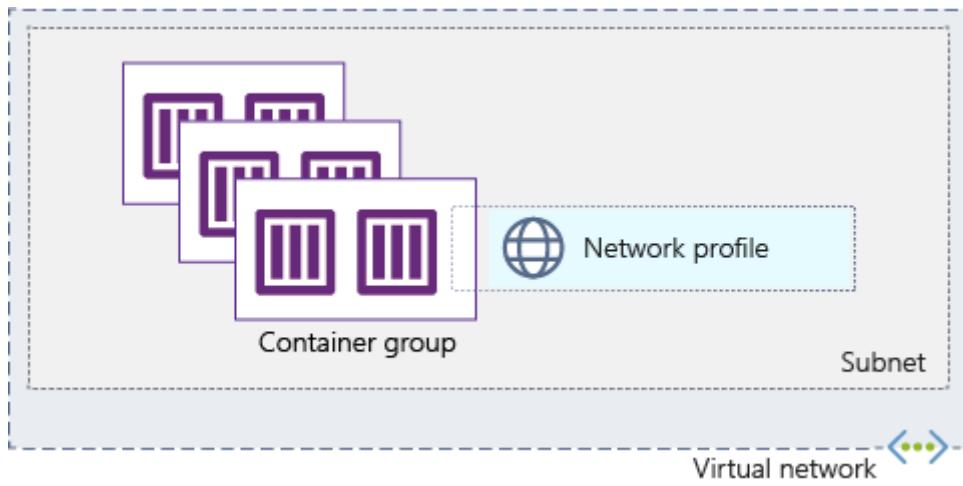
### Important

[Network profiles](#) have been retired as of the `2021-07-01` API version. If you're using this or a more recent version, ignore any steps and actions related to network profiles.

A network profile is a network configuration template for Azure resources. It specifies certain network properties for the resource, for example, the subnet into which it should be deployed. When you first use the [az container create](#) command to deploy a container group to a subnet (and thus a virtual network), Azure creates a network profile for you. You can then use that network profile for future deployments to the subnet.

To use a Resource Manager template, YAML file, or a programmatic method to deploy a container group to a subnet, you need to provide the full Resource Manager resource ID of a network profile. You can use a profile previously created using [az container create](#), or create a profile using a Resource Manager template (see [template example](#) and [reference](#)). To get the ID of a previously created profile, use the [az network profile list](#) command.

The following diagram depicts several container groups deployed to a subnet delegated to Azure Container Instances. Once you deploy one container group to a subnet, you can deploy more container groups to it by specifying the same network profile.



## Next steps

- For deployment examples with the Azure CLI, see [Deploy container instances into an Azure virtual network](#).
- To deploy a new virtual network, subnet, network profile, and container group using a Resource Manager template, see [Create an Azure container group with virtual network](#).
- When using the [Azure portal](#) to create a container instance, you can also provide settings for a new or existing virtual network on the **Networking** tab.

---

## Feedback

Was this page helpful?

Yes

No

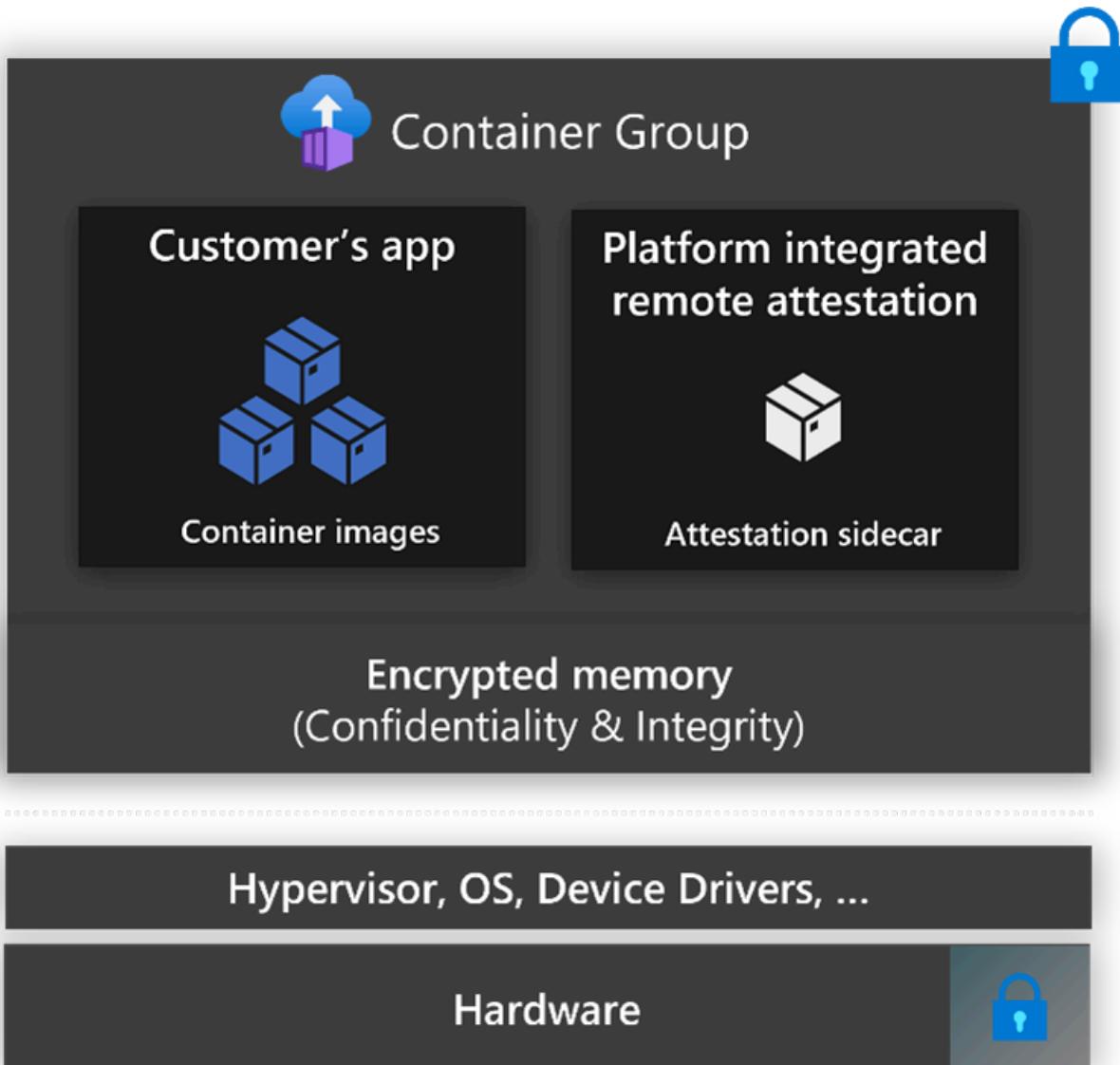
[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

# Confidential containers on Azure Container Instances

Article • 08/29/2024

This article introduces how confidential containers on Azure Container Instances (ACI) can enable you to secure your workloads running in the cloud. This article provides background about the feature set, scenarios, limitations, and resources.

Confidential containers on Azure Container Instances enable customers to run Linux containers within a hardware-based and attested Trusted Execution Environment (TEE). Customers can lift and shift their containerized Linux applications or build new confidential computing applications without needing to adopt any specialized programming models to achieve the benefits of confidentiality in a TEE. Confidential containers on Azure Container Instances protect data-in-use and encrypts data being used in memory. Azure Container Instances extends this capability through verifiable execution policies, and verifiable hardware root of trust assurances through guest attestation.



## Features of confidential containers on Azure Container Instances

### Lift and shift applications

Customers can lift and shift their containerized Linux applications or build new confidential computing applications without needing to adopt specialized programming models to achieve the benefits of confidentiality in a TEE.

### Hardware based trusted execution environment

Confidential containers on Azure Container Instances are deployed in a container group with a Hyper-V isolated TEE, which includes a memory encryption key generated and managed by an AMD SEV-SNP capable processor. Data in use in memory is encrypted.

with this key to help provide protection against data replay, corruption, remapping, and aliasing-based attacks.

## Verifiable execution policies

Confidential containers on Azure Container Instances can run with verifiable execution policies that enable customers to have control over what software and actions are allowed to run within the TEE. These execution policies help to protect against bad actors creating unexpected application modifications that could potentially leak sensitive data. Customers author execution policies through provided [tooling](#), and cryptographic proofs verify the policies.

## Remote guest attestation

Confidential containers on ACI provide support for remote guest attestation, which is used to verify the trustworthiness of your container group before creating a secure channel with a relying party. Container groups can generate an SNP hardware attestation report, which the hardware signs and which includes information about the hardware and software. The Microsoft Azure Attestation service can then verify this generated hardware attestation via an [open-source sidecar application](#) or by another attestation service before any sensitive data is released to the TEE.

## Confidential computing enforcement policies

Confidential containers support container-level integrity and attestation via confidential computing enforcement (CCE) policies. Confidential computing enforcement policies prescribe the components that are permitted to run within the container group, which the container runtime enforces.

## Azure CLI confcom extension

The Azure CLI confcom extension enables customers to generate confidential computing enforcement policies using an ARM template as an input and providing a base 64 string policy as an output. This output is included in the definition of the container group to enforce which components are permitted to run. For more information on authoring confidential computing execution policies, see [Azure CLI confcom extension](#).

# Secure key release and encrypted file system sidecars

Confidential containers on Azure Container Instances integrate with two open source sidecars to support confidential functionality within the container group. You can find these sidecars and more information in the [confidential sidebar repository](#).

## Secure key release sidebar

Confidential containers on Azure Container Instances provide a sidebar open source container for attestation and secure key release. This sidebar instantiates a web server, which exposes a REST API so that other containers can retrieve a hardware attestation report or a Microsoft Azure Attestation token via the POST method. The sidebar integrates with Azure Key vault for releasing a key to the container group after validation completes.

## Encrypted file system sidebar

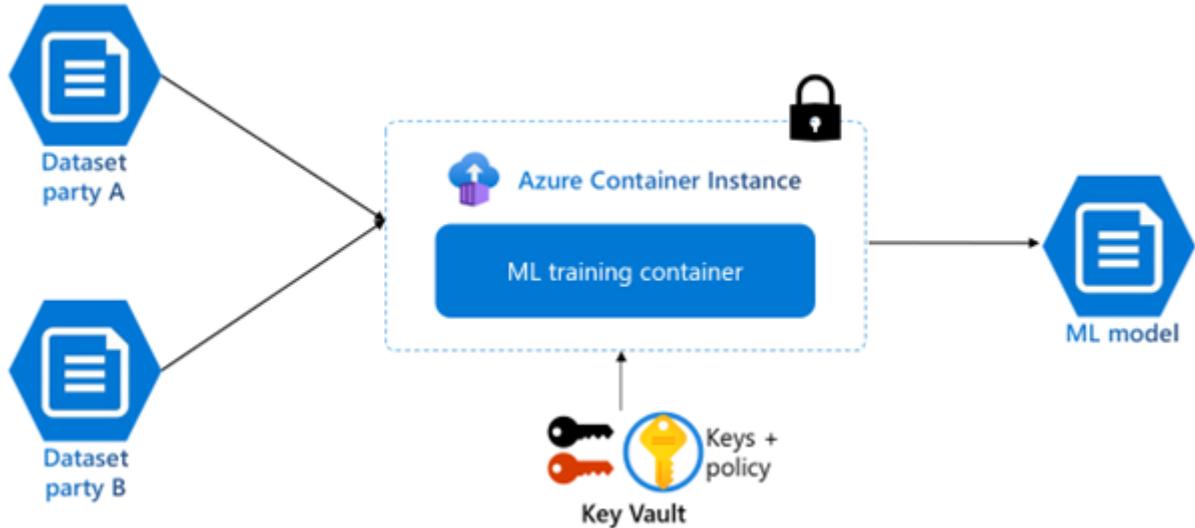
Confidential containers on Azure Container Instances provide a sidebar container to mount a remote encrypted filesystem previously uploaded to Azure Blob Storage. The sidebar container transparently retrieves the hardware attestation and the certificate chain endorsing the attestation's signing key. It then requests Microsoft Azure Attestation to authorize an attestation token, which is required for securely releasing the filesystem's encryption key from the managed HSM. The key is released to the sidebar container only if the expected authority signed the attestation token and the attestation claims match the key's release policy. The sidebar container transparently uses the key to mount the remote encrypted filesystem; this process preserves the confidentiality and integrity of the filesystem upon any operation from a container that is running within the container group.

## Scenarios

### Data clean rooms for multi-party data analytics and machine learning training

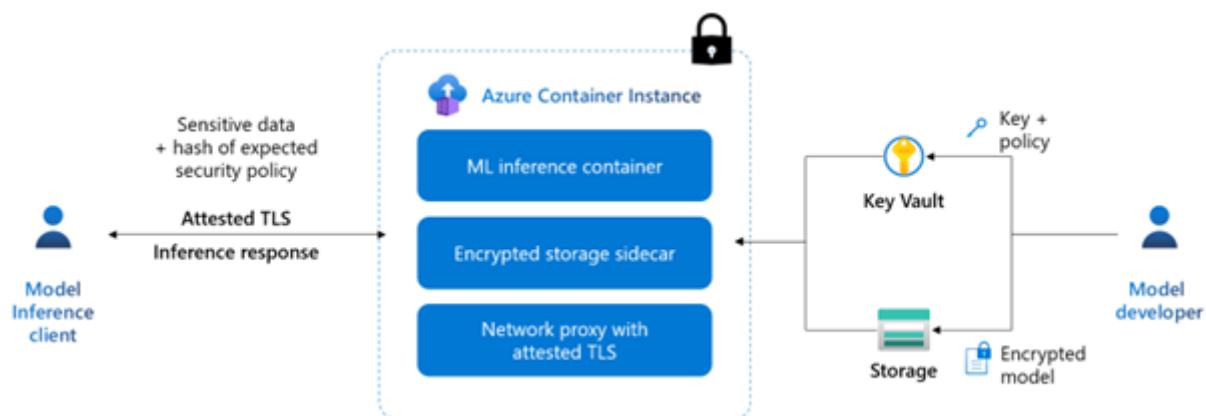
Business transactions and project collaborations often require sharing confidential data among multiple parties. This data may include personal information, financial information, and medical records, which need to be protected from unauthorized access. Confidential containers on Azure Container Instances provide the necessary

features (hardware-based TEEs, remote attestation) for customers to process training data from multiple sources without exposing input data to other parties. These features enable organizations to get more value from their or their partners' datasets while maintaining control over access to their sensitive information. This offering makes confidential containers on Azure Container Instances ideal for multi-party data analytics scenarios such as confidential machine learning.



## Confidential inference

ACI provides fast and easy deployments, flexible resource allocation, and pay per use pricing, which positions it as a great platform for confidential inference workloads. With confidential containers on Azure Container Instances, model developers and data owners can collaborate while protecting the intellectual property of the model developer and keeping the data used for inferencing secure and private. Check out a [sample](#) deployment of confidential inference using confidential containers on Azure Container Instances.



## Unsupported Scenarios

- The Azure CLI confcom extension must generate the confidential computing enforcement policies.
- Confidential computing enforcement policies can't be manually created.

## Resources

- [Azure CLI confcom extension ↗](#)
- [Confidential sidecar containers ↗](#)
- [Confidential hello world application ↗](#)
- [Machine learning inference demo ↗](#)

## Next Steps

- For a deployment example, see [Deploy a confidential container group with Azure Resource Manager](#)
- 

## Feedback

Was this page helpful?



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

# Attestation in Confidential containers on Azure Container Instances

Article • 08/29/2024

Attestation is an essential part of confidential computing. Attestation appears in the definition by the Confidential Computing Consortium: "Confidential Computing is the protection of data in use by performing computation in a hardware-based, attested Trusted Execution Environment."

According to the [Remote ATtestation procedureS \(RATS\) Architecture](#) In remote attestation, "one peer (the "Attester") produces believable information about itself ("Evidence") to enable a remote peer (the "Relying Party") to decide whether to consider that Attester a trustworthy peer. Remote attestation procedures are facilitated by an additional vital party (the "Verifier")." In simpler terms, attestation is a way of proving that a computer system is trustworthy.

In Confidential Containers on Azure Container Instances (ACI), you can use an attestation token to verify that container group:

- Is running on confidential computing hardware. In this case AMD SEV-SNP.
- Is running on an Azure compliant utility VM.
- Is enforcing the expected confidential computing enforcement (CCE) policy that was generated using [tooling](#).

## Full attestation

Expanding upon this concept of attestation. Full attestation captures all the components that are part of the Trusted Execution Environment that is remotely verifiable. To achieve full attestation, in Confidential Containers, we introduce the notion of a CCE policy, which defines a set of rules, which is enforced in the utility VM. The security policy is encoded in the attestation report as an SHA-256 digest stored in the HostData attribute, as provided to the AMD SEV-SNP hardware by the host operating system during the VM boot-up. This structure means that the security policy enforced by the utility VM is immutable throughout the lifetime of the utility VM.

The exhaustive list of attributes that are part of the SEV-SNP attestation can be found [here](#).

Some important fields to consider in an attestation token returned by [Microsoft Azure Attestation \(MAA\)](#)

 Expand table

| Claim                       | Sample value                                                     | Description                                                                                     |
|-----------------------------|------------------------------------------------------------------|-------------------------------------------------------------------------------------------------|
| x-ms-attestation-type       | sevsnpvm                                                         | String value that describes the attestation type. For example, in this scenario sevsnp hardware |
| x-ms-compliance-status      | azure-compliant-uvm                                              | Compliance status of the utility VM that runs the container group.                              |
| x-ms-sevsnpvm-hostdata      | 670fff86714a650a49b58fadc1e90fedae0eb32dd51e34931c1e7a1839c08f6f | Hash of the CCE policy that was generated using tooling during deployment.                      |
| x-ms-sevsnpvm-is-debuggable | false                                                            | Flag to indicate whether the underlying hardware is running in debug mode                       |

# Sample attestation token generated by MAA

JSON

```
{
 "header": {
 "alg": "RS256",
 "jku": "https://sharedeus2.eus2.test.attest.azure.net/certs",
```



```
"a489c8578fb2f54d895fc8d000a85b2ff4855c015e4fb7216495c4dba4598345",
 "x-ms-sevsnpvm-smt-allowed": true,
 "x-ms-sevsnpvm-snpfw-svn": 8,
 "x-ms-sevsnpvm-tee-svn": 0,
 "x-ms-sevsnpvm-uvm-endorsement": {
 "x-ms-sevsnpvm-guestsvn": "100",
 "x-ms-sevsnpvm-launchmeasurement":
 "a1e1a4b64e8de5c664ceee069010441f74cf039065b5b847e82b9d1a7629aaaf33d5591c6b18
 cee48a4dde481aa88d0fb"
 },
 "x-ms-sevsnpvm-vmpl": 0,
 "x-ms-ver": "1.0"
}
}
```

## Generating an attestation token

We have open-source sidecar container implementations that provide an easy REST interface to get a raw SNP (Secure Nested Paging) report produced by the hardware or a MAA token. The sidecar is available at this [repository](#) and can be deployed with your container group.

## Next steps

- [Learn how to use attestation to release a secret to your container group](#)
- [Deploy a confidential container group with Azure Resource Manager](#)

---

## Feedback

Was this page helpful?



Yes



No

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

# Azure Container Instances Spot containers (preview)

Article • 08/29/2024

This article introduces the concept of Azure Container Instances (ACI) Spot containers, which allows you to run interruptible workloads in containerized form on unused Azure capacity. By utilizing Spot containers, you can enjoy up to a 70% discount compared to regular-priority ACI containers.

Spot containers offer the best of both worlds by combining the simplicity of ACI with the cost-effectiveness of Spot VMs. This enables customers to easily and affordably scale their containerized interruptible workloads. It's important to note that Spot containers may be preempted at any time, particularly when Azure limits surplus capacity.

Customers are billed based on per-second memory and core usage.

This feature is designed for customers who need to run interruptible workloads with no strict availability requirements. Azure Container Instances Spot Containers support both Linux and Windows containers, providing flexibility for different operating system environments.

This article provides background about the feature, limitations, and resources. To see the availability of Spot containers in Azure regions, see [Resource and quota limits](#).

## ⓘ Note

Spot containers with Azure Container Instances is in preview and is not recommended for production scenarios.

## Azure Container Instances Spot containers overview

### Lift and shift applications

ACI Spot containers are a cost-effective option for running containerized applications or parallelizable offline workloads including image rendering, Genomic processing, Monte Carlo simulations, etc. Customers can lift and shift their containerized Linux or Windows applications without needing to adopt specialized programming models to achieve the benefits of standard ACI containers along with low cost.

# Eviction policies

For Spot containers, customers can't choose eviction types or policies like Spot VMs. If an eviction occurs, the container groups hosting the customer workloads are automatically restarted without requiring any action from the customer.

## Unsupported features

ACI Spot containers preview release includes these limitations such as:

- **Public IP Endpoint:** ACI Spot container groups aren't assigned a public IP endpoint. This means the container groups can't be accessed directly from the internet.
- **Deployment Behind Virtual Network:** Spot container groups can't be deployed behind a virtual network.
- **Confidential SKU Support:** ACI Spot containers don't support the Confidential SKU, which means that you can't use the Confidential Computing capabilities provided by Azure.
- **Availability Zone Pinning:** ACI Spot containers don't support the ability to pin Availability Zones per container group deployment.

## Next steps

- For a deployment example with the Azure portal, see [Deploy a Spot container with Azure Container Instances using the Azure portal](#)
- For a deployment example with the Azure CLI, see [Deploy a Spot container with Azure Container Instances using the Azure CLI](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# Virtual nodes on Azure Container Instances

Article • 11/13/2024

This article introduces how virtual nodes on Azure Container Instances (ACI) enable you to orchestrate containerized workloads. This article provides background about the feature set, availability, and resources.

Virtual nodes on Azure Container Instances allow you to deploy pods in your Azure Kubernetes Service (AKS) cluster that run as container groups in ACI. Since virtual nodes are backed by ACI's serverless infrastructure, you can quickly scale up your workload without needing to wait for the Kubernetes cluster autoscaler to deploy VM compute nodes. You can rapidly scale your application workloads and only pay for the time that your pods are running.

Virtual nodes on Azure Container Instances is an evolution on the existing [virtual nodes](#) offering for AKS. It offers significantly better support for Kubernetes functionality, removing most of the limitations of the prior implementation. Additionally, it enables you to take advantage of advanced ACI functionality like [confidential containers](#).

Virtual nodes on Azure Container Instances was featured in the AKS keynote at KubeCon NA 2023, with a [segment showing some of its many improvements ↗](#).

## Features of virtual nodes on Azure Container Instances

### Rapidly scale your application workloads

Pods deployed in a virtual node run on ACI's serverless infrastructure. You don't need to first provision additional VM compute nodes in your AKS cluster when scaling up your workload.

### Only pay per second of execution time

With standard AKS cluster nodes, you pay for provisioned VMs even if you are not actively utilizing their capacity. Pods in a virtual node use ACI's per-second billing, so you pay based on what you need.

# Deploy confidential container groups

Virtual nodes can be configured to run your pods as [confidential container groups in ACI](#). Confidential containers on Azure Container Instances can run with verifiable execution policies that enable customers to have control over what software and actions are allowed to run within the TEE. These execution policies help to protect against bad actors creating unexpected application modifications that could potentially leak sensitive data. Customers author execution policies through provided [tooling ↗](#), and cryptographic proofs verify the policies.

## Region availability

Virtual nodes on Azure Container Instances is supported in all regions where ACI is available. Virtual nodes on confidential Azure Container Instances is supported in all regions where confidential ACI is available.

For more information, see [Resource availability & quota limits for ACI](#).

## Resources

- [Helm chart for deploying virtual nodes on Azure Container Instances ↗](#)
- [Guidance for customizing pods running on virtual nodes ↗](#)
- [Guidance for customizing your virtual nodes installation ↗](#)
- [Guidance for deploying high-scale workloads on virtual nodes ↗](#)

## Next steps

- To try out virtual nodes on Azure Container Instances, see [Deploy virtual nodes on Azure Container Instances in your AKS cluster](#)
- If you want to manage multi-instance container groups, see [About NGroups](#).

---

## Feedback

Was this page helpful?



Yes



No

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

# Deploy container instances into an Azure virtual network

Article • 12/11/2024

Azure Virtual Network provides secure, private networking for your Azure and on-premises resources. By deploying container groups into an Azure virtual network, your containers can communicate securely with other resources in the virtual network.

This article shows how to use the [az container create](#) command in the Azure CLI to deploy container groups to either a new virtual network or an existing virtual network.

## Important

- Subnets must be delegated before using a virtual network
- Before deploying container groups in virtual networks, we suggest checking the limitation first. For networking scenarios and limitations, see [Virtual network scenarios and resources for Azure Container Instances](#).
- Container group deployment to a virtual network is generally available for Linux and Windows containers, in most regions where Azure Container Instances is available. For details, see [available-regions](#).

## Important

[Network profiles](#) have been retired as of the 2021-07-01 API version. If you're using this or a more recent version, ignore any steps and actions related to network profiles.

Examples in this article are formatted for the Bash shell. If you prefer another shell such as PowerShell or Command Prompt, adjust the line continuation characters accordingly.

## Prerequisites

### Define environment variables

The automated deployment pathway uses the following environment variables and resource names throughout this guide. Users proceeding through the guide manually can use their own variables and names as preferred.

## Azure CLI

```
export RANDOM_ID="$(openssl rand -hex 3)"
export MY_RESOURCE_GROUP_NAME="myACIResourceGroup$RANDOM_ID"
export MY_VNET_NAME="aci-vnet"
export MY_SUBNET_NAME="aci-subnet"
export MY_SUBNET_ID="/subscriptions/$(az account show --query id --output tsv)/resourceGroups/$MY_RESOURCE_GROUP_NAME/providers/Microsoft.Network/virtualNetworks/$MY_VNET_NAME/subnets/$MY_SUBNET_NAME"
export MY_APP_CONTAINER_NAME="appcontainer"
export MY_COMM_CHECKER_NAME="commchecker"
export MY_YAML_APP_CONTAINER_NAME="appcontaineryaml"
```

## Create a resource group

You need a resource group to manage all the resources used in the following examples. To create a resource group, use [az group create](#):

### Azure CLI

```
az group create --name $MY_RESOURCE_GROUP_NAME --location eastus
```

A successful operation should produce output similar to the following JSON:

Results:

### JSON

```
{
 "id": "/subscriptions/xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxx/resourceGroups/myACIResourceGroup123abc",
 "location": "abcdef",
 "managedBy": null,
 "name": "myACIResourceGroup123",
 "properties": {
 "provisioningState": "Succeeded"
 },
 "tags": null,
 "type": "Microsoft.Resources/resourceGroups"
}
```

## Deploy to new virtual network

### ⓘ Note

If you are using subnet IP range /29 to have only 3 IP addresses, we recommend always to go one range above (never below). For example, use subnet IP range /28 so you can have at least 1 or more IP buffer per container group. By doing this, you can avoid containers in stuck, not able to start, restart or even not able to stop states.

To deploy to a new virtual network and have Azure create the network resources for you automatically, specify the following when you execute [az container create](#):

- Virtual network name
- Virtual network address prefix in CIDR format
- Subnet name
- Subnet address prefix in CIDR format

The virtual network and subnet address prefixes specify the address spaces for the virtual network and subnet, respectively. These values are represented in Classless Inter-Domain Routing (CIDR) notation, for example `10.0.0.0/16`. For more information about working with subnets, see [Add, change, or delete a virtual network subnet](#).

Once you deploy your first container group with this method, you can deploy to the same subnet by specifying the virtual network and subnet names, or the network profile that Azure automatically creates for you. Because Azure delegates the subnet to Azure Container Instances, you can deploy *only* container groups to the subnet.

## Example

The following [az container create](#) command specifies settings for a new virtual network and subnet. Provide the name of a resource group that was created in a region where container group deployments in a virtual network are [available](#). This command deploys the public Microsoft aci-helloworld container that runs a small Node.js webserver serving a static web page. In the next section, you'll deploy a second container group to the same subnet, and test communication between the two container instances.

Azure CLI

```
az container create \
--name $MY_APP_CONTAINER_NAME \
--resource-group $MY_RESOURCE_GROUP_NAME \
--image mcr.microsoft.com/azuredocs/aci-helloworld \
--vnet $MY_VNET_NAME \
--vnet-address-prefix 10.0.0.0/16 \
--subnet $MY_SUBNET_NAME \
--subnet-address-prefix 10.0.0.0/24
```

A successful operation should produce output similar to the following JSON:

## Results:

## JSON

```
 },
 "livenessProbe": null,
 "name": "appcontainer",
 "ports": [
 {
 "port": 80,
 "protocol": "TCP"
 }
],
 "readinessProbe": null,
 "resources": {
 "limits": null,
 "requests": {
 "cpu": 1.0,
 "gpu": null,
 "memoryInGb": 1.5
 }
 },
 "securityContext": null,
 "volumeMounts": null
 }
],
"diagnostics": null,
"dnsConfig": null,
"encryptionProperties": null,
"extensions": null,
"id": "/subscriptions/xxxxxxxx-xxxx-xxxx-xxxx-
xxxxxxxx/resourceGroups/myACIResourceGroup123/providers/Microsoft.Containe
rInstance/containerGroups/appcontainer",
"identity": null,
"imageRegistryCredentials": null,
"initContainers": [],
"instanceView": {
 "events": [],
 "state": "Running"
},
"ipAddress": {
 "autoGeneratedDomainNameLabelScope": null,
 "dnsNameLabel": null,
 "fqdn": null,
 "ip": "10.0.0.4",
 "ports": [
 {
 "port": 80,
 "protocol": "TCP"
 }
],
 "type": "Private"
},
"location": "eastus",
"name": "appcontainer",
"osType": "Linux",
"priority": null,
"provisioningState": "Succeeded",
"resourceGroup": "myACIResourceGroup123abc",
```

```

 "restartPolicy": "Always",
 "sku": "Standard",
 "subnetIds": [
 {
 "id": "/subscriptions/xxxxxxxx-xxxx-xxxx-xxxx-
xxxxxxxx/resourceGroups/myACIResourceGroup123/providers/Microsoft.Network/
virtualNetworks/aci-vnet/subnets/aci-subnet",
 "name": null,
 "resourceGroup": "myACIResourceGroup123abc"
 }
],
 "tags": {},
 "type": "Microsoft.ContainerInstance/containerGroups",
 "volumes": null,
 "zones": null
}

```

When you deploy to a new virtual network by using this method, the deployment can take a few minutes while the network resources are created. After the initial deployment, further container group deployments to the same subnet complete more quickly.

## Deploy to existing virtual network

To deploy a container group to an existing virtual network:

1. Create a subnet within your existing virtual network, use an existing subnet in which a container group is already deployed, or use an existing subnet emptied of *all* other resources and configuration. The subnet that you use for container groups can contain only container groups. Before you deploy a container group to a subnet, you must explicitly delegate the subnet before provisioning. Once delegated, the subnet can be used only for container groups. If you attempt to deploy resources other than container groups to a delegated subnet, the operation fails.
2. Deploy a container group with [az container create](#) and specify one of the following:
  - Virtual network name and subnet name
  - Virtual network resource ID and subnet resource ID, which allows using a virtual network from a different resource group

## Deploy using a YAML file

You can also deploy a container group to an existing virtual network by using a YAML file, a [Resource Manager template](#), or another programmatic method such as with the Python SDK.

For example, when using a YAML file, you can deploy to a virtual network with a subnet delegated to Azure Container Instances. Specify the following properties:

- `ipAddress`: The private IP address settings for the container group.
  - `ports`: The ports to open, if any.
  - `protocol`: The protocol (TCP or UDP) for the opened port.
- `subnetIds`: The resource IDs of the subnets to be deployed to
  - `id`: The resource ID of the subnet
  - `name`: The name of the subnet

This YAML creates a container group in your virtual network. Enter your container group name in the name fields and your subnet ID in the subnet ID field. We use `appcontaineryaml` for the name. If you need to find your subnet ID and no longer have access to previous outputs, you can use the [az container show](#) command to view it. Look for the `id` field under `subnetIds`.

```
YAML

apiVersion: '2021-07-01'
location: eastus
name: appcontaineryaml
properties:
 containers:
 - name: appcontaineryaml
 properties:
 image: mcr.microsoft.com/azuredocs/aci-helloworld
 ports:
 - port: 80
 protocol: TCP
 resources:
 requests:
 cpu: 1.0
 memoryInGB: 1.5
 ipAddress:
 type: Private
 ports:
 - protocol: tcp
 port: '80'
 osType: Linux
 restartPolicy: Always
 subnetIds:
 - id: <subnet_id>
 name: default
 tags: null
 type: Microsoft.ContainerInstance/containerGroups
```

The following Bash command is for the automated deployment pathway.

#### Bash

```
echo -e "apiVersion: '2021-07-01'\nlocation: eastus\nname: $MY_APP_CONTAINER_NAME\nproperties:\n containers:\n - name: $MY_APP_CONTAINER_NAME\n properties:\n image: mcr.microsoft.com/azuredocs/aci-helloworld\n ports:\n - port: 80\n protocol: TCP\n resources:\n requests:\n cpu: 1.0\n memoryInGB: 1.5\n ipAddress:\n type: Private\n ports:\n -\n protocol: tcp\n port: '80'\n osType: Linux\n restartPolicy: Always\n subnetIds:\n - id: $MY_SUBNET_ID\n name: default\n tags: null\n type: Microsoft.ContainerInstance/containerGroups" > container-instances-vnet.yaml
```

Deploy the container group with the [az container create](#) command, specifying the YAML file name for the `--file` parameter:

#### Azure CLI

```
az container create --resource-group $MY_RESOURCE_GROUP_NAME \
--file container-instances-vnet.yaml
```

The following Bash command is for the automated deployment pathway.

#### Bash

```
rm container-instances-vnet.yaml
```

Once the deployment completes, run the [az container show](#) command to display its status:

#### Azure CLI

```
az container list --resource-group $MY_RESOURCE_GROUP_NAME --output table
```

The output should resemble the sample below:

Results:

#### Output

| Name                                       | ResourceGroup            | Status     | Image  |
|--------------------------------------------|--------------------------|------------|--------|
| IP:ports                                   | Network                  | CPU/Memory | OsType |
| appcontainer                               | myACIResourceGroup123abc | Succeeded  |        |
| mcr.microsoft.com/azuredocs/aci-helloworld | 10.0.0.4:80,80           | Private    | 1.0    |
| core/1.5 gb                                | Linux                    | abcdef     |        |

```
appcontaineryaml myACIResourceGroup123abc Succeeded
mcr.microsoft.com/azuredocs/aci-helloworld 10.0.0.5:80,80 Private 1.0
core/1.5 gb Linux abcdef
```

## Demonstrate communication between container instances

The following example deploys a third container group to the same subnet created previously. Using an Alpine Linux image, it verifies communication between itself and the first container instance.

### ⓘ Note

Due to rate limiting in effect for pulling public Docker images like the Alpine Linux one used here, you may receive an error in the form:

(RegistryErrorResponse) An error response is received from the docker registry 'index.docker.io'. Please retry later. Code: RegistryErrorResponse Message: An error response is received from the docker registry 'index.docker.io'. Please retry later.

The following Bash command is for the automated deployment pathway.

#### Bash

```
echo -e "Due to rate limiting in effect for pulling public Docker images
like the Alpine Linux one used here, you may receive an error in the
form:\n\n(RegistryErrorResponse) An error response is received from the
docker registry 'index.docker.io'. Please retry later.\nCode:
RegistryErrorResponse\nMessage: An error response is received from the
docker registry 'index.docker.io'. Please retry later.\n\nIf this occurs,
the automated deployment will exit. You can try again or go to the end of
the guide to see instructions for cleaning up your resources."
```

First, get the IP address of the first container group you deployed, the *appcontainer*:

#### Azure CLI

```
az container show --resource-group $MY_RESOURCE_GROUP_NAME \
--name $MY_APP_CONTAINER_NAME \
--query ipAddress.ip --output tsv
```

The output displays the IP address of the container group in the private subnet. For example:

Results:

Output

10.0.0.4

Now, set `CONTAINER_GROUP_IP` to the IP you retrieved with the `az container show` command, and execute the following `az container create` command. This second container, *commchecker*, runs an Alpine Linux-based image and executes `wget` against the first container group's private subnet IP address.

Azure CLI

```
az container create \
--resource-group $MY_RESOURCE_GROUP_NAME \
--name $MY_COMM_CHECKER_NAME \
--image alpine:3.4 \
--command-line "wget 10.0.0.4" \
--restart-policy never \
--vnet $MY_VNET_NAME \
--subnet $MY_SUBNET_NAME
```

After this second container deployment completes, pull its logs so you can see the output of the `wget` command it executed:

Azure CLI

```
az container logs --resource-group $MY_RESOURCE_GROUP_NAME --name
$MY_COMM_CHECKER_NAME
```

If the second container communicated successfully with the first, output is similar to:

Output

```
Connecting to 10.0.0.4 (10.0.0.4:80)
index.html 100% |*****| 1663 0:00:00
ETA
```

The log output should show that `wget` was able to connect and download the index file from the first container using its private IP address on the local subnet. Network traffic between the two container groups remained within the virtual network.

## Clean up resources

If you don't plan to continue using these resources, you can delete them to avoid Azure charges. You can clean up all the resources you used in this guide by deleting the resource group with the [az group delete](#) command. Once deleted, **these resources are unrecoverable**.

## Next steps

- To deploy a new virtual network, subnet, network profile, and container group using a Resource Manager template, see [Create an Azure container group with virtual network](#).
  - To deploy Azure Container Instances that can pull images from an Azure Container Registry through a private endpoint, see [Deploy to Azure Container Instances from Azure Container Registry using a managed identity](#).
- 

## Feedback

Was this page helpful?

 Yes

 No

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

# Deploy to Azure Container Instances from Azure Container Registry using a service principal

Article • 08/29/2024

Azure Container Registry is an Azure-based, managed container registry service used to store private Docker container images. This article describes how to pull container images stored in an Azure container registry when deploying to Azure Container Instances. One way to configure registry access is to create a Microsoft Entra service principal and password, and store the sign-in credentials in an Azure key vault.

## Prerequisites

**Azure container registry:** You need an Azure container registry--and at least one container image in the registry--to complete the steps in this article. If you need a registry, see [Create a container registry using the Azure CLI](#).

**Azure CLI:** The command-line examples in this article use the [Azure CLI](#) and are formatted for the Bash shell. You can [install the Azure CLI](#) locally, or use the [Azure Cloud Shell](#) ↗.

## Limitations

- Windows containers don't support system-assigned managed identity-authenticated image pulls with ACR, only user-assigned.

## Configure registry authentication

In a production scenario where you provide access to "headless" services and applications, we recommend you configure registry access by using a [service principal](#). A service principal allows you to provide [Azure role-based access control \(Azure RBAC\)](#) to your container images. For example, you can configure a service principal with pull-only access to a registry.

Azure Container Registry provides more [authentication options](#).

In the following section, you create an Azure key vault and a service principal, and store the service principal's credentials in the vault.

## Create key vault

If you don't already have a vault in [Azure Key Vault](#), create one with the Azure CLI using the following commands.

Update the `RES_GROUP` variable with the name of an existing resource group in which to create the key vault, and `ACR_NAME` with the name of your container registry. For brevity, commands in this article assume that your registry, key vault, and container instances are all created in the same resource group.

Specify a name for your new key vault in `AKV_NAME`. The vault name must be unique within Azure and must be 3-24 alphanumeric characters in length, begin with a letter, end with a letter or digit, and can't contain consecutive hyphens.

### Azure CLI

```
RES_GROUP=myresourcegroup # Resource Group name
ACR_NAME=myregistry # Azure Container Registry registry name
AKV_NAME=mykeyvault # Azure Key Vault vault name

az keyvault create -g $RES_GROUP -n $AKV_NAME
```

## Create service principal and store credentials

Now create a service principal and store its credentials in your key vault.

The following commands use `az ad sp create-for-rbac` to create the service principal, and `az keyvault secret set` to store the service principal's `password` in the vault. Be sure to take note of the service principal's `appId` upon creation.

### Azure CLI

```
Create service principal
az ad sp create-for-rbac \
--name http://$ACR_NAME-pull \
--scopes $(az acr show --name $ACR_NAME --query id --output tsv) \
--role acrpull

SP_ID=xxxx # Replace with your service principal's appId

Store the registry *password* in the vault
az keyvault secret set \
--vault-name $AKV_NAME \
--name $ACR_NAME-pull-pwd \
--value $(az ad sp show --id $SP_ID --query password --output tsv)
```

The `--role` argument in the preceding command configures the service principal with the *acrpull* role, which grants it pull-only access to the registry. To grant both push and pull access, change the `--role` argument to *acrpush*.

Next, store the service principal's *appId* in the vault, which is the **username** you pass to Azure Container Registry for authentication.

#### Azure CLI

```
Store service principal ID in vault (the registry *username*)
az keyvault secret set \
 --vault-name $AKV_NAME \
 --name $ACR_NAME-pull-usr \
 --value $(az ad sp show --id $SP_ID --query appId --output tsv)
```

You created an Azure key vault and stored two secrets in it:

- `$ACR_NAME-pull-usr`: The service principal ID, for use as the container registry **username**.
- `$ACR_NAME-pull-pwd`: The service principal password, for use as the container registry **password**.

You can now reference these secrets by name when you or your applications and services pull images from the registry.

## Deploy container with Azure CLI

Now that the service principal credentials are stored in Azure Key Vault secrets, your applications and services can use them to access your private registry.

First get the registry's login server name by using the `az acr show` command. The login server name is all lowercase and similar to `myregistry.azurecr.io`.

#### Azure CLI

```
ACR_LOGIN_SERVER=$(az acr show --name $ACR_NAME --resource-group $RES_GROUP \
--query "loginServer" --output tsv)
```

Execute the following `az container create` command to deploy a container instance. The command uses the service principal's credentials stored in Azure Key Vault to authenticate to your container registry, and assumes you previously pushed the `aci-helloworld` image to your registry. Update the `--image` value if you'd like to use a different image from your registry.

## Azure CLI

```
az container create \
 --name aci-demo \
 --resource-group $RES_GROUP \
 --image $ACR_LOGIN_SERVER/aci-helloworld:v1 \
 --registry-login-server $ACR_LOGIN_SERVER \
 --registry-username $(az keyvault secret show --vault-name $AKV_NAME -n $ACR_NAME-pull-usr --query value -o tsv) \
 --registry-password $(az keyvault secret show --vault-name $AKV_NAME -n $ACR_NAME-pull-pwd --query value -o tsv) \
 --dns-name-label aci-demo-$RANDOM \
 --query ipAddress.fqdn
```

The `--dns-name-label` value must be unique within Azure, so the preceding command appends a random number to the container's DNS name label. The output from the command displays the container's fully qualified domain name (FQDN), for example:

### Output

```
"aci-demo-25007.eastus.azurecontainer.io"
```

Once the container starts successfully, you can navigate to its FQDN in your browser to verify the application is running successfully.

## Deploy with Azure Resource Manager template

You can specify the properties of your Azure container registry in an Azure Resource Manager template by including the `imageRegistryCredentials` property in the container group definition. For example, you can specify the registry credentials directly:

### JSON

```
[...]
"imageRegistryCredentials": [
 {
 "server": "imageRegistryLoginServer",
 "username": "imageRegistryUsername",
 "password": "imageRegistryPassword"
 }
]
[...]
```

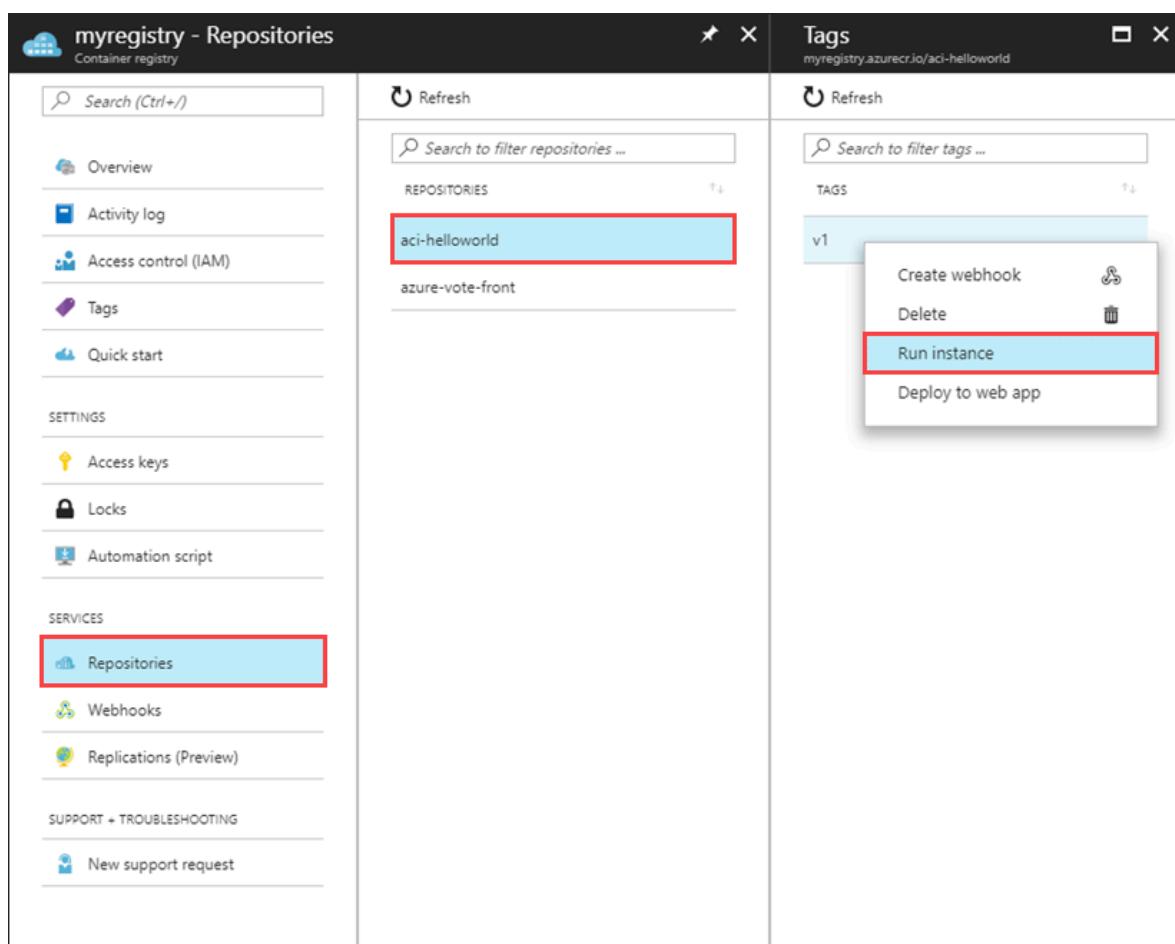
For complete container group settings, see the [Resource Manager template reference](#).

For details on referencing Azure Key Vault secrets in a Resource Manager template, see [Use Azure Key Vault to pass secure parameter value during deployment](#).

## Deploy with Azure portal

If you maintain container images in an Azure container registry, you can easily create a container in Azure Container Instances using the Azure portal. When using the portal to deploy a container instance from a container registry, you must enable the registry's [admin account](#). The admin account is designed for a single user to access the registry, mainly for testing purposes.

1. In the Azure portal, navigate to your container registry.
2. To confirm that the admin account is enabled, select **Access keys**, and under **Admin user** select **Enable**.
3. Select **Repositories**, then select the repository that you want to deploy from, right-click the tag for the container image you want to deploy, and select **Run instance**.



4. Enter a name for the container and a name for the resource group. You can also change the default values if you wish.

**Create container instance**

**Container name \***  
mycontainer ✓

**Container image**  
myregistry.azurecr.io/aci-helloworld:v1

**OS type**  
 Linux  Windows

**Subscription \***  
Visual Studio Enterprise with MSDN ▾

**Resource group \***  
myresourcegroup ▾  
[Create new](#)

**Location \***  
East US ▾

**Number of cores**  
1 ▾

**Memory (GB) \***  
1.5

**Public IP address**  
 Yes  No

**Port \***  
80

**OK**

5. Once the deployment completes, you can navigate to the container group from the notifications pane to find its IP address and other properties.

mycontainer  
Container group

Search (Ctrl+ /)

Overview

Activity log

Access control (IAM)

Tags

SETTINGS

Properties

Locks

Automation script

Delete

Resource group (change)  
myResourceGroup

Status  
Running

Location  
East US

Subscription (change)  
Visual Studio Enterprise with MSDN

Subscription ID  
<Subscription ID>

OS type  
Linux

IP address  
52.170.115.171

1 container

| NAME        | IMAGE                          | STATE   | START TIME           | RESTART COUNT |
|-------------|--------------------------------|---------|----------------------|---------------|
| mycontainer | myregistry.azurecr.io/aci-h... | Running | 2017-12-22T22:44:41Z | 0             |

## Next steps

For more information about Azure Container Registry authentication, see [Authenticate with an Azure container registry](#).

---

## Feedback

Was this page helpful?

Yes

No

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

# Deploy to Azure Container Instances from Azure Container Registry using a managed identity

Article • 08/29/2024

Azure Container Registry (ACR) is an Azure-based, managed container registry service used to store private Docker container images. This article describes how to pull container images stored in an Azure container registry when deploying to container groups with Azure Container Instances. One way to configure registry access is to create a Microsoft Entra managed identity.

When access to an Azure Container Registry (ACR) is [restricted using a private endpoint](#), using a managed identity allows Azure Container Instances [deployed into a virtual network](#) to access the container registry through the private endpoint.

## Prerequisites

**Azure container registry:** You need a premium SKU Azure container registry with at least one image. If you need to create a registry, see [Create a container registry using the Azure CLI](#). Be sure to take note of the registry's `id` and `loginServer`.

**Azure CLI:** The command-line examples in this article use the [Azure CLI](#) and are formatted for the Bash shell. You can [install the Azure CLI](#) locally, or use the [Azure Cloud Shell](#).

## Limitations

- Windows containers don't support system-assigned managed identity-authenticated image pulls with ACR, only user-assigned.

## Configure registry authentication

Your container registry must have Trusted Services enabled. To find instructions on how to enable trusted services, see [Allow trusted services to securely access a network-restricted container registry](#).

## Create an identity

Create an identity in your subscription using the [az identity create](#) command. You can use the same resource group you used previously to create the container registry, or a different one.

Azure CLI

```
az identity create --resource-group myResourceGroup --name myACRId
```

To configure the identity in the following steps, use the [az identity show](#) command to store the identity's resource ID and service principal ID in variables.

In order to properly configure the identity in future steps, use [az identity show](#) to obtain and store the identity's resource ID and service principal ID in variables.

Azure CLI

```
Get resource ID of the user-assigned identity
USERID=$(az identity show --resource-group myResourceGroup --name myACRId --query id --output tsv)
Get service principal ID of the user-assigned identity
SPID=$(az identity show --resource-group myResourceGroup --name myACRId --query principalId --output tsv)
```

You need the identity's resource ID to sign in to the CLI from your virtual machine. To show the value:

Bash

```
echo $USERID
```

The resource ID is of the form:

Bash

```
/subscriptions/xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx/resourcegroups/myResourceGroup/providers/Microsoft.ManagedIdentity/userAssignedIdentities/myACRId
```

You'll also need the service principal ID to grant the managed identity access to your container registry. To show the value:

Bash

```
echo $SPID
```

The service principal ID is of the form:

Bash

```
xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx
```

## Grant the identity a role assignment

In order for your identity to access your container registry, you must grant it a role assignment. Use the following command to grant the `acrpull` role to the identity you created, making sure to provide your registry's ID and the service principal we obtained earlier:

Azure CLI

```
az role assignment create --assignee $SPID --scope <registry-id> --role acrpull
```

## Deploy using an Azure Resource Manager (ARM) template

Start by copying the following JSON into a new file named `azuredeploy.json`. In Azure Cloud Shell, you can use Visual Studio Code to create the file in your working directory:

Bash

```
code azuredeploy.json
```

You can specify the properties of your Azure container registry in an ARM template by including the `imageRegistryCredentials` property in the container group definition. For example, you can specify the registry credentials directly:

### ⓘ Note

This is not a comprehensive ARM template, but rather an example of what the `resources` section of a complete template would look like.

JSON

```
{
 "type": "Microsoft.ContainerInstance/containerGroups",
 "apiVersion": "2021-09-01",
 "name": "myContainerGroup",
 "location": "norwayeast",
 "identity": {
 "type": "UserAssigned",
 "userAssignedIdentities": {
 "/subscriptions/xxxxxxxx-xxxx-xxxx-xxxx-
xxxxxxxxxxxx/resourcegroups/myResourceGroup/providers/Microsoft.ManagedIdentity/userAssignedIdentities/myACRId": {}
 }
 },
 "properties": {
 "containers": [
 {
 "name": "mycontainer",
 "properties": {
 "image": "myacr.azurecr.io/hello-world:latest",
 "ports": [
 {
 "port": 80,
 "protocol": "TCP"
 }
],
 "resources": {
 "requests": {
 "cpu": 1,
 "memoryInGB": 1
 }
 }
 }
 }
],
 "imageRegistryCredentials": [
 {
 "server": "myacr.azurecr.io",
 "identity": "/subscriptions/xxxxxxxx-xxxx-xxxx-xxxx-
xxxxxxxxxxxx/resourcegroups/myResourceGroup/providers/Microsoft.ManagedIdentity/userAssignedIdentities/myACRId"
 }
],
 "ipAddress": {
 "ports": [
 {
 "port": 80,
 "protocol": "TCP"
 }
],
 "type": "public"
 },
 "osType": "Linux"
 }
}
```

```
}
```

## Deploy the template

Deploy your Resource Manager template with the following command:

Azure CLI

```
az deployment group create --resource-group myResourceGroup --template-file
azuredeploy.json
```

## Deploy using the Azure CLI

To deploy a container group using managed identity to authenticate image pulls via the Azure CLI, use the following command, making sure that your <dns-label> is globally unique:

Azure CLI

```
az container create --name my-containergroup --resource-group
myResourceGroup --image <loginServer>/hello-world:v1 --acr-identity $USERID
--assign-identity $USERID --ports 80 --dns-name-label <dns-label>
```

## Deploy in a virtual network using the Azure CLI

To deploy a container group to a virtual network using managed identity to authenticate image pulls from an ACR that runs behind a private endpoint via the Azure CLI, use the following command:

Azure CLI

```
az container create --name my-containergroup --resource-group
myResourceGroup --image <loginServer>/hello-world:v1 --acr-identity $USERID
--assign-identity $USERID --vnet
"/subscriptions/$SUBSCRIPTION_ID/resourceGroups/" /subscriptions/xxxxxxxx-
xxxx-xxxx-xxxx-xxxxxxxxxxxx/resourceGroups/myVNetResourceGroup/providers/ -
-subnet mySubnetName
```

For more info on how to deploy to a virtual network see [Deploy container instances into an Azure virtual network](#).

# Deploy a multi-container group in a virtual network using YAML and the Azure CLI

To deploy a multi-container group to a virtual network using managed identity to authenticate image pulls from an ACR that runs behind a private endpoint via the Azure CLI, you can specify the container group configuration in a YAML file. Then pass the YAML file as a parameter to the command.

YAML

```
apiVersion: '2021-10-01'
location: eastus
type: Microsoft.ContainerInstance/containerGroups
identity:
 type: UserAssigned
 userAssignedIdentities: {
 '/subscriptions/xxxxxxxx-xxxx-xxxx-xxxx-
xxxxxxxxxxxx/resourcegroups/myResourceGroup/providers/Microsoft.ManagedIden
tity/userAssignedIdentities/myACRId': {}
 }
properties:
 osType: Linux
 imageRegistryCredentials:
 - server: myacr.azurecr.io
 identity: '/subscriptions/xxxxxxxx-xxxx-xxxx-xxxx-
xxxxxxxxxxxx/resourcegroups/myResourceGroup/providers/Microsoft.ManagedIden
tity/userAssignedIdentities/myACRId'
 subnetIds:
 - id: '/subscriptions/xxxxxxxx-xxxx-xxxx-xxxx-
xxxxxxxxxxxx/resourceGroups/myVNetResourceGroup/providers/Microsoft.Network
/virtualNetworks/myVNetName/subnets/mySubnetName'
 name: mySubnetName
 containers:
 - name: myContainer-1
 properties:
 resources:
 requests:
 cpu: '.4'
 memoryInGb: '1'
 environmentVariables:
 - name: CONTAINER
 value: 1
 image: 'myacr.azurecr.io/myimage:latest'
 - name: myContainer-2
 properties:
 resources:
 requests:
 cpu: '.4'
 memoryInGb: '1'
 environmentVariables:
 - name: CONTAINER
```

```
 value: 2
 image: 'myacr.azurecr.io/myimage:latest'
```

#### Azure CLI

```
az container create --name my-containergroup --resource-group
myResourceGroup --file my-YAML-file.yaml
```

For more info on how to deploy to a multi-container group see [Deploy a multi-container group](#).

## Clean up resources

To remove all resources from your Azure subscription, delete the resource group:

#### Azure CLI

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

## Next Steps

- Learn how to deploy to Azure Container Instances from Azure Container Registry using a service principal

## Feedback

Was this page helpful?

 Yes

 No

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

# Configure a GitHub Action to create a container instance

Article • 08/29/2024

[GitHub Actions](#) is a suite of features in GitHub to automate your software development workflows in the same place you store code and collaborate on pull requests and issues.

Use the [Deploy to Azure Container Instances](#) GitHub Actions to automate deployment of a single container to Azure Container Instances. The action allows you to set properties for a container instance similar to those in the `az container create` command.

This article shows how to set up a workflow in a GitHub repo that performs the following actions:

- Build an image from a Dockerfile
- Push the image to an Azure container registry
- Deploy the container image to an Azure container instance

This article shows two ways to set up the workflow:

- [Configure GitHub workflow](#) - Create a workflow in a GitHub repo using the Deploy to Azure Container Instances action and other actions.
- [Use CLI extension](#) - Use the `az container app up` command in the [Deploy to Azure](#) extension in the Azure CLI. This command streamlines creation of the GitHub workflow and deployment steps.

## Important

The GitHub Actions for Azure Container Instances is currently in preview. Previews are made available to you on the condition that you agree to the [supplemental terms of use](#). Some aspects of this feature may change prior to general availability (GA).

## Prerequisites

- **GitHub account** - Create an account on <https://github.com> if you don't already have one.
- **Azure CLI** - You can use the Azure Cloud Shell or a local installation of the Azure CLI to complete the Azure CLI steps. If you need to install or upgrade, see [Install](#)

## Azure CLI.

- **Azure container registry** - If you don't have one, create an Azure container registry in the Basic tier using the [Azure CLI](#), [Azure portal](#), or other methods. Take note of the resource group used for the deployment, which is used for the GitHub workflow.

## Set up repo

- For the examples in this article, use GitHub to fork the following repository:  
<https://github.com/Azure-Samples/acr-build-helloworld-node>

This repo contains a Dockerfile and source files to create a container image of a small web app.



- Ensure Actions is enabled for your repository. Navigate to your forked repository and select **Settings > Actions**. In **Actions permissions**, ensure that **Allow all actions** is selected.

## Configure GitHub workflow

### Create credentials for Azure authentication

In the GitHub workflow, you need to supply Azure credentials to authenticate to the Azure CLI. The following example creates a service principal with the Contributor role scoped to the resource group for your container registry.

First, get the resource ID of your resource group. Substitute the name of your group in the following [az group show](#) command:

Azure CLI

```
groupId=$(az group show \
--name <resource-group-name> \
--query id --output tsv)
```

Use [az ad sp create-for-rbac](#) to create the service principal:

Azure CLI

```
az ad sp create-for-rbac \
--scope $groupId \
--role Contributor \
--sdk-auth
```

Output is similar to:

JSON

```
{
 "clientId": "xxxx6ddc-xxxx-xxxx-xxx-ef78a99dxxxx",
 "clientSecret": "xxxx79dc-xxxx-xxxx-xxxx-aaaaaec5xxxx",
 "subscriptionId": "xxxx251c-xxxx-xxxx-xxxx-bf99a306xxxx",
 "tenantId": "xxxx88bf-xxxx-xxxx-xxxx-2d7cd011xxxx",
 "activeDirectoryEndpointUrl": "https://login.microsoftonline.com",
 "resourceManagerEndpointUrl": "https://management.azure.com/",
 "activeDirectoryGraphResourceId": "https://graph.windows.net/",
 "sqlManagementEndpointUrl": "https://management.core.windows.net:8443/",
 "galleryEndpointUrl": "https://gallery.azure.com/",
 "managementEndpointUrl": "https://management.core.windows.net/"
}
```

Save the JSON output because it's used in a later step. Also, take note of the `clientId`, which you need to update the service principal in the next section.

## Update for registry authentication

Update the Azure service principal credentials to allow push and pull access to your container registry. This step enables the GitHub workflow to use the service principal to [authenticate with your container registry](#) and to push and pull a Docker image.

Get the resource ID of your container registry. Substitute the name of your registry in the following `az acr show` command:

Azure CLI

```
registryId=$(az acr show \
--name <registry-name> \
--resource-group <resource-group-name> \
--query id --output tsv)
```

Use `az role assignment create` to assign the AcrPush role, which gives push and pull access to the registry. Substitute the client ID of your service principal:

Azure CLI

```
az role assignment create \
--assignee <ClientId> \
--scope $registryId \
--role AcrPush
```

## Save credentials to GitHub repo

1. In the GitHub UI, navigate to your forked repository and select **Security > Secrets and variables > Actions**.
2. Select **New repository secret** to add the following secrets:

[+] Expand table

| Secret                | Value                                                                                            |
|-----------------------|--------------------------------------------------------------------------------------------------|
| AZURE_CREDENTIALS     | The entire JSON output from the service principal creation step                                  |
| REGISTRY_LOGIN_SERVER | The login server name of your registry (all lowercase). Example:<br><i>myregistry.azurecr.io</i> |
| REGISTRY_USERNAME     | The <code>clientId</code> from the JSON output from the service principal creation               |
| REGISTRY_PASSWORD     | The <code>clientSecret</code> from the JSON output from the service principal creation           |
| RESOURCE_GROUP        | The name of the resource group you used to scope the service principal                           |

## Create workflow file

1. In the GitHub UI, select **Actions**.
2. Select **set up a workflow yourself**.
3. In **Edit new file**, paste the following YAML contents to overwrite the sample code.  
Accept the default filename `main.yml`, or provide a filename you choose.
4. Select **Start commit**, optionally provide short and extended descriptions of your commit, and select **Commit new file**.

```
yml

on: [push]
name: Linux Container Workflow

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

```

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

- name: 'Login via Azure CLI'
 uses: azure/login@v1
 with:
 creds: ${{ secrets.AZURE_CREDENTIALS }}

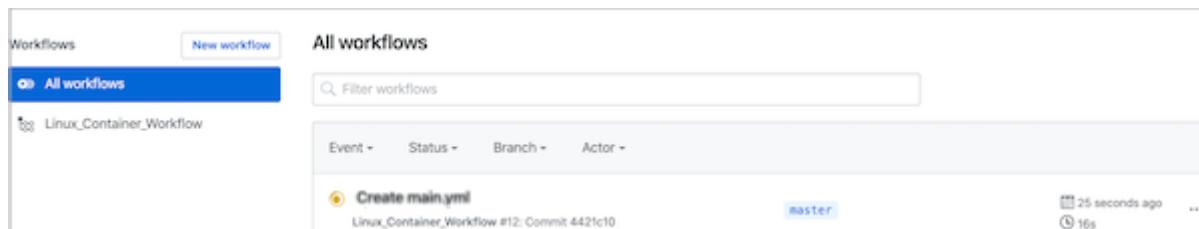
- name: 'Build and push image'
 uses: azure/docker-login@v1
 with:
 login-server: ${{ secrets.REGISTRY_LOGIN_SERVER }}
 username: ${{ secrets.REGISTRY_USERNAME }}
 password: ${{ secrets.REGISTRY_PASSWORD }}
- run:
 docker build . -t ${{ secrets.REGISTRY_LOGIN_SERVER }}/sampleapp:${{ github.sha }}
 docker push ${{ secrets.REGISTRY_LOGIN_SERVER }}/sampleapp:${{ github.sha }}

- name: 'Deploy to Azure Container Instances'
 uses: 'azure/aci-deploy@v1'
 with:
 resource-group: ${{ secrets.RESOURCE_GROUP }}
 dns-name-label: ${{ secrets.RESOURCE_GROUP }}${{ github.run_number }}
 image: ${{ secrets.REGISTRY_LOGIN_SERVER }}/sampleapp:${{ github.sha }}
 registry-login-server: ${{ secrets.REGISTRY_LOGIN_SERVER }}
 registry-username: ${{ secrets.REGISTRY_USERNAME }}
 registry-password: ${{ secrets.REGISTRY_PASSWORD }}
 name: aci-sampleapp
 location: 'west us'

```

## Validate workflow

After you commit the workflow file, the workflow is triggered. To review workflow progress, navigate to **Actions > Workflows**.



See [Viewing workflow run history](#) for information about viewing the status and results of each step in your workflow. If the workflow doesn't complete, see [Viewing logs to diagnose failures](#).

When the workflow completes successfully, get information about the container instance named *aci-sampleapp* by running the [az container show](#) command. Substitute the name of your resource group:

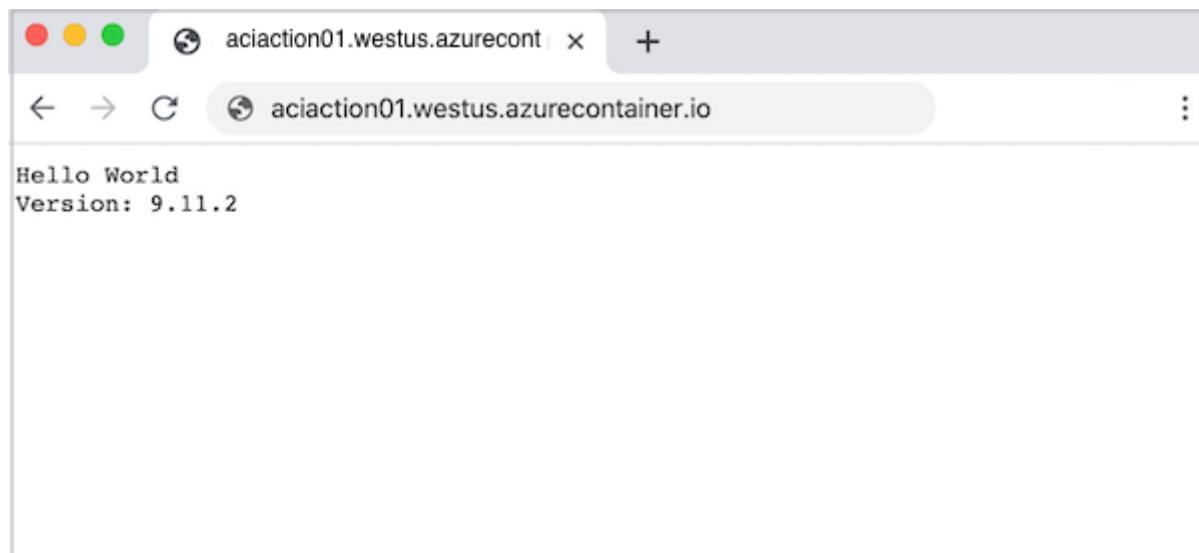
```
Azure CLI

az container show \
--resource-group <resource-group-name> \
--name aci-sampleapp \
--query "{FQDN:ipAddress.fqdn,ProvisioningState:provisioningState}" \
--output table
```

Output is similar to:

| Output                                |                   |
|---------------------------------------|-------------------|
| FQDN                                  | ProvisioningState |
| aci-action01.westus.azurecontainer.io | Succeeded         |

After the instance is provisioned, navigate to the container's FQDN in your browser to view the running web app.



## Use Deploy to Azure extension

Alternatively, use the [Deploy to Azure extension](#) in the Azure CLI to configure the workflow. The `az container app up` command in the extension takes input parameters from you to set up a workflow to deploy to Azure Container Instances.

The workflow created by the Azure CLI is similar to the workflow you can [create manually using GitHub](#).

## Additional prerequisites

In addition to the [prerequisites](#) and [repo setup](#) for this scenario, you need to install the [Deploy to Azure extension](#) for the Azure CLI.

Run the [az extension add](#) command to install the extension:

```
Azure CLI

az extension add \
 --name deploy-to-azure
```

For information about finding, installing, and managing extensions, see [Use extensions with Azure CLI](#).

## Run `az container app up`

To run the [az container app up](#) command, provide at minimum:

- The name of your Azure container registry, for example, *myregistry*
- The URL to your GitHub repo, for example, `https://github.com/<your-GitHub-ID>/acr-build-helloworld-node`

Sample command:

```
Azure CLI

az container app up \
 --acr myregistry \
 --repository https://github.com/myID/acr-build-helloworld-node
```

## Command progress

- When prompted, provide your GitHub credentials or provide a [GitHub personal access token](#) (PAT) that has *repo* and *user* scopes to authenticate with your GitHub account. If you provide GitHub credentials, the command creates a PAT for you. Follow additional prompts to configure the workflow.
- The command creates repo secrets for the workflow:
  - Service principal credentials for the Azure CLI
  - Credentials to access the Azure container registry

- After the command commits the workflow file to your repo, the workflow is triggered.

Output is similar to:

```
Output

[...]
Checking in file github/workflows/main.yml in the GitHub repository
myid/acr-build-helloworld-node
Creating workflow...
GitHub Action Workflow has been created - https://github.com/myid/acr-build-
helloworld-node/runs/515192398
GitHub workflow completed.
Workflow succeeded
Your app is deployed at: http://acr-build-helloworld-
node.eastus.azurecontainer.io:8080/
```

To view the workflow status and results of each step in the GitHub UI, see [Viewing workflow run history](#).

## Validate workflow

The workflow deploys an Azure container instance with the base name of your GitHub repo, in this case, *acr-build-helloworld-node*. When the workflow completes successfully, get information about the container instance named *acr-build-helloworld-node* by running the [az container show](#) command. Substitute the name of your resource group:

```
Azure CLI

az container show \
--resource-group <resource-group-name> \
--name acr-build-helloworld-node \
--query "{FQDN:ipAddress.fqdn,ProvisioningState:provisioningState}" \
--output table
```

Output is similar to:

| Output                                             |                   |
|----------------------------------------------------|-------------------|
| FQDN                                               | ProvisioningState |
| -----                                              | -----             |
| acr-build-helloworld-node.westus.azurecontainer.io | Succeeded         |

After the instance is provisioned, navigate to the container's FQDN in your browser to view the running web app.

# Clean up resources

Stop the container instance with the [az container delete](#) command:

```
Azure CLI

az container delete \
 --name <instance-name>
 --resource-group <resource-group-name>
```

To delete the resource group and all the resources in it, run the [az group delete](#) command:

```
Azure CLI

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

## Next steps

Browse the [GitHub Marketplace](#) ↗ for more actions to automate your development workflow

---

## Feedback

Was this page helpful?



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

# Encrypt deployment data

Article • 08/29/2024

When you run Azure Container Instances (ACI) resources in the cloud, the ACI service collects and persists data related to your containers. ACI automatically encrypts this data when it persists in the cloud. This encryption protects your data to help meet your organization's security and compliance commitments. ACI also gives you the option to encrypt this data with your own key, giving you greater control over the data related to your ACI deployments.

## ACI data encryption

Data in ACI is encrypted and decrypted using 256-bit AES encryption. It's enabled for all ACI deployments, and you don't need to modify your deployment or containers to take advantage of this encryption. This coverage includes metadata about the deployment, environment variables, keys being passed into your containers, and logs persisted after your containers are stopped so you can still see them. Encryption doesn't affect your container group performance, and there's no further cost for encryption.

You can rely on Microsoft-managed keys for the encryption of your container data, or you can manage the encryption with your own keys. The following table compares these options:

[+] Expand table

|                                         | Microsoft-managed keys | Customer-managed keys |
|-----------------------------------------|------------------------|-----------------------|
| <b>Encryption/decryption operations</b> | Azure                  | Azure                 |
| <b>Key storage</b>                      | Microsoft key store    | Azure Key Vault       |
| <b>Key rotation responsibility</b>      | Microsoft              | Customer              |
| <b>Key access</b>                       | Microsoft only         | Microsoft, Customer   |

This article reviews two flows for encrypting data with a customer-managed key:

- Encrypt data with a customer-managed key stored in a standard Azure Key Vault
- Encrypt data with a customer-managed key stored in a network-protected Azure Key Vault with [Trusted Services](#) enabled.

# Encrypt data with a customer-managed key stored in a standard Azure Key Vault

## Prerequisites

- Use the Bash environment in [Azure Cloud Shell](#). For more information, see [Quickstart for Bash in Azure Cloud Shell](#).

 [Launch Cloud Shell](#) 

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

## Create Service Principal for ACI

The first step is to ensure that your [Azure tenant](#) has a service principal assigned for granting permissions to the Azure Container Instances service.

 **Important**

In order to run the following command and create a service principal successfully, confirm that you have permissions to create service principals in your tenant.

The following CLI command sets up the ACI SP in your Azure environment:

Azure CLI

```
az ad sp create --id 6bb8e274-af5d-4df2-98a3-4fd78b4caf9
```

The output from running this command should show you a service principal set up with "displayName": "Azure Container Instance Service."

In case you're unable to successfully create the service principal:

- confirm that you have permissions to do so in your tenant
- check to see if a service principal already exists in your tenant for deploying to ACI.

You can do that by running `az ad sp show --id 6bb8e274-af5d-4df2-98a3-4fd78b4caf9` and use that service principal instead

## Create a Key Vault resource

Create an Azure Key Vault using [Azure portal](#), [Azure CLI](#), or [Azure PowerShell](#).

For the properties of your key vault, use the following guidelines:

- Name: A unique name is required.
- Subscription: Choose a subscription.
- Under Resource Group, either choose an existing resource group, or create new and enter a resource group name.
- In the Location pull-down menu, choose a location.
- You can leave the other options to their defaults or pick based on additional requirements.

### ⓘ Important

When using customer-managed keys to encrypt an ACI deployment template, it is recommended that the following two properties be set on the key vault, Soft Delete and Do Not Purge. These properties are not enabled by default, but can be enabled using either PowerShell or Azure CLI on a new or existing key vault.

## Generate a new key

Once your key vault is created, navigate to the resource in Azure portal. On the left navigation menu of the resource blade, under Settings, select **Keys**. On the view for "Keys," select "Generate/Import" to generate a new key. Use any unique Name for this key, and any other preferences based on your requirements.

## Create a key

### Options

Generate

### Name ⓘ

ACI-Encrypt



### Key Type ⓘ

RSA EC

### RSA Key Size

2048 3072 4096

### Set activation date? ⓘ



### Set expiration date? ⓘ



### Enabled?

Yes No

## Set access policy

Create a new access policy for allowing the ACI service to access your Key.

- Once your key generates, back in your key vault resource blade, under Settings, select **Access Policies**.
- On the "Access Policies" page for your key vault, choose **Add Access Policy**.

- Set the *Key Permissions* to include Get and Unwrap Key

The screenshot shows the 'Add access policy' dialog. In the 'Key permissions' section, 'Get' and 'Unwrap Key' are checked. Other options like List, Update, Create, Import, Delete, Recover, Backup, and Restore are unchecked. The 'Secret permissions', 'Certificate permissions', and 'Select principal' sections are empty. The 'Authorized application' section shows '(None)'.

- For *Select Principal*, select Azure Container Instance Service
- Select **Add** at the bottom

The access policy should now show up in your key vault's access policies.

| Current Access Policies                      |             |       |                 |                    |                         |               |
|----------------------------------------------|-------------|-------|-----------------|--------------------|-------------------------|---------------|
| Name                                         | Category    | Email | Key Permissions | Secret Permissions | Certificate Permissions | Action        |
| <b>APPLICATION</b>                           |             |       |                 |                    |                         |               |
| Azure Container Instance Serv... APPLICATION | APPLICATION |       | 2 selected      | 0 selected         | 0 selected              | <b>Delete</b> |

## Modify your JSON deployment template

### Important

Encrypting deployment data with a customer-managed key is available in the latest API version (2019-12-01) that is currently rolling out. Specify this API version in your deployment template. If you have any issues with this, please reach out to Azure Support.

Once the key vault key and access policy are set up, add the following properties to your ACI deployment template. Learn more about deploying ACI resources with a template in the [Tutorial: Deploy a multi-container group using a Resource Manager template](#).

- Under `resources`, set `apiVersion` to `2019-12-01`.
- Under the container group properties section of the deployment template, add an `encryptionProperties`, which contains the following values:
  - `vaultBaseUrl`: the DNS Name of your key vault, which can be found on the overview blade of the key vault resource in Portal
  - `keyName`: the name of the key generated earlier
  - `keyVersion`: the current version of the key. This field can be found by going into the key itself (under "Keys" in the Settings section of your key vault resource)
- Under the container group properties, add a `sku` property with value `Standard`. The `sku` property is required in API version 2019-12-01.

The following template snippet shows these additional properties to encrypt deployment data:

JSON

```
[...]
"resources": [
 {
 "name": "[parameters('containerGroupName')]",
 "type": "Microsoft.ContainerInstance/containerGroups",
 "apiVersion": "2019-12-01",
 "location": "[resourceGroup().location]",
 "properties": {
 "encryptionProperties": {
 "vaultBaseUrl": "https://example.vault.azure.net",
 "keyName": "acikey",
 "keyVersion": "xxxxxxxxxxxxxxxxxx"
 },
 "sku": "Standard",
 "containers": [
 [...]
]
 }
 }
]
```

Following is a complete template, adapted from the template in [Tutorial: Deploy a multi-container group using a Resource Manager template](#).

JSON

```
{
 "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",
 "contentVersion": "1.0.0.0",
 "parameters": {
 "containerGroupName": {
 "type": "string",
 "defaultValue": "myContainerGroup",
 "metadata": {
 "description": "Container Group name."
 }
 }
 },
 "variables": {
 "container1name": "aci-tutorial-app",
 "container1image": "mcr.microsoft.com/azuredocs/aci-helloworld:latest",
 "container2name": "aci-tutorial-sidecar",
 "container2image": "mcr.microsoft.com/azuredocs/aci-tutorial-sidecar"
 },
 "resources": [
 {
 "name": "[parameters('containerGroupName')]",
 "type": "Microsoft.ContainerInstance/containerGroups",
 "apiVersion": "2019-12-01",
 "location": "[resourceGroup().location]",
 "properties": {
 "encryptionProperties": {
 "vaultBaseUrl": "https://example.vault.azure.net",
 "keyName": "acikey",
 "keyVersion": "xxxxxxxxxxxxxxxxxx"
 },
 "sku": "Standard",
 "containers": [
 {
 "name": "[variables('container1name')]",
 "properties": {
 "image": "[variables('container1image')]",
 "resources": {
 "requests": {
 "cpu": 1,
 "memoryInGb": 1.5
 }
 },
 "ports": [
 {
 "port": 80
 },
 {
 "port": 8080
 }
]
 }
 },
 {
 "name": "[variables('container2name')]"
 }
]
 }
 }
]
}
```

```

 "name": "[variables('container2name')]",
 "properties": {
 "image": "[variables('container2image')]",
 "resources": [
 "requests": {
 "cpu": 1,
 "memoryInGb": 1.5
 }
]
 }
],
 "osType": "Linux",
 "ipAddress": {
 "type": "Public",
 "ports": [
 {
 "protocol": "tcp",
 "port": "80"
 },
 {
 "protocol": "tcp",
 "port": "8080"
 }
]
 }
],
"outputs": {
 "containerIPv4Address": {
 "type": "string",
 "value": "[reference(resourceId('Microsoft.ContainerInstance/containerGroups/', parameters('containerGroupName'))).ipAddress.ip]"
 }
}
}

```

## Deploy your resources

If you created and edited the template file on your desktop, you can upload it to your Cloud Shell directory by dragging the file into it.

Create a resource group with the `az group create` command.

Azure CLI

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

Deploy the template with the [az deployment group create](#) command.

#### Azure CLI

```
az deployment group create --resource-group myResourceGroup --template-file deployment-template.json
```

Within a few seconds, you should receive an initial response from Azure. Once the deployment completes, all data related to it persisted by the ACI service is encrypted with the key you provided.

## Encrypt data with a customer-managed key in a network protected Azure Key Vault with Trusted Services enabled

### Create a Key Vault resource

Create an Azure Key Vault using [Azure portal](#), [Azure CLI](#), or [Azure PowerShell](#). To start, don't apply any network-limitations so we can add necessary keys to the vault. In subsequent steps, we add network-limitations and enable trusted services.

For the properties of your key vault, use the following guidelines:

- Name: A unique name is required.
- Subscription: Choose a subscription.
- Under Resource Group, either choose an existing resource group, or create new and enter a resource group name.
- In the Location pull-down menu, choose a location.
- You can leave the other options to their defaults or pick based on additional requirements.

#### Important

When using customer-managed keys to encrypt an ACI deployment template, it is recommended that the following two properties be set on the key vault, Soft Delete and Do Not Purge. These properties are not enabled by default, but can be enabled using either PowerShell or Azure CLI on a new or existing key vault.

### Generate a new key

Once your key vault is created, navigate to the resource in Azure portal. On the left navigation menu of the resource blade, under Settings, select **Keys**. On the view for "Keys," choose "Generate/Import" to generate a new key. Use any unique Name for this key, and any other preferences based on your requirements. Make sure to capture key name and version for subsequent steps.

The screenshot shows the 'Create a key' page in the Azure portal. At the top, the breadcrumb navigation is: Home > dekapurkv - Overview > dekapurkv - Keys > Create a key. The main title is 'Create a key'. Below it, there's a 'Options' section with a dropdown set to 'Generate'. The 'Name' field is required and contains 'ACI-Encrypt'. The 'Key Type' section shows 'RSA' selected. Under 'RSA Key Size', '2048' is selected. There are checkboxes for 'Set activation date?' and 'Set expiration date?', both of which are unchecked. At the bottom, there's a 'Enabled?' section with 'Yes' selected.

## Create a user-assigned managed identity for your container group

Create an identity in your subscription using the [az identity create](#) command. You can use the same resource group used to create the key vault, or use a different one.

```
Azure CLI
az identity create \
--resource-group myResourceGroup \
--name myACIID
```

To use the identity in the following steps, use the [az identity show](#) command to store the identity's service principal ID and resource ID in variables.

```
Azure CLI
Get service principal ID of the user-assigned identity
spID=$(az identity show \
```

```
--resource-group myResourceGroup \
--name myACIID \
--query principalId --output tsv)
```

## Set access policy

Create a new access policy for allowing the user-assigned identity to access and unwrap your key for encryption purposes.

Azure CLI

```
az keyvault set-policy \
--name mykeyvault \
--resource-group myResourceGroup \
--object-id $spID \
--key-permissions get unwrapKey
```

## Modify Azure Key Vault's network permissions

The following commands set up an Azure Firewall for your Azure Key Vault and allow Azure Trusted Services such as ACI access.

Azure CLI

```
az keyvault update \
--name mykeyvault \
--resource-group myResourceGroup \
--default-action Deny
```

Azure CLI

```
az keyvault update \
--name mykeyvault \
--resource-group myResourceGroup \
--bypass AzureServices
```

## Modify your JSON deployment template

### ⓘ Important

Encrypting deployment data with a customer-managed key is available in the 2022-09-01 API version or newer. The 2022-09-01 API version is only available via ARM or REST. If you have any issues with this, please reach out to Azure Support. Once

the key vault key and access policy are set up, add the following properties to your ACI deployment template. Learn more about deploying ACI resources with a template in the [Tutorial: Deploy a multi-container group using a Resource Manager template](#).

- Under `resources`, set `apiVersion` to `2022-09-01`.
- Under the container group properties section of the deployment template, add an `encryptionProperties`, which contains the following values:
  - `vaultBaseUrl`: the DNS Name of your key vault. This property can be found on the overview blade of the key vault resource in Portal
  - `keyName`: the name of the key generated earlier
  - `keyVersion`: the current version of the key. This property can be found by clicking into the key itself (under "Keys" in the Settings section of your key vault resource)
  - `identity`: this property is the resource URI of the Managed Identity instance created earlier
- Under the container group properties, add a `sku` property with value `Standard`. The `sku` property is required in API version 2022-09-01.
- Under `resources`, add the `identity` object required to use Managed Identity with ACI, which contains the following values:
  - `type`: the type of the identity being used (either user-assigned or system-assigned). This case is set to "UserAssigned"
  - `userAssignedIdentities`: the resourceURI of the same user-assigned identity used in the `encryptionProperties` object.

The following template snippet shows these additional properties to encrypt deployment data:

JSON

```
[...]
"resources": [
 {
 "name": "[parameters('containerGroupName')]",
 "type": "Microsoft.ContainerInstance/containerGroups",
 "apiVersion": "2019-12-01",
 "location": "[resourceGroup().location]",
 "identity": {
 "type": "UserAssigned",
 "userAssignedIdentities": {
 "/subscriptions/XXXXXXXXXXXXXXXXXXXX/resourcegroups/XXXXXXXXXXXXXXXXXXXX/providers/Microsoft.ManagedIdentity/userAssignedIdentities/myACIID": {}
 }
 }
 }
]
```

```

 },
 "properties": {
 "encryptionProperties": {
 "vaultBaseUrl": "https://example.vault.azure.net",
 "keyName": "acikey",
 "keyVersion": "xxxxxxxxxxxxxxxxxxxx",
 "identity": [
 "/subscriptions/XXXXXXXXXXXXXXXXXXXX/resourcegroups/XXXXXXXXXXXXXXXXXXXXXX
 /providers/Microsoft.ManagedIdentity/userAssignedIdentities/myACIId"
],
 "sku": "Standard",
 "containers": [
 [...]
]
 }
 }
]

```

Following is a complete template, adapted from the template in [Tutorial: Deploy a multi-container group using a Resource Manager template](#).

JSON

```
{
 "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",
 "contentVersion": "1.0.0.0",
 "parameters": {
 "containerGroupName": {
 "type": "string",
 "defaultValue": "myContainerGroup",
 "metadata": {
 "description": "Container Group name."
 }
 }
 },
 "variables": {
 "container1name": "aci-tutorial-app",
 "container1image": "mcr.microsoft.com/azuredocs/aci-helloworld:latest",
 "container2name": "aci-tutorial-sidecar",
 "container2image": "mcr.microsoft.com/azuredocs/aci-tutorial-sidecar"
 },
 "resources": [
 {
 "name": "[parameters('containerGroupName')]",
 "type": "Microsoft.ContainerInstance/containerGroups",
 "apiVersion": "2022-09-01",
 "location": "[resourceGroup().location]",
 "identity": {
 "type": "UserAssigned",
 "userAssignedIdentities": [
 "/subscriptions/XXXXXXXXXXXXXXXXXXXX/resourcegroups/XXXXXXXXXXXXXXXXXXXXXX
 /providers/Microsoft.ManagedIdentity/userAssignedIdentities/myACIId"
]
 }
 }
]
}
```

```
/providers/Microsoft.ManagedIdentity/userAssignedIdentities/myACIId": {}
 }
},
"properties": {
 "encryptionProperties": {
 "vaultBaseUrl": "https://example.vault.azure.net",
 "keyName": "acikey",
 "keyVersion": "xxxxxxxxxxxxxxxxxxxx",
 "identity":
 "/subscriptions/XXXXXXXXXXXXXXXXXXXX/resourcegroups/XXXXXXXXXXXXXXXXXXXX
/providers/Microsoft.ManagedIdentity/userAssignedIdentities/myACIId"
 },
 "sku": "Standard",
 "containers": [
 {
 "name": "[variables('container1name')]",
 "properties": {
 "image": "[variables('container1image')]",
 "resources": {
 "requests": {
 "cpu": 1,
 "memoryInGb": 1.5
 }
 },
 "ports": [
 {
 "port": 80
 },
 {
 "port": 8080
 }
]
 }
 },
 {
 "name": "[variables('container2name')]",
 "properties": {
 "image": "[variables('container2image')]",
 "resources": {
 "requests": {
 "cpu": 1,
 "memoryInGb": 1.5
 }
 }
 }
 }
],
 "osType": "Linux",
 "ipAddress": {
 "type": "Public",
 "ports": [
 {
 "protocol": "tcp",
 "port": "80"
 },
 {
 "protocol": "tcp",
 "port": "443"
 }
]
 }
}
```

```
 {
 "protocol": "tcp",
 "port": "8080"
 }
]
 }
],
 "outputs": {
 "containerIPv4Address": {
 "type": "string",
 "value": "[reference(resourceId('Microsoft.ContainerInstance/containerGroups/', parameters('containerGroupName'))).ipAddress.ip]"
 }
 }
}
```

## Deploy your resources

If you created and edited the template file on your desktop, you can upload it to your Cloud Shell directory by dragging the file into it.

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

Azure CLI

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

Deploy the template with the [az deployment group create](#) command.

Azure CLI

```
az deployment group create --resource-group myResourceGroup --template-file deployment-template.json
```

Within a few seconds, you should receive an initial response from Azure. Once the deployment completes, all data related to it persisted by the ACI service is encrypted with the key you provided.

## Feedback

Was this page helpful?

 Yes

 No



# Reliability in Azure Container Instances

Article • 10/14/2024

This article describes reliability support in Azure Container Instances (ACI) and covers both intra-regional resiliency with [availability zones](#) and information on 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 Instances supports *zonal* container group deployments, meaning the instance is pinned to a specific, self-selected availability zone. The availability zone is specified at the container group level. Containers within a container group can't have unique availability zones. To change your container group's availability zone, you must delete the container group and create another container group with the new availability zone.

## Prerequisites

- Zonal container group deployments are supported in most regions where ACI is available for Linux and Windows Server 2019 container groups. For details, see [Regions and resource availability](#).
- If using Azure CLI, ensure version `2.30.0` or later is installed.

- If using PowerShell, ensure version 2.1.1-preview or later is installed.
- If using the Java SDK, ensure version 2.9.0 or later is installed.
- Availability zone support is only available on ACI API version 09-01-2021 or later.

 **Important**

Container groups with GPU resources don't support availability zones at this time.

## Availability zone redeployment and migration

To change your container group's availability zone, you must delete the container group and create another container group with the new availability zone.

## Create a resource with availability zone enabled

To create a Container Instance resource with availability zone enabled, you'll need to deploy a container group using an Azure Resource Manager (ARM) template.

 **Note**

Examples in this article are formatted for the Bash shell. If you prefer another shell, adjust the line continuation characters accordingly.

To deploy a container with ARM:

1. Copy-paste the following JSON into a new file named `azuredeploy.json`. This example template deploys a container group with a single container into availability zone 1 in East US.

JSON

```
{
 "$schema": "https://schema.management.azure.com/schemas/2019-04-01/deploymentTemplate.json#",
 "contentVersion": "1.0.0.0",
 "metadata": {
 "_generator": {
 "name": "bicep",
 "version": "0.4.1.14562",
 "templateHash": "12367894147709986470"
 }
 },
 "parameters": {
```

```
 "name": {
 "type": "string",
 "defaultValue": "acilinuxpublicipcontainergroup",
 "metadata": {
 "description": "Name for the container group"
 }
 },
 "image": {
 "type": "string",
 "defaultValue": "mcr.microsoft.com/azuredocs/aci-helloworld",
 "metadata": {
 "description": "Container image to deploy. Should be of the form repoName/imagename:tag for images stored in public Docker Hub, or a fully qualified URI for other registries. Images from private registries require additional registry credentials."
 }
 },
 "port": {
 "type": "int",
 "defaultValue": 80,
 "metadata": {
 "description": "Port to open on the container and the public IP address."
 }
 },
 "cpuCores": {
 "type": "int",
 "defaultValue": 1,
 "metadata": {
 "description": "The number of CPU cores to allocate to the container."
 }
 },
 "memoryInGb": {
 "type": "int",
 "defaultValue": 2,
 "metadata": {
 "description": "The amount of memory to allocate to the container in gigabytes."
 }
 },
 "restartPolicy": {
 "type": "string",
 "defaultValue": "Always",
 "allowedValues": [
 "Always",
 "Never",
 "OnFailure"
],
 "metadata": {
 "description": "The behavior of Azure runtime if container has stopped."
 }
 },
 },
}
```

```
"location": {
 "type": "string",
 "defaultValue": "eastus",
 "metadata": {
 "description": "Location for all resources."
 }
},
"functions": [],
"resources": [
{
 "type": "Microsoft.ContainerInstance/containerGroups",
 "apiVersion": "2021-09-01",
 "zones": [
 "1"
],
 "name": "[parameters('name')]",
 "location": "[parameters('location')]",
 "properties": {
 "containers": [
 {
 "name": "[parameters('name')]",
 "properties": {
 "image": "[parameters('image')]",
 "ports": [
 {
 "port": "[parameters('port')]",
 "protocol": "TCP"
 }
],
 "resources": {
 "requests": {
 "cpu": "[parameters('cpuCores')]",
 "memoryInGB": "
[parameters('memoryInGb')]"
 }
 }
 }
 }
],
 "osType": "Linux",
 "restartPolicy": "[parameters('restartPolicy')]",
 "ipAddress": {
 "type": "Public",
 "ports": [
 {
 "port": "[parameters('port')]",
 "protocol": "TCP"
 }
]
 }
 }
},
"outputs": {
```

```
 "containerIPv4Address": {
 "type": "string",
 "value": "
[reference(resourceId('Microsoft.ContainerInstance/containerGroups',
parameters('name'))).ipAddress.ip]"
 }
 }
}
```

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

Azure CLI

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

3. Deploy the template with the [az deployment group create](#) command:

Azure CLI

```
az deployment group create \
--resource-group myResourceGroup \
--template-file azuredeploy.json
```

4. To verify the container group deployed successfully into an availability zone, view the container group details with the [az container show](#) command:

Azure CLI

```
az container show --name acilinuxpublicipcontainergroup --resource-
group myResourceGroup
```

## Zonal failover support

A container group of container instances is assigned to a single availability zone. As a result, that group of container instances won't be impacted by an outage that occurs in any other availability zone of the same region.

If, however, an outage occurs in the availability zone of the container group, you can expect downtime for all the container instances within that group.

To avoid container instance downtime, we recommend that you create a minimum of two container groups across two different availability zones in a given region. This ensures that your container instance resources are up and running whenever any single zone in that region experiences outage.

# Disaster recovery

When an entire Azure region or datacenter experiences downtime, your mission-critical code needs to continue processing in a different region. Azure Container Instances deployed with zonal configuration run in a specific zone within a specific region. There's no built-in redundancy available. To avoid loss of execution during region wide outages, you can redundantly deploy the container instances in other regions.

## Next steps

- [Azure Architecture Center's guide on availability zones.](#)
  - [Reliability in Azure](#)
- 

## Feedback

Was this page helpful?

 Yes

 No

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

# Deploy an Azure Container Instances (ACI) container group with DNS name reuse policy (preview)

Article • 08/29/2024

DNS name reuse is convenient for DevOps within any modern company. The idea of redeploying an application by reusing the DNS name fulfills an on-demand philosophy that secures cloud development. Therefore, it's important to note that DNS names that are available to anyone become a problem when one customer releases a name only to have that same name taken by another customer. This is called subdomain takeover. A customer releases a resource using a particular name, and another customer creates a new resource with that same DNS name. If there were any records pointing to the old resource, they now also point to the new resource.

In order to avoid this, ACI now allows customers to reuse DNS names while preventing different customers from reusing DNS names. ACI secures DNS names by randomly generating a hash value to associate with the DNS name, making it difficult for another customer to accidentally create an ACI with the same name and get linked to the past customer's ACI information.

## ⓘ Important

DNS name reuse policy support is only available on ACI API version `10-01-2021` or later.

## Prerequisites

- An **active Azure subscription**. If you don't have an active Azure subscription, create a [free account](#) before you begin.
- A **resource group** to manage all the resources you use in this how-to guide. We use the example resource group name **ACIResourceGroup** throughout this article.

Azure CLI

```
az group create --name ACIResourceGroup --location westus
```

# Understand the DNS name reuse policy

You now have the choice when creating an ACI to choose what level of reuse you want your DNS name label to have.

  Expand table

| Policy name        | Policy definition                                                                                                                                                          |
|--------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| unsecure           | Hash isn't generated. Object's domain label can't be reused within resource group, subscription, or tenant.                                                                |
| tenantReuse        | <b>Default</b> Hash generates based on the DNS name and the tenant ID. Object's domain name label can be reused within the same tenant.                                    |
| subscriptionReuse  | Hash generated bases on the DNS name and the tenant ID and subscription ID. Object's domain name label can be reused within the same subscription.                         |
| resourceGroupReuse | Hash generates based on the DNS name and the tenant ID, subscription ID, and resource group name. Object's domain name label can be reused within the same resource group. |
| noReuse            | Hash generates using a GUID. Object's domain label can be reused freely, as this hash is always unique.                                                                    |

## Create a container instance

### ⓘ Important

Setting the DNS name reuse policy is not currently supported through the Azure CLI .

The process for deploying your container instance remains same. If you want the full process of how to deploy a container instance, see the quickstart featuring your preferred deployment method. For example, the [ARM template quickstart](#).

For [Azure portal](#) users, you can set the DNS name reuse policy on the **Networking** tab during the container instance creation process using the **DNS name label scope reuse** field.

## Create container instance

- 'Public' will create a public IP address for your container instance.
- 'Private' will allow you to choose a new or existing virtual network for your container instance. This is not yet available for Windows containers.
- 'None' will not create either a public IP or virtual network. You will still be able to access your container logs using the command line.

Networking type  Public  Private  None

DNS name label ⓘ aci-dns-name ✓

DNS name label scope reuse \* ⓘ Tenant

Ports ⓘ

Ports 80

Tenant

Tenant

Subscription

Resource Group

No reuse

Any reuse (unsecure)

For ARM template users, see the [Resource Manager reference](#) to see how the dnsNameLabelReusePolicy field fits into the existing schema.

For YAML template users, see the [YAML reference](#) to see how the dnsNameLabelReusePolicy field fits into the existing schema.

## Next steps

See the Azure Quickstart Template [Create an Azure container group with virtual network](#), to deploy a container group within a virtual network.

## Feedback

Was this page helpful?

 Yes

 No

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

# Deploy and manage Azure Container Instances by using Azure Logic Apps

Article • 01/05/2024

Applies to: Azure Logic Apps (Consumption)

With Azure Logic Apps and the Azure Container Instance connector, you can set up automated tasks and workflows that deploy and manage [container groups](#). The Container Instance connector supports the following actions:

- Create or delete a container group
- Get the properties of a container group
- Get a list of container groups
- Get the logs of a container instance

Use these actions in your logic apps for tasks such as running a container workload in response to a Logic Apps trigger. You can also have other actions use the output from Container Instance actions.

This connector provides only actions, so to start your logic app, use a separate trigger, such as a **Recurrence** trigger to run a container workload on a regular schedule. Or, you might have need to trigger a container group deployment after an event such as arrival of an Outlook e-mail.

If you're new to logic apps, review [What is Azure Logic Apps?](#)

## Prerequisites

- An Azure account and subscription. If you don't have an Azure subscription, [sign up for a free Azure account](#).
- Basic knowledge about how to create logic apps and how to create container instances
  - [Create an example Consumption logic app workflow](#)
  - [Create and manage container instances](#)
- The logic app where you want to access your container instances. To use an action, start your logic app with another trigger, for example, the **Recurrence** trigger.

# Add a Container Instance action

When you add a trigger or action that connects to a service or system, and you don't have an existing or active connection, Azure Logic Apps prompts you to provide the connection information, which varies based on the connection type, for example:

- Your account credentials
- A name to use for the connection
- The name for the server or system
- The authentication type to use
- A connection string

1. Sign in to the [Azure portal](#), and open your logic app in Logic App Designer, if not open already.

2. Choose a path:

- Under the last step where you want to add an action, choose **New step**.

-or-

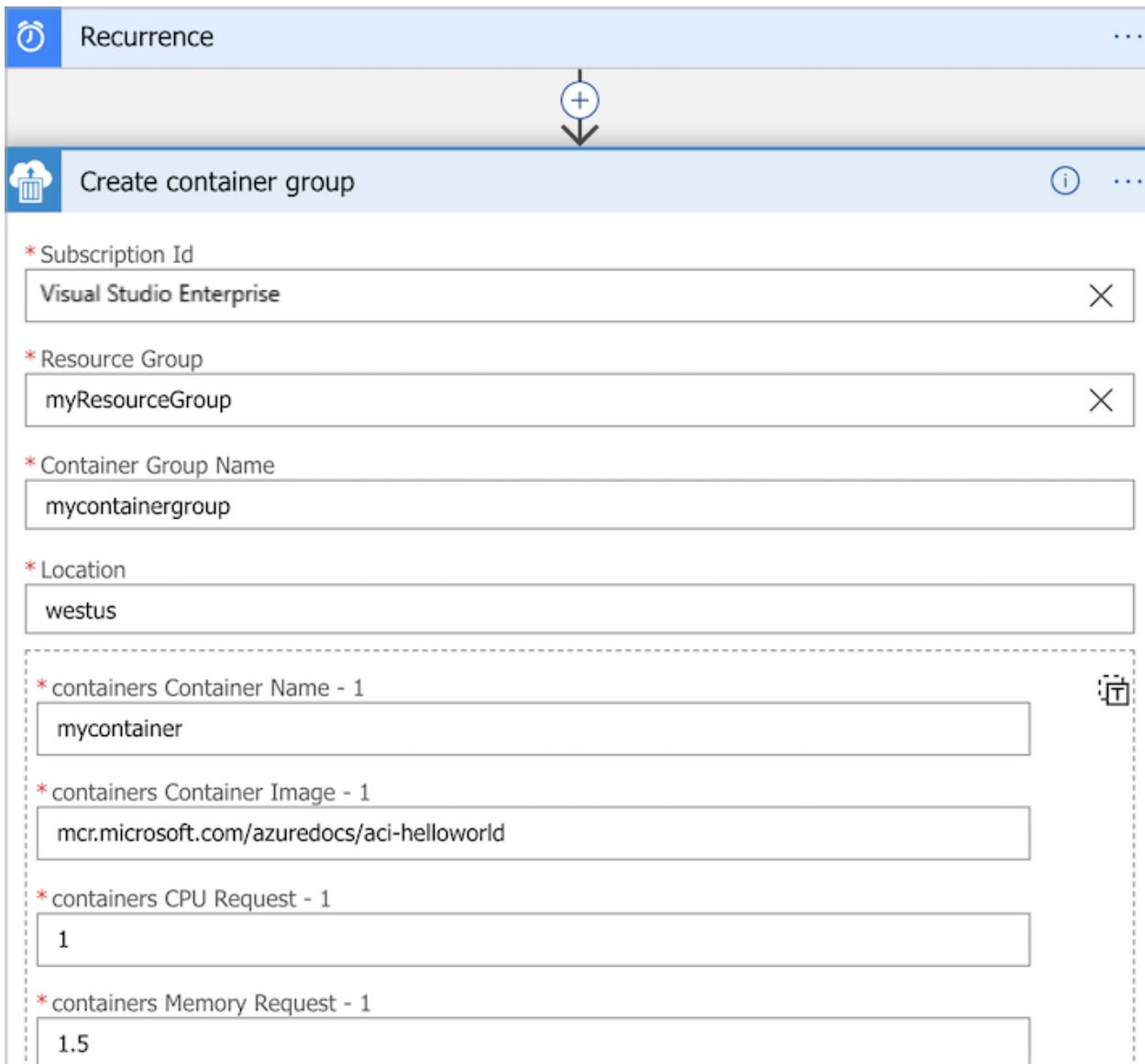
- Between the steps where you want to add an action, move your pointer over the arrow between steps. Choose the plus sign (+) that appears, and then select **Add an action**.

3. In the search box, enter "container instance" as your filter. Under the actions list, select the Azure Container Instance connector action you want.

4. Provide a name for your connection.

5. Provide the necessary details for your selected action and continue building your logic app's workflow.

For example, select **Create container group** and enter the properties for a container group and one or more container instances in the group, as shown in the following image (partial detail):



## Connector reference

For technical details about triggers, actions, and limits, which are described by the connector's OpenAPI (formerly Swagger) description, review the connector's [reference page](#) or container group [YAML reference](#).

## Next steps

- See a [sample logic app](#) that runs a container in Azure Container Instances to analyze the sentiment of e-mail or Twitter text
- [Managed connectors for Azure Logic Apps](#)
- [Built-in connectors for Azure Logic Apps](#)
- [What are connectors in Azure Logic Apps](#)

# Tutorial: Deploy a confidential container to Azure Container Instances via Azure portal

Article • 08/29/2024

In this tutorial, you use Azure portal to deploy a confidential container to Azure Container Instances with a development confidential computing enforcement policy. After deploying the container, you can browse to the running application.

## ⓘ Note

When deploying confidential containers on Azure Container Instances via Portal you will only be able to deploy with a development confidential computing enforcement policy. This policy is only recommended for development and test workloads. Logging, and exec functionality are still available in the container group when using this policy and software components are not validated. To full attest your container group while running production workloads, it is recommended that you deploy with a custom confidential computing enforcement policy via an Azure Resource Manager template. For more information, see the [tutorial](#).

Welcome to Confidential containers on Azure Container Instances!



AMD SEV-SNP Attestation Report:

```
version:
0000002
guest_svn:
0000002
policy:
000000000003001f
family_id:
00000000000000000000000000000001
image_id:
00000000000000000000000000000002
vmpk:
00000000
signature_algo:
00000001
platform_version:
030000000000873
platform_info:
0100000000000000
author_key_en:
00000000
reserved1:
00000000
report_data:
00000000000000000000000000000000 00000000000000000000000000000000
00000000000000000000000000000000 00000000000000000000000000000000
measurement:
00000000000000000000000000000000
```

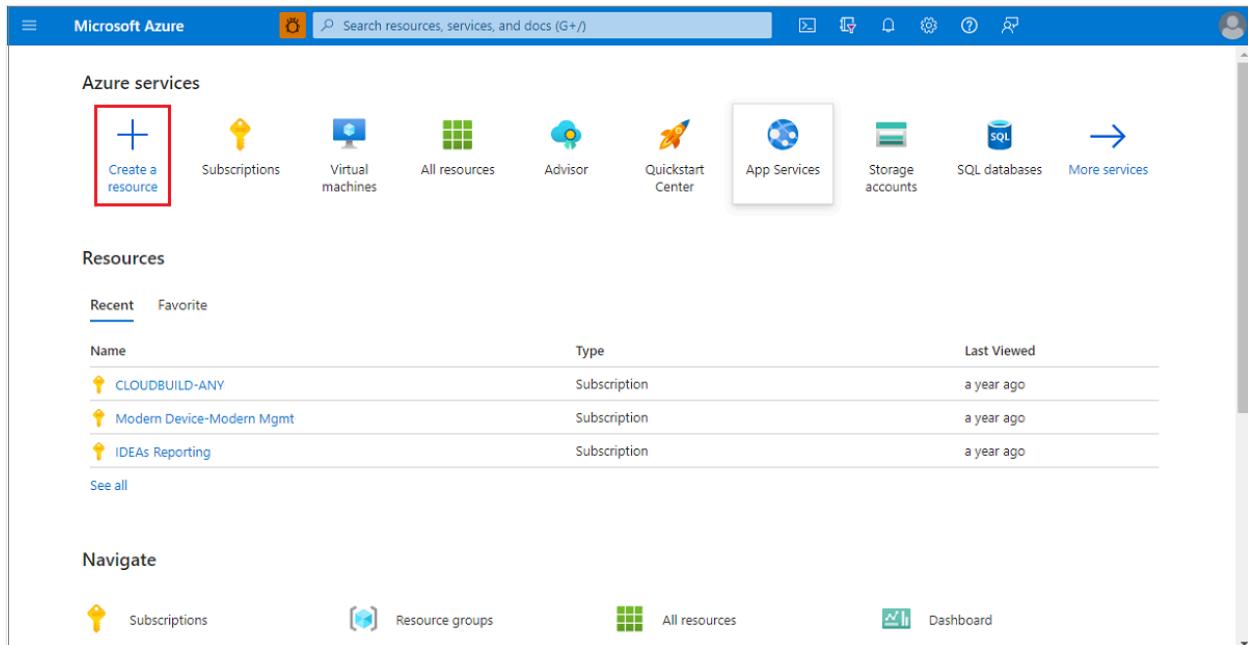
# Sign in to Azure

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

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

## Create a confidential container on Azure Container Instances

On the Azure portal homepage, select **Create a resource**.



The screenshot shows the Microsoft Azure portal interface. At the top, there's a navigation bar with icons for search, notifications, and settings. Below it, the 'Azure services' section features a 'Create a resource' button with a red box around it. Other service icons include Subscriptions, Virtual machines, All resources, Advisor, Quickstart Center, App Services, Storage accounts, SQL databases, and More services. The 'Resources' section displays a table of recent subscriptions, all of which are 'Subscription' type and were last viewed 'a year ago'. Below this is a 'Recent' tab and a 'See all' link. The 'Navigate' section at the bottom includes links for Subscriptions, Resource groups, All resources, and Dashboard.

| Name                      | Type         | Last Viewed |
|---------------------------|--------------|-------------|
| CLOUDBUILD-ANY            | Subscription | a year ago  |
| Modern Device-Modern Mgmt | Subscription | a year ago  |
| IDEAs Reporting           | Subscription | a year ago  |

Select **Containers > Container Instances**.

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

Home > Create a resource ...

Get Started Search services and marketplace

Recently created

Popular Azure services See more in All services

Categories

- AI + Machine Learning
- Analytics
- Blockchain
- Compute
- Containers** Create | Docs | MS Learn
- Databases
- Developer Tools
- DevOps
- Identity
- Integration
- Internet of Things
- IT & Management Tools
- Media
- Migration
- Mixed Reality
- Monitoring & Diagnostics

**Kubernetes Service** Create | Docs | MS Learn

**Web App for Containers** Create | Docs | MS Learn

**Batch Service** Create | Docs | MS Learn

**Kubernetes - Azure Arc** Create | Learn more

**Container App** Create | Learn more

**Container Instances** Create | Learn more

**Container Registry** Create | Docs | MS Learn

**Service Fabric Cluster** Create | Docs

On the **Basics** page, choose a subscription and enter the following values for **Resource group**, **Container name**, **Image source**, and **Container image**.

- Resource group: **Create new** > `myresourcegroup`
- Container name: `helloworld`
- Region: One of `West Europe / North Europe / East US`
- SKU: `Confidential (development policy)`
- Image source: **Quickstart images**
- Container image: `mcr.microsoft.com/aci/aci-confidential-helloworld:v1 (Linux)`

## Create container instance

**Project details**

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

Subscription \* ⓘ Subscription Name

Resource group \* ⓘ (New) myACIRG    
[Create new](#)

**Container details**

Container name \* ⓘ helloworld

Region \* ⓘ (Europe) North Europe

Availability zones ⓘ None

i No availability zones are available for the SKU you have selected.

SKU Confidential (development policy)

! Container groups deployed from the portal use development policies and are recommended for dev/test workloads. Logging, and exec functionality are still available in the container group when using this policy. For production workloads, it is highly recommended you generate a more restrictive custom policy.  
[Learn more ↗](#)

Image source \* ⓘ  Quickstart images  Container image  Custom image

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

### i Note

When deploying confidential containers on Azure Container Instances via Portal you will only be able to deploy with a development confidential computing enforcement policy. This policy is only recommended for development and test workloads. Logging, and exec functionality are still available in the container group when using this policy and software components are not validated. To fully attest your container group while running production workloads, it is recommended that you deploy with a custom confidential computing enforcement policy via an Azure Resource Manager template. For more information, see the [tutorial](#).

Leave all other settings as their defaults, then select **Review + create**.

When the validation completes, you're shown a summary of the container's settings. Select **Create** to submit your container deployment request.

## Create container instance

...

Validation passed

Basics Networking Advanced Tags **Review + create**

### Basics

|                     |                                                   |
|---------------------|---------------------------------------------------|
| Subscription        | Subscription Name                                 |
| Resource group      | (new) myACIRG                                     |
| Region              | North Europe                                      |
| Container name      | helloworld                                        |
| SKU                 | Confidential (development policy)                 |
| Image type          | Public                                            |
| Image               | mcr.microsoft.com/azuredocs/aci-helloworld:latest |
| OS type             | Linux                                             |
| Memory (GiB)        | 1.5                                               |
| Number of CPU cores | 1                                                 |
| GPU type (preview)  | None                                              |
| GPU count           | 0                                                 |

### Networking

|                            |                      |
|----------------------------|----------------------|
| Networking type            | Public               |
| Ports                      | 80 (TCP)             |
| DNS name label scope reuse | Any reuse (unsecure) |

**Create**

< Previous

Next >

Download a template for automation

When deployment starts, a notification appears that indicates the deployment is in progress. Another notification is displayed when the container group deployed.

Open the overview for the container group by navigating to **Resource Groups > myACIRG > helloworld**. Make a note of the **IP** of the container instance and its **Status**.

1. On the **Overview** page, note the **Status** of the instance and its **IP address**.

Home > Resource groups > myACIRG >

**helloworld** ⚡ ⋆ ...

Container instances

Search Start Restart Stop Delete Refresh

**Overview** View Cost

Activity log Access control (IAM) Tags Settings Containers Identity Properties Locks Monitoring Metrics Alerts Tasks (preview) Export template Support + troubleshooting New Support Request

**Essentials**

Resource group (move) : [myACIRG](#)  
Status : Running  
Location : North Europe  
Subscription (move) : [your subscription](#)  
Subscription ID : <your subscription id>  
Tags (edit) : [Click here to add tags](#)

OS type : Linux  
IP address (Public) : [20.166.162.14](#)  
FQDN : ...  
Container count : 1

**CPU**

100  
90  
80  
70  
60  
50  
40  
30  
20  
10  
0

10:15 AM 10:30 AM 10:45 AM 11 AM UTC-08:00

GPU Usage (Avg)  
helloworld  
0

**Memory**

25MB  
20MB  
15MB  
10MB  
5MB  
0B

10:15 AM 10:30 AM 10:45 AM 11 AM UTC-08:00

Memory Usage (Avg)  
helloworld  
23.52 MB

**Network bytes received**

60B  
50B  
40B  
30B  
20B  
10B  
0B

10:15 AM 10:30 AM 10:45 AM 11 AM UTC-08:00

Network Bytes Received Per Second (Avg)  
helloworld  
31.52 B

2. Once its status is *Running*, navigate to the IP address in your browser.

The presence of the attestation report below the Azure Container Instances logo confirms that the container is running on hardware that supports a hardware-based and attested trusted execution environment (TEE). If you deploy to hardware that doesn't support a TEE, for example by choosing a region where the [ACI Confidential SKU isn't available](#), no attestation report is shown.

Congratulations! You deployed a confidential container on Azure Container Instances, which displays a hardware attestation report in your browser.

# Clean up resources

When you're done with the container, select **Overview** for the *helloworld* container instance, then select **Delete**.

## Next steps

In this tutorial, you created a confidential container on Azure Container instances with a development confidential computing enforcement policy. If you would like to deploy a confidential container group with a custom computing enforcement policy continue to the confidential containers on Azure Container Instances - deploy with Azure Resource Manager template tutorial.

- [Azure Container Instances Azure Resource Manager template tutorial](#)
  - [Confidential computing enforcement policies overview](#)
  - [Azure CLI confcom extension examples ↗](#)
  - [Confidential Hello World application ↗](#)
- 

## Feedback

Was this page helpful?

 Yes

 No

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

# Tutorial: Prepare a deployment for a confidential container on Azure Container Instances

Article • 08/29/2024

In Azure Container Instances, you can use confidential containers on the serverless platform to run container applications in a hardware-based and attested trusted execution environment (TEE). This capability can help protect data in use and provides in-memory encryption via Secure Nested Paging.

In this tutorial, you learn how to:

- ✓ Create an Azure Resource Manager template (ARM template) for a confidential container group.
- ✓ Generate a confidential computing enforcement (CCE) policy.
- ✓ Deploy the confidential container group to Azure.

## Prerequisites

To complete this tutorial, you must satisfy the following requirements:

- **Azure CLI:** You must have Azure CLI version 2.44.1 or later installed on your local computer. To find your version, run `az --version`. If you need to install or upgrade, see [Install the Azure CLI](#).
- **Azure CLI confcom extension:** You must have Azure CLI confcom extension version 0.30+ installed to generate confidential computing enforcement policies.

Bash

```
az extension add -n confcom
```

- **Docker:** You need Docker installed locally. Docker provides packages that configure the Docker environment on [macOS](#), [Windows](#), and [Linux](#).

This tutorial assumes a basic understanding of core Docker concepts like containers, container images, and basic `docker` commands. For a primer on Docker and container basics, see the [Docker overview](#).

 **Important**

Because Azure Cloud Shell doesn't include the Docker daemon, you must install both the Azure CLI and Docker Engine on your *local computer* to complete this tutorial. You can't use Azure Cloud Shell for this tutorial.

# Create an ARM template for a Container Instances container group

In this tutorial, you deploy a Hello World application that generates a hardware attestation report. You start by creating an ARM template with a container group resource to define the properties of this application. You then use this ARM template with the Azure CLI confcom tooling to generate a CCE policy for attestation.

This tutorial uses [this ARM template](#) as an example. To view the source code for this application, see [Azure Confidential Container Instances Hello World](#).

The example template adds two properties to the Container Instances resource definition to make the container group confidential:

- `sku`: Enables you to select between confidential and standard container group deployments. If you don't add this property to the resource, the container group is a standard deployment.
- `confidentialComputeProperties`: Enables you to pass in a custom CCE policy for attestation of your container group. If you don't add this object to the resource, the software components that run within the container group won't validate.

## ⓘ Note

The `ccePolicy` parameter under `confidentialComputeProperties` is blank. You'll fill it in when you generate the policy later in the tutorial.

Use your preferred text editor to save this ARM template on your local machine as `template.json`.

ARM

```
{
 "$schema": "https://schema.management.azure.com/schemas/2019-04-
01/deploymentTemplate.json#",
 "contentVersion": "1.0.0.0",
 "parameters": {
 "name": {
 "type": "string",
```

```
 "defaultValue": "helloworld",
 "metadata": {
 "description": "Name for the container group"
 }
 },
 "location": {
 "type": "string",
 "defaultValue": "North Europe",
 "metadata": {
 "description": "Location for all resources."
 }
 },
 "image": {
 "type": "string",
 "defaultValue":
"mcr.microsoft.com/public/acc/samples/aci/helloworld:2.7",
 "metadata": {
 "description": "Container image to deploy. Should be of the form repoName/imagename:tag for images stored in public Docker Hub, or a fully qualified URI for other registries. Images from private registries require additional registry credentials."
 }
 },
 "port": {
 "type": "int",
 "defaultValue": 80,
 "metadata": {
 "description": "Port to open on the container and the public IP address."
 }
 },
 "cpuCores": {
 "type": "int",
 "defaultValue": 1,
 "metadata": {
 "description": "The number of CPU cores to allocate to the container."
 }
 },
 "memoryInGb": {
 "type": "int",
 "defaultValue": 1,
 "metadata": {
 "description": "The amount of memory to allocate to the container in gigabytes."
 }
 },
 "restartPolicy": {
 "type": "string",
 "defaultValue": "Never",
 "allowedValues": [
 "Always",
 "Never",
 "OnFailure"
],
 }
},
```

```
 "metadata": {
 "description": "The behavior of Azure runtime if container has stopped."
 }
},
"resources": [
{
 "type": "Microsoft.ContainerInstance/containerGroups",
 "apiVersion": "2023-05-01",
 "name": "[parameters('name')]",
 "location": "[parameters('location')]",
 "properties": {
 "confidentialComputeProperties": {
 "ccePolicy": ""
 },
 "containers": [
{
 "name": "[parameters('name')]",
 "properties": {
 "image": "[parameters('image')]",
 "ports": [
{
 "port": "[parameters('port')]",
 "protocol": "TCP"
}
],
 "resources": {
 "requests": {
 "cpu": "[parameters('cpuCores')]",
 "memoryInGB": "[parameters('memoryInGb')]"
}
}
}
]
}
],
 "sku": "Confidential",
 "osType": "Linux",
 "restartPolicy": "[parameters('restartPolicy')]",
 "ipAddress": {
 "type": "Public",
 "ports": [
{
 "port": "[parameters('port')]",
 "protocol": "TCP"
}
]
}
}
],
 "outputs": {
 "containerIPv4Address": {
 "type": "string",
 "value": "

```

```
[reference(resourceId('Microsoft.ContainerInstance/containerGroups',
parameters('name'))).ipAddress.ip]"
}
}
}
```

## Create a custom CCE policy

With the ARM template that you crafted and the Azure CLI confcom extension, you can generate a custom CCE policy. The CCE policy is used for attestation. The tool takes the ARM template as an input to generate the policy. The policy enforces the specific container images, environment variables, mounts, and commands, which can then be validated when the container group starts up. For more information on the Azure CLI confcom extension, see the [documentation on GitHub](#).

1. To generate the CCE policy, run the following command by using the ARM template as input:

```
Azure CLI

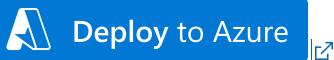
az confcom acipolicygen -a .\template.json
```

When this command finishes, a Base64 string generated as output will automatically appear in the `ccePolicy` property of the ARM template.

## Deploy the template

In the following steps, you use the Azure portal to deploy the template. You can also use Azure PowerShell, the Azure CLI, or the REST API. To learn about other deployment methods, see [Deploy templates](#).

1. Select the **Deploy to Azure** button to sign in to Azure and begin a Container Instances deployment.



2. Select **Build your own template in the editor**.

## Custom deployment ...

Deploy from a custom template

### Select a template

Basics

Review + create

Automate deploying resources with Azure Resource Manager templates in a single, coordinated operation. Create or select a template below to get started. [Learn more about template deployment](#)

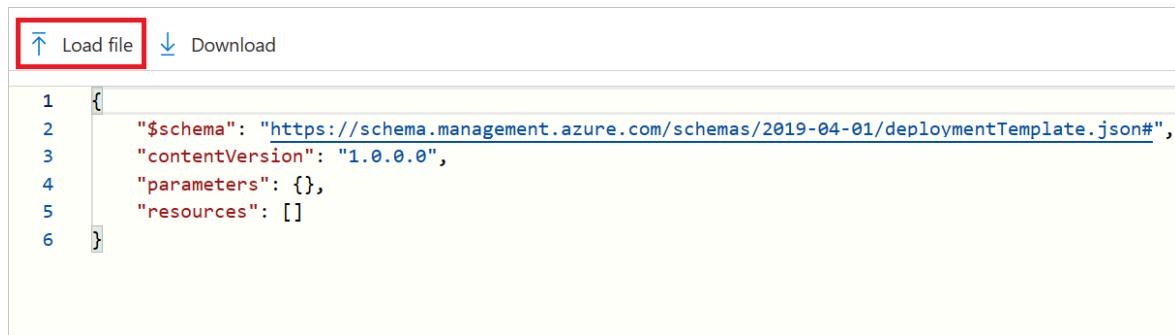


Build your own template in the editor

### Common templates

The template JSON that appears is mostly blank.

3. Select **Load file** and upload *template.json*, which you modified by adding the CCE policy in the previous steps.



```
1 [
2 "$schema": "https://schema.management.azure.com/schemas/2019-04-01/deploymentTemplate.json#",
3 "contentVersion": "1.0.0.0",
4 "parameters": {},
5 "resources": []
6]
```

4. Select **Save**.

5. Select or enter the following values:

- **Subscription:** Select an Azure subscription.
- **Resource group:** Select **Create new**, enter a unique name for the resource group, and then select **OK**.
- **Name:** Accept the generated name for the instance, or enter a name.
- **Location:** Select a location for the resource group. Choose a region where **confidential containers are supported**. Example: **North Europe**.
- **Image:** Accept the default image name. This sample Linux image displays a hardware attestation.

Accept default values for the remaining properties, and then select **Review + create**.

# Custom deployment

Deploy from a custom template



Customized template

1 resource



Edit template



Edit parameters



Visualize

## Project details

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

Subscription \* ⓘ

Windows Fabric - Internal Consumption (PM Team)

Resource group \* ⓘ

(New) myACIRG

Create new

## Instance details

Region \* ⓘ

North Europe

Name ⓘ

helloworld

Location ⓘ

North Europe

Image ⓘ

mcr.microsoft.com/aci/aci-confidential-helloworld:v1

Port ⓘ

80

Cpu Cores ⓘ

1

Memory In Gb ⓘ

1

Restart Policy ⓘ

Never

**Review + create**

< Previous

Next : Review + create >

6. Review the terms and conditions. If you agree, select **I agree to the terms and conditions stated above**.

7. Wait until the **Deployment succeeded** notification appears. It confirms that you successfully created the instance.

**Deployment succeeded**

Deployment 'Microsoft.Template-20230217150208' to resource group 'myACIRG' was successful.

**Pin to dashboard**

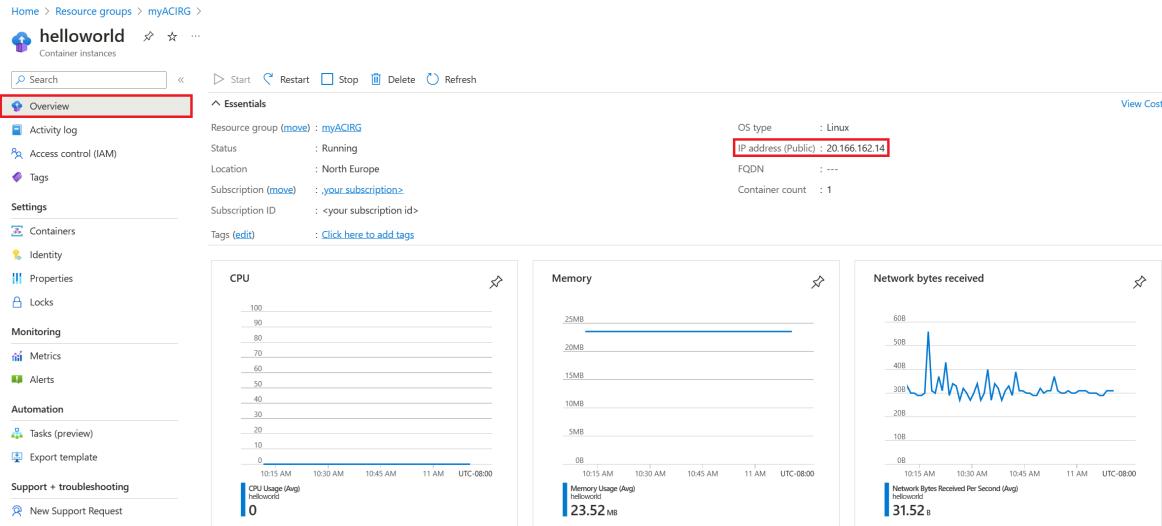
**Go to resource group**

a few seconds ago

# Review deployed resources

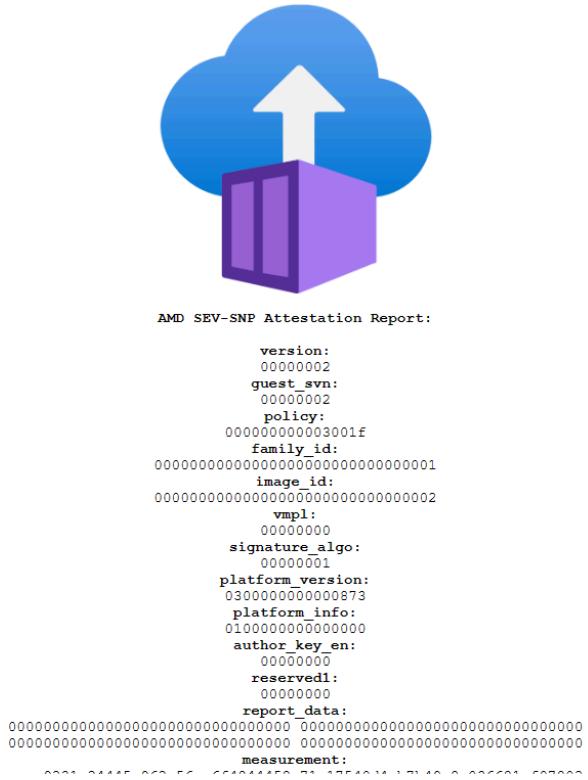
In the following steps, you use the Azure portal to review the properties of the container instance. You can also use a tool such as the [Azure CLI](#).

1. In the portal, search for **Container Instances**, and then select the container instance that you created.
2. On the **Overview** page, note the status of the instance and its IP address.



3. When the status of the instance is **Running**, go to the IP address in your browser.

**Welcome to Confidential containers on Azure Container Instances!**



The presence of the attestation report below the Azure Container Instances logo confirms that the container is running on hardware that supports a TEE.

If you deploy to hardware that doesn't support a TEE (for example, by choosing a region where Confidential Container Instances isn't available), no attestation report appears.

## Related content

Now that you deployed a confidential container group on Container Instances, you can learn more about how policies are enforced:

- [Confidential containers on Azure Container Instances](#)
  - [Azure CLI confcom extension examples ↗](#)
  - [Confidential Hello World application ↗](#)
- 

## Feedback

Was this page helpful?

 Yes

 No

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

# Run containerized tasks with restart policies

Article • 08/29/2024

The ease and speed of deploying containers in Azure Container Instances provides a compelling platform for executing run-once tasks like build, test, and image rendering in a container instance.

With a configurable restart policy, you can specify that your containers are stopped when their processes complete. Because Azure bills container instances by the second, you're only charged for the compute resources used while the container executing your task is running.

The examples presented in this article use the Azure CLI. You must have Azure CLI version 2.0.21 or greater [installed locally](#), or use the CLI in the [Azure Cloud Shell](#).

## Container restart policy

When you create a [container group](#) in Azure Container Instances, you can specify one of three restart policy settings.

[+] Expand table

| Restart policy | Description                                                                                                                                                                                |
|----------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Always         | Containers in the container group are always restarted. This policy is the <b>default</b> setting applied when no restart policy is specified at container creation.                       |
| Never          | Containers in the container group are never restarted. The containers run at most once.                                                                                                    |
| OnFailure      | Containers in the container group are restarted only when the process executed in the container fails (when it terminates with a nonzero exit code). The containers are run at least once. |

### Note

If your container group is configured with an IP address, that IP address can change when the container group is restarted.

# Specify a restart policy

How you specify a restart policy depends on how you create your container instances, such as with the Azure CLI, Azure PowerShell cmdlets, or in the Azure portal. In the Azure CLI, specify the `--restart-policy` parameter when you call [az container create](#).

Azure CLI

```
az container create \
--resource-group myResourceGroup \
--name mycontainer \
--image mycontainerimage \
--restart-policy OnFailure
```

## Run to completion example

To see the restart policy in action, create a container instance from the Microsoft aci-wordcount image, and specify the `OnFailure` restart policy. This example container runs a Python script that, by default, analyzes the text of Shakespeare's [Hamlet](#), writes the 10 most common words to STDOUT, and then exits.

Run the example container with the following [az container create](#) command:

Azure CLI

```
az container create \
--resource-group myResourceGroup \
--name mycontainer \
--image mcr.microsoft.com/azuredocs/aci-wordcount:latest \
--restart-policy OnFailure
```

Azure Container Instances starts the container, and then stops it when its application (or script, in this case) exits. When Azure Container Instances stops a container whose restart policy is `Never` or `OnFailure`, the container's status is set to **Terminated**. You can check a container's status with the [az container show](#) command:

Azure CLI

```
az container show \
--resource-group myResourceGroup \
--name mycontainer \
--query containers[0].instanceView.currentState.state
```

Example output:

## Output

```
"Terminated"
```

Once the example container's status shows *Terminated*, you can see its task output by viewing the container logs. Run the [az container logs](#) command to view the script's output:

## Azure CLI

```
az container logs --resource-group myResourceGroup --name mycontainer
```

Output:

## Output

```
[('the', 990),
 ('and', 702),
 ('of', 628),
 ('to', 610),
 ('I', 544),
 ('you', 495),
 ('a', 453),
 ('my', 441),
 ('in', 399),
 ('HAMLET', 386)]
```

This example shows the output that the script sent to STDOUT. Your containerized tasks, however, might instead write their output to persistent storage for later retrieval. For example, to an [Azure file share](#).

## Next steps

Task-based scenarios, such as batch processing a large dataset with several containers, can take advantage of custom [environment variables](#) or [command lines](#) at runtime.

For details on how to persist the output of your containers that run to completion, see [Mounting an Azure file share with Azure Container Instances](#).

## Feedback

Was this page helpful?

Yes

No



# Set environment variables in container instances

Article • 08/29/2024

Setting environment variables in your container instances allows you to provide dynamic configuration of the application or script run by the container. This feature is similar to the `--env` command-line argument to `docker run`.

To set environment variables in a container, specify them when you create a container instance. This article shows examples of setting environment variables when you start a container with the [Azure CLI](#), [Azure PowerShell](#), and the [Azure portal](#).

For example, if you run the Microsoft aci-wordcount container image, you can modify its behavior by specifying the following environment variables:

*NumWords*: The number of words sent to STDOUT.

*MinLength*: The minimum number of characters in a word for it to be counted. A higher number ignores common words like "of" and "the."

If you need to pass secrets as environment variables, Azure Container Instances supports [secure values](#) for both Windows and Linux containers.

## ⓘ Note

We recommend that you use the Azure Az PowerShell module to interact with Azure. To get started, see [Install Azure PowerShell](#). To learn how to migrate to the Az PowerShell module, see [Migrate Azure PowerShell from AzureRM to Az](#).

## Azure CLI example

To see the default output of the aci-wordcount container, run it first with this `az container create` command (no environment variables specified):

Azure CLI

```
az container create \
 --resource-group myResourceGroup \
 --name mycontainer1 \
 --image mcr.microsoft.com/azuredocs/aci-wordcount:latest \
 --restart-policy OnFailure
```

To modify the output, start a second container with the `--environment-variables` argument added, specifying values for the `NumWords` and `MinLength` variables. (This example assumes you run the CLI in a Bash shell or Azure Cloud Shell. If you use the Windows Command Prompt, specify the variables with double-quotes, such as `--environment-variables "NumWords"="5" "MinLength"="8"`.)

Azure CLI

```
az container create \
 --resource-group myResourceGroup \
 --name mycontainer2 \
 --image mcr.microsoft.com/azuredocs/aci-wordcount:latest \
 --restart-policy OnFailure \
 --environment-variables 'NumWords'='5' 'MinLength'='8'
```

Once both containers' state shows as *Terminated* (use [az container show](#) to check state), display their logs with [az container logs](#) to see the output.

Azure CLI

```
az container logs --resource-group myResourceGroup --name mycontainer1
az container logs --resource-group myResourceGroup --name mycontainer2
```

The outputs of the containers show how you modified the second container's script behavior by setting environment variables.

**mycontainer1**

Output

```
[('the', 990),
 ('and', 702),
 ('of', 628),
 ('to', 610),
 ('I', 544),
 ('you', 495),
 ('a', 453),
 ('my', 441),
 ('in', 399),
 ('HAMLET', 386)]
```

**mycontainer2**

Output

```
[('CLAUDIUS', 120),
 ('POLONIUS', 113),
```

```
('GERTRUDE', 82),
('ROSENCRANTZ', 69),
('GUILDENSTERN', 54)]
```

## Azure PowerShell example

Setting environment variables in PowerShell is similar to the CLI, but uses the `-EnvironmentVariable` command-line argument.

First, launch the aci-wordcount container in its default configuration with this [New-AzContainerGroup](#) command:

```
Azure PowerShell

New-AzContainerGroup
 -ResourceGroupName myResourceGroup
 -Name mycontainer1
 -Image mcr.microsoft.com/azuredocs/aci-wordcount:latest
```

Now run the following [New-AzContainerGroup](#) command. This one specifies the *NumWords* and *MinLength* environment variables after populating an array variable, `envVars`:

```
Azure PowerShell

$envVars = @(
 New-AzContainerInstanceEnvironmentVariableObject -Name "NumWords" -Value
 "5"
 New-AzContainerInstanceEnvironmentVariableObject -Name "MinLength" -
 Value "8"
)

$containerGroup = New-AzContainerGroup -ResourceGroupName "myResourceGroup"
 -Name "mycontainer2"
 -Image "mcr.microsoft.com/azuredocs/aci-wordcount:latest"
 -RestartPolicy "OnFailure"
 -Container @(
 New-AzContainerGroupContainer -Name "mycontainer2"
 -EnvironmentVariable $envVars
)
```

Once both containers' state is *Terminated* (use [Get-AzContainerInstanceLog](#) to check state), pull their logs with the [Get-AzContainerInstanceLog](#) command.

```
Azure PowerShell
```

```
Get-AzContainerInstanceLog -ResourceGroupName myResourceGroup -
ContainerGroupName mycontainer1
Get-AzContainerInstanceLog -ResourceGroupName myResourceGroup -
ContainerGroupName mycontainer2
```

The output for each container shows how you've modified the script run by the container by setting environment variables.

Console

```
PS Azure:\> Get-AzContainerInstanceLog -ResourceGroupName myResourceGroup -
ContainerGroupName mycontainer1
[('the', 990),
 ('and', 702),
 ('of', 628),
 ('to', 610),
 ('I', 544),
 ('you', 495),
 ('a', 453),
 ('my', 441),
 ('in', 399),
 ('HAMLET', 386)]

Azure:\
PS Azure:\> Get-AzContainerInstanceLog -ResourceGroupName myResourceGroup -
ContainerGroupName mycontainer2
[('CLAUDIUS', 120),
 ('POLONIUS', 113),
 ('GERTRUDE', 82),
 ('ROSENCRANTZ', 69),
 ('GUILDENSTERN', 54)]

Azure:\
```

## Azure portal example

To set environment variables when you start a container in the Azure portal, specify them in the **Advanced** page when you create the container.

1. On the **Advanced** page, set the **Restart policy** to *On failure*
2. Under **Environment variables**, enter `NumWords` with a value of `5` for the first variable, and enter `MinLength` with a value of `8` for the second variable.
3. Select **Review + create** to verify and then deploy the container.

**Create container instance**

Basics Networking Advanced Tags Review + create

Configure additional container properties and variables.

Restart policy: On failure

Environment variables:

| KEY       | VALUE |
|-----------|-------|
| NumWords  | 5     |
| MinLength | 8     |

Command override: Example: /bin/bash -c "echo hello", /bin/bash -c "echo have a good day"

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

To view the container's logs, under **Settings** select **Containers**, then **Logs**. Similar to the output shown in the previous CLI and PowerShell sections, you can see how the environment variables change the script's behavior. Only five words are displayed, each with a minimum length of eight characters.

Home > mycontainer - Containers

### mycontainer - Containers

Container instances

Search (Ctrl+ /)

Refresh

1 container

| NAME        | IMAGE                   | STATE      | START TIME           | RESTART COUNT |
|-------------|-------------------------|------------|----------------------|---------------|
| mycontainer | microsoft/aci-wordcount | Terminated | 2018-05-15T17:23:34Z | 0             |

SETTINGS

- Containers
- Properties
- Locks
- Automation script

MONITORING

Events Properties Logs

```
[('CLAUDIUS', 120),
 ('POLONIUS', 113),
 ('GERTRUDE', 82),
 ('ROSENCRANTZ', 69),
 ('GUILDENSTERN', 54)]
```

## Secure values

Objects with secure values are intended to hold sensitive information like passwords or keys for your application. Using secure values for environment variables is both safer and more flexible than including it in your container's image. Another option is to use secret volumes, described in [Mount a secret volume in Azure Container Instances](#).

Environment variables with secure values aren't visible in your container's properties-- their values can be accessed only from within the container. For example, container properties viewed in the Azure portal or Azure CLI display only a secure variable's name, not its value.

Set a secure environment variable by specifying the `secureValue` property instead of the regular `value` for the variable's type. The two variables defined in the following YAML demonstrate the two variable types.

## YAML deployment

Create a `secure-env.yaml` file with the following snippet.

YAML

```
apiVersion: 2019-12-01
location: eastus
name: securetest
properties:
 containers:
 - name: mycontainer
 properties:
 environmentVariables:
 - name: 'NOTSECRET'
 value: 'my-exposed-value'
 - name: 'SECRET'
 secureValue: 'my-secret-value'
 image: mcr.microsoft.com/oss/nginx/nginx:1.15.5-alpine
 ports: []
 resources:
 requests:
 cpu: 1.0
 memoryInGB: 1.5
 osType: Linux
 restartPolicy: Always
 tags: null
 type: Microsoft.ContainerInstance/containerGroups
```

Run the following command to deploy the container group with YAML (adjust the resource group name as necessary):

Azure CLI

```
az container create --resource-group myResourceGroup --file secure-env.yaml
```

## Verify environment variables

Run the [az container show](#) command to query your container's environment variables:

Azure CLI

```
az container show --resource-group myResourceGroup --name securetest --query
'containers[].environmentVariables'
```

The JSON response shows both the insecure environment variable's key and value, but only the name of the secure environment variable:

JSON

```
[
 [
 {
 "name": "NOTSECRET",
 "secureValue": null,
 "value": "my-exposed-value"
 },
 {
 "name": "SECRET",
 "secureValue": null,
 "value": null
 }
]
]
```

With the [az container exec](#) command, which enables executing a command in a running container, you can verify that the secure environment variable is set. Run the following command to start an interactive bash session in the container:

Azure CLI

```
az container exec --resource-group myResourceGroup --name securetest --exec-
command "/bin/sh"
```

Once you open an interactive shell within the container, you can access the `SECRET` variable's value:

Console

```
root@caas-ef3ee231482549629ac8a40c0d3807fd-3881559887-53741:/# echo $SECRET
my-secret-value
```

## Next steps

Task-based scenarios, such as batch processing a large dataset with several containers, can benefit from custom environment variables at runtime. For more information about running task-based containers, see [Run containerized tasks with restart policies](#).

---

## Feedback

Was this page helpful?



Yes



No

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

# Set the command line in a container instance to override the default command line operation

Article • 08/29/2024

When you create a container instance, optionally specify a command to override the default command line instruction baked into the container image. This behavior is similar to the `--entrypoint` command-line argument to `docker run`.

Like setting [environment variables](#) for container instances, specifying a starting command line is useful for batch jobs where you need to prepare each container dynamically with task-specific configuration.

## Command line guidelines

- By default, the command line specifies a *single process that starts without a shell* in the container. For example, the command line might run a Python script or executable file. The process can specify additional parameters or arguments.
- To execute multiple commands, begin your command line by setting a shell environment that is supported in the container operating system. Examples:

[+] [Expand table](#)

| Operating system | Default shell |
|------------------|---------------|
| Ubuntu           | /bin/bash     |
| Alpine           | /bin/sh       |
| Windows          | cmd           |

Follow the conventions of the shell to combine multiple commands to run in sequence.

- Depending on the container configuration, you might need to set a full path to the command line executable or arguments.
- Set an appropriate [restart policy](#) for the container instance, depending on whether the command-line specifies a long-running task or a run-once task. For example, a restart policy of `Never` or `OnFailure` is recommended for a run-once task.

- If you need information about the default entrypoint set in a container image, use the [docker image inspect](#) command.

## Command line syntax

The command line syntax varies depending on the Azure API or tool used to create the instances. If you specify a shell environment, also observe the command syntax conventions of the shell.

- [az container create](#) command: Pass a string with the `--command-line` parameter.  
Example: `--command-line "python myscript.py arg1 arg2"`.
- [New-AzureRmContainerGroup](#) Azure PowerShell cmdlet: Pass a string with the `-Command` parameter. Example: `-Command "echo hello"`.
- Azure portal: In the **Command override** property of the container configuration, provide a comma-separated list of strings, without quotes. Example: `python, myscript.py, arg1, arg2`.
- Resource Manager template or YAML file, or one of the Azure SDKs: Specify the command line property as an array of strings. Example: the JSON array `["python", "myscript.py", "arg1", "arg2"]` in a Resource Manager template.

If you're familiar with [Dockerfile](#) syntax, this format is similar to the *exec* form of the CMD instruction.

## Examples

[\[ \]](#) Expand table

|                          | Azure CLI                                                                                     | Portal                                                                                                               | Template                                                                                        |
|--------------------------|-----------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------|
| <b>Single command</b>    | <code>--command-line "python myscript.py arg1 arg2"</code>                                    | <b>Command override:</b><br><code>python, myscript.py, arg1, arg2</code>                                             | <code>"command": ["python", "myscript.py", "arg1", "arg2"]</code>                               |
| <b>Multiple commands</b> | <code>--command-line "/bin/bash -c 'mkdir test; touch test/myfile; tail -f /dev/null'"</code> | <b>Command override:</b><br><code>/bin/bash, -c, mkdir test; touch test; touch test/myfile; tail -f /dev/null</code> | <code>"command": ["/bin/bash", "-c", "mkdir test; touch test/myfile; tail -f /dev/null"]</code> |

# Azure CLI example

As an example, modify the behavior of the microsoft/aci-wordcount container image, which analyzes text in Shakespeare's *Hamlet* to find the most frequently occurring words. Instead of analyzing *Hamlet*, you could set a command line that points to a different text source.

To see the output of the microsoft/aci-wordcount container when it analyzes the default text, run it with the following [az container create](#) command. No start command line is specified, so the default container command runs. For illustration purposes, this example sets [environment variables](#) to find the top three words that are at least five letters long:

```
Azure CLI

az container create \
 --resource-group myResourceGroup \
 --name mycontainer1 \
 --image mcr.microsoft.com/azuredocs/aci-wordcount:latest \
 --environment-variables NumWords=3 MinLength=5 \
 --restart-policy OnFailure
```

Once the container's state shows as *Terminated* (use [az container show](#) to check state), display the log with [az container logs](#) to see the output.

```
Azure CLI

az container logs --resource-group myResourceGroup --name mycontainer1

Output

[('HAMLET', 386), ('HORATIO', 127), ('CLAUDIUS', 120)]
```

Now set up a second example container to analyze different text by specifying a different command line. The Python script executed by the container, *wordcount.py*, accepts a URL as an argument, and processes that page's content instead of the default.

For example, to determine the top three words that are at least five letters long in *Romeo and Juliet*:

```
Azure CLI

az container create \
 --resource-group myResourceGroup \
 --name mycontainer2 \
 --image mcr.microsoft.com/azuredocs/aci-wordcount:latest \
```

```
--restart-policy OnFailure \
--environment-variables NumWords=3 MinLength=5 \
--command-line "python wordcount.py
http://shakespeare.mit.edu/romeo_juliet/full.html"
```

Again, once the container is *Terminated*, view the output by showing the container's logs:

Azure CLI

```
az container logs --resource-group myResourceGroup --name mycontainer2
```

Output

```
[('ROMEO', 177), ('JULIET', 134), ('CAPULET', 119)]
```

## Next steps

Task-based scenarios, such as batch processing a large dataset with several containers, can benefit from custom command lines at runtime. For more information about running task-based containers, see [Run containerized tasks with restart policies](#).

## Feedback

Was this page helpful?

 Yes

 No

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

# Execute a command in a running Azure container instance

Article • 08/29/2024

Azure Container Instances supports executing a command in a running container. Running a command in a started container is especially helpful during application development and troubleshooting. The most common use of this feature is to launch an interactive shell so that you can debug issues in a running container.

## Run a command with Azure CLI

Execute a command in a running container with [az container exec](#) in the [Azure CLI](#):

Azure CLI

```
az container exec --resource-group <group-name> --name <container-group-name> --exec-command "<command>"
```

For example, to launch a Bash shell in an Nginx container:

Azure CLI

```
az container exec --resource-group myResourceGroup --name mynginx --exec-command "/bin/bash"
```

In the following example output, the Bash shell is launched in a running Linux container, providing a terminal in which `ls` is executed:

Output

```
root@caas-83e6c883014b427f9b277a2bba3b7b5f-708716530-2qv47:/# ls
bin dev home lib64 mnt proc run srv tmp var
boot etc lib media opt root sbin sys usr
root@caas-83e6c883014b427f9b277a2bba3b7b5f-708716530-2qv47:/# exit
exit
Bye.
```

In this example, Command Prompt is launched in a running Nanoserver container:

Azure CLI

```
az container exec --resource-group myResourceGroup --name myiis --exec-command "cmd.exe"
```

## Output

```
Microsoft Windows [Version 10.0.14393]
(c) 2016 Microsoft Corporation. All rights reserved.
```

```
C:\>dir
Volume in drive C has no label.
Volume Serial Number is 76E0-C852
```

```
Directory of C:\
```

```
03/23/2018 09:13 PM <DIR> inetpub
11/20/2016 11:32 AM 1,894 License.txt
03/23/2018 09:13 PM <DIR> Program Files
07/16/2016 12:09 PM <DIR> Program Files (x86)
03/13/2018 08:50 PM 171,616 ServiceMonitor.exe
03/23/2018 09:13 PM <DIR> Users
03/23/2018 09:12 PM <DIR> var
03/23/2018 09:22 PM <DIR> Windows
 2 File(s) 173,510 bytes
 6 Dir(s) 21,171,609,600 bytes free
```

```
C:\>exit
Bye.
```

## Multi-container groups

If your [container group](#) has multiple containers, such as an application container and a logging sidecar, specify the name of the container in which to run the command with `--container-name`.

For example, in the container group *mynginx* are two containers, *nginx-app* and *logger*. To launch a shell on the *nginx-app* container:

### Azure CLI

```
az container exec --resource-group myResourceGroup --name mynginx --container-name nginx-app --exec-command "/bin/bash"
```

## Restrictions

Azure Container Instances currently supports launching a single process with [az container exec](#), and you can't pass command arguments. For example, you can't chain commands like in `sh -c "echo FOO && echo BAR"`, or execute `echo FOO`.

## Next steps

Learn about other troubleshooting tools and common deployment issues in [Troubleshoot container and deployment issues in Azure Container Instances](#).

---

## Feedback

Was this page helpful?

 Yes

 No

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

# Run an init container for setup tasks in a container group

Article • 08/29/2024

Azure Container Instances supports *init containers* in a container group. Init containers run to completion before the application container or containers start. Similar to [Kubernetes init containers](#), use one or more init containers to perform initialization logic for your app containers, such as setting accounts, running setup scripts, or configuring databases.

This article shows how to use an Azure Resource Manager template to configure a container group with an init container.

## Things to know

- **API version** - You need at least Azure Container Instances API version 2019-12-01 to deploy init containers. Deploy using an `initContainers` property in a [YAML file](#) or a [Resource Manager template](#).
- **Order of execution** - Init containers are executed in the order specified in the template, and before other containers. By default, you can specify a maximum of 59 init containers per container group. At least one non-init container must be in the group.
- **Host environment** - Init containers run on the same hardware as the rest of the containers in the group.
- **Resources** - You don't specify resources for init containers. They're granted the total resources such as CPUs and memory available to the container group. While an init container runs, no other containers run in the group.
- **Supported properties** - Init containers can use some group properties such as volumes and secrets. However, they can't use ports, IP address and managed identities if configured for the container group.
- **Restart policy** - Each init container must exit successfully before the next container in the group starts. If an init container doesn't exit successfully, its restart action depends on the [restart policy](#) configured for the group:

[+] Expand table

| Policy in group | Policy in init |
|-----------------|----------------|
| Always          | OnFailure      |
| OnFailure       | OnFailure      |
| Never           | Never          |

- **Charges** - The container group incurs charges from the first deployment of an init container.

## Resource Manager template example

Start by copying the following JSON into a new file named `azuredeploy.json`. The template sets up a container group with one init container and two application containers:

- The *init1* container runs the [busybox](#) image. It sleeps for 60 seconds and then writes a command-line string to a file in an [emptyDir volume](#).
- Both application containers run the Microsoft `aci-wordcount` container image:
  - The *hamlet* container runs the wordcount app in its default configuration, counting word frequencies in Shakespeare's play *Hamlet*.
  - The *juliet* app container reads the command-line string from the `emptyDir` volume to run the wordcount app instead on Shakespeare's *Romeo and Juliet*.

For more information and examples using the `aci-wordcount` image, see [Set environment variables in container instances](#).

JSON

```
{
 "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",
 "contentVersion": "1.0.0.0",
 "parameters": {
 "containerGroupName": {
 "type": "string",
 "defaultValue": "myContainerGroup",
 "metadata": {
 "description": "Container Group name."
 }
 }
 },
 "resources": [
 {
 "name": "[parameters('containerGroupName')]",
 "type": "Microsoft.ContainerInstance/containerGroups",
 "properties": {
 "containers": [
 {
 "name": "init1",
 "image": "busybox",
 "type": "init",
 "command": [
 "sh",
 "-c",
 "sleep 60 & echo 'Hello world' > /tmp/test.txt"
],
 "volumeMounts": [
 {
 "name": "emptyDir",
 "mountPath": "/tmp"
 }
]
 },
 {
 "name": "hamlet",
 "image": "microsoft/aci-wordcount:latest",
 "volumeMounts": [
 {
 "name": "emptyDir",
 "mountPath": "/app"
 }
]
 },
 {
 "name": "juliet",
 "image": "microsoft/aci-wordcount:latest",
 "environmentVariables": [
 {
 "name": "WORDCOUNT_STRING",
 "value": "Hello world"
 }
],
 "volumeMounts": [
 {
 "name": "emptyDir",
 "mountPath": "/app"
 }
]
 }
],
 "osType": "Linux"
 }
 }
]
}
```

```
"apiVersion": "2019-12-01",
"location": "[resourceGroup().location]",
"properties": {
 "sku": "Standard",
 "initContainers": [
 {
 "name": "init1",
 "properties": {
 "image": "mcr.microsoft.com/aks/e2e/library-
busybox:master.210714.1",
 "environmentVariables": [],
 "volumeMounts": [
 {
 "name": "emptydir1",
 "mountPath": "/mnt/emptydir"
 }
],
 "command": [
 "/bin/sh",
 "-c",
 "sleep 60; echo python wordcount.py
http://shakespeare.mit.edu/romeo_juliet/full.html >
/mnt/emptydir/command_line.txt"
]
 }
 }
],
 "containers": [
 {
 "name": "hamlet",
 "properties": {
 "image": "mcr.microsoft.com/azuredocs/aci-
wordcount",
 "volumeMounts": [
 {
 "name": "emptydir1",
 "mountPath": "/mnt/emptydir"
 }
],
 "environmentVariables": [
 {
 "name": "NumWords",
 "value": "3"
 },
 {
 "name": "MinLength",
 "value": "5"
 }
],
 "resources": {
 "requests": {
 "memoryInGB": 1.0,
 "cpu": 1
 }
 }
 }
 }
]
}
```

```

 },
 {
 "name": "juliet",
 "properties": {
 "image": "mcr.microsoft.com/azuredocs/aci-
wordcount",
 "volumeMounts": [
 {
 "name": "emptydir1",
 "mountPath": "/mnt/emptydir"
 }
],
 "command": [
 "/bin/sh",
 "-c",
 "$(cat /mnt/emptydir/command_line.txt)"
],
 "environmentVariables": [
 {
 "name": "NumWords",
 "value": "3"
 },
 {
 "name": "MinLength",
 "value": "5"
 }
],
 "resources": {
 "requests": {
 "memoryInGB": 1.0,
 "cpu": 1
 }
 }
 }
],
 "restartPolicy": "OnFailure",
 "osType": "Linux",
 "volumes": [
 {
 "name": "emptydir1",
 "emptyDir": {}
 }
],
 "tags": {}
 }
]
}

```

## Deploy the template

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

Azure CLI

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

Deploy the template with the [az deployment group create](#) command.

Azure CLI

```
az deployment group create \
--resource-group myResourceGroup \
--template-file azuredeploy.json
```

In a group with an init container, the deployment time is increased because of the time it takes for the init container or containers to complete.

## View container logs

To verify the init container ran successfully, view the log output of the app containers using the [az container logs](#) command. The `--container-name` argument specifies the container from which to pull logs. In this example, pull the logs for the *hamlet* and *juliet* containers, which show different command output:

Azure CLI

```
az container logs \
--resource-group myResourceGroup \
--name myContainerGroup \
--container-name hamlet
```

Output:

Console

```
[('HAMLET', 386), ('HORATIO', 127), ('CLAUDIUS', 120)]
```

Azure CLI

```
az container logs \
--resource-group myResourceGroup \
--name myContainerGroup \
--container-name juliet
```

Output:

Console

```
[('ROMEO', 177), ('JULIET', 134), ('CAPULET', 119)]
```

## Next steps

Init containers help you perform setup and initialization tasks for your application containers. For more information about running task-based containers, see [Run containerized tasks with restart policies](#).

Azure Container Instances provides other options to modify the behavior of application containers. Examples include:

- [Set environment variables in container instances](#)
- [Set the command line in a container instance to override the default command line operation](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# Config maps for Azure Container Instances (Preview)

Article • 11/19/2024

A config map is a property that can be used to apply container configurations similar to environment variables and secret volumes. However, unlike when using environment variables or secret volumes where restarting the pod to apply the settings is required, applying settings using a config map does not require any restarts for the changes to take effect.

Azure Container Instances can be created with or without config maps and can be updated at any point in time post creation using config maps. Updating config maps in an existing running container group can be accomplished quickly without compromising uptime of the container.

## How it works

A config map can be included in the container properties or in a container group profile. Creating a container group profile with the config map settings makes applying those settings simple and easy to automate.

## Create a container group profile with config map settings

CLI

Create a container group profile with config map settings using [az container container-group-profile create](#).

Azure CLI

```
az container container-group-profile create \
 --resource-group myResourceGroup \
 --name myContainerGroupProfile \
 --location WestCentralUS \
 --image nginx \
 --os-type Linux \
 --ip-address Public \
 --ports 8000 \
 --cpu 1 \
 --memory 1.5 \
```

```
--restart-policy never \
--config-map key1=value1 key2=value2
```

## Apply config map settings using a container group profile

Applying the config map settings stored in a container group profile requires updating the container and specifying the container group profile that should be associated with the update.

CLI

Apply the config map settings stored in the container group profile using [az container create](#).

Azure CLI

```
az container create \
 --resource-group myResourceGroup \
 --name myContainer \
 --location WestCentralUS \
 --container-group-profile-id
"/subscriptions/{SubscriptionID}/resourceGroups/myResourceGroup/providers/Microsoft.ContainerInstance/containerGroupProfiles/myContainerGroupProfile" \
 --container-group-profile-revision 1
```

## Apply config map settings without container group profile

Config map settings can also be applied directly to the instance by specifying the config map settings in the create commands.

CLI

Apply the config map settings using [az container create](#).

Azure CLI

```
az container create \
 --resource-group myResourceGroup \
 --name myContainer \
 --location WestCentralUS \
```

```
--config-map key1=value1 key2=value2
```

Once the update has been applied to an existing container and you will see the values mounted in the container without requiring a restart.

```
/mnt/configmap/<containername>/key1 with value as "value1"
```

```
/mnt/configmap/<containername>/key2 with value as "value2"
```

## Next steps

[Learn how to use config maps with standby pools to increase scale and availability](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# How to use managed identities with Azure Container Instances

Article • 08/29/2024

Use [managed identities for Azure resources](#) to run code in Azure Container Instances that interacts with other Azure services - without maintaining any secrets or credentials in code. The feature provides an Azure Container Instances deployment with an automatically managed identity in Microsoft Entra ID.

In this article, you learn more about managed identities in Azure Container Instances and:

- ✓ Enable a user-assigned or system-assigned identity in a container group
- ✓ Grant the identity access to an Azure key vault
- ✓ Use the managed identity to access a key vault from a running container

Adapt the examples to enable and use identities in Azure Container Instances to access other Azure services. These examples are interactive. However, in practice your container images would run code to access Azure services.

## Why use a managed identity?

Use a managed identity in a running container to authenticate to any [service that supports Microsoft Entra authentication](#) without managing credentials in your container code. For services that don't support AD authentication, you can store secrets in an Azure key vault and use the managed identity to access the key vault to retrieve credentials. For more information about using a managed identity, see [What is managed identities for Azure resources?](#)

## Enable a managed identity

When you create a container group, enable one or more managed identities by setting a [ContainerGroupIdentity](#) property. You can also enable or update managed identities after a container group is running - either action causes the container group to restart. To set the identities on a new or existing container group, use the Azure CLI, a Resource Manager template, a YAML file, or another Azure tool.

Azure Container Instances supports both types of managed Azure identities: user-assigned and system-assigned. On a container group, you can enable a system-assigned

identity, one or more user-assigned identities, or both types of identities. If you're unfamiliar with managed identities for Azure resources, see the [overview](#).

## Use a managed identity

To use a managed identity, the identity must be granted access to one or more Azure service resources (such as a web app, a key vault, or a storage account) in the subscription. Using a managed identity in a running container is similar to using an identity in an Azure Virtual Machine (VM). See the VM guidance for using a [token](#), [Azure PowerShell](#) or [Azure CLI](#), or the [Azure SDKs](#).

## Prerequisites

- Use the Bash environment in [Azure Cloud Shell](#). For more information, see [Quickstart for Bash in Azure Cloud Shell](#).  
A blue rectangular button with a white 'A' icon and the text 'Launch Cloud Shell'. To the right of the button is a small blue square icon with a white arrow pointing outwards.
- If you prefer to run CLI reference commands locally, [install](#) the Azure CLI. If you're running on Windows or macOS, consider running Azure CLI in a Docker container. For more information, see [How to run the Azure CLI in a Docker container](#).
  - If you're using a local installation, sign in to the Azure CLI by using the `az login` command. To finish the authentication process, follow the steps displayed in your terminal. For other sign-in options, see [Sign in with the Azure CLI](#).
  - When you're prompted, install the Azure CLI extension on first use. For more information about extensions, see [Use extensions with the Azure CLI](#).
  - Run `az version` to find the version and dependent libraries that are installed. To upgrade to the latest version, run `az upgrade`.
- This article requires version 2.0.49 or later of the Azure CLI. If using Azure Cloud Shell, the latest version is already installed.

## Create an Azure key vault

The examples in this article use a managed identity in Azure Container Instances to access an Azure key vault secret.

First, create a resource group named `myResourceGroup` in the `eastus` location with the following `az group create` command:

Azure CLI

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

Use the [az keyvault create](#) command to create a key vault. Be sure to specify a unique key vault name.

Azure CLI

```
az keyvault create \
--name mykeyvault \
--resource-group myResourceGroup \
--location eastus
```

Store a sample secret in the key vault using the [az keyvault secret set](#) command:

Azure CLI

```
az keyvault secret set \
--name SampleSecret \
--value "Hello Container Instances" \
--description ACIsecret --vault-name mykeyvault
```

Continue with the following examples to access the key vault using either a user-assigned or system-assigned managed identity in Azure Container Instances.

## Example 1: Use a user-assigned identity to access Azure key vault

### Create an identity

First create an identity in your subscription using the [az identity create](#) command. You can use the same resource group used to create the key vault, or use a different one.

Azure CLI

```
az identity create \
--resource-group myResourceGroup \
--name myACIID
```

To use the identity in the following steps, use the [az identity show](#) command to store the identity's service principal ID and resource ID in variables.

## Azure CLI

```
Get service principal ID of the user-assigned identity
$SP_ID=$(az identity show \
 --resource-group myResourceGroup \
 --name myACIId \
 --query principalId --output tsv)

Get resource ID of the user-assigned identity
$RESOURCE_ID=$(az identity show \
 --resource-group myResourceGroup \
 --name myACIId \
 --query id --output tsv)
```

## Grant user-assigned identity access to the key vault

Run the following [az keyvault set-policy](#) command to set an access policy on the key vault. The following example allows the user-assigned identity to get secrets from the key vault:

## Azure CLI

```
az keyvault set-policy \
 --name mykeyvault \
 --resource-group myResourceGroup \
 --object-id $SP_ID \
 --secret-permissions get
```

## Enable user-assigned identity on a container group

Run the following [az container create](#) command to create a container instance based on Microsoft's `azure-cli` image. This example provides a single-container group that you can use interactively to run the Azure CLI to access other Azure services. In this section, only the base operating system is used. For an example to use the Azure CLI in the container, see [Enable system-assigned identity on a container group](#).

The `--assign-identity` parameter passes your user-assigned managed identity to the group. The long-running command keeps the container running. This example uses the same resource group used to create the key vault, but you could specify a different one.

## Azure CLI

```
az container create \
 --resource-group myResourceGroup \
 --name mycontainer \
```

```
--image mcr.microsoft.com/azure-cli \
--assign-identity $RESOURCE_ID \
--command-line "tail -f /dev/null"
```

Within a few seconds, you should get a response from the Azure CLI indicating that the deployment completed. Check its status with the [az container show](#) command.

Azure CLI

```
az container show \
--resource-group myResourceGroup \
--name mycontainer
```

The `identity` section in the output looks similar to the following, showing the identity is set in the container group. The `principalID` under `userAssignedIdentities` is the service principal of the identity you created in Microsoft Entra ID:

Output

```
[...]
"identity": {
 "principalId": "null",
 "tenantId": "xxxxxxxx-f292-4e60-9122-xxxxxxxxxxxx",
 "type": "UserAssigned",
 "userAssignedIdentities": {
 "/subscriptions/xxxxxxxx-0903-4b79-a55a-
xxxxxxxxxxxx/resourcegroups/danlep1018/providers/Microsoft.ManagedIdentity/u
serAssignedIdentities/myACIId": {
 "clientId": "xxxxxxxx-5523-45fc-9f49-xxxxxxxxxxxx",
 "principalId": "xxxxxxxx-f25b-4895-b828-xxxxxxxxxxxx"
 }
 }
},
[...]
```

## Use user-assigned identity to get secret from key vault

Now you can use the managed identity within the running container instance to access the key vault. First launch a bash shell in the container:

Azure CLI

```
az container exec \
--resource-group myResourceGroup \
--name mycontainer \
--exec-command "/bin/bash"
```

Run the following commands in the bash shell in the container. To get an access token to use Microsoft Entra ID to authenticate to key vault, run the following command:

Bash

```
client_id="xxxxxxxx-5523-45fc-9f49-xxxxxxxxxxxx"
curl "http://169.254.169.254/metadata/identity/oauth2/token?api-
version=2018-02-
01&resource=https%3A%2F%2Fvault.azure.net&client_id=$client_id" -H
Metadata:true -s
```

Output:

Bash

```
{"access_token": "xxxxxxxxxxxxxxxxxxxxxxxxxxxxx1QiLCJhbGciOiJSUzI1NiIsIng1
dCI6Imk2bEdrM0ZaenhSY1ViMkMzbkVRN3N5SEpsWSIsImtpZCI6Imk2bEdrM0ZaenhSY1ViMkMz
bkVRN3N5SEpsWSJ9.....xxxxxxxxxxxxxx", "refresh_token": "", "expires_in": "28
799", "expires_on": "1539927532", "not_before": "1539898432", "resource": "https:/
/vault.azure.net/", "token_type": "Bearer"}
```

For Windows containers, metadata server (169.254.169.254) isn't available. Run the following or equivalent commands to get an access token.

Console

```
curl -G -v %IDENTITY_ENDPOINT% --data-urlencode
resource=https://vault.azure.net --data-urlencode principalId=<principal id>
-H secret:%IDENTITY_HEADER%
```

To store the access token in a variable to use in subsequent commands to authenticate, run the following command:

Bash

```
TOKEN=$(curl 'http://169.254.169.254/metadata/identity/oauth2/token?api-
version=2018-02-01&resource=https%3A%2F%2Fvault.azure.net' -H Metadata:true
| jq -r '.access_token')
```

Now use the access token to authenticate to key vault and read a secret. Be sure to substitute the name of your key vault in the URL ([https://mykeyvault.vault.azure.net/...](https://mykeyvault.vault.azure.net/)):

Bash

```
curl https://mykeyvault.vault.azure.net/secrets/SampleSecret/?api-version=7.4 -H "Authorization: Bearer $TOKEN"
```

The response looks similar to the following, showing the secret. In your code, you would parse this output to obtain the secret. Then, use the secret in a subsequent operation to access another Azure resource.

Bash

```
{"value":"Hello Container Instances","contentType":"ACIsecret","id":"https://mykeyvault.vault.azure.net/secrets/SampleSecret/xxxxxxxxxxxxxxxxxxxx","attributes":{"enabled":true,"created":1539965967,"updated":1539965967,"recoveryLevel":"Purgeable"},"tags":{"file-encoding":"utf-8"}}
```

## Example 2: Use a system-assigned identity to access Azure key vault

### Enable system-assigned identity on a container group

Run the following `az container create` command to create a container instance based on Microsoft's `azure-cli` image. This example provides a single-container group that you can use interactively to run the Azure CLI to access other Azure services.

The `--assign-identity` parameter with no additional value enables a system-assigned managed identity on the group. The identity is scoped to the resource group of the container group. The long-running command keeps the container running. This example uses the same resource group used to create the key vault, which is in the scope of the identity.

Azure CLI

```
Get the resource ID of the resource group
RG_ID=$(az group show --name myResourceGroup --query id --output tsv)

Create container group with system-managed identity
az container create \
--resource-group myResourceGroup \
--name mycontainer \
--image mcr.microsoft.com/azure-cli \
--assign-identity --scope $RG_ID \
--command-line "tail -f /dev/null"
```

Within a few seconds, you should get a response from the Azure CLI indicating that the deployment completed. Check its status with the [az container show](#) command.

```
Azure CLI

az container show \
--resource-group myResourceGroup \
--name mycontainer
```

The `identity` section in the output looks similar to the following, showing that a system-assigned identity is created in Microsoft Entra ID:

```
Output

[...]
"identity": {
 "principalId": "xxxxxxxx-528d-7083-b74c-xxxxxxxxxxxx",
 "tenantId": "xxxxxxxx-f292-4e60-9122-xxxxxxxxxxxx",
 "type": "SystemAssigned",
 "userAssignedIdentities": null
},
[...]
```

Set a variable to the value of `principalId` (the service principal ID) of the identity, to use in later steps.

```
Azure CLI

SP_ID=$(az container show \
--resource-group myResourceGroup \
--name mycontainer \
--query identity.principalId --out tsv)
```

## Grant container group access to the key vault

Run the following [az keyvault set-policy](#) command to set an access policy on the key vault. The following example allows the system-managed identity to get secrets from the key vault:

```
Azure CLI

az keyvault set-policy \
--name mykeyvault \
--resource-group myResourceGroup \
```

```
--object-id $SP_ID \
--secret-permissions get
```

## Use container group identity to get secret from key vault

Now you can use the managed identity to access the key vault within the running container instance. First launch a bash shell in the container:

Azure CLI

```
az container exec \
--resource-group myResourceGroup \
--name mycontainer \
--exec-command "/bin/bash"
```

Run the following commands in the bash shell in the container. First, sign in to the Azure CLI using the managed identity:

Azure CLI

```
az login --identity
```

From the running container, retrieve the secret from the key vault:

Azure CLI

```
az keyvault secret show \
--name SampleSecret \
--vault-name mykeyvault --query value
```

The value of the secret is retrieved:

Output

```
"Hello Container Instances"
```

For Windows containers, the 'az login' command won't work because the metadata server is unavailable. Additionally, a managed identity token can't be generated in a Windows virtual network container.

## Enable managed identity using Resource Manager template

To enable a managed identity in a container group using a [Resource Manager template](#), set the `identity` property of the `Microsoft.ContainerInstance/containerGroups` object with a `ContainerGroupIdentity` object. The following snippets show the `identity` property configured for different scenarios. See the [Resource Manager template reference](#). Specify a minimum `apiVersion` of `2018-10-01`.

## User-assigned identity

A user-assigned identity is a resource ID of the form:

```
"/subscriptions/{subscriptionId}/resourceGroups/{resourceGroupName}/providers/Microsoft.ManagedIdentity/userAssignedIdentities/{identityName}"
```

You can enable one or more user-assigned identities.

JSON

```
"identity": {
 "type": "UserAssigned",
 "userAssignedIdentities": {
 "myResourceId1": {}
 }
}
```

## System-assigned identity

JSON

```
"identity": {
 "type": "SystemAssigned"
}
```

## System- and user-assigned identities

On a container group, you can enable both a system-assigned identity and one or more user-assigned identities.

JSON

```
"identity": {
 "type": "SystemAssigned, UserAssigned",
 "userAssignedIdentities": {
 "myResourceId1": {}
 }
}
...
}
```

## Enable managed identity using YAML file

To enable a managed identity in a container group deployed using a [YAML file](#), include the following YAML. Specify a minimum `apiVersion` of `2018-10-01`.

### User-assigned identity

A user-assigned identity is a resource ID of the form

```
'/subscriptions/{subscriptionId}/resourceGroups/{resourceGroupName}/providers/Microsoft.ManagedIdentity/userAssignedIdentities/{identityName}'
```

You can enable one or more user-assigned identities.

YAML

```
identity:
 type: UserAssigned
 userAssignedIdentities:
 {'myResourceId1':{}}
```

### System-assigned identity

YAML

```
identity:
 type: SystemAssigned
```

## System- and user-assigned identities

On a container group, you can enable both a system-assigned identity and one or more user-assigned identities.

yml

```
identity:
 type: SystemAssigned, UserAssigned
 userAssignedIdentities:
 {'myResourceId1':{}}
```

## Next steps

In this article, you learned about managed identities in Azure Container Instances and how to:

- ✓ Enable a user-assigned or system-assigned identity in a container group
- ✓ Grant the identity access to an Azure key vault
- ✓ Use the managed identity to access a key vault from a running container
  - Learn more about [managed identities for Azure resources](#).
  - See an [Azure Go SDK example](#) of using a managed identity to access a key vault from Azure Container Instances.

---

## Feedback

Was this page helpful?

 Yes

 No

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

# Deploy container instances that use GPU resources

Article • 08/29/2024

To run certain compute-intensive workloads on Azure Container Instances, deploy your [container groups](#) with *GPU resources*. The container instances in the group can access one or more NVIDIA Tesla GPUs while running container workloads such as CUDA and deep learning applications.

This article shows how to add GPU resources when you deploy a container group by using a [YAML file](#) or [Resource Manager template](#). You can also specify GPU resources when you deploy a container instance using the Azure portal.

## ⓘ Important

K80 and P100 GPU SKUs are retiring by August 31st, 2023. This is due to the retirement of the underlying VMs used: [NC Series](#) and [NCv2 Series](#). Although V100 SKUs will be available, it is recommended to use Azure Kubernetes Service instead. GPU resources are not fully supported and should not be used for production workloads. Use the following resources to migrate to AKS today: [How to Migrate to AKS](#).

## ⓘ Important

This feature is currently in preview, and some [limitations apply](#). Previews are made available to you on the condition that you agree to the [supplemental terms of use](#). Some aspects of this feature may change prior to general availability (GA).

## Prerequisites

### ⓘ Note

Due to some current limitations, not all limit increase requests are guaranteed to be approved.

- If you would like to use this sku for your production container deployments, create an [Azure Support request](#) to increase the limit.

# Preview limitations

In preview, the following limitations apply when using GPU resources in container groups.

## Region availability

[+] Expand table

| Regions                                                        | OS    | Available GPU SKUs |
|----------------------------------------------------------------|-------|--------------------|
| East US, West Europe, West US 2, Southeast Asia, Central India | Linux | V100               |

Support will be added for additional regions over time.

**Supported OS types:** Linux only

**Additional limitations:** GPU resources can't be used when deploying a container group into a [virtual network](#).

## About GPU resources

### Count and SKU

To use GPUs in a container instance, specify a *GPU resource* with the following information:

- **Count** - The number of GPUs: 1, 2, or 4.
- **SKU** - The GPU SKU: **V100**. Each SKU maps to the NVIDIA Tesla GPU in one of the following Azure GPU-enabled VM families:

[+] Expand table

| SKU  | VM family |
|------|-----------|
| V100 | NCv3      |

### Maximum resources per SKU

[+] Expand table

| OS    | GPU SKU | GPU count | Max CPU | Max Memory (GB) | Storage (GB) |
|-------|---------|-----------|---------|-----------------|--------------|
| Linux | V100    | 1         | 6       | 112             | 50           |
| Linux | V100    | 2         | 12      | 224             | 50           |
| Linux | V100    | 4         | 24      | 448             | 50           |

When deploying GPU resources, set CPU and memory resources appropriate for the workload, up to the maximum values shown in the preceding table. These values are currently larger than the CPU and memory resources available in container groups without GPU resources.

### ⓘ Important

Default [subscription limits](#) (quotas) for GPU resources differ by SKU. The default CPU limits for V100 SKUs are initially set to 0. To request an increase in an available region, please submit an [Azure support request](#).

## Things to know

- **Deployment time** - Creation of a container group containing GPU resources takes up to **8-10 minutes**. This is due to the additional time to provision and configure a GPU VM in Azure.
- **Pricing** - Similar to container groups without GPU resources, Azure bills for resources consumed over the *duration* of a container group with GPU resources. The duration is calculated from the time to pull your first container's image until the container group terminates. It doesn't include the time to deploy the container group.

See [pricing details](#).

- **CUDA drivers** - Container instances with GPU resources are pre-provisioned with NVIDIA CUDA drivers and container runtimes, so you can use container images developed for CUDA workloads.

We support up through CUDA 11 at this stage. For example, you can use the following base images for your Dockerfile:

- [nvidia/cuda:11.4.2-base-ubuntu20.04](#)
- [tensorflow/tensorflow:devel-gpu](#)

 **Note**

To improve reliability when using a public container image from Docker Hub, import and manage the image in a private Azure container registry, and update your Dockerfile to use your privately managed base image. [Learn more about working with public images](#).

## YAML example

One way to add GPU resources is to deploy a container group by using a [YAML file](#). Copy the following YAML into a new file named *gpu-deploy-aci.yaml*, then save the file. This YAML creates a container group named *gpucontainergroup* specifying a container instance with a V100 GPU. The instance runs a sample CUDA vector addition application. The resource requests are sufficient to run the workload.

 **Note**

The following example uses a public container image. To improve reliability, import and manage the image in a private Azure container registry, and update your YAML to use your privately managed base image. [Learn more about working with public images](#).

### YAML

```
additional_properties: {}
apiVersion: '2021-09-01'
name: gpucontainergroup
properties:
 containers:
 - name: gpucontainer
 properties:
 image: k8s-gcrio.azureedge.net/cuda-vector-add:v0.1
 resources:
 requests:
 cpu: 1.0
 memoryInGB: 1.5
 gpu:
 count: 1
 sku: V100
 osType: Linux
 restartPolicy: OnFailure
```

Deploy the container group with the `az container create` command, specifying the YAML file name for the `--file` parameter. You need to supply the name of a resource group and a location for the container group such as *eastus* that supports GPU resources.

Azure CLI

```
az container create --resource-group myResourceGroup --file gpu-deploy-aci.yaml --location eastus
```

The deployment takes several minutes to complete. Then, the container starts and runs a CUDA vector addition operation. Run the `az container logs` command to view the log output:

Azure CLI

```
az container logs --resource-group myResourceGroup --name gpuscontainergroup --container-name gpuscontainer
```

Output:

Output

```
[Vector addition of 50000 elements]
Copy input data from the host memory to the CUDA device
CUDA kernel launch with 196 blocks of 256 threads
Copy output data from the CUDA device to the host memory
Test PASSED
Done
```

## Resource Manager template example

Another way to deploy a container group with GPU resources is by using a [Resource Manager template](#). Start by creating a file named `gpudeploy.json`, then copy the following JSON into it. This example deploys a container instance with a V100 GPU that runs a [TensorFlow](#) training job against the MNIST dataset. The resource requests are sufficient to run the workload.

JSON

```
{
 "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",
 "contentVersion": "1.0.0.0",
 "parameters": {
 "containerGroupName": {
```

```

 "type": "string",
 "defaultValue": "gpucontainergroup",
 "metadata": {
 "description": "Container Group name."
 }
},
"variables": {
 "containername": "gpucontainer",
 "containerimage": "mcr.microsoft.com/azuredocs/samples-tf-mnist-
demo:gpu"
},
"resources": [
{
 "name": "[parameters('containerGroupName')]",
 "type": "Microsoft.ContainerInstance/containerGroups",
 "apiVersion": "2021-09-01",
 "location": "[resourceGroup().location]",
 "properties": {
 "containers": [
{
 "name": "[variables('containername')]",
 "properties": {
 "image": "[variables('containerimage')]",
 "resources": {
 "requests": {
 "cpu": 4.0,
 "memoryInGb": 12.0,
 "gpu": {
 "count": 1,
 "sku": "V100"
 }
 }
 }
 }
 }
],
 "osType": "Linux",
 "restartPolicy": "OnFailure"
}
]
}
}

```

Deploy the template with the [az deployment group create](#) command. You need to supply the name of a resource group that was created in a region such as *eastus* that supports GPU resources.

#### Azure CLI

```
az deployment group create --resource-group myResourceGroup --template-file
gpudeploy.json
```

The deployment takes several minutes to complete. Then, the container starts and runs the TensorFlow job. Run the [az container logs](#) command to view the log output:

Azure CLI

```
az container logs --resource-group myResourceGroup --name
gpucontainergroup --container-name gpucontainer
```

Output:

Output

```
2018-10-25 18:31:10.155010: I
tensorflow/core/platform/cpu_feature_guard.cc:137] Your CPU supports
instructions that this TensorFlow binary was not compiled to use: SSE4.1
SSE4.2 AVX AVX2 FMA
2018-10-25 18:31:10.305937: I
tensorflow/core/common_runtime/gpu/gpu_device.cc:1030] Found device 0 with
properties:
name: Tesla V100 major: 3 minor: 7 memoryClockRate(GHz): 0.8235
pciBusID: ccb6:00:00.0
totalMemory: 11.92GiB freeMemory: 11.85GiB
2018-10-25 18:31:10.305981: I
tensorflow/core/common_runtime/gpu/gpu_device.cc:1120] Creating TensorFlow
device (/device:GPU:0) -> (device: 0, name: Tesla V100, pci bus id:
ccb6:00:00.0, compute capability: 3.7)
2018-10-25 18:31:14.941723: I tensorflow/stream_executor/dso_loader.cc:139]
successfully opened CUDA library libcupti.so.8.0 locally
Successfully downloaded train-images-idx3-ubyte.gz 9912422 bytes.
Extracting /tmp/tensorflow/input_data/train-images-idx3-ubyte.gz
Successfully downloaded train-labels-idx1-ubyte.gz 28881 bytes.
Extracting /tmp/tensorflow/input_data/train-labels-idx1-ubyte.gz
Successfully downloaded t10k-images-idx3-ubyte.gz 1648877 bytes.
Extracting /tmp/tensorflow/input_data/t10k-images-idx3-ubyte.gz
Successfully downloaded t10k-labels-idx1-ubyte.gz 4542 bytes.
Extracting /tmp/tensorflow/input_data/t10k-labels-idx1-ubyte.gz
Accuracy at step 0: 0.097
Accuracy at step 10: 0.6993
Accuracy at step 20: 0.8208
Accuracy at step 30: 0.8594
...
Accuracy at step 990: 0.969
Adding run metadata for 999
```

## Clean up resources

Because using GPU resources may be expensive, ensure that your containers don't run unexpectedly for long periods. Monitor your containers in the Azure portal, or check the status of a container group with the [az container show](#) command. For example:

Azure CLI

```
az container show --resource-group myResourceGroup --name gpucontainergroup
--output table
```

When you're done working with the container instances you created, delete them with the following commands:

Azure CLI

```
az container delete --resource-group myResourceGroup --name
gpucontainergroup -y
az container delete --resource-group myResourceGroup --name
gpucontainergroupm -y
```

## Next steps

- Learn more about deploying a container group using a [YAML file](#) or [Resource Manager template](#).
- Learn more about [GPU optimized VM sizes](#) in Azure.

---

## Feedback

Was this page helpful?

 Yes

 No

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

# Enable a TLS endpoint in a sidecar container

Article • 08/29/2024

This article shows how to create a [container group](#) with an application container and a sidecar container running a TLS/SSL provider. By setting up a container group with a separate TLS endpoint, you enable TLS connections for your application without changing your application code.

You set up an example container group consisting of two containers:

- An application container that runs a simple web app using the public Microsoft aci-helloworld image.
- A sidecar container running the public [Nginx](#) image, configured to use TLS.

In this example, the container group only exposes port 443 for Nginx with its public IP address. Nginx routes HTTPS requests to the companion web app, which listens internally on port 80. You can adapt the example for container applications that listen on other ports.

See [Next steps](#) for other approaches to enabling TLS in a container group.

## Prerequisites

- Use the Bash environment in [Azure Cloud Shell](#). For more information, see [Quickstart for Bash in Azure Cloud Shell](#).  
A blue rectangular button with a white 'A' icon on the left, the text 'Launch Cloud Shell' in the center, and a small blue square with a white right-pointing arrow icon on the right.
- If you prefer to run CLI reference commands locally, [install](#) the Azure CLI. If you're running on Windows or macOS, consider running Azure CLI in a Docker container. For more information, see [How to run the Azure CLI in a Docker container](#).
  - If you're using a local installation, sign in to the Azure CLI by using the `az login` command. To finish the authentication process, follow the steps displayed in your terminal. For other sign-in options, see [Sign in with the Azure CLI](#).
  - When you're prompted, install the Azure CLI extension on first use. For more information about extensions, see [Use extensions with the Azure CLI](#).
  - Run `az version` to find the version and dependent libraries that are installed. To upgrade to the latest version, run `az upgrade`.

- This article requires version 2.0.55 or later of the Azure CLI. If using Azure Cloud Shell, the latest version is already installed.

## Create a self-signed certificate

To set up Nginx as a TLS provider, you need a TLS/SSL certificate. This article shows how to create and set up a self-signed TLS/SSL certificate. For production scenarios, you should obtain a certificate from a certificate authority.

To create a self-signed TLS/SSL certificate, use the [OpenSSL](#) tool available in Azure Cloud Shell and many Linux distributions, or use a comparable client tool in your operating system.

First create a certificate request (.csr file) in a local working directory:

Console

```
openssl req -new -newkey rsa:2048 -nodes -keyout ssl.key -out ssl.csr
```

Follow the prompts to add the identification information. For Common Name, enter the hostname associated with the certificate. When prompted for a password, press Enter without typing, to skip adding a password.

Run the following command to create the self-signed certificate (.crt file) from the certificate request. For example:

Console

```
openssl x509 -req -days 365 -in ssl.csr -signkey ssl.key -out ssl.crt
```

You should now see three files in the directory: the certificate request (`ssl.csr`), the private key (`ssl.key`), and the self-signed certificate (`ssl.crt`). You use `ssl.key` and `ssl.crt` in later steps.

## Configure Nginx to use TLS

### Create Nginx configuration file

In this section, you create a configuration file for Nginx to use TLS. Start by copying the following text into a new file named `nginx.conf`. In Azure Cloud Shell, you can use Visual Studio Code to create the file in your working directory:

Console

code nginx.conf

In `location`, be sure to set `proxy_pass` with the correct port for your app. In this example, we set port 80 for the `aci-helloworld` container.

Console

```
nginx Configuration File
https://wiki.nginx.org/Configuration

Run as a less privileged user for security reasons.
user nginx;

worker_processes auto;

events {
 worker_connections 1024;
}

pid /var/run/nginx.pid;

http {

 #Redirect to https, using 307 instead of 301 to preserve post data

 server {
 listen [::]:443 ssl;
 listen 443 ssl;

 server_name localhost;

 # Protect against the BEAST attack by not using SSLv3 at all. If you
 need to support older browsers (IE6) you may need to add
 # SSLv3 to the list of protocols below.
 ssl_protocols TLSv1.2;

 # Ciphers set to best allow protection from Beast, while providing
 forwarding secrecy, as defined by Mozilla -
 https://wiki.mozilla.org/Security/Server_Side_TLS#Nginx
 ssl_ciphers ECDHE-RSA-AES128-GCM-SHA256:ECDHE-ECDSA-
 AES128-GCM-SHA256:ECDHE-RSA-AES256-GCM-SHA384:ECDHE-ECDSA-AES256-GCM-
 SHA384:DHE-RSA-AES128-GCM-SHA256:DHE-DSS-AES128-GCM-
 SHA256:kEDH+AESGCM:ECDHE-RSA-AES128-SHA256:ECDHE-ECDSA-AES128-SHA256:ECDHE-
 RSA-AES128-SHA:ECDHE-ECDSA-AES128-SHA:ECDHE-RSA-AES256-SHA384:ECDHE-ECDSA-
 AES256-SHA384:ECDHE-RSA-AES256-SHA:ECDHE-ECDSA-AES256-SHA:DHE-RSA-AES128-
 SHA256:DHE-RSA-AES128-SHA:DHE-DSS-AES128-SHA256:DHE-RSA-AES256-SHA256:DHE-
 DSS-AES256-SHA:DHE-RSA-AES256-SHA:AES128-GCM-SHA256:AES256-GCM-SHA384:ECDHE-
 RSA-RC4-SHA:ECDHE-ECDSA-RC4-SHA:AES128:AES256:RC4-
 SHA:HIGH:!aNULL:!eNULL:!EXPORT:!DES:!3DES:!MD5:!PSK;
 ssl_prefer_server_ciphers on;
```

```

 # Optimize TLS/SSL by caching session parameters for 10 minutes.
This cuts down on the number of expensive TLS/SSL handshakes.

 # The handshake is the most CPU-intensive operation, and by default
it is re-negotiated on every new/parallel connection.

 # By enabling a cache (of type "shared between all Nginx workers"),
we tell the client to re-use the already negotiated state.

 # Further optimization can be achieved by raising keepalive_timeout,
but that shouldn't be done unless you serve primarily HTTPS.

 ssl_session_cache shared:SSL:10m; # a 1mb cache can hold about
4000 sessions, so we can hold 40000 sessions
 ssl_session_timeout 24h;

Use a higher keepalive timeout to reduce the need for repeated
handshakes
 keepalive_timeout 300; # up from 75 secs default

remember the certificate for a year and automatically connect to
HTTPS
 add_header Strict-Transport-Security 'max-age=31536000;
includeSubDomains';

 ssl_certificate /etc/nginx/ssl.crt;
 ssl_certificate_key /etc/nginx/ssl.key;

 location / {
 proxy_pass http://localhost:80; # TODO: replace port if app
listens on port other than 80

 proxy_set_header Connection "";
 proxy_set_header Host $host;
 proxy_set_header X-Real-IP $remote_addr;
 proxy_set_header X-Forwarded-For $remote_addr;
 }
 }
}

```

## Base64-encode secrets and configuration file

Base64-encode the Nginx configuration file, the TLS/SSL certificate, and the TLS key. In the next section, you enter the encoded contents in a YAML file used to deploy the container group.

Console

```

cat nginx.conf | base64 > base64-nginx.conf
cat ssl.crt | base64 > base64-ssl.crt
cat ssl.key | base64 > base64-ssl.key

```

# Deploy container group

Now deploy the container group by specifying the container configurations in a [YAML file](#).

## Create YAML file

Copy the following YAML into a new file named `deploy-aci.yaml`. In Azure Cloud Shell, you can use Visual Studio Code to create the file in your working directory:

```
Console
code deploy-aci.yaml
```

Enter the contents of the base64-encoded files where indicated under `secret`. For example, `cat` each of the base64-encoded files to see its contents. During deployment, these files are added to a [secret volume](#) in the container group. In this example, the secret volume is mounted to the Nginx container.

```
YAML
api-version: 2019-12-01
location: westus
name: app-with-ssl
properties:
 containers:
 - name: nginx-with-ssl
 properties:
 image: mcr.microsoft.com/oss/nginx/nginx:1.15.5-alpine
 ports:
 - port: 443
 protocol: TCP
 resources:
 requests:
 cpu: 1.0
 memoryInGB: 1.5
 volumeMounts:
 - name: nginx-config
 mountPath: /etc/nginx
 - name: my-app
 properties:
 image: mcr.microsoft.com/azuredocs/aci-helloworld
 ports:
 - port: 80
 protocol: TCP
 resources:
 requests:
 cpu: 1.0
```

```
 memoryInGB: 1.5
 volumes:
 - secret:
 ssl.crt: <Enter contents of base64-ssl.crt here>
 ssl.key: <Enter contents of base64-ssl.key here>
 nginx.conf: <Enter contents of base64-nginx.conf here>
 name: nginx-config
 ipAddress:
 ports:
 - port: 443
 protocol: TCP
 type: Public
 osType: Linux
 tags: null
 type: Microsoft.ContainerInstance/containerGroups
```

## Deploy the container group

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

Azure CLI

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

Deploy the container group with the [az container create](#) command, passing the YAML file as an argument.

Azure CLI

```
az container create --resource-group <myResourceGroup> --file deploy-aci.yaml
```

## View deployment state

To view the state of the deployment, use the following [az container show](#) command:

Azure CLI

```
az container show --resource-group <myResourceGroup> --name app-with-ssl --output table
```

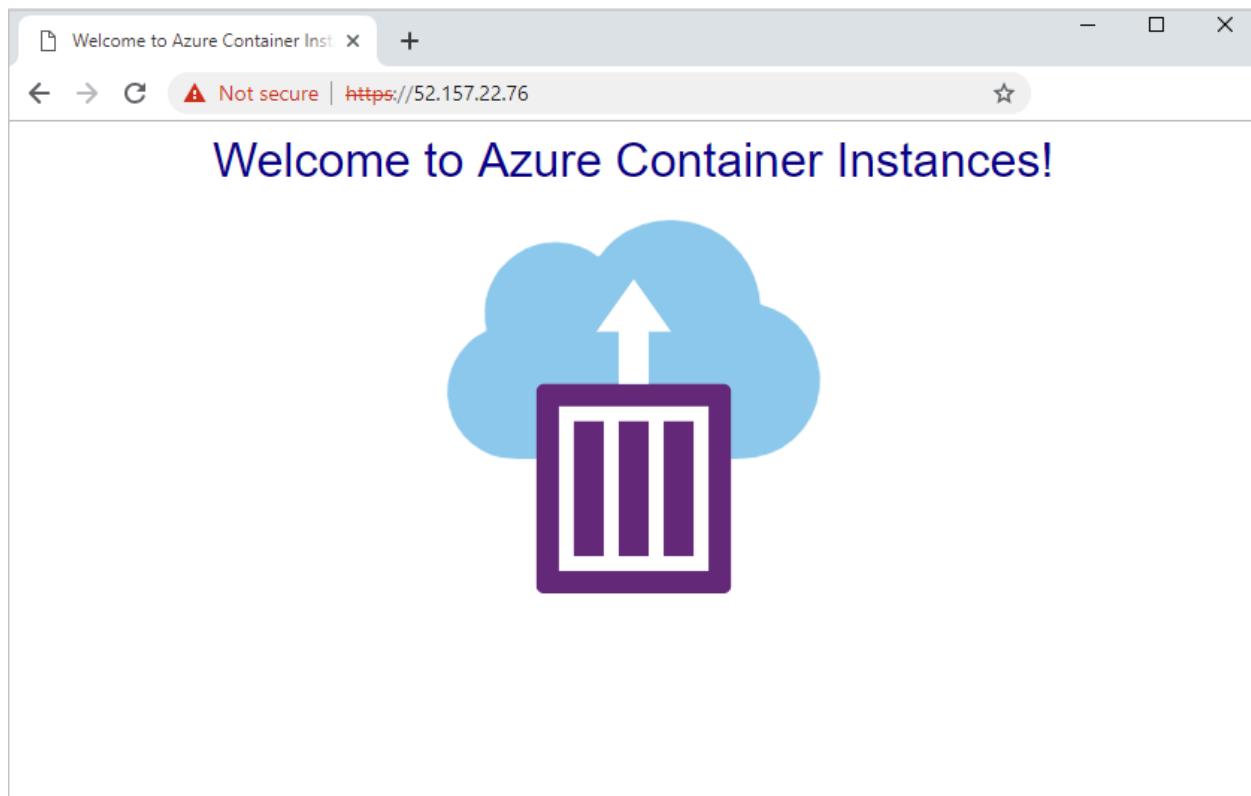
For a successful deployment, output is similar to the following:

Console

| Name         | ResourceGroup   | Status      | Image                                             | OsType | Location         |
|--------------|-----------------|-------------|---------------------------------------------------|--------|------------------|
| IP:ports     | Network         | CPU/Memory  |                                                   |        |                  |
| app-with-ssl | myresourcegroup | Running     | nginx, mcr.microsoft.com/azuredocs/aci-helloworld |        | 52.157.22.76:443 |
| Public       | 1.0             | core/1.5 gb | Linux                                             |        | westus           |

## Verify TLS connection

Use your browser to navigate to the public IP address of the container group. The IP address shown in this example is 52.157.22.76, so the URL is <https://52.157.22.76>. You must use HTTPS to see the running application, because of the Nginx server configuration. Attempts to connect over HTTP fail.



### ⓘ Note

Because this example uses a self-signed certificate and not one from a certificate authority, the browser displays a security warning when connecting to the site over HTTPS. You might need to accept the warning or adjust browser or certificate settings to proceed to the page. This behavior is expected.

# Next steps

This article showed you how to set up an Nginx container to enable TLS connections to a web app running in the container group. You can adapt this example for apps that listen on ports other than port 80. You can also update the Nginx configuration file to automatically redirect server connections on port 80 (HTTP) to use HTTPS.

While this article uses Nginx in the sidecar, you can use another TLS provider such as [Caddy](#).

If you deploy your container group in an [Azure virtual network](#), you can consider other options to enable a TLS endpoint for a backend container instance, including:

- [Azure Functions Proxies](#)
- [Azure API Management](#)
- [Azure Application Gateway](#) - see a sample [deployment template](#).

---

## Feedback

Was this page helpful?

 Yes

 No

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

# Enable automatic HTTPS with Caddy in a sidecar container

Article • 03/04/2025

This article describes how Caddy can be used as a sidecar container in a [container group](#) and act as a reverse proxy to provide an automatically managed HTTPS endpoint for your application.

Caddy is a powerful, enterprise-ready, open source web server with automatic HTTPS written in Go and represents an alternative to Nginx.

The automatization of certificates is possible because Caddy supports the ACMEv2 API ([RFC 8555](#)) that interacts with [Let's Encrypt](#) to issue certificates.

In this example, only the Caddy container gets exposed on ports 80/TCP and 443/TCP. The application behind the reverse proxy remains private. The network communication between Caddy and your application happens via localhost.

## ⓘ Note

This stands in contrast to the intra container group communication known from docker compose, where containers can be referenced by name.

The example mounts the [Caddyfile](#), which is required to configure the reverse proxy, from a file share hosted on an Azure Storage account.

## ⓘ Note

For production deployments, most users want to bake the Caddyfile into a custom docker image based on [caddy](#). This way, there is no need to mount files into the container.

## Prerequisites

- Use the Bash environment in [Azure Cloud Shell](#). For more information, see [Quickstart for Bash in Azure Cloud Shell](#).

 [Launch Cloud Shell](#)



- If you prefer to run CLI reference commands locally, [install](#) the Azure CLI. If you're running on Windows or macOS, consider running Azure CLI in a Docker container. For more information, see [How to run the Azure CLI in a Docker container](#).
  - If you're using a local installation, sign in to the Azure CLI by using the `az login` command. To finish the authentication process, follow the steps displayed in your terminal. For other sign-in options, see [Sign in with the Azure CLI](#).
  - When you're prompted, install the Azure CLI extension on first use. For more information about extensions, see [Use extensions with the Azure CLI](#).
  - Run `az version` to find the version and dependent libraries that are installed. To upgrade to the latest version, run `az upgrade`.
- This article requires version 2.0.55 or later of the Azure CLI. If using Azure Cloud Shell, the latest version is already installed.

## Prepare the Caddyfile

Create a file called `Caddyfile` and paste the following configuration. This configuration creates a reverse proxy configuration, pointing to your application container listening on 5000/TCP.

Console

```
my-app.westeurope.azurecontainer.io {
 reverse_proxy http://localhost:5000
}
```

It's important to note, that the configuration references a domain name instead of an IP address. Caddy needs to be reachable by this URL to carry out the challenge step required by the ACME protocol and to successfully retrieve a certificate from Let's Encrypt.

### ⓘ Note

For production deployment, users might want to use a domain name they control, for example, `api.company.com` and create a CNAME record pointing to `my-app.westeurope.azurecontainer.io`. If so, it needs to be ensured, that the custom domain name is also used in the Caddyfile, instead of the one assigned by Azure (for example, `*.westeurope.azurecontainer.io`). Further, the custom domain name,

needs to be referenced in the ACI YAML configuration described later in this example.

## Prepare storage account

Create a storage account

Azure CLI

```
az storage account create \
--name <storage-account> \
--resource-group <resource-group> \
--location westeurope
```

Create the file shares required to store the container state and caddy configuration.

Azure CLI

```
az storage share create \
--name proxy-caddyfile \
--account-name <storage-account>

az storage share create \
--name proxy-config \
--account-name <storage-account>

az storage share create \
--name proxy-data \
--account-name <storage-account>
```

Retrieve the storage account keys and make a note for later use

Azure CLI

```
az storage account keys list -g <resource-group> -n <storage-account>
```

## Deploy container group

### Create YAML file

Create a file called `ci-my-app.yaml` and paste the following content. Ensure to replace `<account-key>` with one of the access keys previously received and `<storage-account>` accordingly.

This YAML file defines two containers `reverse-proxy` and `my-app`. The `reverse-proxy` container mounts the three previously created file shares. The configuration also exposes port 80/TCP and 443/TCP of the `reverse-proxy` container. The communication between both containers happens on localhost only.

### ⓘ Note

It's important to note, that the `dnsNameLabel` key, defines the public DNS name, under which the container instance group will be reachable, it needs to match the FQDN defined in the `Caddyfile`

```
yml
```

```
name: ci-my-app
apiVersion: "2021-10-01"
location: westeurope
properties:
 containers:
 - name: reverse-proxy
 properties:
 image: caddy:2.6
 ports:
 - protocol: TCP
 port: 80
 - protocol: TCP
 port: 443
 resources:
 requests:
 memoryInGB: 1.0
 cpu: 1.0
 limits:
 memoryInGB: 1.0
 cpu: 1.0
 volumeMounts:
 - name: proxy-caddyfile
 mountPath: /etc/caddy
 - name: proxy-data
 mountPath: /data
 - name: proxy-config
 mountPath: /config
 - name: my-app
 properties:
 image: mcr.microsoft.com/azuredocs/aci-helloworld
 ports:
 - port: 5000
 protocol: TCP
 environmentVariables:
 - name: PORT
 value: 5000
 resources:
```

```
requests:
 memoryInGB: 1.0
 cpu: 1.0
limits:
 memoryInGB: 1.0
 cpu: 1.0
ipAddress:
 ports:
 - protocol: TCP
 port: 80
 - protocol: TCP
 port: 443
type: Public
dnsNameLabel: my-app
osType: Linux
volumes:
 - name: proxy-caddyfile
 azureFile:
 shareName: proxy-caddyfile
 storageAccountName: "<storage-account>"
 storageAccountKey: "<account-key>"
 - name: proxy-data
 azureFile:
 shareName: proxy-data
 storageAccountName: "<storage-account>"
 storageAccountKey: "<account-key>"
 - name: proxy-config
 azureFile:
 shareName: proxy-config
 storageAccountName: "<storage-account>"
 storageAccountKey: "<account-key>"
```

## Deploy the container group

Create a resource group with the `az group create` command:

Azure CLI

```
az group create --name <resource-group> --location westeurope
```

Deploy the container group with the `az container create` command, passing the YAML file as an argument.

Azure CLI

```
az container create --resource-group <resource-group> --file ci-my-app.yaml
```

## View the deployment state

To view the state of the deployment, use the following [az container show](#) command:

Azure CLI

```
az container show --resource-group <resource-group> --name ci-my-app --output table
```

## Verify TLS connection

Before verifying if everything went well, give the container group some time to fully start and for Caddy to request a certificate.

## OpenSSL

We can use the `s_client` subcommand of OpenSSL for that purpose.

Bash

```
echo "Q" | openssl s_client -connect my-app.westeurope.azurecontainer.io:443
```

Console

```
CONNECTED(00000188)

Certificate chain
 0 s:CN = my-app.westeurope.azurecontainer.io
 i:C = US, O = Let's Encrypt, CN = R3
 1 s:C = US, O = Let's Encrypt, CN = R3
 i:C = US, O = Internet Security Research Group, CN = ISRG Root X1
 2 s:C = US, O = Internet Security Research Group, CN = ISRG Root X1
 i:O = Digital Signature Trust Co., CN = DST Root CA X3

Server certificate
-----BEGIN CERTIFICATE-----
MIIEgTCCA2mgAwIBAgISAxixidSnpH4vVuCZk9UNG/pd2MA0GCSqGSIb3DQEBCwUA
MDIxCzAJBgNVBAYTA1VTMRYwFAYDVQQKEw1MZXQncyBFbmNyeXB0MQswCQYDVQQD
EwJSMzAeFw0yMzA0MDYxODAzMzNaFw0yMzA3MDUxODAzMzJaMC4xLDAqBgNVBAMT
I215LWFwcC53ZXN0ZXVyb3B1LmF6dXJ1Y29udGFpbmVyLmlvMFkwEwYHKoZIzj0C
AQYIKoZIzj0DAQcDQgAEaaN/wGyFcimM+104WzbFg06vI1XxXqp9vgmLZHpfRnwV
a08JbaB7hE+M5EAg34LDY80RyHgY+Ff4vTh2Z96rVqOCA14wggJaMA4GA1UdDwEB
/wQEAWiHgDAdBgNVHSUEfjAUBggrBgeFBQcDAQYIKwYBBQUHAWIwDAYDVR0TAQH/
BAIwADAdBgNVHQ4EFgQUoL5DP+4PWiyE79hL5o+v8uyHdAwHwYDVR0jBBgwFoAU
FC6zF7dYVsuumA1A5h+vnYsUwsYwVQYIKwYBBQUHAQEESTBHMCEGCCsGAQUFBzAB
hhVodHRwOi8vcjMuby5sZW5jci5vcmcwIgYIKwYBBQUHMAKGFmh0dHA6Ly9yMy5p
Lmx1bmNyLm9yZy8wLgYDVR0RBCCwJYIjbXktYXBwLndlc3RldXJvcGUuYXp1cmVj
b250YWluZXIuaW8wTAYDVR0gBEUwQzAIBgZngQwBAgEwNwYLKwYBBAGC3xMBAQEW
KDAmBggxBgEFBQcCARYaaHR0cDovL2Nwcy5sZXrZw5jcn1wdC5vcmcwggeeBgor
BgEEAdZ5AgQCBIH1BIHyAPAAdgC3PvsK35xNunXy0cW6WPRsXfxCz3qfNcSeHQmB
```

```
Je20mQAAAYdX8+CQAAAЕAwBHMEUCIQC9Ztqd3DXoJhOИHW+P7ketGrK1VA6nPZ1
9Ci0rn6t8gIgXHcrbBqItemndRMv+UJ3DaBfTkY0qECecOJCgLhSYNUAdgDoPtDa
PvUGNTLnVyi8iWvJA9PL0RFr70tp4Xd9bQa9bgAAAYdX8+CAAAAЕAwBHMEUCIBJ1
24z44vKFUOLCi1a7ymVuWErkмLb/GtysvcxILaj0AiEAr49hyKfen4BbSTwC8Fg4
/LgZnn2F3uHI+9p+ZM09xTAwDQYJKoZIhvcNAQELBQADggEBACqxa21eiW3JrZwk
FHgpd6SxhUeeсrYXxFNva1Y6G//q2qCmGeKK3GK+ZGPqDtcoASH5t5ghV4dIT4WU
auVDLFVywXzR8PT6QUu3W8QxU+W7406twBf23qMIgrF8PIWhStI5mn1uCpeqlnf5
HpRaj2f5/5n19pcCZcrRx94G9qhPYdMzuy4mZRhxXRqrpIsabqX3DC21d8dszCvD
pkV61iuARgm3MIQz1yL/x5Bn4nywjnhYZA4KFktC0Ti55cPRh1mkzGQAsYQDdWrq
dVav+U9d0LQ4Sq4suaDmzDzApr+hpQSJhwgRN16+tLMyZ6INAU2JWKDxiyDTd0uH
jz456og=
-----END CERTIFICATE-----
subject=CN = my-app.westeurope.azurecontainer.io

issuer=C = US, O = Let's Encrypt, CN = R3

No client certificate CA names sent
Peer signing digest: SHA256
Peer signature type: ECDSA
Server Temp Key: X25519, 253 bits

SSL handshake has read 4208 bytes and written 401 bytes
Verification error: unable to get local issuer certificate

New, TLSv1.3, Cipher is TLS_AES_128_GCM_SHA256
Server public key is 256 bit
Secure Renegotiation IS NOT supported
Compression: NONE
Expansion: NONE
No ALPN negotiated
Early data was not sent
Verify return code: 20 (unable to get local issuer certificate)

Post-Handshake New Session Ticket arrived:
SSL-Session:
 Protocol : TLSv1.3
 Cipher : TLS_AES_128_GCM_SHA256
 Session-ID:
85F1A4290F99A0DD28C8CB21EF4269E7016CC5D23485080999A8548057729B24
 Session-ID-ctx:
 Resumption PSK:
752D438C19A5DBDBF10781F863D5E5D9A8859230968A9EAFF7BBA86937D004F
 PSK identity: None
 PSK identity hint: None
 SRP username: None
 TLS session ticket lifetime hint: 604800 (seconds)
 TLS session ticket:
 0000 - 2f 25 98 90 9d 46 9b 01-03 78 db bd 4d 64 b3 a6
 /%...F...x..Md..
 0010 - 52 c0 7a 8a b6 3d b8 4b-c0 d7 fc 04 e8 63 d4 bb
R.z..=.K.....c..
 0020 - 15 b3 25 b7 be 64 3d 30-2b d7 dc 7a 1a d1 22 63
 ..%..d=0+..z.."c
```

```
0030 - 42 30 90 65 6b b5 e1 83-a3 6c 76 c8 f6 ae e9 31
B0.ek....lv....1
0040 - 45 91 33 57 8e 9f 4b 6a-2e 2c 9b f9 87 5f 71 1d
E.3W..Kj.,....q.
0050 - 5a 84 59 50 17 31 1f 62-2b 0e 1e e5 70 03 d9 e9
Z.YP.1.b+....p...
0060 - 50 1c 5d 1f a4 3c 8a 0e-f4 c5 7d ce 9e 5c 98 de P.]..
<....}..\..
0070 - e5 .
```

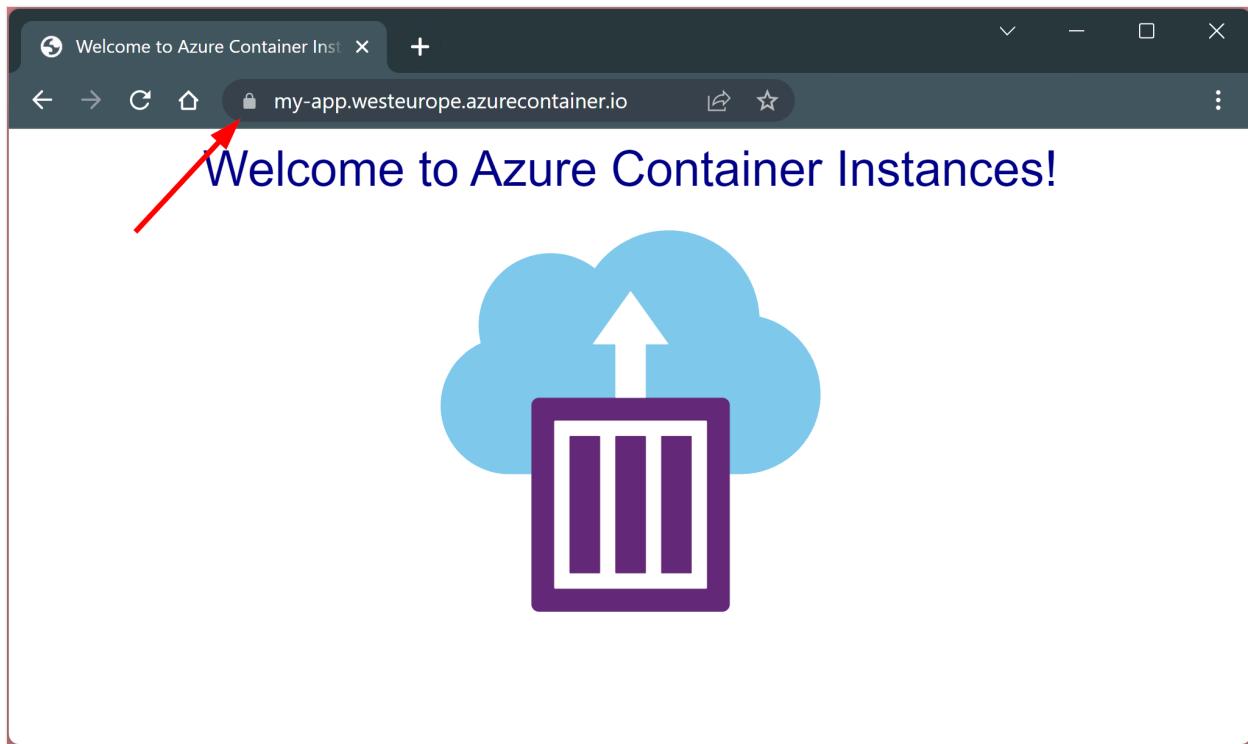
Start Time: 1680808973  
Timeout : 7200 (sec)  
Verify return code: 20 (unable to get local issuer certificate)  
Extended master secret: no  
Max Early Data: 0

---

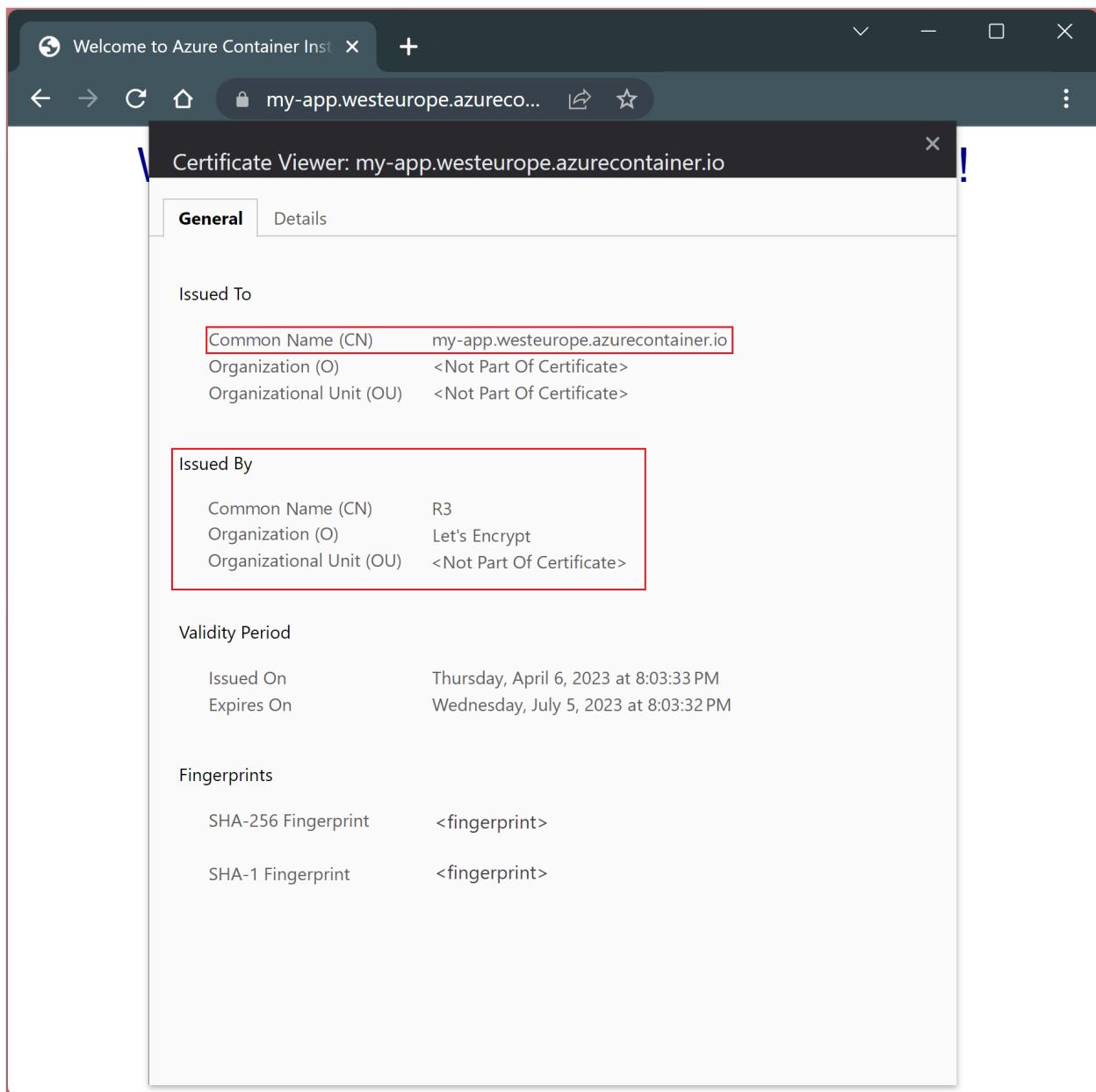
read R BLOCK

## Chrome browser

Navigate to <https://my-app.westeurope.azurecontainer.io> and verify the certificate by clicking on the padlock next to the URL.



To see the certificate details, select on "Connection is secure" followed by "certificate is valid".



## Next steps

- [Caddy documentation ↗](#)
- [GitHub aci-helloworld ↗](#)
- [YAML reference: Azure Container Instances](#)
- [Secure your codeless REST API with automatic HTTPS using Data API builder and Caddy ↗](#)

## Feedback

Was this page helpful?

Yes

No

# Expose a static IP address for a container group

Article • 08/29/2024

This article shows one way to expose a static, public IP address for a [container group](#) by using an Azure [application gateway](#). Follow these steps when you need a static entry point for an external-facing containerized app that runs in Azure Container Instances.

In this article, you use the Azure CLI to create the resources for this scenario:

- An Azure virtual network
- A container group deployed [in the virtual network](#) that hosts a small web app
- An application gateway with a public frontend IP address, a listener to host a website on the gateway, and a route to the backend container group

As long as the application gateway runs and the container group exposes a stable private IP address in the network's delegated subnet, the container group is accessible at this public IP address.

## ⓘ Note

Azure Application Gateway [supports HTTP, HTTPS, HTTP/2, and WebSocket protocols](#).

Azure charges for an application gateway based on the amount of time that the gateway is provisioned and available, as well as the amount of data it processes. See [pricing](#).

## Create virtual network

In a typical case, you might already have an Azure virtual network. If you don't have one, create one as shown with the following example commands. The virtual network needs separate subnets for the application gateway and the container group.

If you need one, create an Azure resource group. For example:

Azure CLI

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

Create a virtual network with the [az network vnet create](#) command. This command creates the *myAGSubnet* subnet in the network.

Azure CLI

```
az network vnet create \
--name myVNet \
--resource-group myResourceGroup \
--location eastus \
--address-prefix 10.0.0.0/16 \
--subnet-name myAGSubnet \
--subnet-prefix 10.0.1.0/24
```

Use the [az network vnet subnet create](#) command to create a subnet for the backend container group. Here, its name is *myACISubnet*.

Azure CLI

```
az network vnet subnet create \
--name myACISubnet \
--resource-group myResourceGroup \
--vnet-name myVNet \
--address-prefix 10.0.2.0/24
```

Use the [az network public-ip create](#) command to create a static public IP resource. In a later step, this address is configured as the front end of the application gateway.

Azure CLI

```
az network public-ip create \
--resource-group myResourceGroup \
--name myAGPublicIPAddress \
--allocation-method Static \
--sku Standard
```

## Create container group

Run the following [az container create](#) to create a container group in the virtual network you configured in the previous step.

The group is deployed in the *myACISubnet* subnet and contains a single instance named *appcontainer* that pulls the `aci-helloworld` image. As shown in other articles in the documentation, this image packages a small web app written in Node.js that serves a static HTML page.

#### Azure CLI

```
az container create \
--name appcontainer \
--resource-group myResourceGroup \
--image mcr.microsoft.com/azuredocs/aci-helloworld \
--vnet myVNet \
--subnet myACISubnet
```

When successfully deployed, the container group is assigned a private IP address in the virtual network. For example, run the following `az container show` command to retrieve the group's IP address:

#### Azure CLI

```
az container show \
--name appcontainer --resource-group myResourceGroup \
--query ipAddress.ip --output tsv
```

Output is similar to: `10.0.2.4`.

For use in a later step, save the IP address in an environment variable:

#### Azure CLI

```
ACI_IP=$(az container show \
--name appcontainer \
--resource-group myResourceGroup \
--query ipAddress.ip --output tsv)
```

#### ⓘ Important

If the container group is stopped, started, or restarted, the container group's private IP is subject to change. If this happens, you will need to update the application gateway configuration.

## Create application gateway

Create an application gateway in the virtual network, following the steps in the [application gateway quickstart](#). The following `az network application-gateway create` command creates a gateway with a public frontend IP address and a route to the backend container group. See the [Application Gateway documentation](#) for details about the gateway settings.

## Azure CLI

```
az network application-gateway create \
--name myAppGateway \
--location eastus \
--resource-group myResourceGroup \
--capacity 2 \
--sku Standard_v2 \
--http-settings-protocol http \
--public-ip-address myAGPublicIPAddress \
--vnet-name myVNet \
--subnet myAGSubnet \
--servers "$ACI_IP" \
--priority 100
```

It can take up to 15 minutes for Azure to create the application gateway.

## Test public IP address

Now you can test access to the web app running in the container group behind the application gateway.

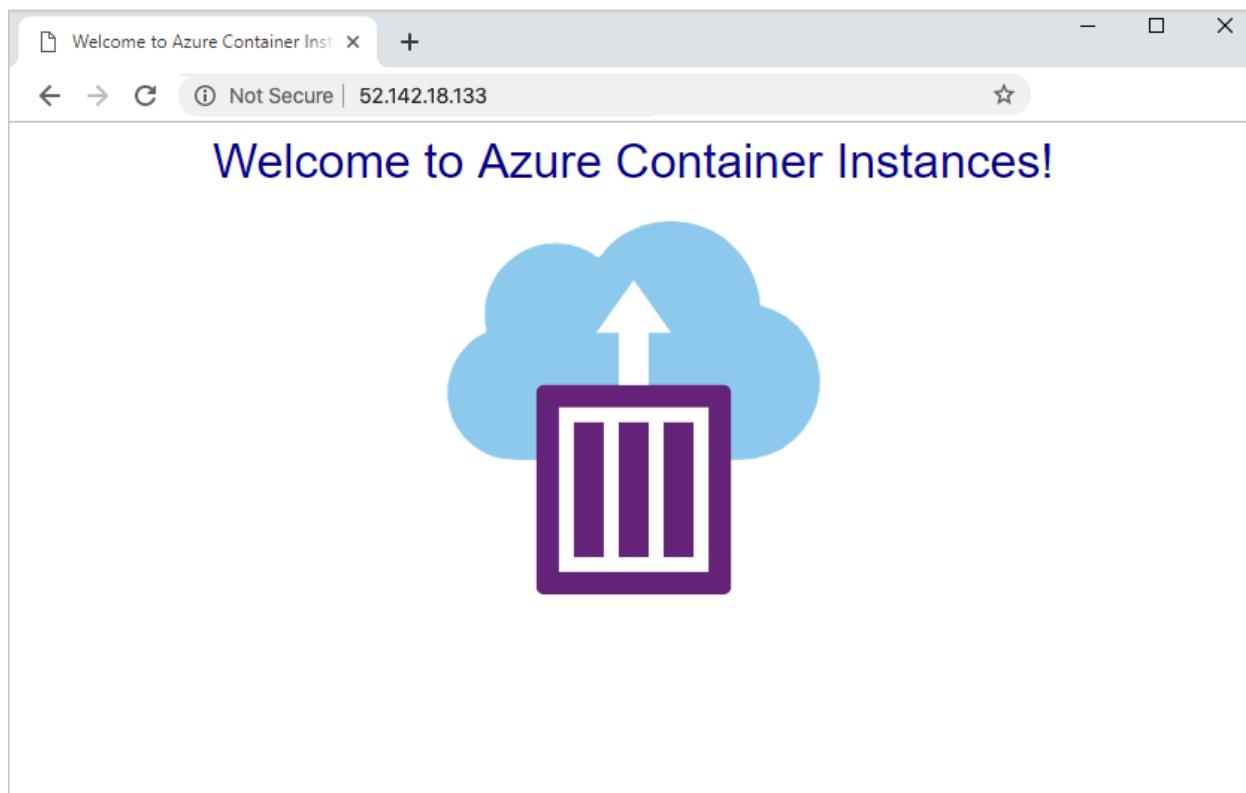
Run the [az network public-ip show](#) command to retrieve the frontend public IP address of the gateway:

## Azure CLI

```
az network public-ip show \
--resource-group myresourcegroup \
--name myAGPublicIPAddress \
--query [ipAddress] \
--output tsv
```

Output is a public IP address, similar to: `52.142.18.133`.

To view the running web app when successfully configured, navigate to the gateway's public IP address in your browser. Successful access is similar to:



## Next steps

- See a [quickstart template](#) ↗ to create a container group with a WordPress container instance as a backend server behind an application gateway.
- You can also configure an application gateway with a certificate for SSL termination. See the [overview](#) and the [tutorial](#).
- Depending on your scenario, consider using other Azure load-balancing solutions with Azure Container Instances. For example, use [Azure Traffic Manager](#) to distribute traffic across multiple container instances and across multiple regions. See this [blog post](#) ↗.

---

## Feedback

Was this page helpful?

 Yes

 No

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

# Configure a single public IP address for outbound and inbound traffic to a container group

Article • 08/29/2024

Setting up a [container group](#) with an external-facing IP address allows external clients to use the IP address to access a container in the group. For example, a browser can access a web app running in a container. However, currently a container group uses a different IP address for outbound traffic. This egress IP address isn't exposed programmatically, which makes container group monitoring and configuration of client firewall rules more complex.

This article provides steps to configure a container group in a [virtual network](#) integrated with [Azure Firewall](#). By setting up a user-defined route to the container group and firewall rules, you can route and identify traffic to and from the container group. Container group ingress and egress use the public IP address of the firewall. Multiple container groups deployed in the virtual network's subnet can use a single egress IP address.

In this article, you use the Azure CLI to create the resources for this scenario:

- Container groups deployed on a delegated subnet [in the virtual network](#)
- An Azure firewall deployed in the network with a static public IP address
- A user-defined route on the container groups' subnet
- A NAT rule for firewall ingress and an application rule for egress

You then validate ingress and egress from example container groups through the firewall.

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

## Prerequisites

- Use the Bash environment in [Azure Cloud Shell](#). For more information, see [Quickstart for Bash in Azure Cloud Shell](#).



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

## Launch Azure Cloud Shell

The Azure Cloud Shell is a free interactive shell that you can use to run the steps in this article. It has common Azure tools preinstalled and configured to use with your account.

To open the Cloud Shell, just select **Try it** from the upper right corner of a code block. You can also launch Cloud Shell in a separate browser tab by going to <https://shell.azure.com>.

When Cloud Shell opens, verify that **Bash** is selected for your environment. Subsequent sessions will use Azure CLI in a Bash environment. Select **Copy** to copy the blocks of code, paste it into the Cloud Shell, and press **Enter** to run it.

## Sign in to Azure

Cloud Shell is automatically authenticated under the initial account signed-in with. Use the following script to sign in using a different subscription, replacing `<Subscription ID>` with your Azure Subscription ID. If you don't have an [Azure subscription](#), create an [Azure free account](#) before you begin.

```
Azure CLI

subscription=<subscriptionId> # add subscription here

az account set -s $subscription # ...or use 'az login'
```

For more information, see [set active subscription](#) or [log in interactively](#)

### ⓘ Note

To download the complete script, go to [full script](#).

## Get started

This tutorial makes use of a randomized variable. If you used an existing resource group, modify the value of this variable appropriately.

Azure CLI

```
resourceGroup=resourceGroup$RANDOM
```

**Azure resource group:** If you don't have an Azure resource group already, create a resource group with the [az group create](#) command. Modify the location value as appropriate.

Azure CLI

```
az group create --name $resourceGroup --location eastus
```

## Deploy ACI in a virtual network

In a typical case, you might already have an Azure virtual network in which to deploy a container group. For demonstration purposes, the following commands create a virtual network and subnet when the container group is created. The subnet is delegated to Azure Container Instances.

The container group runs a small web app from the `aci-helloworld` image. As shown in other articles in the documentation, this image packages a small web app written in Node.js that serves a static HTML page.

Create the container group with the [az container create](#) command:

Azure CLI

```
az container create \
--name appcontainer \
--resource-group $resourceGroup \
--image mcr.microsoft.com/azuredocs/aci-helloworld \
--vnet aci-vnet \
--vnet-address-prefix 10.0.0.0/16 \
--subnet aci-subnet \
--subnet-address-prefix 10.0.0.0/24
```

## 💡 Tip

Adjust the value of `--subnet address-prefix` for the IP address space you need in your subnet. The smallest supported subnet is /29, which provides eight IP addresses. Some IP addresses are reserved for use by Azure.

For use in a later step, get the private IP address of the container group by running the [az container show][az-container-show] command:

### Azure CLI

```
aciPrivateIp=$(az container show --name appcontainer \
--resource-group $resourceGroup \
--query ipAddress.ip --output tsv)"
```

## Deploy Azure Firewall in network

In the following sections, use the Azure CLI to deploy an Azure firewall in the virtual network. For background, see [Tutorial: Deploy and configure Azure Firewall using the Azure portal](#).

First, use the `az network vnet subnet create` to add a subnet named AzureFirewallSubnet for the firewall. AzureFirewallSubnet is the *required* name of this subnet.

### Azure CLI

```
az network vnet subnet create \
--name AzureFirewallSubnet \
--resource-group $resourceGroup \
--vnet-name aci-vnet \
--address-prefix 10.0.1.0/26
```

Use the following [Azure CLI commands](#) to create a firewall in the subnet.

If not already installed, add the firewall extension to the Azure CLI using the `az extension add` command:

### Azure CLI

```
az extension add --name azure-firewall
```

Create the firewall resources using the `az network firewall create` command:

#### Azure CLI

```
az network firewall create \
--name myFirewall \
--resource-group $resourceGroup \
--location eastus

az network public-ip create \
--name fw-pip \
--resource-group $resourceGroup \
--location eastus \
--allocation-method static \
--sku standard

az network firewall ip-config create \
--firewall-name myFirewall \
--name FW-config \
--public-ip-address fw-pip \
--resource-group $resourceGroup \
--vnet-name aci-vnet
```

Update the firewall configuration using the [az network firewall update](#) command:

#### Azure CLI

```
az network firewall update \
--name myFirewall \
--resource-group $resourceGroup
```

Get the firewall's private IP address using the [az network firewall ip-config list](#) command. This private IP address is used in a later command.

#### Azure CLI

```
fwPrivateIp=$(az network firewall ip-config list \
--resource-group $resourceGroup \
--firewall-name myFirewall \
--query "[].privateIpAddress" --output tsv)"
```

Get the firewall's public IP address using the [az network public-ip show](#) command. This public IP address is used in a later command.

#### Azure CLI

```
fwPublicIp=$(az network public-ip show \
--name fw-pip \
--resource-group $resourceGroup \
--query ipAddress --output tsv)"
```

# Define user-defined route on ACI subnet

To divert traffic to the Azure firewall, define a user-defined route on the ACI subnet. For more information, see [Route network traffic](#).

## Create route table

First, run the following [az network route-table create](#) command to create the route table. Create the route table in the same region as the virtual network.

Azure CLI

```
az network route-table create \
--name Firewall-rt-table \
--resource-group $resourceGroup \
--location eastus \
--disable-bgp-route-propagation true
```

## Create route

Run [az network-route-table route create](#) to create a route in the route table. To route traffic to the firewall, set the next hop type to `VirtualAppliance`, and pass the firewall's private IP address as the next hop address.

Azure CLI

```
az network route-table route create \
--resource-group $resourceGroup \
--name DG-Route \
--route-table-name Firewall-rt-table \
--address-prefix 0.0.0.0/0 \
--next-hop-type VirtualAppliance \
--next-hop-ip-address $fwPrivateIp
```

## Associate route table to ACI subnet

Run the [az network vnet subnet update](#) command to associate the route table with the subnet delegated to Azure Container Instances.

Azure CLI

```
az network vnet subnet update \
--name aci-subnet \
--resource-group $resourceGroup \
```

```
--vnet-name aci-vnet \
--address-prefixes 10.0.0.0/24 \
--route-table Firewall-rt-table
```

## Configure rules on firewall

By default, Azure Firewall denies (blocks) inbound and outbound traffic.

### Configure NAT rule on firewall to ACI subnet

Create a [NAT rule](#) on the firewall to translate and filter inbound internet traffic to the application container you started previously in the network. For details, see [Filter inbound Internet traffic with Azure Firewall DNAT](#)

Create a NAT rule and collection by using the [az network firewall nat-rule create](#) command:

Azure CLI

```
az network firewall nat-rule create \
--firewall-name myFirewall \
--collection-name myNATCollection \
--action dnat \
--name myRule \
--protocols TCP \
--source-addresses '*' \
--destination-addresses $fwPublicIp \
--destination-ports 80 \
--resource-group $resourceGroup \
--translated-address $aciPrivateIp \
--translated-port 80 \
--priority 200
```

Add NAT rules as needed to filter traffic to other IP addresses in the subnet. For example, other container groups in the subnet could expose IP addresses for inbound traffic, or other internal IP addresses could be assigned to the container group after a restart.

### Create outbound application rule on the firewall

Run the following [az network firewall application-rule create](#) command to create an outbound rule on the firewall. This sample rule allows access from the subnet delegated to Azure Container Instances to the FQDN `checkip.dyndns.org`. HTTP access to the site is used in a later step to confirm the egress IP address from Azure Container Instances.

## Azure CLI

```
az network firewall application-rule create \
--collection-name myAppCollection \
--firewall-name myFirewall \
--name Allow-CheckIP \
--protocols Http=80Https=443 \
--resource-group $resourceGroup \
--target-fqdns checkip.dyndns.org \
--source-addresses 10.0.0.0/24 \
--priority 200 \
--action Allow
```

# Test container group access through the firewall

The following sections verify that the subnet delegated to Azure Container Instances is properly configured behind the Azure firewall. The previous steps routed both incoming traffic to the subnet and outgoing traffic from the subnet through the firewall.

## Test ingress to a container group

Test inbound access to the `appcontainer` running in the virtual network by browsing to the firewall's public IP address. Previously, you stored the public IP address in variable `$FW_PUBLIC_IP`:

## Azure CLI

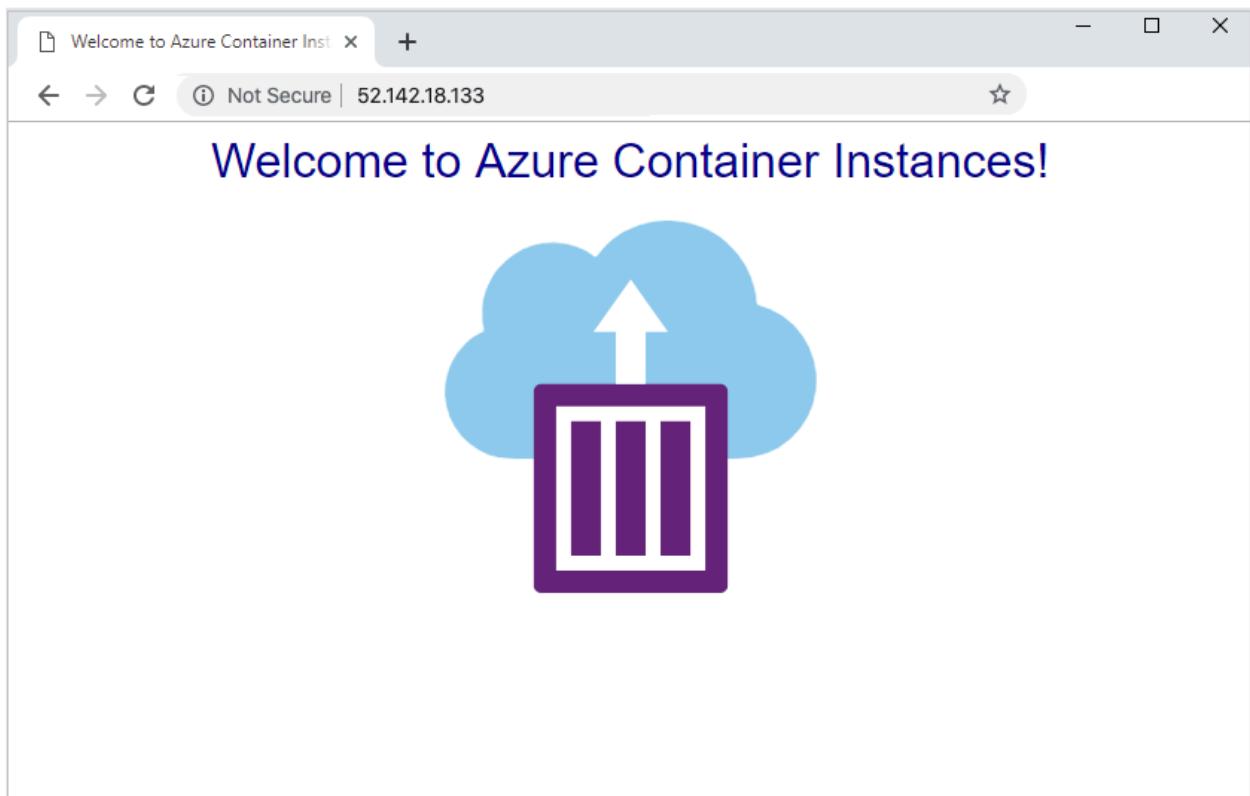
```
echo $fwPublicIp
```

Output is similar to:

## Console

```
52.142.18.133
```

If the NAT rule on the firewall is configured properly, you see the following when you enter the firewall's public IP address in your browser:



## Test egress from a container group

Deploy the following sample container into the virtual network. When it runs, it sends a single HTTP request to <http://checkip.dyndns.org>, which displays the IP address of the sender (the egress IP address). If the application rule on the firewall is configured properly, the firewall's public IP address is returned.

```
Azure CLI

az container create \
--resource-group $resourceGroup \
--name testegress \
--image mcr.microsoft.com/azuredocs/aci-tutorial-sidecar \
--command-line "curl -s http://checkip.dyndns.org" \
--restart-policy OnFailure \
--vnet aci-vnet \
--subnet aci-subnet
```

View the container logs to confirm the IP address is the same as the public IP address of the firewall.

```
Azure CLI

az container logs \
--resource-group $resourceGroup \
--name testegress
```

Output is similar to:

Console

```
<html><head><title>Current IP Check</title></head><body>Current IP Address:
52.142.18.133</body></html>
```

## Clean up resources

When no longer needed, you can use [az group delete](#) to remove the resource group and all related resources as follows. The `--no-wait` parameter returns control to the prompt without waiting for the operation to complete. The `--yes` parameter confirms that you wish to delete the resources without another prompt to do so.

Azure CLI

```
az group delete --name $resourceGroup --yes --no-wait
```

## Next steps

In this article, you set up container groups in a virtual network behind an Azure firewall. You configured a user-defined route and NAT and application rules on the firewall. By using this configuration, you set up a single, static IP address for ingress and egress from Azure Container Instances.

For more information about managing traffic and protecting Azure resources, see the [Azure Firewall](#) documentation.

---

## Feedback

Was this page helpful?

 Yes

 No

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

# Configure a NAT gateway for static IP address for outbound traffic from a container group

Article • 08/29/2024

Setting up a [container group](#) with an external-facing IP address allows external clients to use the IP address to access a container in the group. For example, a browser can access a web app running in a container. However, currently a container group uses a different IP address for outbound traffic. This egress IP address isn't exposed programmatically, which makes container group monitoring and configuration of client firewall rules more complex.

This article provides steps to configure a container group in a [virtual network](#) integrated with a [Network Address Translation \(NAT\) gateway](#). By configuring a NAT gateway to SNAT a subnet address range delegated to Azure Container Instances (ACI), you can identify outbound traffic from your container groups. The container group egress traffic uses the public IP address of the NAT gateway. Multiple container groups deployed in the virtual network's subnet can use a single NAT gateway.

In this article, you use the Azure CLI to create the resources for this scenario:

- Container groups deployed on a delegated subnet [in the virtual network](#)
- A NAT gateway deployed in the network with a static public IP address

You then validate egress from example container groups through the NAT gateway.

## ⓘ Note

The ACI service recommends integrating with a NAT gateway for containerized workloads that have static egress but not static ingress requirements. For ACI architecture that supports both static ingress and egress, please see the following tutorial: [Use Azure Firewall for ingress and egress](#).

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

## Prerequisites

- Use the Bash environment in [Azure Cloud Shell](#). For more information, see [Quickstart for Bash in Azure Cloud Shell](#).

 [Launch Cloud Shell](#) 

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

## Launch Azure Cloud Shell

The Azure Cloud Shell is a free interactive shell that you can use to run the steps in this article. It has common Azure tools preinstalled and configured to use with your account.

To open the Cloud Shell, just select **Try it** from the upper right corner of a code block. You can also launch Cloud Shell in a separate browser tab by going to <https://shell.azure.com>.

When Cloud Shell opens, verify that **Bash** is selected for your environment. Subsequent sessions will use Azure CLI in a Bash environment. Select **Copy** to copy the blocks of code, paste it into the Cloud Shell, and press **Enter** to run it.

## Sign in to Azure

Cloud Shell is automatically authenticated under the initial account signed-in with. Use the following script to sign in using a different subscription, replacing <Subscription ID> with your Azure Subscription ID. If you don't have an [Azure subscription](#), create an [Azure free account](#) before you begin.

Azure CLI

```
subscription=<subscriptionId> # add subscription here
```

```
az account set -s $subscription # ...or use 'az login'
```

For more information, see [set active subscription](#) or [log in interactively](#)

 **Note**

To download the complete script, go to [full script ↗](#).

## Get started

This tutorial makes use of a randomized variable. If you used an existing resource group, modify the value of this variable appropriately.

Azure CLI

```
resourceGroup=resourceGroup$RANDOM
```

**Azure resource group:** If you don't have an Azure resource group already, create a resource group with the [az group create](#) command. Modify the location value as appropriate.

Azure CLI

```
az group create --name $resourceGroup --location eastus
```

## Deploy ACI in a virtual network

In a typical case, you might already have an Azure virtual network in which to deploy a container group. For demonstration purposes, the following commands create a virtual network and subnet when the container group is created. The subnet is delegated to Azure Container Instances.

The container group runs a small web app from the `aci-helloworld` image. As shown in other articles in the documentation, this image packages a small web app written in Node.js that serves a static HTML page.

Create the container group with the [az container create](#) command:

Azure CLI

```
az container create \
--name appcontainer \
--resource-group $resourceGroup \
--image mcr.microsoft.com/azuredocs/aci-helloworld \
--vnet aci-vnet \
--vnet-address-prefix 10.0.0.0/16 \
--subnet aci-subnet \
--subnet-address-prefix 10.0.0.0/24
```

### ⓘ Note

Adjust the value of `--subnet address-prefix` for the IP address space you need in your subnet. The smallest supported subnet is /29, which provides eight IP addresses. Some IP addresses are reserved for use by Azure, which you can read more about [here](#).

## Create a public IP address

In the following sections, use the Azure CLI to deploy an Azure NAT gateway in the virtual network. For background, see [Quickstart: Create a NAT gateway using Azure CLI](#).

First, use the `az network vnet public-ip create` to create a public IP address for the NAT gateway. The gateway uses this public IP to access the Internet. You receive a warning about an upcoming breaking change where Standard SKU IP addresses are availability zone aware by default. You can learn more about the use of availability zones and public IP addresses [here](#).

### Azure CLI

```
az network public-ip create \
--name myPublicIP \
--resource-group $resourceGroup \
--sku standard \
--zone 1 \
--allocation static
```

Store the public IP address in a variable for use during the validation step later in this script.

### Azure CLI

```
ngPublicIp=$(az network public-ip show \
--name myPublicIP \
```

```
--resource-group $resourceGroup \
--query ipAddress --output tsv)"
```

## Deploy a NAT gateway into a virtual network

Use the following [az network nat gateway create](#) to create a NAT gateway that uses the public IP you created in the previous step.

Azure CLI

```
az network nat gateway create \
--resource-group $resourceGroup \
--name myNATgateway \
--public-ip-addresses myPublicIP \
--idle-timeout 10
```

## Configure NAT service for source subnet

We configure the source subnet **aci-subnet** to use a specific NAT gateway resource **myNATgateway** with [az network vnet subnet update](#). This command activates the NAT service on the specified subnet.

Azure CLI

```
az network vnet subnet update \
--resource-group $resourceGroup \
--vnet-name aci-vnet \
--name aci-subnet \
--nat-gateway myNATgateway
```

## Test egress from a container group

Test inbound access to the **appcontainer** running in the virtual network by browsing to the firewall's public IP address. Previously, you stored the public IP address in variable **\$NG\_PUBLIC\_IP**

Deploy the following sample container into the virtual network. When it runs, it sends a single HTTP request to <http://checkip.dyndns.org>, which displays the IP address of the sender (the egress IP address). If the application rule on the firewall is configured properly, the firewall's public IP address is returned.

Azure CLI

```
az container create \
--resource-group $resourceGroup \
--name testegress \
--image mcr.microsoft.com/azuredocs/aci-tutorial-sidecar \
--command-line "curl -s http://checkip.dyndns.org" \
--restart-policy OnFailure \
--vnet aci-vnet \
--subnet aci-subnet
```

View the container logs to confirm the IP address is the same as the public IP address we created in the first step of the tutorial.

Azure CLI

```
az container logs \
--resource-group $resourceGroup \
--name testegress
```

Output is similar to:

Console

```
<html><head><title>Current IP Check</title></head><body>Current IP Address:
52.142.18.133</body></html>
```

This IP address should match the public IP address created in the first step of the tutorial.

Azure CLI

```
echo $ngPublicIp
```

## Clean up resources

When no longer needed, you can use `az group delete` to remove the resource group and all related resources as follows. The `--no-wait` parameter returns control to the prompt without waiting for the operation to complete. The `--yes` parameter confirms that you wish to delete the resources without another prompt to do so.

Azure CLI

```
az group delete --name $resourceGroup --yes --no-wait
```

# Next steps

In this article, you set up container groups in a virtual network behind an Azure NAT gateway. By using this configuration, you set up a single, static IP address egress from Azure Container Instances container groups.

For troubleshooting assistance, see the [Troubleshoot Azure Virtual Network NAT connectivity](#).

---

## Feedback

Was this page helpful?

 Yes

 No

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

# Deploy a container group with custom DNS settings

Article • 10/21/2024

In [Azure Virtual Network](#), you can deploy container groups using the `az container create` command in the Azure CLI. You can also provide advanced configuration settings to the `az container create` command using a YAML configuration file.

This article demonstrates how to deploy a container group with custom DNS settings using a YAML configuration file.

For more information on deploying container groups to a virtual network, see the [Deploy in a virtual network article](#).

## ⓘ Important

Previously, the process of deploying container groups on virtual networks used [network profiles](#) for configuration. However, network profiles have been retired as of the `2021-07-01` API version. We recommend you use the latest API version, which relies on [subnet IDs](#) instead.

## Prerequisites

- An **active Azure subscription**. If you don't have an active Azure subscription, create a [free account](#) before you begin.
- **Azure CLI**. The command-line examples in this article use the [Azure CLI](#) and are formatted for the Bash shell. You can [install the Azure CLI](#) locally or use the [Azure Cloud Shell](#).
- A **resource group** to manage all the resources you use in this how-to guide. We use the example resource group name **ACIResourceGroup** throughout this article.

Azure CLI

```
az group create --name ACIResourceGroup --location westus
```

## Limitations

The Azure DNS Resolver IP, 168.63.129.16, is automatically added to the /etc/resolv.conf file in container instances even when a custom DNS configuration is applied. The presence of the Azure DNS Resolver IP can cause incorrect DNS resolutions for distributions using musl-libc, such as Alpine Linux. This incorrect resolution is because [musl-libc sends DNS queries in parallel and caches the fastest response](#). To avoid this issue, we recommend you use distributions that handle DNS queries sequentially, like Ubuntu and RHEL, which use glibc instead of musl-libc.

For other networking scenarios and limitations, see [Virtual network scenarios and resources for Azure Container Instances](#).

### Important

Container group deployment to a virtual network is available for Linux containers in most regions where Azure Container Instances is available. For details, see [Regions and resource availability](#). Examples in this article are formatted for the Bash shell.

For PowerShell or command prompt, adjust the line continuation characters accordingly.

## Create your virtual network

You need a virtual network to deploy a container group with a custom DNS configuration. This virtual network requires a subnet with permissions to create Azure Container Instances resources and a linked private DNS zone to test name resolution.

This guide uses a virtual network named `aci-vnet`, a subnet named `aci-subnet`, and a private DNS zone named `private.contoso.com`. We use **Azure Private DNS Zones**, which you can learn about in the [Private DNS Overview](#).

If you have an existing virtual network that meets these criteria, you can skip to [Deploy your container group](#).

### Tip

You can modify the following commands with your own information as needed.

1. Create the virtual network using the `az network vnet create` command. Enter address prefixes in Classless Inter-Domain Routing (CIDR) format (for example: `10.0.0.0/16`).

```
az network vnet create \
--name aci-vnet \
--resource-group ACIResourceGroup \
--location westus \
--address-prefix 10.0.0.0/16
```

2. Create the subnet using the [az network vnet subnet create](#) command. The following command creates a subnet in your virtual network with a delegation that permits it to create container groups. For more information about working with subnets, see the [Add, change, or delete a virtual network subnet](#). For more information about subnet delegation, see the [Virtual Network Scenarios and Resources article section on delegated subnets](#).

#### Azure CLI

```
az network vnet subnet create \
--name aci-subnet \
--resource-group ACIResourceGroup \
--vnet-name aci-vnet \
--address-prefixes 10.0.0.0/24 \
--delegations Microsoft.ContainerInstance/containerGroups
```

3. Record the subnet ID key-value pair from the output of this command. You use this key-value pair in your YAML configuration file later. It takes the form `"id":`

```
"/subscriptions/<subscription-
ID>/resourceGroups/ACIResourceGroup/providers/Microsoft.Network/virtualNetwork
s/aci-vnet/subnets/aci-subnet".
```

4. Create the private DNS Zone using the [az network private-dns zone create](#) command.

#### Azure CLI

```
az network private-dns zone create -g ACIResourceGroup -n
private.contoso.com
```

5. Link the DNS zone to your virtual network using the [az network private-dns link vnet create](#) command. The DNS server is only required to test name resolution. The `-e` flag enables automatic hostname registration, which is unneeded, so we set it to `false`.

#### Azure CLI

```
az network private-dns link vnet create \
 -g ACIResourceGroup \
 -n aciDNSLink \
 -z private.contoso.com \
 -v aci-vnet \
 -e false
```

Once you complete the previous steps, you should see an output with a final key-value pair that reads `"virtualNetworkLinkState": "Completed"`.

## Deploy your container group

### ⓘ Note

Custom DNS settings are not currently available in the Azure portal for container group deployments. They must be provided with YAML file, Resource Manager template, [REST API](#), or an [Azure SDK](#).

Copy the following YAML into a new file named *custom-dns-deploy-aci.yaml*. Edit the following configurations with your values:

- `dnsConfig`: DNS settings for your containers within your container group.
  - `nameServers`: A list of name servers to be used for DNS lookups.
  - `searchDomains`: DNS suffixes to be appended for DNS lookups.
- `ipAddress`: The private IP address settings for the container group.
  - `ports`: The ports to open, if any.
  - `protocol`: The protocol (TCP or UDP) for the opened port.
- `subnetIDs`: Network settings for the subnet(s) in the virtual network.
  - `id`: The full Resource Manager resource ID of the subnet, which you obtained earlier.

### ⓘ Note

The DNS config fields aren't automatically queried at this time, so these fields must be explicitly filled out.

### YAML

```
apiVersion: '2021-07-01'
location: westus
```

```
name: pwsh-vnet-dns
properties:
 containers:
 - name: pwsh-vnet-dns
 properties:
 command:
 - /bin/bash
 - -c
 - echo hello; sleep 10000
 environmentVariables: []
 image: mcr.microsoft.com/powershell:latest
 ports:
 - port: 80
 resources:
 requests:
 cpu: 1.0
 memoryInGB: 2.0
 dnsConfig:
 nameServers:
 - 10.0.0.10 # DNS Server 1
 - 10.0.0.11 # DNS Server 2
 searchDomains: contoso.com # DNS search suffix
 ipAddress:
 type: Private
 ports:
 - port: 80
 subnetIds:
 - id: /subscriptions/<subscription-ID>/resourceGroups/ACIResourceGroup/providers/Microsoft.Network/virtualNetworks/aci-vnet/subnets/aci-subnet
 osType: Linux
 tags: null
 type: Microsoft.ContainerInstance/containerGroups
```

Deploy the container group with the `az container create` command, specifying the YAML file name with the `--file` parameter:

Azure CLI

```
az container create --resource-group ACIResourceGroup \
--file custom-dns-deploy-aci.yaml
```

Once the deployment is complete, run the `az container show` command to display its status. Sample output:

Azure CLI

```
az container show --resource-group ACIResourceGroup --name pwsh-vnet-dns -o
table
```

## Output

Name	ResourceGroup	Status	Image
IP:ports	Network	CPU/Memory	OsType
pwsh-vnet-dns	ACIResourceGroup	Running	mcr.microsoft.com/powershell
10.0.0.5:80	Private	1.0 core/2.0 gb	Linux

After the status shows `Running`, execute the `az container exec` command to obtain bash access within the container.

## Azure CLI

```
az container exec --resource-group ACIResourceGroup --name pwsh-vnet-dns --exec-command "/bin/bash"
```

Validate that DNS is working as expected from within your container. For example, read the `/etc/resolv.conf` file to ensure proper configuration of the DNS settings provided in the YAML file.

### ⓘ Note

Note that the Azure DNS resolver IP 168.63.129.16 is automatically added to the `/etc/resolv.conf` file in ACIs, even when a custom DNS configuration is applied. This can lead to resolution issues in distributions that handle DNS querying processes in parallel. For more information, see the [Limitations section](#).

## Bash

```
root@wk-caas-81d609b206c541589e11058a6d260b38-90b0aff460a737f346b3b0:/# cat /etc/resolv.conf

nameserver 10.0.0.10
nameserver 10.0.0.11
nameserver 168.63.129.16
search contoso.com
```

## Clean up resources

### Delete container instances

When you're finished with the container instance you created, delete it with the [az container delete](#) command:

```
Azure CLI
```

```
az container delete --resource-group ACIResourceGroup --name pwsh-vnet-dns -y
```

## Delete network resources

If you don't plan to use this virtual network again, you can delete it with the [az network vnet delete](#) command:

```
Azure CLI
```

```
az network vnet delete --resource-group ACIResourceGroup --name aci-vnet
```

## Delete resource group

If you don't plan to use this resource group outside of this guide, you can delete it with [az group delete](#) command:

```
Azure CLI
```

```
az group delete --name ACIResourceGroup
```

Enter `y` when prompted if you're sure you wish to perform the operation.

## Next steps

For more information on how to deploy a container group within a virtual network, see the Azure Quickstart Template [Create an Azure container group with virtual network](#).

## Feedback

Was this page helpful?

 Yes

 No

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

# Mount an Azure file share in Azure Container Instances

Article • 08/29/2024

By default, Azure Container Instances are stateless. If the container is restarted, crashes, or stops, all of its state is lost. To persist state beyond the lifetime of the container, you must mount a volume from an external store. As shown in this article, Azure Container Instances can mount an Azure file share created with [Azure Files](#). Azure Files offers fully managed file shares hosted in Azure Storage that are accessible via the industry standard Server Message Block (SMB) protocol. Using an Azure file share with Azure Container Instances provides file-sharing features similar to using an Azure file share with Azure virtual machines.

## Limitations

- Azure Storage doesn't support SMB mounting of file share using managed identity
- You can only mount Azure Files shares to Linux containers. Review more about the differences in feature support for Linux and Windows container groups in the [overview](#).
- Azure file share volume mount requires the Linux container run as *root*.
- Azure File share volume mounts are limited to CIFS support.

### Note

Mounting an Azure Files share to a container instance is similar to a Docker [bind mount](#)  . If you mount a share into a container directory in which files or directories exist, the mount obscures files or directories, making them inaccessible while the container runs.

### Important

If the outbound connection to the internet is blocked in the delegated subnet, you must add a [service endpoint](#) to Azure Storage on your delegated subnet.

## Create an Azure file share

Before using an Azure file share with Azure Container Instances, you must create it. Run the following script to create a storage account to host the file share, and the share itself. The storage account name must be globally unique, so the script adds a random value to the base string.

```
Azure CLI

Change these four parameters as needed
ACI_PERS_RESOURCE_GROUP=myResourceGroup
ACI_PERS_STORAGE_ACCOUNT_NAME=mystorageaccount$RANDOM
ACI_PERS_LOCATION=eastus
ACI_PERS_SHARE_NAME=acishare

Create the storage account with the parameters
az storage account create \
 --resource-group $ACI_PERS_RESOURCE_GROUP \
 --name $ACI_PERS_STORAGE_ACCOUNT_NAME \
 --location $ACI_PERS_LOCATION \
 --sku Standard_LRS

Create the file share
az storage share create \
 --name $ACI_PERS_SHARE_NAME \
 --account-name $ACI_PERS_STORAGE_ACCOUNT_NAME
```

## Get storage credentials

To mount an Azure file share as a volume in Azure Container Instances, you need three values: the storage account name, the share name, and the storage access key.

- **Storage account name** - If you used the preceding script, the storage account name was stored in the `$ACI_PERS_STORAGE_ACCOUNT_NAME` variable. To see the account name, type:

```
Console

echo $ACI_PERS_STORAGE_ACCOUNT_NAME
```

- **Share name** - This value is already known (defined as `acishare` in the preceding script)
- **Storage account key** - This value can be found using the following command:

```
Azure CLI
```

```
STORAGE_KEY=$(az storage account keys list --resource-group $ACI_PERS_RESOURCE_GROUP --account-name $ACI_PERS_STORAGE_ACCOUNT_NAME --query "[0].value" --output tsv)
echo $STORAGE_KEY
```

## Deploy container and mount volume - CLI

To mount an Azure file share as a volume in a container by using the Azure CLI, specify the share and volume mount point when you create the container with [az container create](#). If you followed the previous steps, you can mount the share you created earlier by using the following command to create a container:

Azure CLI

```
az container create \
--resource-group $ACI_PERS_RESOURCE_GROUP \
--name hellofiles \
--image mcr.microsoft.com/azuredocs/aci-hellofiles \
--dns-name-label aci-demo \
--ports 80 \
--azure-file-volume-account-name $ACI_PERS_STORAGE_ACCOUNT_NAME \
--azure-file-volume-account-key $STORAGE_KEY \
--azure-file-volume-share-name $ACI_PERS_SHARE_NAME \
--azure-file-volume-mount-path /aci/logs/
```

The `--dns-name-label` value must be unique within the Azure region where you create the container instance. Update the value in the preceding command if you receive a **DNS name label** error message when you execute the command.

## Manage files in mounted volume

Once the container starts up, you can use the web app deployed via the Microsoft aci-hellofiles image to create small text files in the Azure file share at the mount path you specified. Obtain the web app's fully qualified domain name (FQDN) with the [az container show](#) command:

Azure CLI

```
az container show --resource-group $ACI_PERS_RESOURCE_GROUP \
--name hellofiles --query ipAddress.fqdn --output tsv
```

After saving text using the app, you can use the [Azure portal](#) or a tool like the [Microsoft Azure Storage Explorer](#) to retrieve and inspect the file or files written to the

file share.

## Deploy container and mount volume - YAML

You can also deploy a container group and mount a volume in a container with the Azure CLI and a [YAML template](#). Deploying by YAML template is a preferred method when deploying container groups consisting of multiple containers.

The following YAML template defines a container group with one container created with the `aci-hellofiles` image. The container mounts the Azure file share `acishare` created previously as a volume. Where indicated, enter the name and storage key for the storage account that hosts the file share.

As in the CLI example, the `dnsNameLabel` value must be unique within the Azure region where you create the container instance. Update the value in the YAML file if needed.

YAML

```
apiVersion: '2019-12-01'
location: eastus
name: file-share-demo
properties:
 containers:
 - name: hellofiles
 properties:
 environmentVariables: []
 image: mcr.microsoft.com/azuredocs/aci-hellofiles
 ports:
 - port: 80
 resources:
 requests:
 cpu: 1.0
 memoryInGB: 1.5
 volumeMounts:
 - mountPath: /aci/logs/
 name: filesharevolume
 osType: Linux
 restartPolicy: Always
 ipAddress:
 type: Public
 ports:
 - port: 80
 dnsNameLabel: aci-demo
volumes:
 - name: filesharevolume
 azureFile:
 sharename: acishare
 storageAccountName: <Storage account name>
 storageAccountKey: <Storage account key>
```

```
tags: {}
type: Microsoft.ContainerInstance/containerGroups
```

To deploy with the YAML template, save the preceding YAML to a file named `deploy-aci.yaml`, then execute the `az container create` command with the `--file` parameter:

Azure CLI

```
Deploy with YAML template
az container create --resource-group myResourceGroup --file deploy-aci.yaml
```

## Deploy container and mount volume - Resource Manager

In addition to CLI and YAML deployment, you can deploy a container group and mount a volume in a container using an Azure [Resource Manager template](#).

First, populate the `volumes` array in the container group `properties` section of the template.

Then, for each container in which you'd like to mount the volume, populate the `volumeMounts` array in the `properties` section of the container definition.

The following Resource Manager template defines a container group with one container created with the `aci-hellofiles` image. The container mounts the Azure file share `acishare` created previously as a volume. Where indicated, enter the name and storage key for the storage account that hosts the file share.

As in the previous examples, the `dnsNameLabel` value must be unique within the Azure region where you create the container instance. Update the value in the template if needed.

JSON

```
{
 "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",
 "contentVersion": "1.0.0.0",
 "variables": {
 "container1name": "hellofiles",
 "container1image": "mcr.microsoft.com/azuredocs/aci-hellofiles"
 },
 "resources": [
 {
 "name": "file-share-demo",
```

```
"type": "Microsoft.ContainerInstance/containerGroups",
"apiVersion": "2019-12-01",
"location": "[resourceGroup().location]",
"properties": {
 "containers": [
 {
 "name": "[variables('container1name')]",
 "properties": {
 "image": "[variables('container1image')]",
 "resources": {
 "requests": {
 "cpu": 1,
 "memoryInGb": 1.5
 }
 },
 "ports": [
 {
 "port": 80
 }
],
 "volumeMounts": [
 {
 "name": "filesharevolume",
 "mountPath": "/aci/logs"
 }
]
 }
 }
],
 "osType": "Linux",
 "ipAddress": {
 "type": "Public",
 "ports": [
 {
 "protocol": "tcp",
 "port": "80"
 }
],
 "dnsNameLabel": "aci-demo"
 },
 "volumes": [
 {
 "name": "filesharevolume",
 "azureFile": {
 "shareName": "acishare",
 "storageAccountName": "<Storage account name>",
 "storageAccountKey": "<Storage account key>"
 }
 }
]
}
```

To deploy with the Resource Manager template, save the preceding JSON to a file named `deploy-aci.json`, then execute the `az deployment group create` command with the `--template-file` parameter:

Azure CLI

```
Deploy with Resource Manager template
az deployment group create --resource-group myResourceGroup --template-file
deploy-aci.json
```

## Mount multiple volumes

To mount multiple volumes in a container instance, you must deploy using an [Azure Resource Manager template](#), a YAML file, or another programmatic method. To use a template or YAML file, provide the share details and define the volumes by populating the `volumes` array in the `properties` section of the file.

For example, if you created two Azure Files shares named `share1` and `share2` in storage account `myStorageAccount`, the `volumes` array in a Resource Manager template would appear similar to the following example:

JSON

```
"volumes": [
 {
 "name": "myvolume1",
 "azureFile": {
 "shareName": "share1",
 "storageAccountName": "myStorageAccount",
 "storageAccountKey": "<storage-account-key>"
 }
 },
 {
 "name": "myvolume2",
 "azureFile": {
 "shareName": "share2",
 "storageAccountName": "myStorageAccount",
 "storageAccountKey": "<storage-account-key>"
 }
 }
]
```

Next, for each container in the container group in which you'd like to mount the volumes, populate the `volumeMounts` array in the `properties` section of the container definition. For example, this mounts the two volumes, `myvolume1` and `myvolume2`, previously defined:

## JSON

```
"volumeMounts": [{}
 "name": "myvolume1",
 "mountPath": "/mnt/share1/"
,
{
 "name": "myvolume2",
 "mountPath": "/mnt/share2/"
}]
```

## Next steps

Learn how to mount other volume types in Azure Container Instances:

- Mount an emptyDir volume in Azure Container Instances
- Mount a gitRepo volume in Azure Container Instances
- Mount a secret volume in Azure Container Instances

---

## Feedback

Was this page helpful?

 Yes

 No

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

# Mount a secret volume in Azure Container Instances

Article • 08/29/2024

Use a *secret* volume to supply sensitive information to the containers in a container group. The *secret* volume stores your secrets in files within the volume, accessible by the containers in the container group. By storing secrets in a *secret* volume, you can avoid adding sensitive data like SSH keys or database credentials to your application code.

- Once deployed with secrets in a container group, a secret volume is *read-only*.
- [Tmpfs](#), a RAM-backed filesystem, backs all secret volumes; their contents are never written to nonvolatile storage.

## ⓘ Note

*Secret* volumes are currently restricted to Linux containers. Learn how to pass secure environment variables for both Windows and Linux containers in [Set environment variables](#). While we're working to bring all features to Windows containers, you can find current platform differences in the [overview](#).

## Mount secret volume - Azure CLI

To deploy a container with one or more secrets by using the Azure CLI, include the `--secrets` and `--secrets-mount-path` parameters in the `az container create` command. This example mounts a *secret* volume consisting of two files containing secrets, "mysecret1" and "mysecret2," at `/mnt/secrets`:

Azure CLI

```
az container create \
 --resource-group myResourceGroup \
 --name secret-volume-demo \
 --image mcr.microsoft.com/azuredocs/aci-helloworld \
 --secrets mysecret1="My first secret FOO" mysecret2="My second secret
 BAR" \
 --secrets-mount-path /mnt/secrets
```

The following `az container exec` output shows opening a shell in the running container, listing the files within the secret volume, then displaying their contents:

## Azure CLI

```
az container exec \
--resource-group myResourceGroup \
--name secret-volume-demo --exec-command "/bin/sh"
```

## Output

```
/usr/src/app # ls /mnt/secrets
mysecret1
mysecret2
/usr/src/app # cat /mnt/secrets/mysecret1
My first secret FOO
/usr/src/app # cat /mnt/secrets/mysecret2
My second secret BAR
/usr/src/app # exit
Bye.
```

## Mount secret volume - YAML

You can also deploy container groups with the Azure CLI and a [YAML template](#).

Deploying by YAML template is the preferred method when deploying container groups consisting of multiple containers.

When you deploy with a YAML template, the secret values must be **Base64-encoded** in the template. However, the secret values appear in plaintext within the files in the container.

The following YAML template defines a container group with one container that mounts a *secret* volume at `/mnt/secrets`. The secret volume has two files containing secrets, "mysecret1" and "mysecret2."

## YAML

```
apiVersion: '2019-12-01'
location: eastus
name: secret-volume-demo
properties:
 containers:
 - name: aci-tutorial-app
 properties:
 environmentVariables: []
 image: mcr.microsoft.com/azuredocs/aci-helloworld:latest
 ports: []
 resources:
 requests:
 cpu: 1.0
```

```
 memoryInGB: 1.5
 volumeMounts:
 - mountPath: /mnt/secrets
 name: secretvolume1
 osType: Linux
 restartPolicy: Always
 volumes:
 - name: secretvolume1
 secret:
 mysecret1: TXkgZmlyc3Qgc2VjcmV0IEZPTwo=
 mysecret2: TXkgc2Vjb25kIHNlY3JldCBCQVIK
 tags: {}
 type: Microsoft.ContainerInstance/containerGroups
```

To deploy with the YAML template, save the preceding YAML to a file named `deploy-aci.yaml`, then execute the `az container create` command with the `--file` parameter:

Azure CLI

```
Deploy with YAML template
az container create \
--resource-group myResourceGroup \
--file deploy-aci.yaml
```

## Mount secret volume - Resource Manager

In addition to CLI and YAML deployment, you can deploy a container group using an Azure [Resource Manager template](#).

First, populate the `volumes` array in the container group `properties` section of the template. When you deploy with a Resource Manager template, the secret values must be **Base64-encoded** in the template. However, the secret values appear in plaintext within the files in the container.

Next, for each container in the container group in which you'd like to mount the `secret` volume, populate the `volumeMounts` array in the `properties` section of the container definition.

The following Resource Manager template defines a container group with one container that mounts a `secret` volume at `/mnt/secrets`. The secret volume has two secrets, "mysecret1" and "mysecret2."

JSON

```
{
 "$schema": "https://schema.management.azure.com/schemas/2019-04-
```

```
01/deploymentTemplate.json#",
 "contentVersion": "1.0.0.0",
 "variables": {
 "container1name": "aci-tutorial-app",
 "container1image": "microsoft/aci-helloworld:latest"
 },
 "resources": [
 {
 "type": "Microsoft.ContainerInstance/containerGroups",
 "apiVersion": "2021-03-01",
 "name": "secret-volume-demo",
 "location": "[resourceGroup().location]",
 "properties": {
 "containers": [
 {
 "name": "[variables('container1name')]",
 "properties": {
 "image": "[variables('container1image')]",
 "resources": {
 "requests": {
 "cpu": 1,
 "memoryInGb": 1.5
 }
 },
 "ports": [
 {
 "port": 80
 }
],
 "volumeMounts": [
 {
 "name": "secretvolume1",
 "mountPath": "/mnt/secrets"
 }
]
 }
 }
],
 "osType": "Linux",
 "ipAddress": {
 "type": "Public",
 "ports": [
 {
 "protocol": "tcp",
 "port": "80"
 }
]
 },
 "volumes": [
 {
 "name": "secretvolume1",
 "secret": {
 "mysecret1": "TXkgZmlyc3Qgc2VjcmV0IEZPTwo=",
 "mysecret2": "TXkgc2Vjb25kIHNlY3JldCBCQVIK"
 }
 }
]
 }
 }
]
}
```

```
 }
]
}
]
}
```

To deploy with the Resource Manager template, save the preceding JSON to a file named `deploy-aci.json`, then execute the `az deployment group create` command with the `--template-file` parameter:

Azure CLI

```
Deploy with Resource Manager template
az deployment group create \
--resource-group myResourceGroup \
--template-file deploy-aci.json
```

## Next steps

### Volumes

Learn how to mount other volume types in Azure Container Instances:

- Mount an Azure file share in Azure Container Instances
- Mount an emptyDir volume in Azure Container Instances
- Mount a gitRepo volume in Azure Container Instances

### Secure environment variables

Another method for providing sensitive information to containers (including Windows containers) is by using [secure environment variables](#).

---

## Feedback

Was this page helpful?

 Yes

 No

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

# Mount an emptyDir volume in Azure Container Instances

Article • 08/29/2024

Learn how to mount an *emptyDir* volume to share data between the containers in a container group in Azure Container Instances. Use *emptyDir* volumes as ephemeral caches for your containerized workloads.

## ⓘ Note

Mounting an *emptyDir* volume is currently restricted to Linux containers. While we are working to bring all features to Windows containers, you can find current platform differences in the [overview](#).

## emptyDir volume

The *emptyDir* volume provides a writable directory accessible to each container in a container group. Containers in the group can read and write the same files in the volume, and it can be mounted using the same or different paths in each container.

Some example uses for an *emptyDir* volume:

- Scratch space
- Checkpointing during long-running tasks
- Store data retrieved by a sidecar container and served by an application container

Data in an *emptyDir* volume is persisted through container crashes. Containers that are restarted, however, aren't guaranteed to persist the data in an *emptyDir* volume. If you stop a container group, the *emptyDir* volume isn't persisted.

The maximum size of a Linux *emptyDir* volume is 50 GB.

## Mount an emptyDir volume

To mount an *emptyDir* volume in a container instance, you can deploy using an [Azure Resource Manager template](#), a [YAML file](#), or other programmatic methods to deploy a container group.

First, populate the `volumes` array in the container group `properties` section of the file.

Next, for each container in the container group in which you'd like to mount the `emptyDir` volume, populate the `volumeMounts` array in the `properties` section of the container definition.

For example, the following Resource Manager template creates a container group consisting of two containers, each of which mounts the `emptyDir` volume:

JSON

```
{
 "$schema": "https://schema.management.azure.com/schemas/2019-04-01/deploymentTemplate.json#",
 "contentVersion": "1.0.0.0",
 "variables": {
 "container1name": "aci-tutorial-app",
 "container1image": "mcr.microsoft.com/azuredocs/aci-helloworld:latest",
 "container2name": "aci-tutorial-sidecar",
 "container2image": "mcr.microsoft.com/azuredocs/aci-tutorial-sidecar"
 },
 "resources": [
 {
 "type": "Microsoft.ContainerInstance/containerGroups",
 "apiVersion": "2021-03-01",
 "name": "volume-demo-emptydir",
 "location": "[resourceGroup().location]",
 "properties": {
 "containers": [
 {
 "name": "[variables('container1name')]",
 "properties": {
 "image": "[variables('container1image')]",
 "resources": {
 "requests": {
 "cpu": 1,
 "memoryInGb": 1.5
 }
 },
 "ports": [
 {
 "port": 80
 }
],
 "volumeMounts": [
 {
 "name": "emptydir1",
 "mountPath": "/mnt/empty"
 }
]
 }
 },
 {
 "name": "[variables('container2name')]",
 "properties": {
 "image": "[variables('container2image')]"
 }
 }
]
 }
 }
]
}
```

```

 "name": "[variables('container2name')]",
 "properties": {
 "image": "[variables('container2image')]",
 "resources": {
 "requests": {
 "cpu": 1,
 "memoryInGb": 1.5
 }
 },
 "volumeMounts": [
 {
 "name": "emptydir1",
 "mountPath": "/mnt/empty"
 }
]
 }
],
"osType": "Linux",
"ipAddress": {
 "type": "Public",
 "ports": [
 {
 "protocol": "tcp",
 "port": "80"
 }
],
 "volumes": [
 {
 "name": "emptydir1",
 "emptyDir": {}
 }
]
}
]
}
}

```

To see examples of container group deployment, see [Deploy a multi-container group using a Resource Manager template](#) and [Deploy a multi-container group using a YAML file](#).

## Next steps

Learn how to mount other volume types in Azure Container Instances:

- [Mount an Azure file share in Azure Container Instances](#)
- [Mount a gitRepo volume in Azure Container Instances](#)
- [Mount a secret volume in Azure Container Instances](#)

---

# Feedback

Was this page helpful?

 Yes

 No

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

# Mount a `gitRepo` volume in Azure Container Instances

Article • 08/29/2024

Learn how to mount a `gitRepo` volume to clone a Git repository into your container instances.

## ⓘ Note

Mounting a `gitRepo` volume is currently restricted to Linux containers. While we are working to bring all features to Windows containers, you can find current platform differences in the [overview](#).

## gitRepo volume

The `gitRepo` volume mounts a directory and clones the specified Git repository into it during container creation. By using a `gitRepo` volume in your container instances, you can avoid adding the code for doing so in your applications.

When you mount a `gitRepo` volume, you can set three properties to configure the volume:

ⓘ [Expand table](#)

Property	Required	Description
<code>repository</code>	Yes	The full URL, including <code>http://</code> or <code>https://</code> , of the Git repository to be cloned.
<code>directory</code>	No	Directory into which the repository should be cloned. The path must not contain or start with " <code>..</code> ". If you specify " <code>..</code> ", the repository is cloned into the volume's directory. Otherwise, the Git repository is cloned into a subdirectory of the given name within the volume directory.
<code>revision</code>	No	The commit hash of the revision to be cloned. If unspecified, the <code>HEAD</code> revision is cloned.

## Mount `gitRepo` volume: Azure CLI

To mount a gitRepo volume when you deploy container instances with the [Azure CLI](#), supply the `--gitrepo-url` and `--gitrepo-mount-path` parameters to the [az container create](#) command. You can optionally specify the directory within the volume to clone into (`--gitrepo-dir`) and the commit hash of the revision to be cloned (`--gitrepo-revision`).

This example command clones the Microsoft [aci-helloworld](#) sample application into `/mnt/aci-helloworld` in the container instance:

```
Azure CLI

az container create \
--resource-group myResourceGroup \
--name hellogitrepo \
--image mcr.microsoft.com/azuredocs/aci-helloworld \
--dns-name-label aci-demo \
--ports 80 \
--gitrepo-url https://github.com/Azure-Samples/aci-helloworld \
--gitrepo-mount-path /mnt/aci-helloworld
```

To verify the gitRepo volume was mounted, launch a shell in the container with [az container exec](#) and list the directory:

```
Azure CLI

az container exec --resource-group myResourceGroup --name hellogitrepo --exec-command /bin/sh
```

```
Output

/usr/src/app # ls -l /mnt/aci-helloworld/
total 16
-rw-r--r-- 1 root root 144 Apr 16 16:35 Dockerfile
-rw-r--r-- 1 root root 1162 Apr 16 16:35 LICENSE
-rw-r--r-- 1 root root 1237 Apr 16 16:35 README.md
drwxr-xr-x 2 root root 4096 Apr 16 16:35 app
```

## Mount gitRepo volume: Resource Manager

To mount a gitRepo volume when you deploy container instances with an [Azure Resource Manager template](#), first populate the `volumes` array in the container group `properties` section of the template. Then, for each container in the container group in which you'd like to mount the *gitRepo* volume, populate the `volumeMounts` array in the `properties` section of the container definition.

For example, the following Resource Manager template creates a container group consisting of a single container. The container clones two GitHub repositories specified by the *gitRepo* volume blocks. The second volume includes additional properties specifying a directory to clone to, and the commit hash of a specific revision to clone.

JSON

```
{
 "$schema": "https://schema.management.azure.com/schemas/2019-04-
01/deploymentTemplate.json#",
 "contentVersion": "1.0.0.0",
 "variables": {
 "container1name": "aci-tutorial-app",
 "container1image": "mcr.microsoft.com/azuredocs/aci-helloworld"
 },
 "resources": [
 {
 "type": "Microsoft.ContainerInstance/containerGroups",
 "apiVersion": "2021-03-01",
 "name": "volume-demo-gitrepo",
 "location": "[resourceGroup().location]",
 "properties": {
 "containers": [
 {
 "name": "[variables('container1name')]",
 "properties": {
 "image": "[variables('container1image')]",
 "resources": {
 "requests": {
 "cpu": 1,
 "memoryInGb": 1.5
 }
 },
 "ports": [
 {
 "port": 80
 }
],
 "volumeMounts": [
 {
 "name": "gitrepo1",
 "mountPath": "/mnt/repo1"
 },
 {
 "name": "gitrepo2",
 "mountPath": "/mnt/repo2"
 }
]
 }
 }
],
 "osType": "Linux",
 "ipAddress": {
 "type": "PublicIP",
 "sku": "Standard",
 "ipAllocationMethod": "Dynamic",
 "idleTimeoutInMinutes": 45
 }
 }
 }
]
}
```

```
 "type": "Public",
 "ports": [
 {
 "protocol": "tcp",
 "port": "80"
 }
],
 "volumes": [
 {
 "name": "gitrepo1",
 "gitRepo": {
 "repository": "https://github.com/Azure-Samples/aci-helloworld"
 }
 },
 {
 "name": "gitrepo2",
 "gitRepo": {
 "directory": "my-custom-clone-directory",
 "repository": "https://github.com/Azure-Samples/aci-helloworld",
 "revision": "d5ccfcfcd0d81f7ca5e3dbe6e5a7705b579101f1"
 }
 }
]
 }
}
```

The resulting directory structure of the two cloned repos defined in the preceding template is:

```
/mnt/repo1/aci-helloworld
/mnt/repo2/my-custom-clone-directory
```

To see an example of container instance deployment with an Azure Resource Manager template, see [Deploy multi-container groups in Azure Container Instances](#).

## Private Git repo authentication

To mount a gitRepo volume for a private Git repository, specify credentials in the repository URL. Typically, credentials are in the form of a user name and a personal access token (PAT) that grants scoped access to the repository.

For example, the Azure CLI `--gitrepo-url` parameter for a private GitHub repository would appear similar to the following (where "gituser" is the GitHub user name, and "abcdef1234fdsa4321abcdef" is the user's personal access token):

```
Console

--gitrepo-url
https://gituser:abcdef1234fdsa4321abcdef@github.com/GitUser/some-private-
repository
```

For an Azure Repos Git repository, specify any user name (you can use "azurereposuser" as in the following example) in combination with a valid PAT:

```
Console

--gitrepo-url
https://azurereposuser:abcdef1234fdsa4321abcdef@dev.azure.com/your-
org/_git/some-private-repository
```

For more information about personal access tokens for GitHub and Azure Repos, see the following guidance:

GitHub: [Creating a personal access token for the command line ↗](#)

Azure Repos: [Create personal access tokens to authenticate access](#)

## Next steps

Learn how to mount other volume types in Azure Container Instances:

- [Mount an Azure file share in Azure Container Instances](#)
- [Mount an emptyDir volume in Azure Container Instances](#)
- [Mount a secret volume in Azure Container Instances](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# Configure liveness probes

Article • 08/29/2024

Containerized applications may run for extended periods of time, resulting in broken states that may need to be repaired by restarting the container. Azure Container Instances supports liveness probes so that you can configure your containers within your container group to restart if critical functionality isn't working. The liveness probe behaves like a [Kubernetes liveness probe](#).

This article explains how to deploy a container group that includes a liveness probe, demonstrating the automatic restart of a simulated unhealthy container.

Azure Container Instances also supports [readiness probes](#), which you can configure to ensure that traffic reaches a container only when it's ready for it.

## YAML deployment

Create a `liveness-probe.yaml` file with the following snippet. This file defines a container group that consists of an NGINX container that eventually becomes unhealthy.

YAML

```
apiVersion: 2019-12-01
location: eastus
name: livenessesstest
properties:
 containers:
 - name: mycontainer
 properties:
 image: mcr.microsoft.com/oss/nginx/nginx:1.15.5-alpine
 command:
 - "/bin/sh"
 - "-c"
 - "touch /tmp/healthy; sleep 30; rm -rf /tmp/healthy; sleep 600"
 ports: []
 resources:
 requests:
 cpu: 1.0
 memoryInGB: 1.5
 livenessProbe:
 exec:
 command:
 - "cat"
 - "/tmp/healthy"
 periodSeconds: 5
 osType: Linux
 restartPolicy: Always
```

```
tags: null
type: Microsoft.ContainerInstance/containerGroups
```

Run the following command to deploy this container group with the preceding YAML configuration:

Azure CLI

```
az container create --resource-group myResourceGroup --name livenessstest -f
liveness-probe.yaml
```

## Start command

The deployment includes a `command` property defining a starting command that runs when the container first starts running. This property accepts an array of strings. This command simulates the container entering an unhealthy state.

First, it starts a bash session and creates a file called `healthy` within the `/tmp` directory. It then sleeps for 30 seconds before deleting the file, then enters a 10-minute sleep:

Bash

```
/bin/sh -c "touch /tmp/healthy; sleep 30; rm -rf /tmp/healthy; sleep 600"
```

## Liveness command

This deployment defines a `livenessProbe` that supports an `exec` liveness command that acts as the liveness check. If this command exits with a nonzero value, the container is killed and restarted, signaling the `healthy` file couldn't be found. If this command exits successfully with exit code 0, no action is taken.

The `periodSeconds` property designates the liveness command should execute every 5 seconds.

## Verify liveness output

Within the first 30 seconds, the `healthy` file created by the start command exists. When the liveness command checks for the `healthy` file's existence, the status code returns 0, signaling success, so no restarting occurs.

After 30 seconds, the `cat /tmp/healthy` command begins to fail, causing unhealthy and killing events to occur.

These events can be viewed from the Azure portal or Azure CLI.

NAME	TYPE	FIRST TIMESTAMP	LAST TIMESTAMP	MESSAGE	COUNT
Pulling	Normal	2018-06-06T22:34:39Z	2018-06-06T22:34:39Z	pulling Image "nginx"	1
Pulled	Normal	2018-06-06T22:34:47Z	2018-06-06T22:34:47Z	Successfully pulled Image "nginx"	1
Created	Normal	2018-06-06T22:34:47Z	2018-06-06T22:34:47Z	Created container with id d5c1d72a3a7...	1
Started	Normal	2018-06-06T22:34:47Z	2018-06-06T22:34:47Z	Started container with id d5c1d72a3a78...	1
Unhealthy	Warning	2018-06-06T22:35:21Z	2018-06-06T22:35:31Z	Liveness probe failed: cat: /tmp/healthy:...	3
Killing	Normal	2018-06-06T22:35:33Z	2018-06-06T22:35:33Z	Killing container with id docker://d5c1d...	1

By viewing the events in the Azure portal, events of type `Unhealthy` are triggered upon the liveness command failing. The subsequent event is of type `Killing`, signifying a container deletion so a restart can begin. The restart count for the container increments each time this event occurs.

Restarts are completed in-place so resources like public IP addresses and node-specific contents are preserved.

NAME	IMAGE	STATE	START TIME	RESTART COUNT
mycontainer	nginx	Running	2018-06-08T20:37:36Z	3

If the liveness probe continuously fails and triggers too many restarts, your container enters an exponential back-off delay.

## Liveness probes and restart policies

Restart policies supersede the restart behavior triggered by liveness probes. For example, if you set a `restartPolicy = Never` and a liveness probe, the container group won't restart because of a failed liveness check. The container group instead adheres to the container group's restart policy of `Never`.

## Next steps

Task-based scenarios may require a liveness probe to enable automatic restarts if a prerequisite function isn't working properly. For more information about running task-

based containers, see [Run containerized tasks in Azure Container Instances](#).

---

## Feedback

Was this page helpful?

 Yes

 No

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

# Configure readiness probes

Article • 08/29/2024

For containerized applications that serve traffic, you might want to verify that your container is ready to handle incoming requests. Azure Container Instances supports readiness probes to include configurations so that your container can't be accessed under certain conditions. The readiness probe behaves like a [Kubernetes readiness probe](#). For example, a container application might need to load a large data set during startup, and you don't want it to receive requests during this time.

This article explains how to deploy a container group that includes a readiness probe, so that a container only receives traffic when the probe succeeds.

Azure Container Instances also supports [liveness probes](#), which you can configure to cause an unhealthy container to automatically restart.

## YAML configuration

As an example, create a `readiness-probe.yaml` file with the following snippet that includes a readiness probe. This file defines a container group that consists of a container running a small web app. The app is deployed from the public

`mcr.microsoft.com/azuredocs/aci-helloworld` image. This containerized app is also demonstrated in [Deploy a container instance in Azure using the Azure CLI](#) and other quickstarts.

YAML

```
apiVersion: 2019-12-01
location: eastus
name: readinesstest
properties:
 containers:
 - name: mycontainer
 properties:
 image: mcr.microsoft.com/azuredocs/aci-helloworld
 command:
 - "/bin/sh"
 - "-c"
 - "node /usr/src/app/index.js & (sleep 240; touch /tmp/ready); wait"
 ports:
 - port: 80
 resources:
 requests:
 cpu: 1.0
 memoryInGB: 1.5
```

```
readinessProbe:
 exec:
 command:
 - "cat"
 - "/tmp/ready"
 periodSeconds: 5
osType: Linux
restartPolicy: Always
ipAddress:
 type: Public
ports:
 - protocol: tcp
 port: '80'
tags: null
type: Microsoft.ContainerInstance/containerGroups
```

## Start command

The deployment includes a `command` property defining a starting command that runs when the container first starts running. This property accepts an array of strings. This command simulates a time when the web app runs but the container isn't ready.

First, it starts a shell session and runs a `node` command to start the web app. It also starts a command to sleep for 240 seconds, after which it creates a file called `ready` within the `/tmp` directory:

```
Bash
```

```
node /usr/src/app/index.js & (sleep 240; touch /tmp/ready); wait
```

## Readiness command

This YAML file defines a `readinessProbe` that supports an `exec` readiness command that acts as the readiness check. This example readiness command tests for the existence of the `ready` file in the `/tmp` directory.

When the `ready` file doesn't exist, the readiness command exits with a nonzero value; the container continues running but can't be accessed. When the command exits successfully with exit code 0, the container is ready to be accessed.

The `periodSeconds` property designates the readiness command should execute every 5 seconds. The readiness probe runs for the lifetime of the container group.

# Example deployment

Run the following command to deploy a container group with the preceding YAML configuration:

```
Azure CLI

az container create --resource-group myResourceGroup --file readiness-probe.yaml
```

## View readiness checks

In this example, during the first 240 seconds, the readiness command fails when it checks for the `ready` file's existence. The status code returned signals that the container isn't ready.

These events can be viewed from the Azure portal or Azure CLI. For example, the portal shows events of type `Unhealthy` are triggered upon the readiness command failing.

Name	Image	State	Previous state	Start time	Restart count
mycontainer	mcr.microsoft.com/azure...	Running	-	2019-10-15T23:46:07Z	0

Name	Type	First timestamp	Last timestamp	Message	Count
Unhealthy	Warning	10/15/2019, 4:46 PM PDT	10/15/2019, 4:47 PM PDT	Readiness probe failed: cat... 16	16
Created	Normal	10/15/2019, 4:46 PM PDT	10/15/2019, 4:46 PM PDT	Created container	1
Started	Normal	10/15/2019, 4:46 PM PDT	10/15/2019, 4:46 PM PDT	Started container	1
Pulled	Normal	10/15/2019, 4:46 PM PDT	10/15/2019, 4:46 PM PDT	Successfully pulled image ...	1
Pulling	Normal	10/15/2019, 4:45 PM PDT	10/15/2019, 4:45 PM PDT	pulling image "mcr.micros..."	1

## Verify container readiness

After starting the container, you can verify that it's not accessible initially. After provisioning, get the IP address of the container group:

```
Azure CLI

az container show --resource-group myResourceGroup --name readinesstest --
```

```
query "ipAddress.ip" --out tsv
```

Try to access the site while the readiness probe fails:

Bash

```
wget <ipAddress>
```

Output shows the site isn't accessible initially:

Bash

```
wget 192.0.2.1
```

Output

```
--2019-10-15 16:46:02-- http://192.0.2.1/
Connecting to 192.0.2.1... connected.
HTTP request sent, awaiting response...
```

After 240 seconds, the readiness command succeeds, signaling the container is ready.

Now, when you run the `wget` command, it succeeds:

Bash

```
wget 192.0.2.1
```

Output

```
--2019-10-15 16:46:02-- http://192.0.2.1/
Connecting to 192.0.2.1... connected.
HTTP request sent, awaiting response...200 OK
Length: 1663 (1.6K) [text/html]
Saving to: 'index.html.1'

index.html.1 100%
[=====] 1.62K
--.-KB/s in 0s

2019-10-15 16:49:38 (113 MB/s) - 'index.html.1' saved [1663/1663]
```

When the container is ready, you can also access the web app by browsing to the IP address using a web browser.

 **Note**

The readiness probe continues to run for the lifetime of the container group. If the readiness command fails at a later time, the container again becomes inaccessible.

## Next steps

A readiness probe could be useful in scenarios involving multi-container groups that consist of dependent containers. For more information about multi-container scenarios, see [Container groups in Azure Container Instances](#).

---

## Feedback

Was this page helpful?

 Yes

 No

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

# Manually stop or start containers in Azure Container Instances

Article • 08/29/2024

The [restart policy](#) setting of a container group determines how container instances start or stop by default. You can override the default setting by manually stopping or starting a container group.

## ⓘ Note

If your container group is configured with an IP address, that IP address can change when the container group is restarted.

## Stop

Manually stop a running container group - for example, by using the [az container stop](#) command or Azure portal. For certain container workloads, you might want to stop a long-running container group after a defined period to save on costs.

*When a container group enters the Stopped state, it terminates and recycles all the containers in the group. It doesn't preserve container state.*

When the containers are recycled, the [resources](#) are deallocated and billing stops for the container group.

The stop action has no effect if the container group already terminated (is in either a Succeeded or Failed state). For example, a container group with run-once container tasks that ran successfully terminates in the Succeeded state. Attempts to stop the group in that state don't change the state.

## Start

When a container group is stopped—either because the containers terminated on their own or you manually stopped the group—you can start the containers. For example, use the [az container start](#) command or Azure portal to manually start the containers in the group. If the container image for any container is updated, a new image is pulled.

Starting a container group begins a new deployment with the same container configuration. This action can help you quickly reuse a known container group

configuration that works as you expect. You don't have to create a new container group to run the same workload.

This action starts all containers in a container group. You can't start a specific container in the group.

After you manually start or restart a container group, the container group runs according to the configured restart policy.

## Restart

You can restart a container group while it's running - for example, by using the [az container restart](#) command. This action restarts all containers in the container group. If the container image for any container is updated, a new image is pulled.

Restarting a container group is helpful when you want to troubleshoot a deployment problem. For example, if a temporary resource limitation prevents your containers from running successfully, restarting the group might solve the problem.

This action restarts all containers in a container group. You can't restart a specific container in the group.

After you manually restart a container group, the container group runs according to the configured restart policy.

## Next steps

Learn more about [restart policy settings](#) in Azure Container Instances.

In addition to manually stopping and starting a container group with the existing configuration, you can [update the settings](#) of a running container group.

---

## Feedback

Was this page helpful?



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

# Update containers in Azure Container Instances

Article • 08/29/2024

During normal operation of your container instances, you could find it necessary to update the running containers in a [container group](#). For example, you might wish to update a property such as an image version, a DNS name, or an environment variable, or refresh a property in a container whose application crashed.

Update the containers in a running container group by redeploying an existing group with at least one modified property. When you update a container group, all running containers in the group are restarted in-place, usually on the same underlying container host.

## ⓘ Note

Terminated or deleted container groups can't be updated. Once a container group has terminated (is in either a Succeeded or Failed state) or has been deleted, the group must be deployed as new. See other [limitations](#).

## Update a container group

To update an existing container group:

- Issue the create command (or use the Azure portal) and specify the name of an existing group
- Modify or add at least one property of the group that supports update when you redeploy. Certain properties [don't support updates](#).
- Set other properties with the values you provided previously. If you don't set a value for a property, it reverts to its default value.

## ⓘ Note

If you set all properties to the values you previously provided and don't modify or add any, the container will restart in response to the create command.

## 💡 Tip

A [YAML file](#) helps maintain a container group's deployment configuration, and provides a starting point to deploy an updated group. If you used a different method to create the group, you can export the configuration to YAML by using [az container export](#)

## Example

The following Azure CLI example updates a container group with a new DNS name label. Because the DNS name label property of the group is one that can be updated, the container group is redeployed, and its containers restarted.

Initial deployment with DNS name label *myapplication-staging*:

Azure CLI

```
Create container group
az container create --resource-group myResourceGroup --name mycontainer \
--image nginx:alpine --dns-name-label myapplication-staging
```

Update the container group with a new DNS name label, *application*, and set the remaining properties with the values used previously:

Azure CLI

```
Update DNS name label (restarts container), leave other properties
unchanged
az container create --resource-group myResourceGroup --name mycontainer \
--image nginx:alpine --dns-name-label myapplication
```

## Update benefits

The primary benefit of updating an existing container group is faster deployment. When you redeploy an existing container group, its container image layers are pulled from layers cached by the previous deployment. Instead of pulling all image layers fresh from the registry as is done with new deployments, only modified layers (if any) are pulled.

Applications based on larger container images like Windows Server Core can see significant improvement in deployment speed when you update instead of delete and deploy new.

## Limitations

- Not all properties of a container group support updates. To change some properties of a container group, you must first delete, then redeploy the group. See [Properties that require container delete](#).
- All containers in a container group are restarted when you update the container group. You can't perform an update or in-place restart of a specific container in a multi-container group.
- The IP address of a container group is typically retained between updates, but isn't guaranteed to remain the same. As long as the container group is deployed to the same underlying host, the container group retains its IP address. Although rare, there are some Azure-internal events that can cause redeployment to a different host. To mitigate this issue, we recommend using a DNS name label for your container instances.
- Terminated or deleted container groups can't be updated. Once a container group is stopped (is in the *Terminated* state) or deleted, the group is deployed as new.

 **Note**

The update command might not work if the Azure Container Group is attached to an Azure Storage profile.

## Properties that require container delete

Not all container group properties can be updated. For example, to change the restart policy of a container, you must first delete the container group, then create it again.

Changes to these properties require container group deletion before redeployment:

- OS type
- CPU, memory, or GPU resources
- Restart policy
- Network profile
- Availability zone

 **Important**

[Network profiles](#) have been retired as of the `2021-07-01` API version. If you're using this or a more recent version, ignore any steps and actions related to network profiles.

When you delete a container group and recreate it, it's not "redeployed," but created new. All image layers are pulled fresh from the registry, not from layers cached by a previous deployment. The IP address of the container might also change due to being deployed to a different underlying host.

## Next steps

This article mentions **container groups** several times. Every container in Azure Container Instances is deployed in a container group, and container groups can contain more than one container. The following articles provide more information about container groups:

- [Container groups in Azure Container Instances](#)
  - [Deploy a multi-container group](#)
  - [Manually stop or start containers in Azure Container Instances](#)
- 

## Feedback

Was this page helpful?

 Yes

 No

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

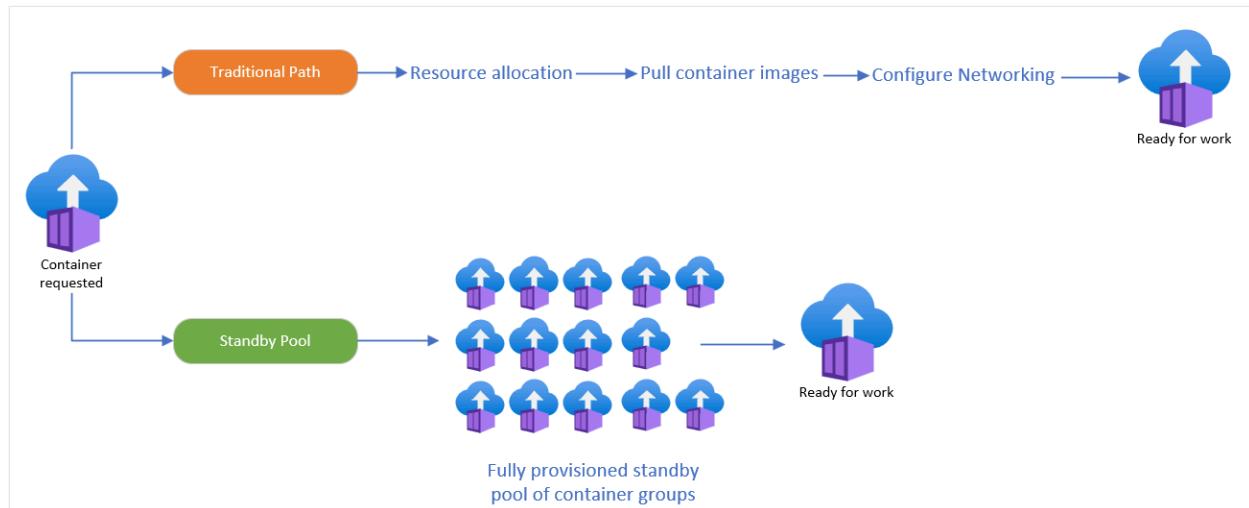
# Standby pools for Azure Container Instances (Preview)

Article • 11/19/2024

## ⓘ Important

Standby pools for Azure Container Instances are currently in preview. Previews are made available to you on the condition that you agree to the [supplemental terms of use](#). Some aspects of this feature may change prior to general availability (GA).

Standby pools for Azure Container Instances enable you to create a pool of pre-provisioned container groups that can be used in response to incoming traffic. The container groups in the pool are fully provisioned, initialized, and ready to receive work.



## Limitations

Standby pools for Azure Container Instances is not available in the Azure portal.

## Prerequisites

### Feature Registration

Register the standby pool resource provider and the standby pool preview feature with your subscription using Azure Cloud Shell. Registration can take up to 30 minutes to successfully show as registered. You can rerun the below commands to determine when the feature is successfully registered.

## Azure PowerShell

```
Register-AzResourceProvider -ProviderNameSpace Microsoft.ContainerInstance
Register-AzResourceProvider -ProviderNamespace Microsoft.StandbyPool
Register-AzProviderFeature -FeatureName StandbyContainerGroupPoolPreview -
ProviderNamespace Microsoft.StandbyPool
```

## Role-based Access Control Permissions

To allow standby pools to create container groups in your subscription, assign the appropriate permissions to the standby pool resource provider.

1. In the Azure portal, navigate to your subscriptions.
2. Select the subscription you want to adjust permissions.
3. Select **Access Control (IAM)**.
4. Select **Add** and **Add role assignment**.
5. Under the **Role** tab, search for **Standby Container Group Pool Contributor** and select it.
6. Move to the **Members** Tab.
7. Select **+ Select members**.
8. Search for **Standby Pool Resource Provider** and select it.
9. Move to the **Review + assign** tab.
10. Apply the changes.
11. Repeat the above steps also assinging the **Azure Container Instances Contributor Role** and the **Network Contributor** to the **Standby Pool Resource Provider**.

For more information on assigning roles, see [assign Azure roles using the Azure portal](#).

## Using a container from the standby pool

When you require a new container group, you can immediately pull one from the standby pool that is provisioned and running.

Standby pools only give out container groups from the pool that are fully provisioned and ready to receive work. For example, when the instances in your pool are still being initialized, they aren't in the running state and are't given out when a container is requested. If no instances in the pool are available, Azure Container Instances will default back to net new container group creation.

## Standby pool size

The number of container groups in a standby pool is determined by setting the `maxReadyCapacity` parameter. When a container group is consumed from the pool, the standby pool automatically begins to refill ensuring that your standby pool maintains the set maximum ready capacity.

The only available refill policy for standby pools on Azure Container instances is `Always`.

 Expand table

Setting	Description
<code>maxReadyCapacity</code>	The maximum number of container groups you want deployed in the pool.
<code>refillPolicy</code>	Tells the standby pool to immediately replenish container groups to maintain the <code>maxReadyCapacity</code> .

## Container group profile

A container group profile tells the standby pool how to configure the containers in the pool. If you make any changes to the container group profile, you also need to update your standby pool to ensure the updates are applied to the instances in the pool.

JSON

```
{
 "location": "[{location}]",
 "properties": {
 "containers": [
 {
 "name": "[mycontainergroupprofile]",
 "properties": {
 "command": [],
 "environmentVariables": [],
 "image": "mcr.microsoft.com/azuredocs/aci-helloworld:latest",
 "ports": [
 {
 "port": 8000
 }
],
 "resources": {
 "requests": {
 "cpu": 1,
 "memoryInGB": 1.5
 }
 }
 }
 }
],
 "imageRegistryCredentials": []
 }
}
```

```
 "ipAddress":{
 "ports": [
 {
 "protocol": "TCP",
 "port": 8000
 }
],
 "type": "Public"
 },
 "osType": "Linux",
 "sku": "Standard"
 }
}
```

## Config maps

A [config map](#) is a property that can be associated with a container group profile and be used to apply container configurations similar to environment variables and secret volumes. However, when using environment variables or secret volumes, restarting the pod is required for the changes to take effect. By using config maps, the configurations can be applied without restarting the container. This enables out of band updates so containers can read the new values without restarting.

Azure Container Instances can be created with or without config maps and can be updated at any point in time post creation using config maps. Updating config maps in an existing running container group can be accomplished quickly and without causing the container to reboot.

For more information, see [use config maps](#).

JSON

```
{
 "properties": {
 "containers": [
 {
 "name": "{mycontainergroupprofile}",
 "properties": {
 "image": "mcr.microsoft.com/azuredocs/aci-helloworld",
 "ports": [
 {
 "port": 80,
 "protocol": "TCP"
 }
],
 "resources": {
 "limits": {
 "cpu": 1.0,
 "memory": 128
 },
 "requests": {
 "cpu": 0.5,
 "memory": 64
 }
 }
 }
 }
]
 }
}
```

```
 "requests": {
 "memoryInGB": 0.5,
 "cpu": 0.5
 }
 },
 "configMap": {
 "keyValuePairs": {
 "key1": "value1",
 "key2": "value2"
 }
 }
}
],
"osType": "Linux",
"ipAddress": {
 "type": "Public",
 "ports": [
 {
 "protocol": "tcp",
 "port": 80
 }
]
},
"location": "{location}"
}
```

## Confidential containers

Standby pools for Azure container instances support confidential containers. To utilize [confidential containers](#) update the `sku` type to `Confidential` in the container group profile.

 **Important**

Values passed using config maps are not included in the security policy or validated by the runtime before the file mount is made available to the container. Any values that could have an impact to data or application security cannot be trusted by the application during execution and instead should be made available to the container using environment variables.

JSON

```
{
 "location": "{location}",
```

```

"properties": {
 "containers": [
 {
 "name": "{mycontainergroupprofile}",
 "properties": {
 "command": [],
 "environmentVariables": [],
 "image": "mcr.microsoft.com/azuredocs/aci-helloworld:latest",
 "ports": [
 {
 "port": 8000
 }
],
 "resources": {
 "requests": {
 "cpu": 1,
 "memoryInGB": 1.5
 }
 }
 }
],
 "imageRegistryCredentials": [],
 "ipAddress": {
 "ports": [
 {
 "protocol": "TCP",
 "port": 8000
 }
],
 "type": "Public"
 },
 "osType": "Linux",
 "sku": "Confidential"
 }
}

```

## Managed Identity

Standby pools for Azure Container Instances support integration with Managed Identity. Applying a managed identity is performed when requesting a container from the standby pool and including the `identity` parameters and settings. Managed Identity is not a property supported directly in the container group profile.

JSON

```
{
 "location": "{location}",

```

```

"properties": {
 "standByPoolProfile": {
 "id": "/subscriptions/{subscriptionId}/resourceGroups/{resourceGroup}/providers/Microsoft.StandbyPool/standbyContainerGroupPools/{standbyPoolName}"
 },
 "containerGroupProfile": {
 "id": "/subscriptions/{subscriptionId}/resourceGroups/{resourceGroup}/providers/Microsoft.ContainerInstance/containerGroupProfiles/{mycontainergroupprofile}",
 "revision": {revisionNumber}
 },
},
"identity": {
 "type": "UserAssigned",
 "userAssignedIdentities": {
 "/subscriptions/{subscriptionId}/resourceGroups/{resourceGroup}/providers/Microsoft.ManagedIdentity/userAssignedIdentities/{identity}": {}
 },
},
"containers": [
 {
 "name": "{mycontainergroupprofile}",
 "properties": {
 "configMap": {
 "keyValuePairs": {
 "{newKey)": "{newValue}"
 }
 }
 }
 }
]
}
}

```

## Availability zones

Standby pools for Azure Container Instances supports creating and requesting containers across availability zones. Creating a zonal standby pool is currently only available using the standby pool [REST APIs](#) using version 2024-08-01-preview and only available in the Central India region.

## Create a zonal standby pool

HTTP

PUT

<https://management.azure.com/subscriptions/{SubscriptionID}/resourceGroups/mResourceGroup/providers/Microsoft.StandbyPool/standbyContainerGroupPools/my>

StandbyPool?api-version=2024-08-01-preview

Request Body

```
{
 "properties": {
 "elasticityProfile": {
 "maxReadyCapacity": 20,
 "refillPolicy": "always"
 },
 "containerGroupProperties": {
 "containerGroupProfile": {
 "id":
 "/subscriptions/{SubscriptionID}/resourceGroups/myResourceGroup/providers/Microsoft.ContainerInstance/containerGroupProfiles/mycontainergroupprofile",
 "revision": 1
 },
 "subnetIds": [
 {
 "id":
 "/subscriptions/{subscriptionId}/resourceGroups/myResourceGroup/providers/Microsoft.Network/virtualNetworks/myVNET/subnets/mySubnet"
 }
]
 },
 "zones": [
 "1",
 "2",
 "3"
]
 },
 "location": "West Central US"
}
```

## Next steps

[Create a standby pool for Azure Container Instances.](#)

## Feedback

Was this page helpful?

 Yes

 No

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

# Create a standby pool for Azure Container Instances (Preview)

Article • 11/19/2024

## ⓘ Important

Standby pools for Azure Container Instances is currently in preview. Previews are made available to you on the condition that you agree to the [supplemental terms of use](#). Some aspects of this feature may change prior to general availability (GA).

This article steps through creating a container group profile and using that profile to configure a standby pool for Azure Container Instances.

## Prerequisites

Before utilizing standby pools, complete the feature registration and configure role based access controls listed in the [Standby Pools for Azure Container Instances](#) overview page.

## Create a container group profile

The container group profile tells the standby pool how to configure the containers in the pool. If you make changes to the container group profile, you also need to update your standby pool to ensure the updates are applied to the instances in the pool.

## ⓘ Note

To use [confidential containers](#) update the `sku` type to `Confidential` when creating your container group profile.

CLI

Create a container group profile using [az container container-group-profile create](#). You can optionally include config map details in the container group profile. For more information on config maps, see [use config maps](#).

Azure CLI

```
az container container-group-profile create \
 --resource-group myResourceGroup \
 --name mycontainergroupprofile \
 --location WestCentralUS \
 --image nginx \
 --os-type Linux \
 --ip-address Public \
 --ports 8000 \
 --cpu 1 \
 --memory 1.5 \
 --restart-policy Never
```

## Create a standby pool

CLI

Create a standby pool and associate it with a container group profile using [az standby-container-group-pool create](#).

Azure CLI

```
az standby-container-group-pool create \
 --resource-group myResourceGroup
 --location WestCentralUS \
 --name myStandbyPool \
 --max-ready-capacity 20 \
 --refill-policy always \
 --container-profile-id
 "/subscriptions/{subscriptionId}/resourceGroups/myResourceGroup/provider
 s/Microsoft.ContainerInstance/containerGroupProfiles/myContainerGroupPro
 file"
```

## Next steps

[Request a container from the standby pool.](#)

## Feedback

Was this page helpful?

 Yes

 No



# Request a container from a standby pool for Azure Container Instances (Preview)

Article • 11/19/2024

## ⓘ Important

Standby pools for Azure Container Instances is currently in preview. Previews are made available to you on the condition that you agree to the [supplemental terms of use](#). Some aspects of this feature may change prior to general availability (GA).

This article steps through requesting a container group from a standby pool for Azure Container Instances.

## Prerequisites

Before utilizing standby pools, complete the feature registration and configure role based access controls listed in the [Standby pools for Azure Container Instances](#) overview page.

## Request a container from the standby pool

CLI

Request a container group from a standby pool using [az container create](#) and specifying the standby pool and container group profile. For more information on using config maps during container requests, see [use config maps](#).

Azure CLI

```
az container create \
 --resource-group myResourceGroup \
 --name mycontainer \
 --location WestCentralUS \
 --config-map key1=value1 key2=value2 \
 --container-group-profile-id
 "/subscriptions/{subscriptionId}/resourceGroups/myResourceGroup/providers/Microsoft.ContainerInstance/containerGroupProfiles/mycontainergroupprofile" \
 --container-group-profile-revision 1 \
```

```
--standby-pool-profile-id
"/subscriptions/{subscriptionId}/resourceGroups/myResourceGroup/provider
s/Microsoft.StandbyPool/standbyContainerGroupPools/myStandbyPool"
```

## Next steps

- Get standby pool and container details using the standby pool runtime view APIs.
- Update or delete your standby pool for Azure Container Instances.

---

## Feedback

Was this page helpful?

 Yes

 No

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

# Update or delete a standby pool for Azure Container Instances (Preview)

Article • 11/19/2024

## ⓘ Important

Standby pools for Azure Container Instances is currently in preview. Previews are made available to you on the condition that you agree to the [supplemental terms of use](#). Some aspects of this feature may change prior to general availability (GA).

This article steps through updating or deleting a standby pool for Azure Container Instances.

## Prerequisites

Before utilizing standby pools, complete the feature registration and configure role based access controls listed in the [Standby pools for Azure Container Instances](#) overview page.

## Update a standby pool

A standby pool can be updated at any point in time. The settings that are adjustable after creation include `maxReadyCapacity` and the associated `containerGroupProfile`. If you update the container group profile of the standby pool, the new profile must also be in the same subscription and location as the standby pool. Once the profile has been updated, the pool will drain all existing instances and replaced them with new ones.

CLI

Update an existing standby pool using [az standby-container-group-pool update](#).

Azure CLI

```
az standby-container-group-pool update \
 --resource-group myResourceGroup
 --location WestCentralUS \
 --name myStandbyPool \
 --max-ready-capacity 20 \
 --refill-policy always \
 --container-profile-id
```

```
"/subscriptions/{subscriptionId}/resourceGroups/myResourceGroup/provider
s/Microsoft.ContainerInstance/containerGroupProfiles/mycontainergrouppro
file"
```

## Delete a standby pool

CLI

Delete an existing standby pool using [az standby-container-group-pool delete](#).

Azure CLI

```
az standby-container-group-pool delete \
--resource-group myResourceGroup \
--name myStandbyPool
```

## Next steps

[Get standby pool and container details using the standby pool runtime view APIs.](#)

## Feedback

Was this page helpful?

 Yes

 No

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

# Get standby pool and instance details (Preview)

Article • 11/19/2024

This article discusses how to retrieve information about your standby pool and the container groups within it.

## Standby pool details

Use the standby pool runtime view APIs to get the current status of your standby pool including how many container groups are available and their current provisioning state.

CLI

```
az standby-container-group-pool status --resource-group myResourceGroup
--name myStandbyPool

{
 "id": "/subscriptions/401ef76a-dea9-45da-b19a-
db3efced675b/resourceGroups/myResourceGroup/providers/Microsoft.StandbyP
ool/standbyContainerGroupPools/myStandbyPool/runtimeViews/latest",
 "instanceCountSummary": [
 {
 "instanceCountsByState": [
 {
 "count": 5,
 "state": "Creating"
 },
 {
 "count": 20,
 "state": "Running"
 },
 {
 "count": 0,
 "state": "Deleting"
 }
]
 }
],
 "name": "latest",
 "provisioningState": "Succeeded",
 "resourceGroup": "myResourceGroup",
 "type":
"Microsoft.StandbyPool/standbyContainerGroupPools/runtimeViews"
```

}

## Next steps

Learn more about [standby pools for Azure Container Instances](#).

---

## Feedback

Was this page helpful?

 Yes

 No

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

# Frequently asked questions about standby pools for Azure Container Instances (Preview)

Article • 11/19/2024

## Important

Standby pools for Azure Container Instances is currently in preview. Previews are made available to you on the condition that you agree to the [supplemental terms of use](#). Some aspects of this feature may change prior to general availability (GA).

Get answers to frequently asked questions about standby pools for Azure Container Instances.

## How long can my standby pool name be?

A standby pool can be anywhere between 3 and 24 characters. For more information, see [Resource naming restrictions for Azure resources](#)

## How many container groups can my standby pool have?

The maximum number of container groups in a standby pool is 2,000.

## Can I change the size of my standby pool without needing to recreate it?

Yes. To change the size of your standby pool, update the max ready capacity setting.

## Can a standby pool resource be moved?

No. Standby pools don't currently support move capabilities. If you need to move a standby pool, you can consider deleting it recreating it in another location.

## I created a standby pool and I noticed that some containers are coming up in a failed state.

Ensure you have enough quota to complete the standby pool creation. Insufficient quota results in the platform attempting to create the containers in the standby pool fail after encountering a quota error. Check for multiple types of quotas such as Cores, Network Interfaces, IP Addresses, etc.

## I requested a container from my pool but it created a new container instead.

Ensure that the container groups in your standby pool are in the running state before issuing a request. For example, if using a standby If containers are in any other states such as creating or deleting, the container request defaults to creating a new container group from scratch.

## Can I create a standby pool with containers spread across availability zones?

Yes. However, standby pools for Azure Container Instances only currently support creating and requesting containers across availability zones using [REST APIs](#) using version 2024-08-01.

## Next steps

Learn more about [standby pools on Azure Container Instances](#).

---

## Feedback

Was this page helpful?



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

# Monitor Azure Container Instances

Article • 08/29/2024

This article describes:

- The types of monitoring data you can collect for this service.
- Ways to analyze that data.

## ⓘ Note

If you're already familiar with this service and/or Azure Monitor and just want to know how to analyze monitoring data, see the [Analyze](#) section near the end of this article.

When you have critical applications and business processes that rely on Azure resources, you need to monitor and get alerts for your system. The Azure Monitor service collects and aggregates metrics and logs from every component of your system. Azure Monitor provides you with a view of availability, performance, and resilience, and notifies you of issues. You can use the Azure portal, PowerShell, Azure CLI, REST API, or client libraries to set up and view monitoring data.

- For more information on Azure Monitor, see the [Azure Monitor overview](#).
- For more information on how to monitor Azure resources in general, see [Monitor Azure resources with Azure Monitor](#).

## Resource types

Azure uses the concept of resource types and IDs to identify everything in a subscription. Resource types are also part of the resource IDs for every resource running in Azure. For example, one resource type for a virtual machine is `Microsoft.Compute/virtualMachines`. For a list of services and their associated resource types, see [Resource providers](#).

Azure Monitor similarly organizes core monitoring data into metrics and logs based on resource types, also called *namespaces*. Different metrics and logs are available for different resource types. Your service might be associated with more than one resource type.

For more information about the resource types for Azure Container Instances, see [Container Instances monitoring data reference](#).

## Data storage

For Azure Monitor:

- Metrics data is stored in the Azure Monitor metrics database.
- Log data is stored in the Azure Monitor logs store. Log Analytics is a tool in the Azure portal that can query this store.
- The Azure activity log is a separate store with its own interface in the Azure portal.

You can optionally route metric and activity log data to the Azure Monitor logs store. You can then use Log Analytics to query the data and correlate it with other log data.

Many services can use diagnostic settings to send metric and log data to other storage locations outside Azure Monitor. Examples include Azure Storage, [hosted partner systems](#), and [non-Azure partner systems](#), by using [Event Hubs](#).

For detailed information on how Azure Monitor stores data, see [Azure Monitor data platform](#).

## Azure Monitor platform metrics

Azure Monitor provides platform metrics for most services. These metrics are:

- Individually defined for each namespace.
- Stored in the Azure Monitor time-series metrics database.
- Lightweight and capable of supporting near real-time alerting.
- Used to track the performance of a resource over time.

**Collection:** Azure Monitor collects platform metrics automatically. No configuration is required.

**Routing:** You can also route some platform metrics to Azure Monitor Logs / Log Analytics so you can query them with other log data. Check the **DS export** setting for each metric to see if you can use a diagnostic setting to route the metric to Azure Monitor Logs / Log Analytics.

- For more information, see the [Metrics diagnostic setting](#).
- To configure diagnostic settings for a service, see [Create diagnostic settings in Azure Monitor](#).

For a list of all metrics it's possible to gather for all resources in Azure Monitor, see [Supported metrics in Azure Monitor](#).

For a list of available metrics for Container Instances, see [Container Instances monitoring data reference](#). These metrics are available for a container group and individual containers. By default, the metrics are aggregated as averages.

All metrics for Container Instances are in the namespace **Container group standard metrics**. In a container group with multiple containers, you can filter on the **containerName** dimension to acquire metrics from a specific container within the group. Containers generate similar data as other Azure resources, but they require a containerized agent to collect required data.

## Get metrics

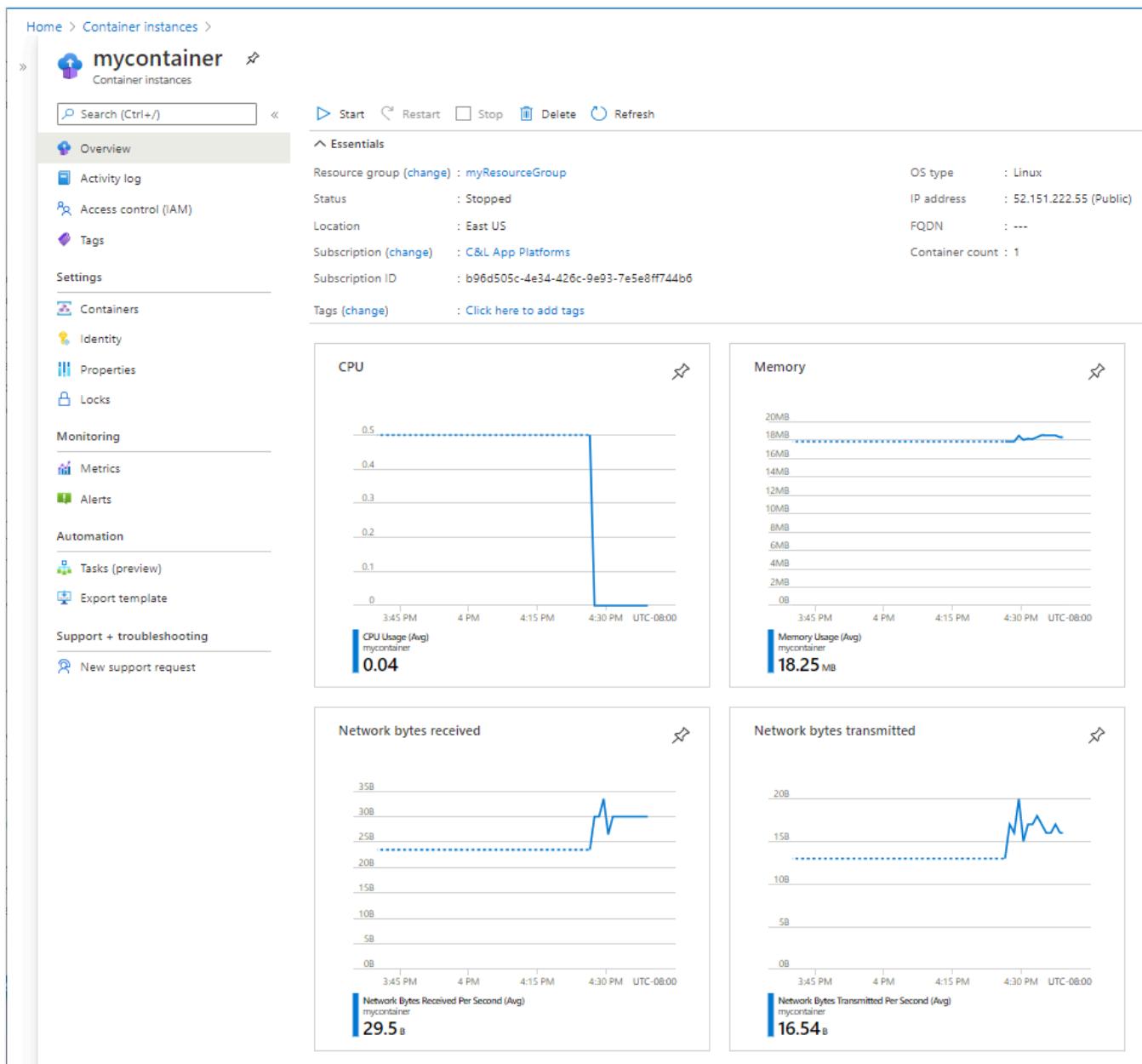
You can gather Azure Monitor metrics for container instances using either the Azure portal or Azure CLI.

### Important

Azure Monitor metrics in Azure Container Instances are currently in preview. At this time, Azure Monitor metrics are only available for Linux containers. Previews are made available to you on the condition that you agree to the [supplemental terms of use](#). Some aspects of this feature may change prior to general availability (GA).

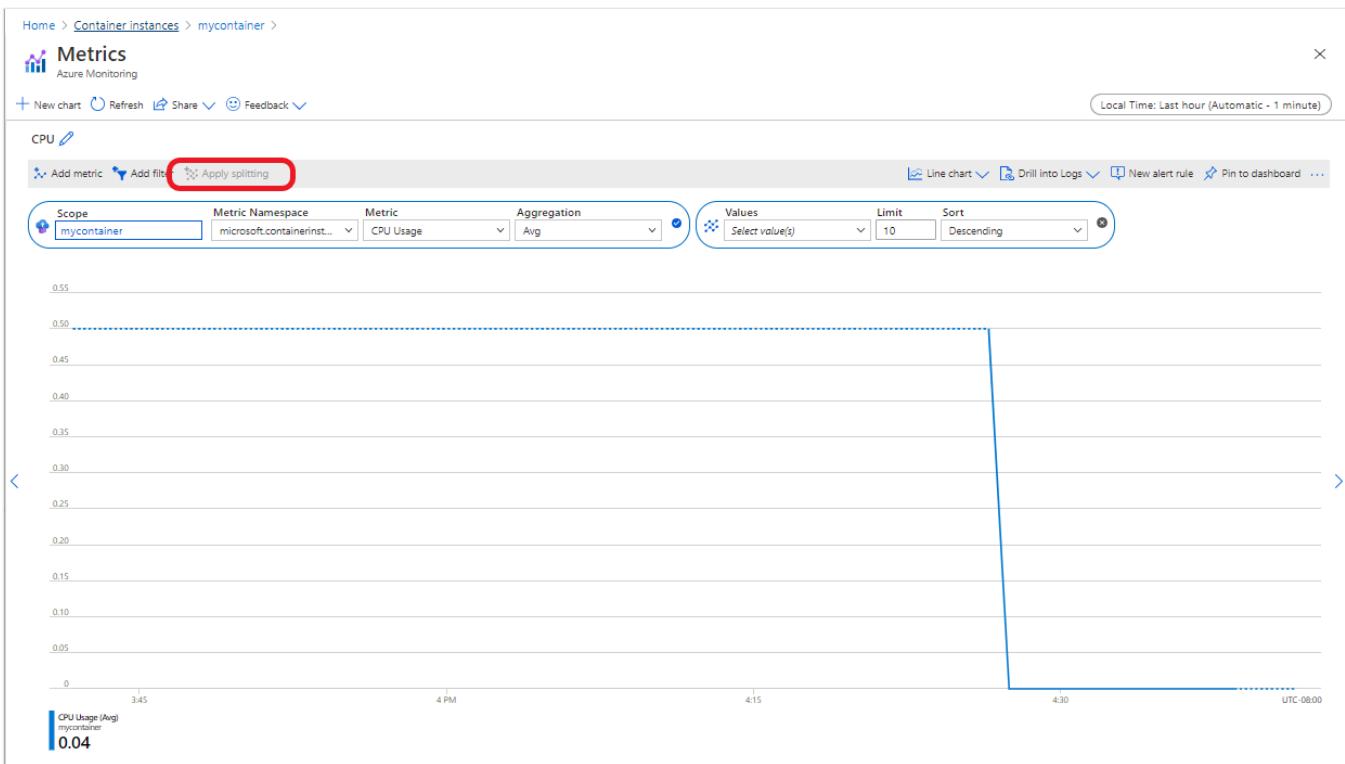
## Use the Azure portal

When a container group is created, Azure Monitor data is available in the Azure portal. To see metrics for a container group, go to the **Overview** page for the container group. Here you can see precreated charts for each of the available metrics.



In a container group that contains multiple containers, use a **dimension** to display metrics by container. To create a chart with individual container metrics, perform the following steps:

1. In the **Overview** page, select one of the metric charts, such as **CPU**.
2. Select the **Apply splitting** button, and select **Container Name**.



## Use Azure CLI

Metrics for container instances can also be gathered using the Azure CLI. First, get the ID of the container group using the following command. Replace <resource-group> with your resource group name and <container-group> with the name of your container group.

### Azure CLI

```
CONTAINER_GROUP=$(az container show --resource-group <resource-group> --name
<container-group> --query id --output tsv)
```

Use the following command to get CPU usage metrics.

### Azure CLI

```
az monitor metrics list --resource $CONTAINER_GROUP --metric CPUUsage --output
table
```

### Output

Timestamp	Name	Average
2020-12-17 23:34:00	CPU Usage	
...		
2020-12-18 00:25:00	CPU Usage	
2020-12-18 00:26:00	CPU Usage	0.4
2020-12-18 00:27:00	CPU Usage	0.0

Change the value of the `--metric` parameter in the command to get other [supported metrics](#).

For example, use the following command to get **memory** usage metrics.

#### Azure CLI

```
az monitor metrics list --resource $CONTAINER_GROUP --metric MemoryUsage --output table
```

#### Output

Timestamp	Name	Average
2019-04-23 22:59:00	Memory Usage	
2019-04-23 23:00:00	Memory Usage	
2019-04-23 23:01:00	Memory Usage	0.0
2019-04-23 23:02:00	Memory Usage	8859648.0
2019-04-23 23:03:00	Memory Usage	9181184.0
2019-04-23 23:04:00	Memory Usage	9580544.0
2019-04-23 23:05:00	Memory Usage	10280960.0
2019-04-23 23:06:00	Memory Usage	7815168.0
2019-04-23 23:07:00	Memory Usage	7739392.0
2019-04-23 23:08:00	Memory Usage	8212480.0
2019-04-23 23:09:00	Memory Usage	8159232.0
2019-04-23 23:10:00	Memory Usage	8093696.0

For a multi-container group, the `containerName` dimension can be added to return metrics per container.

#### Azure CLI

```
az monitor metrics list --resource $CONTAINER_GROUP --metric MemoryUsage --dimension containerName --output table
```

#### Output

Timestamp	Name	Containername	Average
2019-04-23 22:59:00	Memory Usage	aci-tutorial-app	
2019-04-23 23:00:00	Memory Usage	aci-tutorial-app	
2019-04-23 23:01:00	Memory Usage	aci-tutorial-app	0.0
2019-04-23 23:02:00	Memory Usage	aci-tutorial-app	16834560.0
2019-04-23 23:03:00	Memory Usage	aci-tutorial-app	17534976.0
2019-04-23 23:04:00	Memory Usage	aci-tutorial-app	18329600.0
2019-04-23 23:05:00	Memory Usage	aci-tutorial-app	19742720.0
2019-04-23 23:06:00	Memory Usage	aci-tutorial-app	14786560.0
2019-04-23 23:07:00	Memory Usage	aci-tutorial-app	14651392.0
2019-04-23 23:08:00	Memory Usage	aci-tutorial-app	15470592.0
2019-04-23 23:09:00	Memory Usage	aci-tutorial-app	15450112.0
2019-04-23 23:10:00	Memory Usage	aci-tutorial-app	15339520.0

2019-04-23 22:59:00	Memory Usage	aci-tutorial-sidecar
2019-04-23 23:00:00	Memory Usage	aci-tutorial-sidecar
2019-04-23 23:01:00	Memory Usage	aci-tutorial-sidecar 0.0
2019-04-23 23:02:00	Memory Usage	aci-tutorial-sidecar 884736.0
2019-04-23 23:03:00	Memory Usage	aci-tutorial-sidecar 827392.0
2019-04-23 23:04:00	Memory Usage	aci-tutorial-sidecar 831488.0
2019-04-23 23:05:00	Memory Usage	aci-tutorial-sidecar 819200.0
2019-04-23 23:06:00	Memory Usage	aci-tutorial-sidecar 843776.0
2019-04-23 23:07:00	Memory Usage	aci-tutorial-sidecar 827392.0
2019-04-23 23:08:00	Memory Usage	aci-tutorial-sidecar 954368.0
2019-04-23 23:09:00	Memory Usage	aci-tutorial-sidecar 868352.0
2019-04-23 23:10:00	Memory Usage	aci-tutorial-sidecar 847872.0

## Azure Monitor resource logs

Resource logs provide insight into operations that were done by an Azure resource. Logs are generated automatically, but you must route them to Azure Monitor logs to save or query them. Logs are organized in categories. A given namespace might have multiple resource log categories.

**Collection:** Resource logs aren't collected and stored until you create a *diagnostic setting* and route the logs to one or more locations. When you create a diagnostic setting, you specify which categories of logs to collect. There are multiple ways to create and maintain diagnostic settings, including the Azure portal, programmatically, and through Azure Policy.

**Routing:** The suggested default is to route resource logs to Azure Monitor Logs so you can query them with other log data. Other locations such as Azure Storage, Azure Event Hubs, and certain Microsoft monitoring partners are also available. For more information, see [Azure resource logs](#) and [Resource log destinations](#).

For detailed information about collecting, storing, and routing resource logs, see [Diagnostic settings in Azure Monitor](#).

For a list of all available resource log categories in Azure Monitor, see [Supported resource logs in Azure Monitor](#).

All resource logs in Azure Monitor have the same header fields, followed by service-specific fields. The common schema is outlined in [Azure Monitor resource log schema](#).

- For more information about how to get log data for Container Instances, see [Retrieve container logs and events in Azure Container Instances](#).
- For the available resource log categories, associated Log Analytics tables, and the log schemas for Container Instances, see [Container Instances monitoring data reference](#).

# Azure activity log

The activity log contains subscription-level events that track operations for each Azure resource as seen from outside that resource; for example, creating a new resource or starting a virtual machine.

**Collection:** Activity log events are automatically generated and collected in a separate store for viewing in the Azure portal.

**Routing:** You can send activity log data to Azure Monitor Logs so you can analyze it alongside other log data. Other locations such as Azure Storage, Azure Event Hubs, and certain Microsoft monitoring partners are also available. For more information on how to route the activity log, see [Overview of the Azure activity log](#).

## Analyze monitoring data

There are many tools for analyzing monitoring data.

## Azure Monitor tools

Azure Monitor supports the following basic tools:

- [Metrics explorer](#), a tool in the Azure portal that allows you to view and analyze metrics for Azure resources. For more information, see [Analyze metrics with Azure Monitor metrics explorer](#).
- [Log Analytics](#), a tool in the Azure portal that allows you to query and analyze log data by using the [Kusto query language \(KQL\)](#). For more information, see [Get started with log queries in Azure Monitor](#).
- The [activity log](#), which has a user interface in the Azure portal for viewing and basic searches. To do more in-depth analysis, you have to route the data to Azure Monitor logs and run more complex queries in Log Analytics.

Tools that allow more complex visualization include:

- [Dashboards](#) that let you combine different kinds of data into a single pane in the Azure portal.
- [Workbooks](#), customizable reports that you can create in the Azure portal. Workbooks can include text, metrics, and log queries.
- [Grafana](#), an open platform tool that excels in operational dashboards. You can use Grafana to create dashboards that include data from multiple sources other than Azure Monitor.

- Power BI, a business analytics service that provides interactive visualizations across various data sources. You can configure Power BI to automatically import log data from Azure Monitor to take advantage of these visualizations.

## Azure Monitor export tools

You can get data out of Azure Monitor into other tools by using the following methods:

- **Metrics:** Use the [REST API for metrics](#) to extract metric data from the Azure Monitor metrics database. The API supports filter expressions to refine the data retrieved. For more information, see [Azure Monitor REST API reference](#).
- **Logs:** Use the REST API or the [associated client libraries](#).
- Another option is the [workspace data export](#).

To get started with the REST API for Azure Monitor, see [Azure monitoring REST API walkthrough](#).

## Analyze Container Instances logs

You can use Log Analytics to analyze and query container instance logs, and you can also enable diagnostic settings as a preview feature in the Azure portal. Log Analytics and diagnostic settings have slightly different table schemas to use for queries. Once you enable diagnostic settings, you can use either or both schemas at the same time.

For detailed information and instructions for querying logs, see [Container group and instance logging with Azure Monitor logs](#). For the Azure Monitor logs table schemas for Container Instances, see [Azure Monitor Logs tables](#).

## Kusto queries

You can analyze monitoring data in the Azure Monitor Logs / Log Analytics store by using the Kusto query language (KQL).

### Important

When you select **Logs** from the service's menu in the portal, Log Analytics opens with the query scope set to the current service. This scope means that log queries will only include data from that type of resource. If you want to run a query that includes data from other

Azure services, select **Logs** from the Azure Monitor menu. See [Log query scope and time range in Azure Monitor Log Analytics](#) for details.

For a list of common queries for any service, see the [Log Analytics queries interface](#).

The following query examples use the legacy Log Analytics log tables. The basic structure of a query is the source table, `ContainerInstanceLog_CL` or `ContainerEvent_CL`, followed by a series of operators separated by the pipe character (`|`). You can chain several operators to refine the results and perform advanced functions.

In the newer table schema for diagnostic settings, the table names appear without `_CL`, and some columns are different. If you have diagnostic settings enabled, you can use either or both tables.

To see example query results, paste the following query into the query text box, and select the **Run** button to execute the query. This query displays all log entries whose "Message" field contains the word "warn":

```
Kusto

ContainerInstanceLog_CL
| where Message contains "warn"
```

More complex queries are also supported. For example, this query displays only those log entries for the "mycontainergroup001" container group generated within the last hour:

```
Kusto

ContainerInstanceLog_CL
| where (ContainerGroup_s == "mycontainergroup001")
| where (TimeGenerated > ago(1h))
```

## Alerts

Azure Monitor alerts proactively notify you when specific conditions are found in your monitoring data. Alerts allow you to identify and address issues in your system before your customers notice them. For more information, see [Azure Monitor alerts](#).

There are many sources of common alerts for Azure resources. For examples of common alerts for Azure resources, see [Sample log alert queries](#). The [Azure Monitor Baseline Alerts \(AMBA\)](#) site provides a semi-automated method of implementing important platform metric alerts,

dashboards, and guidelines. The site applies to a continually expanding subset of Azure services, including all services that are part of the Azure Landing Zone (ALZ).

The common alert schema standardizes the consumption of Azure Monitor alert notifications. For more information, see [Common alert schema](#).

## Types of alerts

You can alert on any metric or log data source in the Azure Monitor data platform. There are many different types of alerts depending on the services you're monitoring and the monitoring data you're collecting. Different types of alerts have various benefits and drawbacks. For more information, see [Choose the right monitoring alert type](#).

The following list describes the types of Azure Monitor alerts you can create:

- [Metric alerts](#) evaluate resource metrics at regular intervals. Metrics can be platform metrics, custom metrics, logs from Azure Monitor converted to metrics, or Application Insights metrics. Metric alerts can also apply multiple conditions and dynamic thresholds.
- [Log alerts](#) allow users to use a Log Analytics query to evaluate resource logs at a predefined frequency.
- [Activity log alerts](#) trigger when a new activity log event occurs that matches defined conditions. Resource Health alerts and Service Health alerts are activity log alerts that report on your service and resource health.

Some Azure services also support [smart detection alerts](#), [Prometheus alerts](#), or [recommended alert rules](#).

For some services, you can monitor at scale by applying the same metric alert rule to multiple resources of the same type that exist in the same Azure region. Individual notifications are sent for each monitored resource. For supported Azure services and clouds, see [Monitor multiple resources with one alert rule](#).

 **Note**

If you're creating or running an application that runs on your service, [Azure Monitor application insights](#) might offer more types of alerts.

## Container Instances alert rules

The following table lists common and recommended alert rules for Container Instances.

Alert type	Condition	Description
Metrics	vCPU usage, memory usage, or network input and output utilization exceeding a certain threshold	Depending on the function of the container, setting an alert for when the metric exceeds an expected threshold may be useful.
Activity logs	Container Instances operations like create, update, and delete	See the <a href="#">Container Instances monitoring data reference</a> for a list of activities you can track.
Log alerts	<code>stdout</code> and <code>stderr</code> outputs in the logs	Use custom log search to set alerts for specific outputs that appear in logs.

## Advisor recommendations

For some services, if critical conditions or imminent changes occur during resource operations, an alert displays on the service **Overview** page in the portal. You can find more information and recommended fixes for the alert in **Advisor recommendations** under **Monitoring** in the left menu. During normal operations, no advisor recommendations display.

For more information on Azure Advisor, see [Azure Advisor overview](#).

## Related content

- See [Container Instances monitoring data reference](#) for a reference of the metrics, logs, and other important values created for Container Instances.
- See [Monitoring Azure resources with Azure Monitor](#) for general details on monitoring Azure resources.

# View container group logs in a log analytics workspace via Azure portal

Article • 09/09/2024

Log Analytics workspaces provide a centralized location for storing and querying log data not only from Azure resources, but also on-premises resources and resources in other clouds. Azure Container Instances includes built-in support for sending logs and event data to Azure Monitor logs. While creating a container instance within Azure portal, you are able to optionally set up Log Analytics. Once your container group is created with Log Analytics enabled, you can view logs within the page for your Azure Container Instance by navigating to “Logs” on the left panel. If you didn’t set up Log Analytics initially, you can also navigate to “Logs” to set it up after you create your container instance also.

The following sections describe how to set up logging and how to view logs for Azure Container Instances.

## Create Azure container instance

This section shows how to create a container instance with Log Analytics.

### Note

Container instance logs are enabled by default. You can also set up Log Analytics later after you create your container instance by deselecting “Enable container instance logs”. The View Logs section has details on how to do this.

1. Navigate to the create container instance page in Azure portal.
2. Fill out the relevant information in the Basic and Networking tabs.
3. In the Monitoring tab, enable container instance logs.

If you have a Log Analytics workspace, you can enter the information for it.

Home > Container instances >

Create container instance ...

Basics Networking **Monitoring** Advanced Tags Review + create

Configure monitoring options for your container instance.

**Insights**

Enable container instance logs

Subscription for your workspace \*

Log Analytics workspace \*  [Create new](#)

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

If you don't have a Log Analytics workspace, you can select "Create New" to create a new one.

Home > Container instances >

Create container instance ...

Basics Networking **Monitoring** Advanced Tags Review + create

Configure monitoring options for your container instance.

**Insights**

Enable container instance logs

Subscription for your workspace \*

Log Analytics workspace \*  [Create new](#)

**Create new workspace**

Region \*

Resource group

Log Analytics workspace \*

[Create](#)

Lastly, fill out the following tabs then go to the "Review + Create" tab and click "Create" to create your container instance.

## View logs

This section outlines how to view logs for your container instance that you created.

1. Once your container instance is already created, click "Logs" on the left panel.

The screenshot shows the Azure Container Instances blade for the 'aci-for-scenario-tests-with-init-containers' resource group. It displays a table of container instances with columns for Name, Image, State, Previous state, Start time, and Restart count. Three containers are listed: 'juliet' (Running, Terminated at 2024-08-22T21:04:55.57Z, 13133 restarts), 'hamlet' (Running, Terminated at 2024-08-22T21:04:55.579Z, 13127 restarts), and 'init1 (init)' (Terminated, Started at 2024-08-21T21:50:05.505Z, 0 restarts). On the left, the 'Logs' section is selected, showing a preview of log entries. A large search bar and a magnifying glass icon are visible on the right.

2. You'll see the Queries Hub where you can find queries to run in Log Analytics. If these aren't needed, then you can close this screen.

After closing the previous screen, if you created the ACI with the log-generating image, you should see data for your container group here.

The screenshot shows the Azure Log Analytics blade for the 'sktestws' workspace. It displays a table of log entries with columns for TimeGenerated (UTC), ContainerGroup\_s, ContainerGroupInstanceId\_g, ContainerID\_s, ContainerName\_s, Count\_s, Location\_s, and Message. The table lists several events related to container operations, such as pulling images and stopping containers. A search bar, a 'New Query' button, and a 'Results' tab are visible at the top. A large search bar and a magnifying glass icon are visible on the right.

3. If you created a container instance without Log Analytics enabled, then you'll see this screen and be able to enable it.

You can enter an already existing Log Analytics workspace:

The screenshot shows the Azure portal interface for managing container instances. In the left sidebar, under the 'Logs' section, there is a 'Configure' button. A large circular icon with a magnifying glass and a plus sign is located in the bottom right corner of the page.

You can also view logs for your container instances by navigating to “Containers” on the left pane, then selecting “Logs” in the middle pane.

If your container instance is in running state, you’ll be able to see the logs:

The screenshot shows the Azure portal interface for managing containers. The 'Logs' tab is selected in the container instance details. The log viewer displays several log entries from the container's stdout stream.

Name	Image	State	Previous state	Start time	Restart count
juliet	mcr.microsoft.com/azuredocs/aci-wordcount	Running	Terminated	2024-08-22T21:04:05.572Z	13133
hamlet	mcr.microsoft.com/azuredocs/aci-wordcount	Running	Terminated	2024-08-22T21:04:05.579Z	13127
int1 (init)	mcr.microsoft.com/aks/e2e/library-busyboxmaster.21...	Terminated	-	2024-08-22T21:50:05.505Z	0

Now you’re ready to run your container group with Log Analytics.

## Next Steps

### Azure Monitor logs

For more information about querying logs and configuring alerts in Azure Monitor logs, see:

- Understanding log searches in Azure Monitor logs

- Unified alerts in Azure Monitor

## Monitor container CPU and memory

For information about monitoring container instance CPU and memory resources, see:

- [Monitor Azure Container Instances](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# Monitor Azure Container Instances

Article • 08/29/2024

This article describes:

- The types of monitoring data you can collect for this service.
- Ways to analyze that data.

## ⓘ Note

If you're already familiar with this service and/or Azure Monitor and just want to know how to analyze monitoring data, see the [Analyze](#) section near the end of this article.

When you have critical applications and business processes that rely on Azure resources, you need to monitor and get alerts for your system. The Azure Monitor service collects and aggregates metrics and logs from every component of your system. Azure Monitor provides you with a view of availability, performance, and resilience, and notifies you of issues. You can use the Azure portal, PowerShell, Azure CLI, REST API, or client libraries to set up and view monitoring data.

- For more information on Azure Monitor, see the [Azure Monitor overview](#).
- For more information on how to monitor Azure resources in general, see [Monitor Azure resources with Azure Monitor](#).

## Resource types

Azure uses the concept of resource types and IDs to identify everything in a subscription. Resource types are also part of the resource IDs for every resource running in Azure. For example, one resource type for a virtual machine is `Microsoft.Compute/virtualMachines`. For a list of services and their associated resource types, see [Resource providers](#).

Azure Monitor similarly organizes core monitoring data into metrics and logs based on resource types, also called *namespaces*. Different metrics and logs are available for different resource types. Your service might be associated with more than one resource type.

For more information about the resource types for Azure Container Instances, see [Container Instances monitoring data reference](#).

## Data storage

For Azure Monitor:

- Metrics data is stored in the Azure Monitor metrics database.
- Log data is stored in the Azure Monitor logs store. Log Analytics is a tool in the Azure portal that can query this store.
- The Azure activity log is a separate store with its own interface in the Azure portal.

You can optionally route metric and activity log data to the Azure Monitor logs store. You can then use Log Analytics to query the data and correlate it with other log data.

Many services can use diagnostic settings to send metric and log data to other storage locations outside Azure Monitor. Examples include Azure Storage, [hosted partner systems](#), and [non-Azure partner systems](#), by using [Event Hubs](#).

For detailed information on how Azure Monitor stores data, see [Azure Monitor data platform](#).

## Azure Monitor platform metrics

Azure Monitor provides platform metrics for most services. These metrics are:

- Individually defined for each namespace.
- Stored in the Azure Monitor time-series metrics database.
- Lightweight and capable of supporting near real-time alerting.
- Used to track the performance of a resource over time.

**Collection:** Azure Monitor collects platform metrics automatically. No configuration is required.

**Routing:** You can also route some platform metrics to Azure Monitor Logs / Log Analytics so you can query them with other log data. Check the **DS export** setting for each metric to see if you can use a diagnostic setting to route the metric to Azure Monitor Logs / Log Analytics.

- For more information, see the [Metrics diagnostic setting](#).
- To configure diagnostic settings for a service, see [Create diagnostic settings in Azure Monitor](#).

For a list of all metrics it's possible to gather for all resources in Azure Monitor, see [Supported metrics in Azure Monitor](#).

For a list of available metrics for Container Instances, see [Container Instances monitoring data reference](#). These metrics are available for a container group and individual containers. By default, the metrics are aggregated as averages.

All metrics for Container Instances are in the namespace **Container group standard metrics**. In a container group with multiple containers, you can filter on the **containerName** dimension to acquire metrics from a specific container within the group. Containers generate similar data as other Azure resources, but they require a containerized agent to collect required data.

# Get metrics

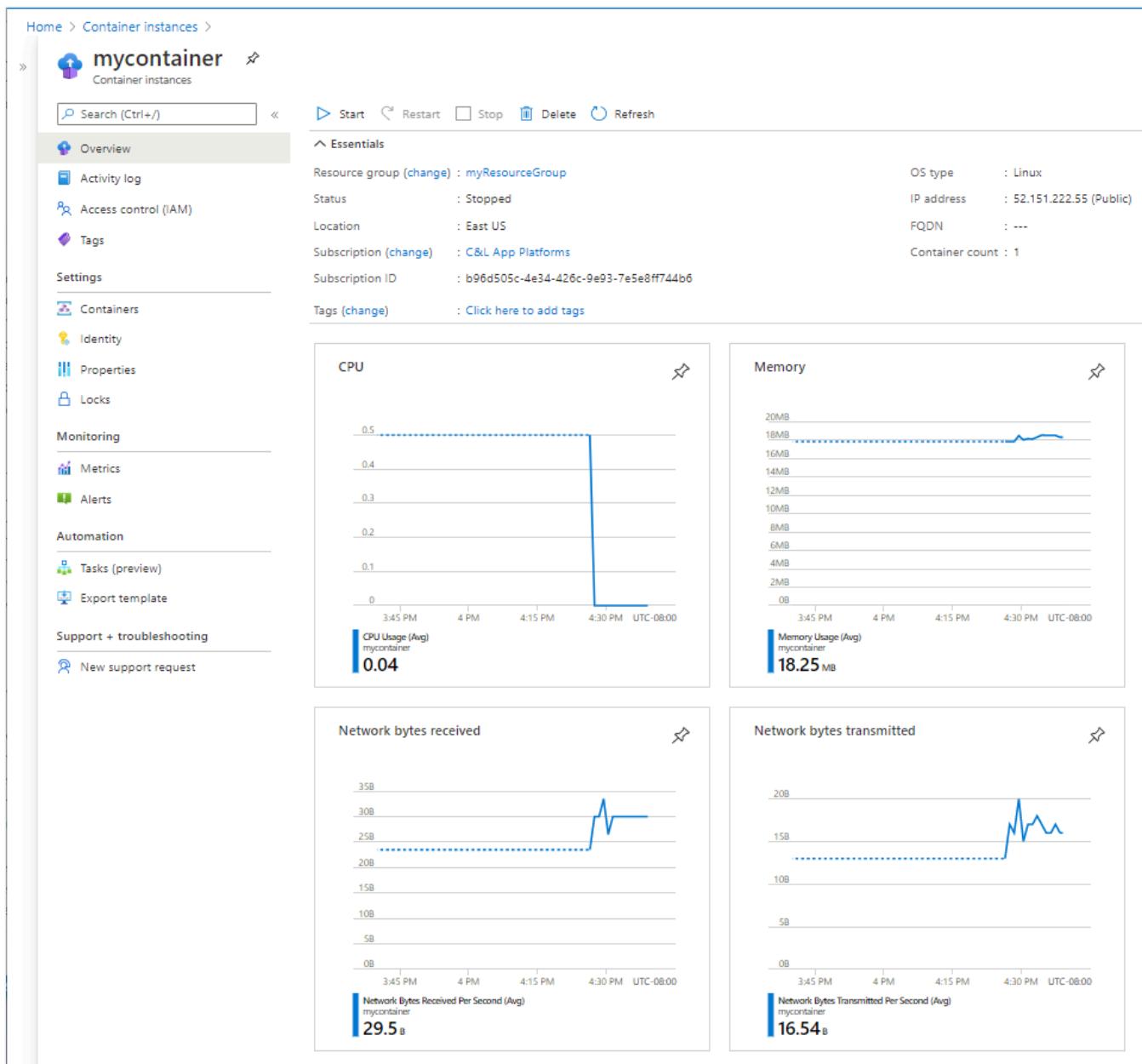
You can gather Azure Monitor metrics for container instances using either the Azure portal or Azure CLI.

## Important

Azure Monitor metrics in Azure Container Instances are currently in preview. At this time, Azure Monitor metrics are only available for Linux containers. Previews are made available to you on the condition that you agree to the [supplemental terms of use](#). Some aspects of this feature may change prior to general availability (GA).

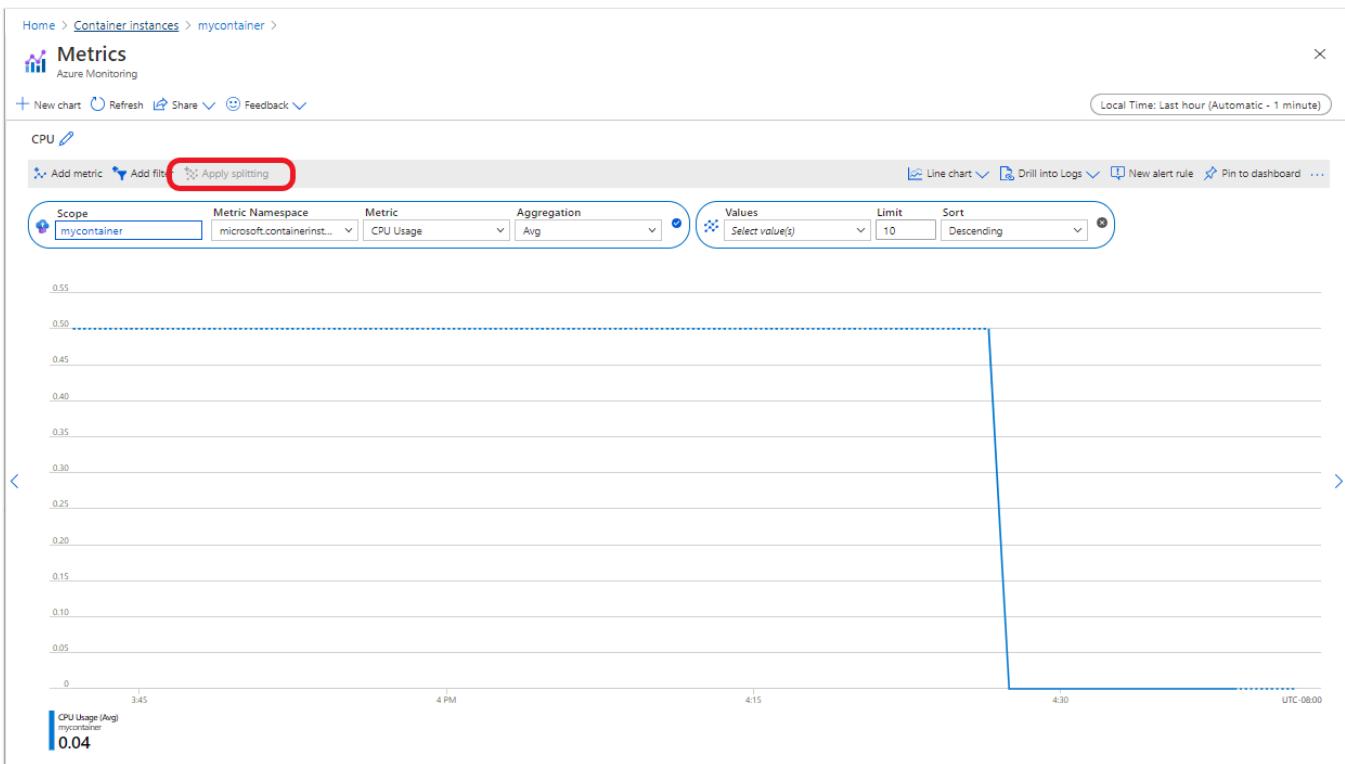
## Use the Azure portal

When a container group is created, Azure Monitor data is available in the Azure portal. To see metrics for a container group, go to the **Overview** page for the container group. Here you can see precreated charts for each of the available metrics.



In a container group that contains multiple containers, use a **dimension** to display metrics by container. To create a chart with individual container metrics, perform the following steps:

1. In the **Overview** page, select one of the metric charts, such as **CPU**.
2. Select the **Apply splitting** button, and select **Container Name**.



## Use Azure CLI

Metrics for container instances can also be gathered using the Azure CLI. First, get the ID of the container group using the following command. Replace <resource-group> with your resource group name and <container-group> with the name of your container group.

### Azure CLI

```
CONTAINER_GROUP=$(az container show --resource-group <resource-group> --name
<container-group> --query id --output tsv)
```

Use the following command to get CPU usage metrics.

### Azure CLI

```
az monitor metrics list --resource $CONTAINER_GROUP --metric CPUUsage --output
table
```

### Output

Timestamp	Name	Average
2020-12-17 23:34:00	CPU Usage	
...		
2020-12-18 00:25:00	CPU Usage	
2020-12-18 00:26:00	CPU Usage	0.4
2020-12-18 00:27:00	CPU Usage	0.0

Change the value of the `--metric` parameter in the command to get other [supported metrics](#).

For example, use the following command to get **memory** usage metrics.

#### Azure CLI

```
az monitor metrics list --resource $CONTAINER_GROUP --metric MemoryUsage --output table
```

#### Output

Timestamp	Name	Average
2019-04-23 22:59:00	Memory Usage	
2019-04-23 23:00:00	Memory Usage	
2019-04-23 23:01:00	Memory Usage	0.0
2019-04-23 23:02:00	Memory Usage	8859648.0
2019-04-23 23:03:00	Memory Usage	9181184.0
2019-04-23 23:04:00	Memory Usage	9580544.0
2019-04-23 23:05:00	Memory Usage	10280960.0
2019-04-23 23:06:00	Memory Usage	7815168.0
2019-04-23 23:07:00	Memory Usage	7739392.0
2019-04-23 23:08:00	Memory Usage	8212480.0
2019-04-23 23:09:00	Memory Usage	8159232.0
2019-04-23 23:10:00	Memory Usage	8093696.0

For a multi-container group, the `containerName` dimension can be added to return metrics per container.

#### Azure CLI

```
az monitor metrics list --resource $CONTAINER_GROUP --metric MemoryUsage --dimension containerName --output table
```

#### Output

Timestamp	Name	Containername	Average
2019-04-23 22:59:00	Memory Usage	aci-tutorial-app	
2019-04-23 23:00:00	Memory Usage	aci-tutorial-app	
2019-04-23 23:01:00	Memory Usage	aci-tutorial-app	0.0
2019-04-23 23:02:00	Memory Usage	aci-tutorial-app	16834560.0
2019-04-23 23:03:00	Memory Usage	aci-tutorial-app	17534976.0
2019-04-23 23:04:00	Memory Usage	aci-tutorial-app	18329600.0
2019-04-23 23:05:00	Memory Usage	aci-tutorial-app	19742720.0
2019-04-23 23:06:00	Memory Usage	aci-tutorial-app	14786560.0
2019-04-23 23:07:00	Memory Usage	aci-tutorial-app	14651392.0
2019-04-23 23:08:00	Memory Usage	aci-tutorial-app	15470592.0
2019-04-23 23:09:00	Memory Usage	aci-tutorial-app	15450112.0
2019-04-23 23:10:00	Memory Usage	aci-tutorial-app	15339520.0

2019-04-23 22:59:00	Memory Usage	aci-tutorial-sidecar
2019-04-23 23:00:00	Memory Usage	aci-tutorial-sidecar
2019-04-23 23:01:00	Memory Usage	aci-tutorial-sidecar 0.0
2019-04-23 23:02:00	Memory Usage	aci-tutorial-sidecar 884736.0
2019-04-23 23:03:00	Memory Usage	aci-tutorial-sidecar 827392.0
2019-04-23 23:04:00	Memory Usage	aci-tutorial-sidecar 831488.0
2019-04-23 23:05:00	Memory Usage	aci-tutorial-sidecar 819200.0
2019-04-23 23:06:00	Memory Usage	aci-tutorial-sidecar 843776.0
2019-04-23 23:07:00	Memory Usage	aci-tutorial-sidecar 827392.0
2019-04-23 23:08:00	Memory Usage	aci-tutorial-sidecar 954368.0
2019-04-23 23:09:00	Memory Usage	aci-tutorial-sidecar 868352.0
2019-04-23 23:10:00	Memory Usage	aci-tutorial-sidecar 847872.0

## Azure Monitor resource logs

Resource logs provide insight into operations that were done by an Azure resource. Logs are generated automatically, but you must route them to Azure Monitor logs to save or query them. Logs are organized in categories. A given namespace might have multiple resource log categories.

**Collection:** Resource logs aren't collected and stored until you create a *diagnostic setting* and route the logs to one or more locations. When you create a diagnostic setting, you specify which categories of logs to collect. There are multiple ways to create and maintain diagnostic settings, including the Azure portal, programmatically, and through Azure Policy.

**Routing:** The suggested default is to route resource logs to Azure Monitor Logs so you can query them with other log data. Other locations such as Azure Storage, Azure Event Hubs, and certain Microsoft monitoring partners are also available. For more information, see [Azure resource logs](#) and [Resource log destinations](#).

For detailed information about collecting, storing, and routing resource logs, see [Diagnostic settings in Azure Monitor](#).

For a list of all available resource log categories in Azure Monitor, see [Supported resource logs in Azure Monitor](#).

All resource logs in Azure Monitor have the same header fields, followed by service-specific fields. The common schema is outlined in [Azure Monitor resource log schema](#).

- For more information about how to get log data for Container Instances, see [Retrieve container logs and events in Azure Container Instances](#).
- For the available resource log categories, associated Log Analytics tables, and the log schemas for Container Instances, see [Container Instances monitoring data reference](#).

# Azure activity log

The activity log contains subscription-level events that track operations for each Azure resource as seen from outside that resource; for example, creating a new resource or starting a virtual machine.

**Collection:** Activity log events are automatically generated and collected in a separate store for viewing in the Azure portal.

**Routing:** You can send activity log data to Azure Monitor Logs so you can analyze it alongside other log data. Other locations such as Azure Storage, Azure Event Hubs, and certain Microsoft monitoring partners are also available. For more information on how to route the activity log, see [Overview of the Azure activity log](#).

## Analyze monitoring data

There are many tools for analyzing monitoring data.

## Azure Monitor tools

Azure Monitor supports the following basic tools:

- [Metrics explorer](#), a tool in the Azure portal that allows you to view and analyze metrics for Azure resources. For more information, see [Analyze metrics with Azure Monitor metrics explorer](#).
- [Log Analytics](#), a tool in the Azure portal that allows you to query and analyze log data by using the [Kusto query language \(KQL\)](#). For more information, see [Get started with log queries in Azure Monitor](#).
- The [activity log](#), which has a user interface in the Azure portal for viewing and basic searches. To do more in-depth analysis, you have to route the data to Azure Monitor logs and run more complex queries in Log Analytics.

Tools that allow more complex visualization include:

- [Dashboards](#) that let you combine different kinds of data into a single pane in the Azure portal.
- [Workbooks](#), customizable reports that you can create in the Azure portal. Workbooks can include text, metrics, and log queries.
- [Grafana](#), an open platform tool that excels in operational dashboards. You can use Grafana to create dashboards that include data from multiple sources other than Azure Monitor.

- Power BI, a business analytics service that provides interactive visualizations across various data sources. You can configure Power BI to automatically import log data from Azure Monitor to take advantage of these visualizations.

## Azure Monitor export tools

You can get data out of Azure Monitor into other tools by using the following methods:

- **Metrics:** Use the [REST API for metrics](#) to extract metric data from the Azure Monitor metrics database. The API supports filter expressions to refine the data retrieved. For more information, see [Azure Monitor REST API reference](#).
- **Logs:** Use the REST API or the [associated client libraries](#).
- Another option is the [workspace data export](#).

To get started with the REST API for Azure Monitor, see [Azure monitoring REST API walkthrough](#).

## Analyze Container Instances logs

You can use Log Analytics to analyze and query container instance logs, and you can also enable diagnostic settings as a preview feature in the Azure portal. Log Analytics and diagnostic settings have slightly different table schemas to use for queries. Once you enable diagnostic settings, you can use either or both schemas at the same time.

For detailed information and instructions for querying logs, see [Container group and instance logging with Azure Monitor logs](#). For the Azure Monitor logs table schemas for Container Instances, see [Azure Monitor Logs tables](#).

## Kusto queries

You can analyze monitoring data in the Azure Monitor Logs / Log Analytics store by using the Kusto query language (KQL).

### Important

When you select **Logs** from the service's menu in the portal, Log Analytics opens with the query scope set to the current service. This scope means that log queries will only include data from that type of resource. If you want to run a query that includes data from other

Azure services, select **Logs** from the Azure Monitor menu. See [Log query scope and time range in Azure Monitor Log Analytics](#) for details.

For a list of common queries for any service, see the [Log Analytics queries interface](#).

The following query examples use the legacy Log Analytics log tables. The basic structure of a query is the source table, `ContainerInstanceLog_CL` or `ContainerEvent_CL`, followed by a series of operators separated by the pipe character (`|`). You can chain several operators to refine the results and perform advanced functions.

In the newer table schema for diagnostic settings, the table names appear without `_CL`, and some columns are different. If you have diagnostic settings enabled, you can use either or both tables.

To see example query results, paste the following query into the query text box, and select the **Run** button to execute the query. This query displays all log entries whose "Message" field contains the word "warn":

```
Kusto

ContainerInstanceLog_CL
| where Message contains "warn"
```

More complex queries are also supported. For example, this query displays only those log entries for the "mycontainergroup001" container group generated within the last hour:

```
Kusto

ContainerInstanceLog_CL
| where (ContainerGroup_s == "mycontainergroup001")
| where (TimeGenerated > ago(1h))
```

## Alerts

Azure Monitor alerts proactively notify you when specific conditions are found in your monitoring data. Alerts allow you to identify and address issues in your system before your customers notice them. For more information, see [Azure Monitor alerts](#).

There are many sources of common alerts for Azure resources. For examples of common alerts for Azure resources, see [Sample log alert queries](#). The [Azure Monitor Baseline Alerts \(AMBA\)](#) site provides a semi-automated method of implementing important platform metric alerts,

dashboards, and guidelines. The site applies to a continually expanding subset of Azure services, including all services that are part of the Azure Landing Zone (ALZ).

The common alert schema standardizes the consumption of Azure Monitor alert notifications. For more information, see [Common alert schema](#).

## Types of alerts

You can alert on any metric or log data source in the Azure Monitor data platform. There are many different types of alerts depending on the services you're monitoring and the monitoring data you're collecting. Different types of alerts have various benefits and drawbacks. For more information, see [Choose the right monitoring alert type](#).

The following list describes the types of Azure Monitor alerts you can create:

- [Metric alerts](#) evaluate resource metrics at regular intervals. Metrics can be platform metrics, custom metrics, logs from Azure Monitor converted to metrics, or Application Insights metrics. Metric alerts can also apply multiple conditions and dynamic thresholds.
- [Log alerts](#) allow users to use a Log Analytics query to evaluate resource logs at a predefined frequency.
- [Activity log alerts](#) trigger when a new activity log event occurs that matches defined conditions. Resource Health alerts and Service Health alerts are activity log alerts that report on your service and resource health.

Some Azure services also support [smart detection alerts](#), [Prometheus alerts](#), or [recommended alert rules](#).

For some services, you can monitor at scale by applying the same metric alert rule to multiple resources of the same type that exist in the same Azure region. Individual notifications are sent for each monitored resource. For supported Azure services and clouds, see [Monitor multiple resources with one alert rule](#).

 **Note**

If you're creating or running an application that runs on your service, [Azure Monitor application insights](#) might offer more types of alerts.

## Container Instances alert rules

The following table lists common and recommended alert rules for Container Instances.

Alert type	Condition	Description
Metrics	vCPU usage, memory usage, or network input and output utilization exceeding a certain threshold	Depending on the function of the container, setting an alert for when the metric exceeds an expected threshold may be useful.
Activity logs	Container Instances operations like create, update, and delete	See the <a href="#">Container Instances monitoring data reference</a> for a list of activities you can track.
Log alerts	<code>stdout</code> and <code>stderr</code> outputs in the logs	Use custom log search to set alerts for specific outputs that appear in logs.

## Advisor recommendations

For some services, if critical conditions or imminent changes occur during resource operations, an alert displays on the service **Overview** page in the portal. You can find more information and recommended fixes for the alert in **Advisor recommendations** under **Monitoring** in the left menu. During normal operations, no advisor recommendations display.

For more information on Azure Advisor, see [Azure Advisor overview](#).

## Related content

- See [Container Instances monitoring data reference](#) for a reference of the metrics, logs, and other important values created for Container Instances.
- See [Monitoring Azure resources with Azure Monitor](#) for general details on monitoring Azure resources.

# Retrieve container logs and events in Azure Container Instances

Article • 08/29/2024

When you have a misbehaving container in Azure Container Instances, start by viewing its logs with [az container logs](#) and streaming its standard out and standard error with [az container attach](#). You can also view logs and events for container instances in the Azure portal, or send log and event data for container groups to [Azure Monitor logs](#).

## View logs

To view logs from your application code within a container, you can use the [az container logs](#) command.

The following sample output is log output from the example task-based container in [Set the command line in a container instance](#), after being provided an invalid URL using a command-line override:

Azure CLI

```
az container logs --resource-group myResourceGroup --name mycontainer
```

Output

```
Traceback (most recent call last):
 File "wordcount.py", line 11, in <module>
 urllib.request.urlretrieve (sys.argv[1], "foo.txt")
 File "/usr/local/lib/python3.6/urllib/request.py", line 248, in
urlretrieve
 with contextlib.closing(urlopen(url, data)) as fp:
 File "/usr/local/lib/python3.6/urllib/request.py", line 223, in urlopen
 return opener.open(url, data, timeout)
 File "/usr/local/lib/python3.6/urllib/request.py", line 532, in open
 response = meth(req, response)
 File "/usr/local/lib/python3.6/urllib/request.py", line 642, in
http_response
 'http', request, response, code, msg, hdrs)
 File "/usr/local/lib/python3.6/urllib/request.py", line 570, in error
 return self._call_chain(*args)
 File "/usr/local/lib/python3.6/urllib/request.py", line 504, in
_call_chain
 result = func(*args)
 File "/usr/local/lib/python3.6/urllib/request.py", line 650, in
http_error_default
```

```
 raise HTTPError(req.full_url, code, msg, hdrs, fp)
urllib.error.HTTPError: HTTP Error 404: Not Found
```

## Attach output streams

The [az container attach](#) command provides diagnostic information during container startup. Once the container starts, it streams STDOUT and STDERR to your local console.

For example, here's output from the task-based container in [Set the command line in a container instance](#), after being supplied a valid URL of a large text file to process:

Azure CLI

```
az container attach --resource-group myResourceGroup --name mycontainer
```

Output

```
Container 'mycontainer' is in state 'Unknown'...
Container 'mycontainer' is in state 'Waiting'...
Container 'mycontainer' is in state 'Running'...
(count: 1) (last timestamp: 2019-03-21 19:42:39+00:00) pulling image
"mcr.microsoft.com/azuredocs/aci-wordcount:latest"
Container 'mycontainer1' is in state 'Running'...
(count: 1) (last timestamp: 2019-03-21 19:42:39+00:00) pulling image
"mcr.microsoft.com/azuredocs/aci-wordcount:latest"
(count: 1) (last timestamp: 2019-03-21 19:42:52+00:00) Successfully pulled
image "mcr.microsoft.com/azuredocs/aci-wordcount:latest"
(count: 1) (last timestamp: 2019-03-21 19:42:55+00:00) Created container
(count: 1) (last timestamp: 2019-03-21 19:42:55+00:00) Started container
```

Start streaming logs:

```
[('the', 22979),
 ('I', 20003),
 ('and', 18373),
 ('to', 15651),
 ('of', 15558),
 ('a', 12500),
 ('you', 11818),
 ('my', 10651),
 ('in', 9707),
 ('is', 8195)]
```

## Get diagnostic events

If your container fails to deploy successfully, review the diagnostic information provided by the Azure Container Instances resource provider. To view the events for your

container, run the [az container show](#) command:

Azure CLI

```
az container show --resource-group myResourceGroup --name mycontainer
```

The output includes the core properties of your container, along with deployment events (shown here truncated):

JSON

```
{
 "containers": [
 {
 "command": null,
 "environmentVariables": [],
 "image": "mcr.microsoft.com/azuredocs/aci-helloworld",
 ...
 "events": [
 {
 "count": 1,
 "firstTimestamp": "2019-03-21T19:46:22+00:00",
 "lastTimestamp": "2019-03-21T19:46:22+00:00",
 "message": "pulling image \"mcr.microsoft.com/azuredocs/aci-helloworld\"",
 "name": "Pulling",
 "type": "Normal"
 },
 {
 "count": 1,
 "firstTimestamp": "2019-03-21T19:46:28+00:00",
 "lastTimestamp": "2019-03-21T19:46:28+00:00",
 "message": "Successfully pulled image \"mcr.microsoft.com/azuredocs/aci-helloworld\"",
 "name": "Pulled",
 "type": "Normal"
 },
 {
 "count": 1,
 "firstTimestamp": "2019-03-21T19:46:31+00:00",
 "lastTimestamp": "2019-03-21T19:46:31+00:00",
 "message": "Created container",
 "name": "Created",
 "type": "Normal"
 },
 {
 "count": 1,
 "firstTimestamp": "2019-03-21T19:46:31+00:00",
 "lastTimestamp": "2019-03-21T19:46:31+00:00",
 "message": "Started container",
 "name": "Started",
 "type": "Normal"
 }
]
 }
]
}
```

```
 },
],
 "previousState": null,
 "restartCount": 0
 },
 "name": "mycontainer",
 "ports": [
 {
 "port": 80,
 "protocol": null
 }
],
 ...
 }
],
...
}
```

## Next steps

Learn how to [troubleshoot common container and deployment issues](#) for Azure Container Instances.

Learn how to send log and event data for container groups to [Azure Monitor logs](#).

---

## Feedback

Was this page helpful?

 Yes

 No

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

# Container group and instance logging with Azure Monitor logs

Article • 08/29/2024

Log Analytics workspaces provide a centralized location for storing and querying log data not only from Azure resources, but also on-premises resources and resources in other clouds. Azure Container Instances includes built-in support for sending logs and event data to Azure Monitor logs.

To send container group log and event data to Azure Monitor logs, specify an existing Log Analytics workspace ID and workspace key when configuring a container group.

The following sections describe how to create a logging-enabled container group and how to query logs. You can also [update a container group](#) with a workspace ID and workspace key to enable logging.

## ⓘ Note

This article was recently updated to use the term Azure Monitor logs instead of Log Analytics. Log data is still stored in a Log Analytics workspace and is still collected and analyzed by the same Log Analytics service. We are updating the terminology to better reflect the role of [logs in Azure Monitor](#). See [Azure Monitor terminology changes](#) for details.

## Prerequisites

To enable logging in your container instances, you need the following prerequisites:

- [Log Analytics workspace](#)
- [Azure CLI \(or Cloud Shell\)](#)

## Get Log Analytics credentials

Azure Container Instances needs permission to send data to your Log Analytics workspace. To grant this permission and enable logging, you must provide the Log Analytics workspace ID and one of its keys (either primary or secondary) when you create the container group.

To obtain the log analytics workspace ID and primary key:

1. Navigate to your Log Analytics workspace in the Azure portal
2. Under **Settings**, select **Agents management**
3. Take note of:
  - **Workspace ID**
  - **Primary key**

## Create container group

Now that you have the log analytics workspace ID and primary key, you're ready to create a logging-enabled container group.

The following examples demonstrate two ways to create a container group that consists of a single [fluentd](#) container: Azure CLI, and Azure CLI with a YAML template. The fluentd container produces several lines of output in its default configuration. Because this output is sent to your Log Analytics workspace, it works well for demonstrating the viewing and querying of logs.

## Deploy with Azure CLI

To deploy with the Azure CLI, specify the `--log-analytics-workspace` and `--log-analytics-workspace-key` parameters in the `az container create` command. Replace the two workspace values with the values you obtained in the previous step (and update the resource group name) before running the following command.

### ⓘ Note

The following example pulls a public container image from Docker Hub. We recommend that you set up a pull secret to authenticate using a Docker Hub account instead of making an anonymous pull request. To improve reliability when working with public content, import and manage the image in a private Azure container registry. [Learn more about working with public images.](#)

### Azure CLI

```
az container create \
 --resource-group myResourceGroup \
 --name mycontainergroup001 \
 --image fluent/fluentd:v1.3-debian-1 \
 --log-analytics-workspace <WORKSPACE_ID> \
 --log-analytics-workspace-key <WORKSPACE_KEY>
```

# Deploy with YAML

Use this method if you prefer to deploy container groups with YAML. The following YAML defines a container group with a single container. Copy the YAML into a new file, then replace `LOG_ANALYTICS_WORKSPACE_ID` and `LOG_ANALYTICS_WORKSPACE_KEY` with the values you obtained in the previous step. Save the file as `deploy-aci.yaml`.

## ⓘ Note

The following example pulls a public container image from Docker Hub. We recommend that you set up a pull secret to authenticate using a Docker Hub account instead of making an anonymous pull request. To improve reliability when working with public content, import and manage the image in a private Azure container registry. [Learn more about working with public images.](#)

### YAML

```
apiVersion: 2019-12-01
location: eastus
name: mycontainergroup001
properties:
 containers:
 - name: mycontainer001
 properties:
 environmentVariables: []
 image: fluent/fluentd:v1.3-debian-1
 ports: []
 resources:
 requests:
 cpu: 1.0
 memoryInGB: 1.5
 osType: Linux
 restartPolicy: Always
 diagnostics:
 logAnalytics:
 workspaceId: LOG_ANALYTICS_WORKSPACE_ID
 workspaceKey: LOG_ANALYTICS_WORKSPACE_KEY
 tags: null
 type: Microsoft.ContainerInstance/containerGroups
```

Next, execute the following command to deploy the container group. Replace `myResourceGroup` with a resource group in your subscription (or first create a resource group named "myResourceGroup"):

Azure CLI

```
az container create --resource-group myResourceGroup --name mycontainergroup001 --file deploy-aci.yaml
```

You should receive a response from Azure containing deployment details shortly after issuing the command.

## View logs

After you deploy the container group, it can take several minutes (up to 10) for the first log entries to appear in the Azure portal.

To view the container group's logs in the `ContainerInstanceLog_CL` table:

1. Navigate to your Log Analytics workspace in the Azure portal
2. Under **General**, select **Logs**
3. Type the following query: `ContainerInstanceLog_CL | limit 50`
4. Select **Run**

You should see several results displayed by the query. If at first you don't see any results, wait a few minutes, then select the **Run** button to execute the query again. By default, log entries are displayed in **Table** format. You can then expand a row to see the contents of an individual log entry.

The screenshot shows the Azure Log Analytics workspace interface. The top navigation bar includes 'Home', 'Log Analytics workspaces', and 'aciworkspace - Logs'. The main area displays a query editor with the following content:

```
ContainerInstanceLog_CL
| limit 50
```

The results table shows log entries with the following columns: TimeGenerated, ContainerName\_s, ContainerImage\_s, Message, and ContainerID\_s. The data is as follows:

TimeGenerated	ContainerName_s	ContainerImage_s	Message	ContainerID_s
2019-08-02T08:56:33.688	mycontainergroup001	fluent/fluentd	2019-08-02 15:56:33 +0000 [info]: parsing config file is succeeded pa...	8b98e4b80ec5a84130044a5debb5e615
2019-08-02T08:56:33.701	mycontainergroup001	fluent/fluentd	2019-08-02 15:56:33 +0000 [warn]: [output_docker1] 'time_format' s...	8b98e4b80ec5a84130044a5debb5e615
2019-08-02T08:56:33.706	mycontainergroup001	fluent/fluentd	2019-08-02 15:56:33 +0000 [warn]: [output1] 'time_format' specified ...	8b98e4b80ec5a84130044a5debb5e615
2019-08-02T08:56:33.728	mycontainergroup001	fluent/fluentd	2019-08-02 15:56:33 +0000 [info]: using configuration file: <ROOT>	8b98e4b80ec5a84130044a5debb5e615
2019-08-02T08:56:33.728	mycontainergroup001	fluent/fluentd	<source>	8b98e4b80ec5a84130044a5debb5e615
2019-08-02T08:56:33.728	mycontainergroup001	fluent/fluentd	@type forward	8b98e4b80ec5a84130044a5debb5e615
2019-08-02T08:56:33.728	mycontainergroup001	fluent/fluentd	@id input1	8b98e4b80ec5a84130044a5debb5e615
2019-08-02T08:56:33.728	mycontainergroup001	fluent/fluentd	@label @mainstream	8b98e4b80ec5a84130044a5debb5e615

## View events

You can also view events for container instances in the Azure portal. Events include the time the instance is created and when it starts. To view the event data in the `ContainerEvent_CL` table:

1. Navigate to your Log Analytics workspace in the Azure portal

2. Under General, select Logs

3. Type the following query: ContainerEvent\_CL | limit 50

4. Select Run

You should see several results displayed by the query. If at first you don't see any results, wait a few minutes, then select the **Run** button to execute the query again. By default, entries are displayed in **Table** format. You can then expand a row to see the contents of an individual entry.

The screenshot shows the Azure Log Analytics workspace interface. In the top navigation bar, it says "Home > Log Analytics workspaces > aciworkspace - Logs". Below the navigation is a search bar with "aciworkspace - Logs" and a "New Query 1\*" button. To the right are links for "Help", "Settings", "Sample queries", "Query explorer", "Save", "Copy", "Export", "New alert rule", and "Pin to dashboard". The main area has tabs for "Schema", "Filter", and "Expl...". A query editor contains the text "ContainerEvent\_CL | limit 50". Below the editor, a message says "Completed. Showing results from the last 24 hours." A table view displays log entries with columns: TimeGenerated [Pacific Time...], ContainerGroup\_s, ContainerName\_s, and Message. The table shows four log entries from August 2, 2019, at 13:53:38.000. The "Message" column contains: "Created container", "Started container", "Successfully pulled image "fluent/fluentd"" and "pulling image "fluent/fluentd"".

## Query container logs

Azure Monitor logs includes an extensive [query language](#) for pulling information from potentially thousands of lines of log output.

The basic structure of a query is the source table (in this article, `ContainerInstanceLog_CL` or `ContainerEvent_CL`) followed by a series of operators separated by the pipe character (`|`). You can chain several operators to refine the results and perform advanced functions.

To see example query results, paste the following query into the query text box, and select the **Run** button to execute the query. This query displays all log entries whose "Message" field contains the word "warn":

```
query

ContainerInstanceLog_CL
| where Message contains "warn"
```

More complex queries are also supported. For example, this query displays only those log entries for the "mycontainergroup001" container group generated within the last hour:

```
query

ContainerEvent_CL
| where ContainerGroup_s == "mycontainergroup001"
| where TimeGenerated > ago(1h)
```

```
ContainerInstanceLog_CL
| where (ContainerGroup_s == "mycontainergroup001")
| where (TimeGenerated > ago(1h))
```

## Log schema

To learn about the Azure Monitor legacy schema for Log Analytics, see [Legacy Log Analytics tables](#).

## Using Diagnostic Settings

The newer tables require use of diagnostic settings to route information to Log Analytics. The diagnostic settings feature for Container Instances is in public preview, and it can be enabled through preview features options in the Azure portal.

To learn more, see [Azure Monitor Log Analytics tables](#).

## Next steps

### Azure Monitor logs

For more information about querying logs and configuring alerts in Azure Monitor logs, see:

- [Understanding log searches in Azure Monitor logs](#)
- [Unified alerts in Azure Monitor](#)

### Monitor container CPU and memory

For information about monitoring container instance CPU and memory resources, see:

- [Monitor Azure Container Instances](#).

---

## Feedback

Was this page helpful?



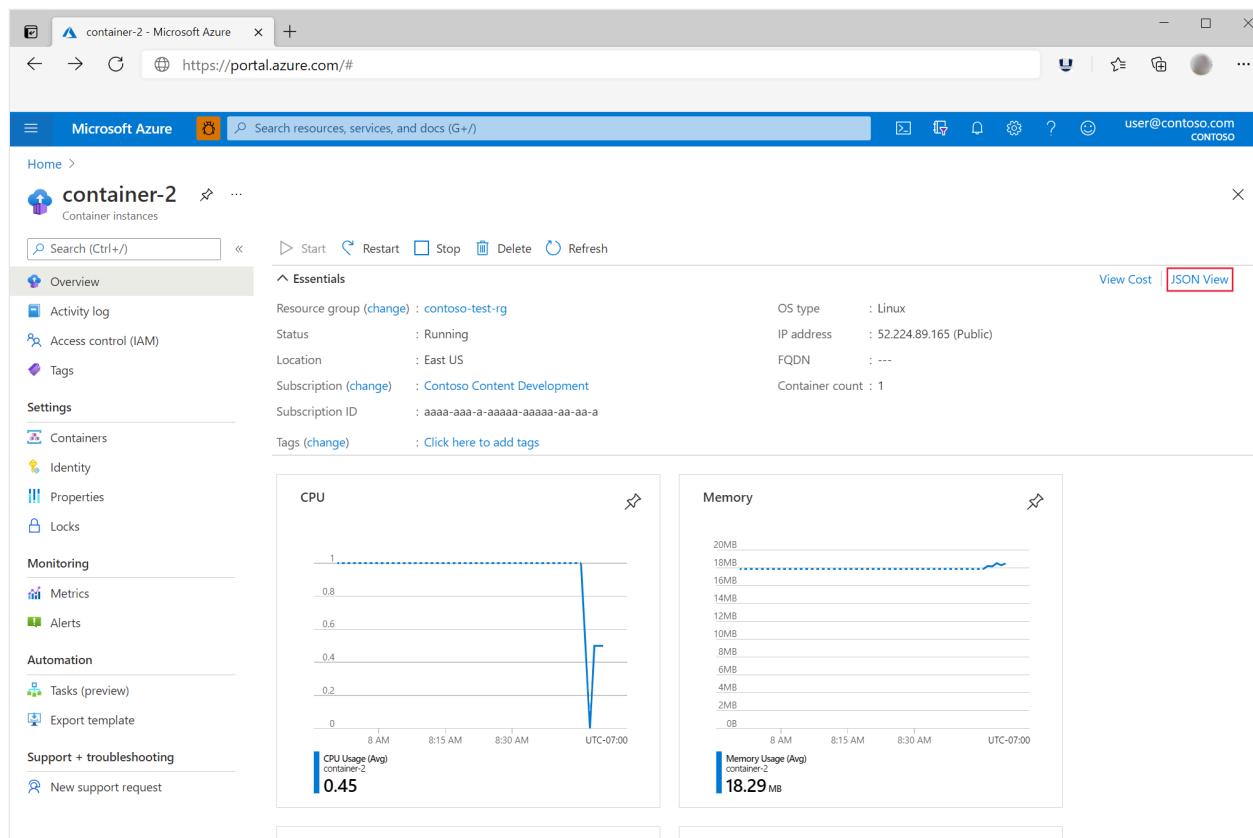
# Azure Container Instances states

Article • 08/29/2024

Azure Container Instances displays several independent state values. This article catalogs those values, where they can be found, and what they indicate.

## Where to find state values

In the Azure portal, state is shown in various locations. All state values are accessible via the JSON definition of the resource. This value can be found under Essentials in the Overview blade, shown in the following image.



The screenshot shows the Azure portal interface for a Container instance named "container-2". The "Overview" blade is open. In the "Essentials" section, you can see details such as the Resource group (contoso-test-rg), Status (Running), Location (East US), Subscription (Contoso Content Development), and Tags. Below the essentials, there are two charts: "CPU" and "Memory". The CPU chart shows usage over time, with a significant spike at UTC-07:00. The Memory chart shows usage in MB, with a value of 18.29 MB. A red box highlights the "JSON View" link in the top right corner of the essentials section.

State is also displayed in other locations in the Azure portal. The following table summarizes where state values can be found:

[Expand table](#)

Name	JSON path	Azure portal location
Container	<code>properties.instanceView.state</code>	Under Essentials in the
Group state		

Name	JSON path	Azure portal location
		Overview blade
Current Container state	<code>properties.containers/initContainers[x].instanceView.currentState.state</code>	Under the Containers blade's table's State column
Previous Container state	<code>properties.containers/initContainers[x].instanceView.previousState.state</code>	Via <i>JSON view</i> under Essentials in the Overview blade
Provisioning state	<code>properties.provisioningState</code>	Via <i>JSON view</i> under Essentials in the Overview blade; HTTP response body

## Container Groups

This value is the state of the deployed container group on the backend.

- **Running:** The container group is running and continues to run until a user action or a stop caused by the restart policy occurs.
- **Stopped:** The container group is stopped and won't run without user action.
- **Pending:** The container group is waiting to initialize (finish running init containers, mount Azure file volumes if applicable). The container continues to attempt to get to the **Running** state unless a user action (stop/delete) happens.
- **Succeeded:** The container group ran to completion successfully. Only applicable for *Never* and *On Failure* restart policies.
- **Failed:** The container group failed to run to completion. Only applicable with a *Never* restart policy. This state indicates either an infrastructure failure (example: incorrect Azure file share credentials) or user application failure (example: application references an environment variable that doesn't exist).

The following table shows what states are applicable to a container group based on the designated restart policy:

[Expand table](#)

Value	Never	On Failure	Always
Running	Yes	Yes	Yes
Stopped	Yes	Yes	Yes
Pending	Yes	Yes	Yes

Value	Never	On Failure	Always
Succeeded	Yes	Yes	No
Failed	Yes	No	No

## Containers

There are two state values for containers- a current state and a previous state. In the Azure portal, shown in the following image, only current state is displayed. All state values are applicable for any given container regardless of the container group's restart policy.

### ⚠ Note

The JSON values of `currentState` and `previousState` contain additional information, such as an exit code or a reason, that is not shown elsewhere in the Azure portal.

Name	Image	State	Previous state	Start time	Restart count
container-2	mcr.microsoft.com/oss/nginx/stable	Running	-	2021-03-25T17:56:33.499Z	0

**Events**

Name	Type	First timestamp	Last timestamp	Message	Count
Started	Normal	3/25/2021, 10:56 AM PDT	3/25/2021, 10:56 AM PDT	Started container	1
Pulled	Normal	3/25/2021, 10:56 AM PDT	3/25/2021, 10:56 AM PDT	Successfully pulled image "mcr.microsoft.com/oss/nginx/stable:latest"	1
Pulling	Normal	3/25/2021, 10:56 AM PDT	3/25/2021, 10:56 AM PDT	pulling image "mcr.microsoft.com/oss/nginx/stable:latest"	1

- **Running:** The container is running.
- **Waiting:** The container is waiting to run. This state indicates either init containers are still running, or the container is backing off due to a crash loop.
- **Terminated:** The container terminated, accompanied with an exit code value.

## Provisioning

This value is the state of the last operation performed on a container group. Generally, this operation is a PUT (create), but it can also be a POST (start/restart/stop) or DELETE (delete).

### ⓘ Important

Additionally, users should not create dependencies on non-terminal provisioning states. Dependencies on **Succeeded** and **Failed** states are acceptable.

In addition to the JSON view, provisioning state can also be found in the [response body of the HTTP call](#).

## Create, start, and restart operations

### ⓘ Important

PUT (create) operations are asynchronous. The returned value from the PUT's response body is not the final state. Making subsequent GET calls on the container group's resourceId or the AsyncOperation (returned in the PUT response headers) is the recommended way to monitor the status of the deployment.

These states are applicable to PUT (create) and POST (start/restart) events.

- **Pending:** The container group is waiting for infrastructure setup, such as a node assignment, virtual network provisioning, or anything else needed before pulling the user image.
- **Creating:** The infrastructure setup finished. The container group is now getting brought up and receiving the resources it needs (mounting Azure file volumes, getting ingress IP address, etc.).
- **Succeeded:** The container group succeeded in getting its containers into the running state and received all resources it needs.
- **Unhealthy:** The container group is unhealthy. For an unexpected state, such as if a node is down, a job is automatically triggered to repair the container group by moving it.
- **Repairing:** The container group is getting moved in order to repair an unhealthy state.

- **Failed:** The container group failed to reach the **Succeeded** provisioning state. Failure can occur for many reasons (low capacity in the designated region, full consumption of user quota, timeout after 30 minutes, etc.). More information on the failure can be found under `events` in the JSON view.

ⓘ Note

A failed state does not mean that the resource is removed or stops attempting to succeed. The container group state will indicate the current state of the group. If you want to ensure the container group does not run after a **Failed** provisioning state, then you will have to stop or delete it.

## Stop and delete operations

These values are applicable to POST (stop) and DELETE (delete) events.

- **Succeeded:** The operation to stop or delete the container group completed successfully.
- **Failed:** The container group failed to reach the **Succeeded** provisioning state, meaning the stop/delete event didn't complete. More information on the failure can be found under `events` in the JSON view.

---

## Feedback

Was this page helpful?

 Yes

 No

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

# Troubleshoot common issues in Azure Container Instances

Article • 08/29/2024

This article shows how to troubleshoot common issues for managing or deploying containers to Azure Container Instances. See also [Frequently asked questions](#).

If you need more support, see available **Help + support** options in the [Azure portal](#).

## Issues during container group deployment

### Naming conventions

When you define your container specification, certain parameters require adherence to naming restrictions. The following table shows the specific requirements for container group properties. For more information, see [Naming conventions](#) in the Azure Architecture Center and [Naming rules and restrictions for Azure resources](#).

[+] Expand table

Scope	Length	Casing	Valid characters	Suggested pattern	Example
Container name <sup>1</sup>	1-63	Lowercase	Alphanumeric, and hyphen anywhere except the first or last character	<name>-<role>-container<number>	web-batch-container1
Container ports	Between 1 and 65535	Integer	Integer between 1 and 65535	<port-number>	443
DNS name label	5-63	Case insensitive	Alphanumeric, and hyphen anywhere except the first or last character	<name>	frontend-site1
Environment variable	1-63	Case insensitive	Alphanumeric, and underscore (_) anywhere except the first or last character	<name>	MY_VARIABLE

Scope	Length	Casing	Valid characters	Suggested pattern	Example
Volume name	5-63	Lowercase	Alphanumeric, and hyphens anywhere except the first or last character. Can't contain two consecutive hyphens.	<name>	batch-output-volume

<sup>1</sup>Restriction also for container group names when not specified independently of container instances, for example with `az container create` command deployments.

## OS version of image not supported

If you specify an image that Azure Container Instances doesn't support, an `OsVersionNotSupported` error is returned. The error is similar to following, where `{0}` is the name of the image you attempted to deploy:

```
JSON
{
 "error": {
 "code": "OsVersionNotSupported",
 "message": "The OS version of image '{0}' is not supported."
 }
}
```

This error is most often encountered when deploying Windows images that are based on Semi-Annual Channel release 1709 or 1803, which aren't supported. For supported Windows images in Azure Container Instances, see [Frequently asked questions](#).

## Unable to pull image

If Azure Container Instances is initially unable to pull your image, it retries for time. If the image pull operation continues to fail, ACI eventually fails the deployment, and you may see a `Failed to pull image` error.

To resolve this issue, delete the container instance and retry your deployment. Ensure that the image exists in the registry and you typed the image name correctly.

If the image can't be pulled, events like the following are shown in the output of `az container show`:

## JSON

```
"events": [
 {
 "count": 3,
 "firstTimestamp": "2017-12-21T22:56:19+00:00",
 "lastTimestamp": "2017-12-21T22:57:00+00:00",
 "message": "pulling image \\"mcr.microsoft.com/azuredocs/aci-helloworld\\\"",
 "name": "Pulling",
 "type": "Normal"
 },
 {
 "count": 3,
 "firstTimestamp": "2017-12-21T22:56:19+00:00",
 "lastTimestamp": "2017-12-21T22:57:00+00:00",
 "message": "Failed to pull image \\"mcr.microsoft.com/azuredocs/aci-helloworld\\": rpc error: code 2 desc Error: image t/aci-helloworld:latest not found",
 "name": "Failed",
 "type": "Warning"
 },
 {
 "count": 3,
 "firstTimestamp": "2017-12-21T22:56:20+00:00",
 "lastTimestamp": "2017-12-21T22:57:16+00:00",
 "message": "Back-off pulling image \\"mcr.microsoft.com/azuredocs/aci-helloworld\\\"",
 "name": "BackOff",
 "type": "Normal"
 }
],
```

## Resource not available error

Due to varying regional resource load in Azure, you might receive the following error when attempting to deploy a container instance:

```
The requested resource with 'x' CPU and 'y.z' GB memory is not available in the location 'example region' at this moment. Please retry with a different resource request or in another location.
```

This error indicates that due to heavy load in the region in which you attempt to deploy, the resources specified for your container can't be allocated at that time. Use one or more of the following mitigation steps to help resolve your issue.

- Verify your container deployment settings fall within the parameters defined in [Region availability for Azure Container Instances](#)

- Specify lower CPU and memory settings for the container
- Deploy to a different Azure region
- Deploy at a later time

## Issues during container group runtime

### Container had an isolated restart without explicit user input

There are two broad categories for why a container group may restart without explicit user input. First, containers may experience restarts caused by an application process crash. The ACI service recommends applying observability solutions such as [Application Insights SDK](#), [container group metrics](#), and [container group logs](#) to determine why the application experienced issues. Second, customers may experience restarts initiated by the ACI infrastructure due to maintenance events. To increase the availability of your application, run multiple container groups behind an ingress component such as an [Application Gateway](#) or [Traffic Manager](#).

### Container continually exits and restarts (no long-running process)

Container groups default to a [restart policy of Always](#), so containers in the container group always restart after they run to completion. You may need to change this to [OnFailure](#) or [Never](#) if you intend to run task-based containers. If you specify [OnFailure](#) and still see continual restarts, there might be an issue with the application or script executed in your container.

When you run container groups without long-running processes, you may see repeated exits and restarts with images such as Ubuntu or Alpine. Connecting via [EXEC](#) won't work as the container has no process keeping it alive. To resolve this problem, include a start command like the following example with your container group deployment to keep the container running.

Azure CLI

```
Deploying a Linux container
az container create -g MyResourceGroup --name myapp --image ubuntu --
command-line "tail -f /dev/null"
```

Azure CLI

```
Deploying a Windows container
az container create -g myResourceGroup --name mywindowsapp --os-type Windows
--image mcr.microsoft.com/windows/servercore:ltsc2019
--command-line "ping -t localhost"
```

The Container Instances API and Azure portal include a `restartCount` property. To check the number of restarts for a container, you can use the `az container show` command in the Azure CLI. In the following example output, which we truncated for brevity, you see the `restartCount` property at the end of the output.

JSON

```
...
"events": [
 {
 "count": 1,
 "firstTimestamp": "2017-11-13T21:20:06+00:00",
 "lastTimestamp": "2017-11-13T21:20:06+00:00",
 "message": "Pulling: pulling image \\\"myregistry.azurecr.io/aci-tutorial-app:v1\\\"",
 "type": "Normal"
 },
 {
 "count": 1,
 "firstTimestamp": "2017-11-13T21:20:14+00:00",
 "lastTimestamp": "2017-11-13T21:20:14+00:00",
 "message": "Pulled: Successfully pulled image \\\"myregistry.azurecr.io/aci-tutorial-app:v1\\\"",
 "type": "Normal"
 },
 {
 "count": 1,
 "firstTimestamp": "2017-11-13T21:20:14+00:00",
 "lastTimestamp": "2017-11-13T21:20:14+00:00",
 "message": "Created: Created container with id
bf25a6ac73a925687cafcec792c9e3723b0776f683d8d1402b20cc9fb5f66a10",
 "type": "Normal"
 },
 {
 "count": 1,
 "firstTimestamp": "2017-11-13T21:20:14+00:00",
 "lastTimestamp": "2017-11-13T21:20:14+00:00",
 "message": "Started: Started container with id
bf25a6ac73a925687cafcec792c9e3723b0776f683d8d1402b20cc9fb5f66a10",
 "type": "Normal"
 }
],
"previousState": null,
"restartCount": 0
```

```
...
}
```

### ⓘ Note

Most container images for Linux distributions set a shell, such as bash, as the default command. Since a shell on its own is not a long-running service, these containers immediately exit and fall into a restart loop when configured with the default **Always** restart policy.

## Container takes a long time to start

The three primary factors that contribute to container startup time in Azure Container Instances are:

- [Image size](#)
- [Image location](#)
- [Cached images](#)

Windows images have [further considerations](#).

### Image size

If your container takes a long time to start, but eventually succeeds, start by looking at the size of your container image. Because Azure Container Instances pulls your container image on demand, the startup time you see is directly related to its size.

You can view the size of your container image by using the `docker images` command in the Docker CLI:

```
Bash
```

```
docker images
```

```
Output
```

REPOSITORY	CREATED	SIZE	TAG	IMAGE ID	15
mcr.microsoft.com/azuredocs/aci-helloworld	months ago	67.6MB	latest	7367f3256b41	

The key to keeping image sizes small is ensuring that your final image doesn't contain anything that isn't required at runtime. One way to do this is with [multi-stage builds](#). Multi-stage builds make it easy to ensure that the final image contains only the artifacts you need for your application, and not any of the extra content that was required at build time.

## Image location

Another way to reduce the impact of the image pull on your container's startup time is to host the container image in [Azure Container Registry](#) in the same region where you intend to deploy container instances. This shortens the network path that the container image needs to travel, significantly shortening the download time.

## Cached images

Azure Container Instances uses a caching mechanism to help speed container startup time for images built on common [Windows base images](#), including `nanoserver:1809`, `servercore:ltsc2019`, and `servercore:1809`. Commonly used Linux images such as `ubuntu:1604` and `alpine:3.6` are also cached. For both Windows and Linux images, avoid using the `latest` tag. Review Container Registry's [Image tag best practices](#) for guidance. For an up-to-date list of cached images and tags, use the [List Cached Images API](#).

### Note

Use of Windows Server 2019-based images in Azure Container Instances is in preview.

## Windows containers slow network readiness

On initial creation, Windows containers may have no inbound or outbound connectivity for up to 30 seconds (or longer, in rare cases). If your container application needs an Internet connection, add delay and retry logic to allow 30 seconds to establish Internet connectivity. After initial setup, container networking should resume appropriately.

## Can't connect to underlying Docker API or run privileged containers

Azure Container Instances doesn't expose direct access to the underlying infrastructure that hosts container groups. This includes access to the container runtime, orchestration technology, and running privileged container operations. To see what operations ACI supports, check the [REST reference documentation](#). If there's something missing, submit a request on the [ACI feedback forums](#).

## Container group IP address may not be accessible due to mismatched ports

Azure Container Instances doesn't yet support port mapping like with regular docker configuration. If you find a container group's IP address isn't accessible when you believe it should be, ensure you configured your container image to listen to the same ports you expose in your container group with the `ports` property.

If you want to confirm that Azure Container Instances can listen on the port you configured in your container image, test a deployment of the `aci-helloworld` image that exposes the port. Also run the `aci-helloworld` app so that it listens on the port. `aci-helloworld` accepts an optional environment variable `PORT` to override the default port 80 it listens on. For example, to test port 9000, set the [environment variable](#) when you create the container group:

1. Set up the container group to expose port 9000, and pass the port number as the value of the environment variable. The example is formatted for the Bash shell. If you prefer another shell such as PowerShell or Command Prompt, you need to adjust variable assignment accordingly.

Azure CLI

```
az container create --resource-group myResourceGroup \
--name mycontainer --image mcr.microsoft.com/azuredocs/aci-helloworld \
--ip-address Public --ports 9000 \
--environment-variables 'PORT='9000'
```

2. Find the IP address of the container group in the command output of `az container create`. Look for the value of `ip`.
3. After the container is provisioned successfully, browse to the IP address and port of the container application in your browser, for example: `192.0.2.0:9000`.

You should see the "Welcome to Azure Container Instances!" message displayed by the web app.

- When you're done with the container, remove it using the `az container delete` command:

```
Azure CLI
```

```
az container delete --resource-group myResourceGroup --name mycontainer
```

## Issues during confidential container group deployments

### Policy errors while using custom CCE policy

Custom CCE policies must be generated the [Azure CLI confcom extension](#). Before generating the policy, ensure that all properties specified in your ARM template are valid and match what you expect to be represented in a confidential computing policy. Some properties to validate include the container image, environment variables, volume mounts, and container commands.

### Missing hash from policy

The Azure CLI confcom extension uses cached images on your local machine that may not match those that are available remotely, which can result in layer mismatch when the policy is validated. Ensure that you remove any old images and pull the latest container images to your local environment. Once you're sure that you have the latest SHA, you should regenerate the CCE policy.

### Process/container terminated with exit code: 139

This exit code occurs due to limitations with the Ubuntu Version 22.04 base image. The recommendation is to use a different base image to resolve this issue.

## Next steps

Learn how to [retrieve container logs and events](#) to help debug your containers.

---

## Feedback

Was this page helpful?

 Yes

 No

Provide product feedback  | Get help at Microsoft Q&A

# Azure Container Instances troubleshooting documentation

Welcome to Azure Container Instances troubleshooting. These articles explain how to determine, diagnose, and fix issues that you might encounter when you use Azure Container Instances. In the navigation pane on the left, browse through the article list or use the search box to find issues and solutions.

**Learn how to use Azure Container Instances by reading these quickstarts, tutorials, and samples**



[Azure Container Instances documentation](#)

**Troubleshoot container group startup**



[Container group deployment remains in Waiting state](#)

[Container group remains in transitioning state](#)

[Inaccessible image failure](#)

[Error codes for Spot container creation](#)

[Error - Web socket is closed or could not be opened](#)

[Image pull takes a long time to run](#)

[ServiceUnavailable - container group quota exceeded in region](#)

[ServiceUnavailable \(409\) - requested resource is not available in the location](#)

**Troubleshoot a running container group**



[Azure Container Instances shows unexpected memory usage metrics](#)

[Can't do operations or the container group is in a bad state](#)

[Can't pull images from Container Registry using a managed identity](#)

[Confidential container issues](#)

[Container group killing event interruptions](#)

[Container group remains in transitioning state](#)

[Container group still runs after a stop or delete command](#)

[Container Instances debugging tools](#)

[Spot container errors](#)

## Troubleshoot container groups in virtual networks

---

### HOW-TO GUIDE

[Can't delete a virtual network or subnet](#)

[Error - Web socket is closed or could not be opened](#)

# az container

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 Instances.

## Commands

[Expand table](#)

Name	Description	Type	Status
<a href="#">az container app</a>	Commands to Manage Azure Container Instances App.	Extension	Preview
<a href="#">az container app up</a>	Deploy to Azure Container Instances using GitHub Actions.	Extension	Preview
<a href="#">az container attach</a>	Attach local standard output and error streams to a container in a container group. This command provides diagnostic information during container startup. Once the container has started, it streams STDOUT and STDERR to your local console.	Core	GA
<a href="#">az container container-group-profile</a>	Manage Azure Container Instance Container Group Profile.	Core	GA
<a href="#">az container container-group-profile create</a>	Create a container group profile.	Core	GA
<a href="#">az container container-group-profile delete</a>	Delete a container group profile.	Core	GA
<a href="#">az container container-group-profile list</a>	List all container group profiles in a resource group.	Core	GA

Name	Description	Type	Status
<a href="#">az container container-group-profile list-revisions</a>	List all revisions for a container group profile.	Core	GA
<a href="#">az container container-group-profile show</a>	Get a container group profile.	Core	GA
<a href="#">az container container-group-profile show-revision</a>	Show a container group profile revision.	Core	GA
<a href="#">az container create</a>	Create a container group.	Core	GA
<a href="#">az container delete</a>	Delete a container group.	Core	GA
<a href="#">az container exec</a>	Execute a command from within a running container of a container group.	Core	GA
<a href="#">az container export</a>	Export a container group in yaml format.	Core	GA
<a href="#">az container list</a>	List container groups.	Core	GA
<a href="#">az container logs</a>	Examine the logs for a container in a container group.	Core	GA
<a href="#">az container restart</a>	Restarts all containers in a container group.	Core	GA
<a href="#">az container show</a>	Get the details of a container group.	Core	GA
<a href="#">az container start</a>	Starts all containers in a container group.	Core	GA
<a href="#">az container stop</a>	Stops all containers in a container group.	Core	GA

## az container attach

 Edit

Attach local standard output and error streams to a container in a container group. This command provides diagnostic information during container startup. Once the container has started, it streams STDOUT and STDERR to your local console.

```
Azure CLI
```

```
az container attach [--container-name]
 [--ids]
 [--name]
 [--resource-group]
 [--subscription]
```

## Examples

Attach local standard output and error streams to a container in a container group.  
( autogenerated )

```
Azure CLI
```

```
az container attach --name MyContainerGroup --resource-group MyResourceGroup
```

## Optional Parameters

### --container-name

The container to attach to. If omitted, the first container in the container group will be chosen.

### --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 container group.

### --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 container create

 Edit

Create a container group.

## Azure CLI

```
az container create --resource-group
 [--acr-identity]
 [--add-capabilities]
 [--allow-escalation]
 [--assign-identity]
 [--azure-file-volume-account-key]
 [--azure-file-volume-account-name]
 [--azure-file-volume-mount-path]
 [--azure-file-volume-share-name]
 [--cce-policy]
 [--command-line]
 [--config-map]
 [--container-group-profile-id]
 [--container-group-profile-revision]
 [--cpu]
 [--dns-name-label]
 [--drop-capabilities]
 [--environment-variables]
 [--fail-container-group-create-on-reuse-failure]
 [--file]
 [--gitrepo-dir]
 [--gitrepo-mount-path]
 [--gitrepo-revision]
 [--gitrepo-url]
 [--image]
 [--ip-address {Private, Public}]
 [--location]
 [--log-analytics-workspace]
 [--log-analytics-workspace-key]
 [--memory]
 [--name]
 [--no-wait]
 [--os-type {Linux, Windows}]
 [--ports]
 [--priority]
 [--privileged]
 [--protocol {TCP, UDP}]
 [--registry-login-server]
 [--registry-password]
 [--registry-username]
 [--restart-policy {Always, Never, OnFailure}]
 [--role]
 [--run-as-group]
 [--run-as-user]
 [--scope]
 [--seccomp-profile]
 [--secrets]
 [--secrets-mount-path]
 [--secure-environment-variables]
 [--sku]
 [--standby-pool-profile-id]
 [--subnet]
```

```
[--subnet-address-prefix]
[--vnet]
[--vnet-address-prefix]
[--zone]
```

## Examples

Create a container in a container group with 1 core and 1Gb of memory.

Azure CLI

```
az container create -g MyResourceGroup --name myapp --image myimage:latest -
--cpu 1 --memory 1
```

Create a container in a container group that runs Windows, with 2 cores and 3.5Gb of memory.

Azure CLI

```
az container create -g MyResourceGroup --name mywinapp --image
winappimage:latest --os-type Windows --cpu 2 --memory 3.5
```

Create a container in a container group with public IP address, ports and DNS name label.

Azure CLI

```
az container create -g MyResourceGroup --name myapp --image myimage:latest -
--ports 80 443 --dns-name-label contoso
```

Create a container in a container group that invokes a script upon start.

Azure CLI

```
az container create -g MyResourceGroup --name myapp --image myimage:latest -
--command-line "/bin/sh -c '/path to/myscript.sh'"
```

Create a container in a container group that runs a command and stop the container afterwards.

Azure CLI

```
az container create -g MyResourceGroup --name myapp --image myimage:latest -
--command-line "echo hello" --restart-policy Never
```

Create a container in a container group with environment variables.

Azure CLI

```
az container create -g MyResourceGroup --name myapp --image myimage:latest --environment-variables key1=value1 key2=value2
```

Create a container in a container group using container image from Azure Container Registry.

Azure CLI

```
az container create -g MyResourceGroup --name myapp --image myAcrRegistry.azurecr.io/myimage:latest --registry-password password
```

Create a container in a container group that mounts an Azure File share as volume.

Azure CLI

```
az container create -g MyResourceGroup --name myapp --image myimage:latest --command-line "cat /mnt/azfile/myfile" --azure-file-volume-share-name myshare --azure-file-volume-account-name mystorageaccount --azure-file-volume-account-key mystoragekey --azure-file-volume-mount-path /mnt/azfile
```

Create a container in a container group that mounts a git repo as volume.

Azure CLI

```
az container create -g MyResourceGroup --name myapp --image myimage:latest --command-line "cat /mnt/gitrepo" --gitrepo-url https://github.com/user/myrepo.git --gitrepo-dir ./dir1 --gitrepo-mount-path /mnt/gitrepo
```

Create a container in a container group using a yaml file.

Azure CLI

```
az container create -g MyResourceGroup -f containerGroup.yaml
```

Create a container group using Log Analytics from a workspace name.

Azure CLI

```
az container create -g MyResourceGroup --name myapp --log-analytics-workspace myworkspace
```

Create a container group with a system assigned identity.

Azure CLI

```
az container create -g MyResourceGroup --name myapp --image myimage:latest --assign-identity
```

Create a container group with a system assigned identity. The group will have a 'Contributor' role with access to a storage account.

Azure CLI

```
az container create -g MyResourceGroup --name myapp --image myimage:latest --assign-identity --scope /subscriptions/99999999-1bf0-4dda-aec3-cb9272f09590/MyResourceGroup/myRG/providers/Microsoft.Storage/storageAccounts/storage1
```

Create a container group with a user assigned identity.

Azure CLI

```
az container create -g MyResourceGroup --name myapp --image myimage:latest --assign-identity /subscriptions/mySubscriptionId/resourcegroups/myRG/providers/Microsoft.ManagedIdentity/userAssignedIdentities/myID
```

Create a container group with both system and user assigned identity.

Azure CLI

```
az container create -g MyResourceGroup --name myapp --image myimage:latest --assign-identity [system] /subscriptions/mySubscriptionId/resourcegroups/myRG/providers/Microsoft.ManagedIdentity/userAssignedIdentities/myID
```

## Required Parameters

**--resource-group -g**

Name of resource group. You can configure the default group using `az configure --defaults group=<name>`.

## Optional Parameters

**--acr-identity**

The identity with access to the container registry.

**--add-capabilities**

A List of security context capabilities to be added.

**--allow-escalation**

Allow whether a process can gain more privileges than its parent process.

Default value: False

**--assign-identity**

Space-separated list of assigned identities. Assigned identities are either user assigned identities (resource IDs) and / or the system assigned identity ('[system]').

See examples for more info.

**--azure-file-volume-account-key**

The storage account access key used to access the Azure File share.

**--azure-file-volume-account-name**

The name of the storage account that contains the Azure File share.

**--azure-file-volume-mount-path**

The path within the container where the azure file volume should be mounted. Must not contain colon ':'.

**--azure-file-volume-share-name**

The name of the Azure File share to be mounted as a volume.

**--cce-policy**

The CCE policy for the confidential container group.

**--command-line**

The command line to run when the container is started, e.g. '/bin/bash -c myscript.sh'.

#### **--config-map**

A list of config map key-value pairs for the container. Space-separated values in 'key=value' format.

#### **--container-group-profile-id**

The reference container group profile ARM resource id.

#### **--container-group-profile-revision**

The reference container group profile revision.

#### **--cpu**

The required number of CPU cores of the containers, accurate to one decimal place.

#### **--dns-name-label**

The dns name label for container group with public IP.

#### **--drop-capabilities**

A List of security context capabilities to be dropped.

#### **--environment-variables -e**

A list of environment variable for the container. Space-separated values in 'key=value' format.

#### **--fail-container-group-create-on-reuse-failure**

The flag indicating whether to fail the container group creation if the standby pool reuse failed.

Default value: False

#### **--file -f**

The path to the input file.

#### **--gitrepo-dir**

The target directory path in the git repository. Must not contain '..'.

Default value: .

#### **--gitrepo-mount-path**

The path within the container where the git repo volume should be mounted. Must not contain colon ':'.

#### **--gitrepo-revision**

The commit hash for the specified revision.

#### **--gitrepo-url**

The URL of a git repository to be mounted as a volume.

#### **--image**

The container image name.

#### **--ip-address**

The IP address type of the container group.

Accepted values: Private, Public

#### **--location -l**

Location. Values from: `az account list-locations`. You can configure the default location using `az configure --defaults location=<location>`.

#### **--log-analytics-workspace**

The Log Analytics workspace name or id. Use the current subscription or use --subscription flag to set the desired subscription.

#### **--log-analytics-workspace-key**

The Log Analytics workspace key.

#### **--memory**

The required memory of the containers in GB, accurate to one decimal place.

#### **--name -n**

The name of the container group.

#### **--no-wait**

Do not wait for the long-running operation to finish.

Default value: False

#### **--os-type**

The OS type of the containers.

Accepted values: Linux, Windows

#### **--ports**

A list of ports to open. Space-separated list of ports.

Default value: [80]

#### **--priority**

The priority of the container group.

#### **--privileged**

The flag to determine if the container permissions is elevated to Privileged.

Default value: False

#### **--protocol**

The network protocol to use.

Accepted values: TCP, UDP

#### **--registry-login-server**

The container image registry login server.

#### **--registry-password**

The password to log in container image registry server.

#### **--registry-username**

The username to log in container image registry server.

#### **--restart-policy**

Restart policy for all containers within the container group.

Accepted values: Always, Never, OnFailure

#### **--role**

Role name or id the system assigned identity will have.

Default value: Contributor

#### **--run-as-group**

Set the User UID for the container.

#### **--run-as-user**

Set the User GID for the container.

#### **--scope**

Scope that the system assigned identity can access.

#### **--seccomp-profile**

A base64 encoded string containing the contents of the JSON in the seccomp profile.

#### **--secrets**

Space-separated secrets in 'key=value' format.

#### **--secrets-mount-path**

The path within the container where the secrets volume should be mounted. Must not contain colon ':'.

#### **--secure-environment-variables**

A list of secure environment variable for the container. Space-separated values in 'key=value' format.

#### **--sku**

The SKU of the container group.

#### **--standby-pool-profile-id**

The standby pool profile ARM resource id from which the container will be reused.

#### --subnet

The name of the subnet when creating a new VNET or referencing an existing one.  
Can also reference an existing subnet by ID.

#### --subnet-address-prefix

The subnet IP address prefix to use when creating a new VNET in CIDR format.  
Default value: 10.0.0.0/24

#### --vnet

The name of the VNET when creating a new one or referencing an existing one. Can also reference an existing vnet by ID. This allows using vnets from other resource groups.

#### --vnet-address-prefix

The IP address prefix to use when creating a new VNET in CIDR format.  
Default value: 10.0.0.0/16

#### --zone

The zone to place the container group.

### ▼ 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 container delete

 Edit

Delete a container group.

Azure CLI

```
az container delete [--ids]
 [--name]
 [--resource-group]
 [--subscription]
 [--yes]
```

## Examples

Delete a container group. (autogenerated)

Azure CLI

```
az container delete --name MyContainerGroup --resource-group 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 container group.

## --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 container exec

 Edit

Execute a command from within a running container of a container group.

The most common use case is to open an interactive bash shell. See examples below.

Azure CLI

```
az container exec --exec-command
 [--container-name]
 [--ids]
 [--name]
 [--resource-group]
 [--subscription]
```

## Examples

Stream a shell from within an nginx container.

Azure CLI

```
az container exec -g MyResourceGroup --name mynginx --container-name nginx -
--exec-command "/bin/bash"
```

# Required Parameters

## --exec-command

The command to run from within the container.

# Optional Parameters

## --container-name

The container name where to execute the command. Can be omitted for container groups with only one container.

## --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 container group.

## --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 container export

 Edit

Export a container group in yaml format.

Azure CLI

```
az container export --file
 [--ids]
 [--name]
 [--resource-group]
 [--subscription]
```

## Examples

Export a container group in yaml.

## Azure CLI

```
az container export -g MyResourceGroup --name mynginx -f output.yaml
```

# Required Parameters

### --file -f

The file path to export the container group.

# 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 container group.

### --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 container list

 Edit

List container groups.

Azure CLI

```
az container list [--resource-group]
```

## Optional Parameters

#### --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 container logs

 Edit

Examine the logs for a container in a container group.

Azure CLI

```
az container logs [--container-name]
 [--follow]
 [--ids]
 [--name]
 [--resource-group]
 [--subscription]
```

## Examples

Examine the logs for a container in a container group. (autogenerated)

Azure CLI

```
az container logs --name MyContainerGroup --resource-group MyResourceGroup
```

## Optional Parameters

### --container-name

The container name to tail the logs. If omitted, the first container in the container group will be chosen.

### --follow

Indicate to stream the tailing logs.

Default value: False

### --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 container group.

### --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 container restart

 Edit

Restarts all containers in a container group.

Restarts all containers in a container group in place. If container image has updates, new image will be downloaded.

Azure CLI

```
az container restart --name
 --resource-group
 [--no-wait]
```

## Required Parameters

**--name -n**

The name of the container group.

**--resource-group -g**

Name of resource group. You can configure the default group using `az configure --defaults group=<name>`.

## Optional Parameters

**--no-wait**

Do not wait for the long-running operation to finish.

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 container show

 Edit

Get the details of a container group.

Azure CLI

```
az container show [--ids]
 [--name]
 [--resource-group]
 [--subscription]
```

## Examples

Get the details of a container group. (autogenerated)

```
az container show --name MyContainerGroup --resource-group 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 container group.

### --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 container start

 Edit

Starts all containers in a container group.

Starts all containers in a container group. Compute resources will be allocated and billing will start.

Azure CLI

```
az container start --name
 --resource-group
 [--no-wait]
```

## Required Parameters

### --name -n

The name of the container group.

### --resource-group -g

Name of resource group. You can configure the default group using `az configure --defaults group=<name>`.

## Optional Parameters

### `--no-wait`

Do not wait for the long-running operation to finish.

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 container stop

 Edit

Stops all containers in a container group.

Stops all containers in a container group. Compute resources will be deallocated and billing will stop.

Azure CLI

```
az container stop --name
 --resource-group
```

## Required Parameters

**--name -n**

The name of the container group.

**--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.

# Azure Container Instances REST API reference

Article • 10/31/2023

Azure Container Instances offers the fastest and simplest way to run a container in Azure, without having to provision any virtual machines and without having to adopt a higher-level service. For more information, see the [Azure Container Instances overview](#).

## See also

[Azure Container Registry](#) - Store and manage container images across all types of Azure deployments.

[Azure Kubernetes Service \(AKS\)](#) - Easily deploy and manage containerized applications in a hosted Kubernetes environment.

# Az.ContainerInstance

Reference

Microsoft Azure PowerShell: ContainerInstance cmdlets

## Container Instances

[Expand table](#)

<a href="#">Add-AzContainerInstanceOutput</a>	Attach to the output stream of a specific container instance in a specified resource group and container group.
<a href="#">Get-AzContainerGroup</a>	Gets the properties of the specified container group in the specified subscription and resource group. The operation returns the properties of each container group including containers, image registry credentials, restart policy, IP address type, OS type, state, and volumes.
<a href="#">Get-AzContainerInstanceCachedImage</a>	Get the list of cached images on specific OS type for a subscription in a region.
<a href="#">Get-AzContainerInstanceCapability</a>	Get the list of CPU/memory/GPU capabilities of a region.
<a href="#">Get-AzContainerInstanceContainerGroupOutboundNetworkDependencyEndpoint</a>	Gets all the network dependencies for this container group to allow complete control of network setting and configuration. For

	container groups, this will always be an empty list.
<a href="#">Get-AzContainerInstanceContainerGroupProfile</a>	Gets the properties of the specified container group profile in the specified subscription and resource group. The operation returns the properties of container group profile including containers, image registry credentials, restart policy, IP address type, OS type, volumes, current revision number, etc.
<a href="#">Get-AzContainerInstanceContainerGroupProfileRevision</a>	Get a list of all the revisions of the specified container group profile in the given subscription and resource group. This operation returns properties of each revision of the specified container group profile including containers, image registry credentials, restart policy, IP address type, OS type volumes, revision number, etc.
<a href="#">Get-AzContainerInstanceLog</a>	Get the logs for a specified container instance in a specified resource group and container group.
<a href="#">Get-AzContainerInstanceUsage</a>	Get the usage for a subscription
<a href="#">Invoke-AzContainerInstanceCommand</a>	Executes a command for a specific container instance in a specified

		resource group and container group.
<a href="#">New-AzContainerGroup</a>		Create or update container groups with specified configurations.
<a href="#">New-AzContainerGroupImageRegistryCredentialObject</a>		Create a in-memory object for ImageRegistryCredential
<a href="#">New-AzContainerGroupPortObject</a>		Create a in-memory object for Port
<a href="#">New-AzContainerGroupVolumeObject</a>		Create an in-memory object for Volume.
<a href="#">New-AzContainerInstanceContainerGroupProfile</a>		Create or update container group profiles with specified configurations.
<a href="#">New-AzContainerInstanceEnvironmentVariableObject</a>		Create a in-memory object for EnvironmentVariable
<a href="#">New-AzContainerInstanceHttpHeaderObject</a>		Create a in-memory object for HttpHeaders
<a href="#">New-AzContainerInstanceInitDefinitionObject</a>		Create a in-memory object for InitContainerDefinition
<a href="#">New-AzContainerInstanceNoDefaultObject</a>		Create a in-memory object for Container with no default values
<a href="#">New-AzContainerInstanceObject</a>		Create a in-memory object for Container
<a href="#">New-AzContainerInstancePortObject</a>		Create a in-memory object for ContainerPort
<a href="#">New-AzContainerInstanceVolumeMountObject</a>		Create a in-memory object for VolumeMount
<a href="#">Remove-AzContainerGroup</a>		Delete the specified container group in the specified subscription and resource group. The operation does not

	delete other resources provided by the user, such as volumes.
<a href="#">Remove-AzContainerInstanceContainerGroupProfile</a>	Delete the specified container group profile in the specified subscription and resource group. The operation does not delete other resources provided by the user, such as volumes.
<a href="#">Remove-AzContainerInstanceSubnetServiceAssociationLink</a>	Delete container group virtual network association links. The operation does not delete other resources provided by the user.
<a href="#">Restart-AzContainerGroup</a>	Restarts all containers in a container group in place. If container image has updates, new image will be downloaded.
<a href="#">Start-AzContainerGroup</a>	Starts all containers in a container group. Compute resources will be allocated and billing will start.
<a href="#">Stop-AzContainerGroup</a>	Stops all containers in a container group. Compute resources will be deallocated and billing will stop.
<a href="#">Update-AzContainerGroup</a>	Updates container group tags with specified values.
<a href="#">Update-AzContainerInstanceContainerGroupProfile</a>	Patches container group profile with specified properties.

# Azure Container Instances libraries for .NET

Article • 10/11/2022

Use the Microsoft Azure Container Instances libraries for .NET to create and manage Azure container instances. Learn more by reading the [Azure Container Instances overview](#).

## Management library

Use the management library to create and manage Azure container instances in Azure.

Install the [NuGet package](#) directly from the Visual Studio [Package Manager console](#) or with the [.NET Core CLI](#).

## Visual Studio Package Manager

PowerShell

```
Install-Package Microsoft.Azure.Management.ContainerInstance.Fluent
```

.NET CLI

```
dotnet add package Microsoft.Azure.Management.ContainerInstance.Fluent
```

## Example source

If you'd like to see the following code examples in context, you can find them in the following GitHub repository:

[Azure-Samples/aci-docs-sample-dotnet](#)

## Authentication

One of the easiest ways to authenticate SDK clients is with [file-based authentication](#). File-based authentication parses a credentials file when instantiating the [IAzure](#) client object, which then uses those credentials when authenticating with Azure. To use file-based authentication:

1. Create a credentials file with the [Azure CLI](#) or [Cloud Shell](#):

```
az ad sp create-for-rbac --sdk-auth > my.azureauth
```

If you use the [Cloud Shell](#) to generate the credentials file, copy its contents into a local file that your .NET application can access.

2. Set the `AZURE_AUTH_LOCATION` environment variable to the full path of the generated credentials file. For example (in the Bash shell):

Bash

```
export AZURE_AUTH_LOCATION=/home/yourusername/my.azureauth
```

Once you've created the credentials file and populated the `AZURE_AUTH_LOCATION` environment variable, use the `Azure.Authenticate` method to initialize the `IAzure` client object. The example project first obtains the `AZURE_AUTH_LOCATION` value, then calls a method that returns an initialized `IAzure` client object:

C#

```
// Set the AZURE_AUTH_LOCATION environment variable with the full
// path to an auth file. Create an auth file with the Azure CLI:
// az ad sp create-for-rbac --sdk-auth > my.azureauth
string authFilePath = Environment.GetEnvironmentVariable("AZURE_AUTH_LOCATION");

// Authenticate with Azure
IAzure azure = GetAzureContext(authFilePath);
```

This method from the sample application returns the initialized `IAzure` instance, which is then passed as the first parameter to all other methods in the sample:

C#

```
/// <summary>
/// Returns an authenticated Azure context using the credentials in the
/// specified auth file.
/// </summary>
/// <param name="authFilePath">The full path to a credentials file on the local
/// filesystem.</param>
/// <returns>Authenticated IAzure context.</returns>
private static IAzure GetAzureContext(string authFilePath)
{
 IAzure azure;
 ISubscription sub;

 try
 {
 Console.WriteLine($"Authenticating with Azure using credentials in file at
{authFilePath}");

 azure = AzureAuthenticate(authFilePath).WithDefaultSubscription();
 }
}
```

```

 sub = azure.GetCurrentSubscription();

 Console.WriteLine($"Authenticated with subscription '{sub.DisplayName}'"
(ID: {sub.SubscriptionId})");
 }
 catch (Exception ex)
 {
 Console.WriteLine($"\\nFailed to authenticate:\\n{ex.Message}");

 if (String.IsNullOrEmpty(authFilePath))
 {
 Console.WriteLine("Have you set the AZURE_AUTH_LOCATION environment
variable?");
 }

 throw;
 }

 return azure;
}

```

For more details about the available authentication methods in the .NET management libraries for Azure, see [Authentication in Azure Management Libraries for .NET](#).

## Create container group - single container

This example creates a container group with a single container.

```

C#

/// <summary>
/// Creates a container group with a single container.
/// </summary>
/// <param name="azure">An authenticated IAzure object.</param>
/// <param name="resourceGroupName">The name of the resource group in which to
create the container group.</param>
/// <param name="containerGroupName">The name of the container group to create.
</param>
/// <param name="containerImage">The container image name and tag, for example
'microsoft\aci-helloworld:latest'.</param>
private static void CreateContainerGroup(IAzure azure,
 string resourceGroupName,
 string containerGroupName,
 string containerImage)
{
 Console.WriteLine($"\\nCreating container group '{containerGroupName}'...");

 // Get the resource group's region
 IResourceGroup resGroup = azure.ResourceGroups.GetByName(resourceGroupName);
 Region azureRegion = resGroup.Region;

```

```

// Create the container group
var containerGroup = azure.ContainerGroups.Define(containerGroupName)
 .WithRegion(azureRegion)
 .WithExistingResourceGroup(resourceGroupName)
 .WithLinux()
 .WithPublicImageRegistryOnly()
 .WithoutVolume()
 .DefineContainerInstance(containerGroupName + "-1")
 .WithImage(containerImage)
 .WithExternalTcpPort(80)
 .WithCpuCoreCount(1.0)
 .WithMemorySizeInGB(1)
 .Attach()
 .WithDnsPrefix(containerGroupName)
 .Create();

Console.WriteLine($"Once DNS has propagated, container group
'{containerGroup.Name}' will be reachable at http://'{containerGroup.Fqdn}'");
}

```

## Create container group - multiple containers

This example creates a container group with two containers: an application container and a sidecar container.

C#

```

/// <summary>
/// Creates a container group with two containers in the specified resource group.
/// </summary>
/// <param name="azure">An authenticated IAzure object.</param>
/// <param name="resourceGroupName">The name of the resource group in which to
/// create the container group.</param>
/// <param name="containerGroupName">The name of the container group to create.
/// </param>
/// <param name="containerImage1">The first container image name and tag, for
/// example 'microsoft\aci-helloworld:latest'.</param>
/// <param name="containerImage2">The second container image name and tag, for
/// example 'microsoft\aci-tutorial-sidecar:latest'.</param>
private static void CreateContainerGroupMulti(IAzure azure,
 string resourceGroupName,
 string containerGroupName,
 string containerImage1,
 string containerImage2)
{
 Console.WriteLine($"{Environment.NewLine}Creating multi-container container group
'{containerGroupName}'{Environment.NewLine}{Environment.NewLine} // Get the resource group's region
 IResourceGroup resGroup = azure.ResourceGroups.GetByName(resourceGroupName);
 Region azureRegion = resGroup.Region;
}

```

```
// Create the container group
var containerGroup = azure.ContainerGroups.Define(containerGroupName)
 .WithRegion(azureRegion)
 .WithExistingResourceGroup(resourceGroupName)
 .WithLinux()
 .WithPublicImageRegistryOnly()
 .WithoutVolume()
 .DefineContainerInstance(containerGroupName + "-1")
 .WithImage(containerImage1)
 .WithExternalTcpPort(80)
 .WithCpuCoreCount(0.5)
 .WithMemorySizeInGB(1)
 .Attach()
 .DefineContainerInstance(containerGroupName + "-2")
 .WithImage(containerImage2)
 .WithoutPorts()
 .WithCpuCoreCount(0.5)
 .WithMemorySizeInGB(1)
 .Attach()
 .WithDnsPrefix(containerGroupName)
 .Create();

Console.WriteLine($"Once DNS has propagated, container group
'{containerGroup.Name}' will be reachable at http://'{containerGroup.Fqdn}'");
}
```

## Asynchronous container create with polling

This example creates a container group with a single container using the `async create` method. It then polls Azure for the container group, and outputs the container group's status until its state is "Running."

```

Console.WriteLine($"\\nCreating container group '{containerGroupName}'...");

// Get the resource group's region
IResourceGroup resGroup = azure.ResourceGroups.GetByName(resourceGroupName);
Region azureRegion = resGroup.Region;

// Create the container group using a fire-and-forget task
Task.Run(() =>

 azure.ContainerGroups.Define(containerGroupName)
 .WithRegion(azureRegion)
 .WithExistingResourceGroup(resourceGroupName)
 .WithLinux()
 .WithPublicImageRegistryOnly()
 .WithoutVolume()
 .DefineContainerInstance(containerGroupName + "-1")
 .WithImage(containerImage)
 .WithExternalTcpPort(80)
 .WithCpuCoreCount(1.0)
 .WithMemorySizeInGB(1)
 .Attach()
 .WithDnsPrefix(containerGroupName)
 .CreateAsync()
);

// Poll for the container group
.IContainerGroup containerGroup = null;
while(containerGroup == null)
{
 containerGroup =
azure.ContainerGroups.GetByResourceGroup(resourceGroupName, containerGroupName);

 Console.Write(".");
 SdkContext.DelayProvider.Delay(1000);
}

Console.WriteLine();

// Poll until the container group is running
while(containerGroup.State != "Running")
{
 Console.WriteLine($"Container group state:
{containerGroup.Refresh().State}");

 Thread.Sleep(1000);
}

Console.WriteLine($"\\nOnce DNS has propagated, container group
'{containerGroup.Name}' will be reachable at http://'{containerGroup.Fqdn}'");
}

```

# Create task-based container group

This example creates a container group with a single task-based container. The container is configured with a [restart policy](#) of "Never" and a [custom command line](#).

If you want to run a single command with several command-line arguments, for example `echo FOO BAR`, you must supply them as a string array to the `WithStartingCommandLines` method. For example:

```
WithStartingCommandLines("echo", "FOO", "BAR")
```

If, however, you want to run multiple commands with (potentially) multiple arguments, you must execute a shell and pass the chained commands as an argument. For example, this executes both an `echo` and a `tail` command:

```
WithStartingCommandLines("/bin/sh", "-c", "echo FOO BAR && tail -f /dev/null")
```

C#

```
/// <summary>
/// Creates a container group with a single task-based container who's
/// restart policy is 'Never'. If specified, the container runs a custom
/// command line at startup.
/// </summary>
/// <param name="azure">An authenticated IAzure object.</param>
/// <param name="resourceGroupName">The name of the resource group in which to
/// create the container group.</param>
/// <param name="containerGroupName">The name of the container group to create.
/// </param>
/// <param name="containerImage">The container image name and tag, for example
/// 'microsoft\aci-wordcount:latest'.</param>
/// <param name="startCommandLine">The command line that should be executed when
/// the container starts. This value can be <c>null</c>.</param>
private static void RunTaskBasedContainer(IAzure azure,
 string resourceGroupName,
 string containerGroupName,
 string containerImage,
 string startCommandLine)
{
 // If a start command wasn't specified, use a default
 if (String.IsNullOrEmpty(startCommandLine))
 {
 startCommandLine = "python wordcount.py
http://shakespeare.mit.edu/romeo_juliet/full.html";
 }

 // Configure some environment variables in the container which the
 // wordcount.py or other script can read to modify its behavior.
 Dictionary<string, string> envVars = new Dictionary<string, string>
 {
```

```

 { "NumWords", "5" },
 { "MinLength", "8" }
 };

 Console.WriteLine($"\\nCreating container group '{containerGroupName}' with
start command '{startCommandLine}'");

 // Get the resource group's region
 IResourceGroup resGroup = azure.ResourceGroups.GetByName(resourceGroupName);
 Region azureRegion = resGroup.Region;

 // Create the container group
 var containerGroup = azure.ContainerGroups.Define(containerGroupName)
 .WithRegion(azureRegion)
 .WithExistingResourceGroup(resourceGroupName)
 .WithLinux()
 .WithPublicImageRegistryOnly()
 .WithoutVolume()
 .DefineContainerInstance(containerGroupName + "-1")
 .WithImage(containerImage)
 .WithExternalTcpPort(80)
 .WithCpuCoreCount(1.0)
 .WithMemorySizeInGB(1)
 .WithStartingCommandLines(startCommandLine.Split())
 .WithEnvironmentVariables(envVars)
 .Attach()
 .WithDnsPrefix(containerGroupName)
 .WithRestartPolicy(ContainerGroupRestartPolicy.Never)
 .Create();

 // Print the container's logs
 Console.WriteLine($"Logs for container '{containerGroupName}-1':");
 Console.WriteLine(containerGroup.GetLogContent(containerGroupName + "-1"));
}

```

## List container groups

This example lists the container groups in a resource group.

C#

```

/// <summary>
/// Prints the container groups in the specified resource group.
/// </summary>
/// <param name="azure">An authenticated IAzure object.</param>
/// <param name="resourceGroupName">The name of the resource group containing the
/// container group(s).</param>
private static void ListContainerGroups(IAzure azure, string resourceGroupName)
{
 Console.WriteLine($"Listing container groups in resource group
'{resourceGroupName}'...");

```

```
foreach (var containerGroup in
azure.ContainerGroups.ListByResourceGroup(resourceGroupName))
{
 Console.WriteLine($"{containerGroup.Name}");
}
}
```

## Get an existing container group

This example gets a specific container group residing in a resource group and then prints a few of its properties and their values.

C#

```
/// <summary>
/// Gets the specified container group and then prints a few of its properties and
/// their values.
/// </summary>
/// <param name="azure">An authenticated IAzure object.</param>
/// <param name="resourceGroupName">The name of the resource group containing the
/// container group.</param>
/// <param name="containerGroupName">The name of the container group whose details
/// should be printed.</param>
private static void PrintContainerGroupDetails(IAzure azure, string
resourceGroupName, string containerGroupName)
{
 Console.Write($"\\nGetting container group details for container group
'{containerGroupName}'...");

 .IContainerGroup containerGroup = null;
 while (containerGroup == null)
 {
 Console.Write(".");

 containerGroup =
azure.ContainerGroups.GetByResourceGroup(resourceGroupName, containerGroupName);

 SdkContext.DelayProvider.Delay(1000);
 }

 Console.WriteLine();
 Console.WriteLine(containerGroup.Name);
 Console.WriteLine("-----");
 Console.WriteLine($"State: {containerGroup.State}");
 Console.WriteLine($"FQDN: {containerGroup.Fqdn}");
 Console.WriteLine($"IP: {containerGroup.IPAddress}");
 Console.WriteLine($"Region: {containerGroup.RegionName}");
}
```

# Delete a container group

This example deletes a container group from a resource group.

C#

```
/// <summary>
/// Deletes the specified container group.
/// </summary>
/// <param name="azure">An authenticated IAzure object.</param>
/// <param name="resourceGroupName">The name of the resource group containing the
/// container group.</param>
/// <param name="containerGroupName">The name of the container group to delete.
/// </param>
private static void DeleteContainerGroup(IAzure azure, string resourceGroupName,
string containerGroupName)
{
 .IContainerGroup containerGroup = null;

 while (containerGroup == null)
 {
 containerGroup =
azure.ContainerGroups.GetByResourceGroup(resourceGroupName, containerGroupName);

 SdkContext.DelayProvider.Delay(1000);
 }

 Console.WriteLine($"Deleting container group '{containerGroupName}'...");
 azure.ContainerGroups.DeleteById(containerGroup.Id);
}
```

## API reference

[Explore the management APIs](#)

## Samples

- The source code for the preceding examples can be found on GitHub:

[Azure-Samples/aci-docs-sample-dotnet ↗](#)

- More Azure Container Instances code samples:

[Azure Code Samples ↗](#)

Explore more [sample .NET code ↗](#) you can use in your apps.



# Azure Container Instances libraries for Python

Article • 03/29/2022

Use the Microsoft Azure Container Instances libraries for Python to create and manage Azure Container Instances. Learn more by reading the [Azure Container Instances overview](#).

## Management APIs

Use the management library to create and manage Azure Container Instances in Azure.

Install the management package via pip:

Bash

```
pip install azure-mgmt-containerinstance
```

## Example source

If you'd like to see the following code examples in context, you can find them in the following GitHub repository:

[Azure-Samples/aci-docs-sample-python](#) ↗

## Authentication

One of the easiest ways to authenticate SDK clients (like the Azure Container Instances and Resource Manager clients in the following example) is with [file-based authentication](#). File-based authentication queries the `AZURE_AUTH_LOCATION` environment variable for the path to a credentials file. To use file-based authentication:

1. Create a credentials file with the [Azure CLI](#) or [Cloud Shell](#) ↗ :

Azure CLI

```
az ad sp create-for-rbac --role Contributor --sdk-auth > my.azureauth
```

If you use the [Cloud Shell](#) ↗ to generate the credentials file, copy its contents into a local file that your Python application can access.

2. Set the `AZURE_AUTH_LOCATION` environment variable to the full path of the generated credentials file. For example (in Bash):

```
Bash
```

```
export AZURE_AUTH_LOCATION=/home/yourusername/my.azureauth
```

Once you've created the credentials file and populated the `AZURE_AUTH_LOCATION` environment variable, use the `get_client_from_auth_file` method of the `client_factory` module to initialize the `ResourceManagementClient` and `ContainerInstanceManagementClient` objects.

```
Python
```

```
Authenticate the management clients with Azure.
Set the AZURE_AUTH_LOCATION environment variable to the full path to an
auth file. Generate an auth file with the Azure CLI or Cloud Shell:
az ad sp create-for-rbac --sdk-auth > my.azureauth
auth_file_path = getenv('AZURE_AUTH_LOCATION', None)
if auth_file_path is not None:
 print("Authenticating with Azure using credentials in file at {}".format(auth_file_path))

 aciclient = get_client_from_auth_file(
 ContainerInstanceManagementClient)
 resclient = get_client_from_auth_file(ResourceManagementClient)
else:
 print("\nFailed to authenticate to Azure. Have you set the"
```

For more details about the available authentication methods in the Python management libraries for Azure, see [Authenticate with the Azure Management Libraries for Python](#).

## Create container group - single container

This example creates a container group with a single container

```
Python
```

```
def create_container_group(aci_client, resource_group,
 container_group_name, container_image_name):
 """Creates a container group with a single container.

 Arguments:
 aci_client
 {azure.mgmt.containerinstance.ContainerInstanceManagementClient}
 -- An authenticated container instance management client.
 resource_group {azure.mgmt.resource.resources.ResourceGroup}
```

```

-- The resource group in which to create the container group.
container_group_name {str}
 -- The name of the container group to create.
container_image_name {str}
 -- The container image name and tag, for example:
 microsoft\aci-helloworld:latest
"""

print("Creating container group '{0}'...".format(container_group_name))

Configure the container
container_resource_requests = ResourceRequests(memory_in_gb=1, cpu=1.0)
container_resource_requirements = ResourceRequirements(
 requests=container_resource_requests)
container = Container(name=container_group_name,
 image=container_image_name,
 resources=container_resource_requirements,
 ports=[ContainerPort(port=80)])

Configure the container group
ports = [Port(protocol=ContainerGroupNetworkProtocol.tcp, port=80)]
group_ip_address = IpAddress(ports=ports,
 dns_name_label=container_group_name,
 type="Public")
group = ContainerGroup(location=resource_group.location,
 containers=[container],
 os_type=OperatingSystemTypes.linux,
 ip_address=group_ip_address)

Create the container group
aci_client.container_groups.create_or_update(resource_group.name,
 container_group_name,
 group)

Get the created container group
container_group = aci_client.container_groups.get(resource_group.name,
 container_group_name)

print("Once DNS has propagated, container group '{0}' will be reachable at"
 " http://{1}".format(container_group_name,

```

## Create container group - multiple containers

This example creates a container group with two containers: an application container and a sidecar container.

Python

```

def create_container_group_multi(aci_client, resource_group,
 container_group_name,
 container_image_1, container_image_2):

```



# Create task-based container group

This example creates a container group with a single task-based container. This example demonstrates several features:

- [Command line override](#) - A custom command line, different from that which is specified in the container's Dockerfile `CMD` line, is specified. Command line override allows you to specify a custom command line to execute at container startup, overriding the default command line baked-in to the container. Regarding executing multiple commands at container startup, the following applies:

If you want to run a **single command** with several command-line arguments, for example `echo FOO BAR`, you must supply them as a string list to the `command` property of the [Container](#). For example:

```
command = ['echo', 'FOO', 'BAR']
```

If, however, you want to run **multiple commands** with (potentially) multiple arguments, you must execute a shell and pass the chained commands as an argument. For example, this executes both an `echo` and a `tail` command:

```
command = ['/bin/sh', '-c', 'echo FOO BAR && tail -f /dev/null']
```

- [Environment variables](#) - Two environment variables are specified for the container in the container group. Use environment variables to modify script or application behavior at runtime, or otherwise pass dynamic information to an application running in the container.
- [Restart policy](#) - The container is configured with a restart policy of "Never," useful for task-based containers that are executed as part of a batch job.
- Operation polling with [AzureOperationPoller](#) - After the create method is invoked, the operation is polled to determine when it has completed and the container group's logs can be obtained.

Python

```
 container_group.ip_address.fqdn))

def run_task_based_container(aci_client, resource_group, container_group_name,
 container_image_name, start_command_line=None):
 """Creates a container group with a single task-based container who's
 restart policy is 'Never'. If specified, the container runs a custom
 command line at startup.
```

```
Arguments:
 aci_client
{azure.mgmt.containerinstance.ContainerInstanceManagementClient}
 -- An authenticated container instance management client.
 resource_group {azure.mgmt.resource.resources.models.ResourceGroup}
 -- The resource group in which to create the container group.
 container_group_name {str}
 -- The name of the container group to create.
 container_image_name {str}
 -- The container image name and tag, for example:
 microsoft\aci-helloworld:latest
 start_command_line {str}
 -- The command line that should be executed when the
 container starts. This value can be None.
 """
If a start command wasn't specified, use a default
if start_command_line is None:
 start_command_line = "python wordcount.py
http://shakespeare.mit.edu/romeo_juliet/full.html"

Configure some environment variables in the container which the
wordcount.py or other script can read to modify its behavior.
env_var_1 = EnvironmentVariable(name='NumWords', value='5')
env_var_2 = EnvironmentVariable(name='MinLength', value='8')

print("Creating container group '{0}' with start command '{1}'"
 .format(container_group_name, start_command_line))

Configure the container
container_resource_requests = ResourceRequests(memory_in_gb=1, cpu=1.0)
container_resource_requirements = ResourceRequirements(
 requests=container_resource_requests)
container = Container(name=container_group_name,
 image=container_image_name,
 resources=container_resource_requirements,
 command=start_command_line.split(),
 environment_variables=[env_var_1, env_var_2])

Configure the container group
group = ContainerGroup(location=resource_group.location,
 containers=[container],
 os_type=OperatingSystemTypes.linux,
 restart_policy=ContainerGroupRestartPolicy.never)

Create the container group
result = aci_client.container_groups.create_or_update(resource_group.name,
 container_group_name,
 group)

Wait for the container create operation to complete. The operation is
"done" when the container group provisioning state is one of:
Succeeded, Canceled, Failed
while result.done() is False:
 sys.stdout.write('.')
 time.sleep(1)
```

```

Get the provisioning state of the container group.
container_group = aci_client.container_groups.get(resource_group.name,
 container_group_name)
if str(container_group.provisioning_state).lower() == 'succeeded':
 print("\nCreation of container group '{}' succeeded."
 .format(container_group_name))
else:
 print("\nCreation of container group '{}' failed. Provisioning state"
 "is: {}".format(container_group_name,
 container_group.provisioning_state))

Get the logs for the container
logs = aci_client.container.list_logs(resource_group.name,
 container_group_name,

```

## List container groups

This example lists the container groups in a resource group and then prints a few of their properties.

When you list container groups, the `instance_view` of each returned group is `None`. To get the details of the containers within a container group, you must then `get` the container group, which returns the group with its `instance_view` property populated. See the next section, [Get an existing container group](#), for an example of iterating over a container group's containers in its `instance_view`.

Python

```

print("Logs for container '{0}':".format(container_group_name))
print("{0}".format(logs.content))

def list_container_groups(aci_client, resource_group):
 """Lists the container groups in the specified resource group.

 Arguments:
 aci_client {azure.mgmt.containerinstance.ContainerInstanceManagementClient}
 -- An authenticated container instance management client.
 resource_group {azure.mgmt.resource.resources.models.ResourceGroup}
 -- The resource group containing the container group(s).
 """
 print("Listing container groups in resource group '{0}'...".format(
 resource_group.name))

```

## Get an existing container group

This example gets a specific container group from a resource group, and then prints a few of its properties (including its containers) and their values.

The [get operation](#) returns a container group with its [instance\\_view](#) populated, which allows you to iterate over each container in the group. Only the `get` operation populates the `instance_view` property of the container group--listing the container groups in a subscription or resource group doesn't populate the instance view due to the potentially expensive nature of the operation (for example, when listing hundreds of container groups, each potentially containing multiple containers). As mentioned previously in the [List container groups](#) section, after a `list`, you must subsequently `get` a specific container group to obtain its container instance details.

Python

```
resource_group.name)

for container_group in container_groups:
 print(" {0}".format(container_group.name))

def print_container_group_details(aci_client, resource_group,
container_group_name):
 """Gets the specified container group and then prints a few of its properties
and their values.

 Arguments:
 aci_client
 {azure.mgmt.containerinstance.ContainerInstanceManagementClient}
 -- An authenticated container instance management client.
 resource_group {azure.mgmt.resource.resources.models.ResourceGroup}
 -- The name of the resource group containing the container
 group.
 container_group_name {str}
 -- The name of the container group whose details should be
 printed.
 """
 print("Getting container group details for container group '{0}'...".
 format(container_group_name))

 container_group = aci_client.container_groups.get(resource_group.name,
 container_group_name)
 print("-----")
 print("Name: {0}".format(container_group.name))
 print("State: {0}".format(container_group.provisioning_state))
 print("FQDN: {0}".format(container_group.ip_address.fqdn))
 print("IP: {0}".format(container_group.ip_address.ip))
 print("Region: {0}".format(container_group.location))
```

# Delete a container group

This example deletes several container groups from a resource group, as well as the resource group.

Python

```
Clean up resources
input("Press ENTER to delete all resources created by this sample: ")
aciClient.container_groups.delete(resource_group_name,
 container_group_name)
aciClient.container_groups.delete(resource_group_name,
 multi_container_group_name)
aciClient.container_groups.delete(resource_group_name,
 task_container_group_name)
resClient.resource_groups.delete(resource_group_name)
```

## Next steps

- The source code for the preceding examples can be found on GitHub:

[Azure-Samples/aci-docs-sample-python ↗](#)

- More Azure Container Instances code samples:

[Azure Code Samples ↗](#)

- Explore more [sample Python code ↗](#) you can use in your apps.

[Explore the management APIs](#)

# Container Instance

Reference

## Packages

 Expand table

Management

# Azure Container Instances SDK for JavaScript - latest

Article • 04/24/2025

## Packages - latest

 Expand table

Reference	Package	Source
<a href="#">Resource Management - Container Instances</a>	<a href="#">@azure/arm-containerinstance</a>	<a href="#">GitHub</a>

# Microsoft.ContainerInstance containerGroups

Article • 12/09/2024

## Bicep resource definition

The containerGroups resource type can be deployed with operations that target:

- **Resource groups** - See [resource group deployment commands](#)

For a list of changed properties in each API version, see [change log](#).

## Resource format

To create a Microsoft.ContainerInstance/containerGroups resource, add the following Bicep to your template.

Bicep

```
resource symbolicname 'Microsoft.ContainerInstance/containerGroups@2024-11-01-preview' = {
 identity: {
 type: 'string'
 userAssignedIdentities: {
 {customized property}: {}
 }
 }
 location: 'string'
 name: 'string'
 properties: {
 confidentialComputeProperties: {
 ccePolicy: 'string'
 }
 containers: [
 {
 name: 'string'
 properties: {
 command: [
 'string'
]
 environmentVariables: [
 {
 name: 'string'
 secureValue: 'string'
 secureValueReference: 'string'
 value: 'string'
 }
]
 }
 }
]
 }
}
```

```
 }
]
image: 'string'
livenessProbe: {
 exec: {
 command: [
 'string'
]
 }
failureThreshold: int
httpGet: {
 httpHeaders: [
 {
 name: 'string'
 value: 'string'
 }
]
 path: 'string'
 port: int
 scheme: 'string'
}
initialDelaySeconds: int
periodSeconds: int
successThreshold: int
timeoutSeconds: int
}
ports: [
 {
 port: int
 protocol: 'string'
 }
]
readinessProbe: {
 exec: {
 command: [
 'string'
]
 }
failureThreshold: int
httpGet: {
 httpHeaders: [
 {
 name: 'string'
 value: 'string'
 }
]
 path: 'string'
 port: int
 scheme: 'string'
}
initialDelaySeconds: int
periodSeconds: int
successThreshold: int
timeoutSeconds: int
}
```

```
resources: {
 limits: {
 cpu: int
 gpu: {
 count: int
 sku: 'string'
 }
 memoryInGB: int
 }
 requests: {
 cpu: int
 gpu: {
 count: int
 sku: 'string'
 }
 memoryInGB: int
 }
}
securityContext: {
 allowPrivilegeEscalation: bool
 capabilities: {
 add: [
 'string'
]
 drop: [
 'string'
]
 }
 privileged: bool
 runAsGroup: int
 runAsUser: int
 seccompProfile: 'string'
}
volumeMounts: [
 {
 mountPath: 'string'
 name: 'string'
 readOnly: bool
 }
]
}
diagnostics: {
 logAnalytics: {
 logType: 'string'
 metadata: {
 {customized property}: 'string'
 }
 workspaceId: 'string'
 workspaceKey: 'string'
 workspaceResourceId: 'string'
 }
}
dnsConfig: {
```

```
nameServers: [
 'string'
]
options: 'string'
searchDomains: 'string'
}
encryptionProperties: {
 identity: 'string'
 keyName: 'string'
 keyVersion: 'string'
 vaultBaseUrl: 'string'
}
extensions: [
{
 name: 'string'
 properties: {
 extensionType: 'string'
 protectedSettings: any(Azure.Bicep.Types.Concrete.AnyType)
 settings: any(Azure.Bicep.Types.Concrete.AnyType)
 version: 'string'
 }
}
]
identityAcls: {
 acls: [
{
 access: 'string'
 identity: 'string'
}
]
 defaultAccess: 'string'
}
imageRegistryCredentials: [
{
 identity: 'string'
 identityUrl: 'string'
 password: 'string'
 passwordReference: 'string'
 server: 'string'
 username: 'string'
}
]
initContainers: [
{
 name: 'string'
 properties: {
 command: [
 'string'
]
 environmentVariables: [
{
 name: 'string'
 secureValue: 'string'
 secureValueReference: 'string'
 value: 'string'
 }
]
}
]
```

```
 }
]
 image: 'string'
 securityContext: {
 allowPrivilegeEscalation: bool
 capabilities: {
 add: [
 'string'
]
 drop: [
 'string'
]
 }
 privileged: bool
 runAsGroup: int
 runAsUser: int
 seccompProfile: 'string'
 }
 volumeMounts: [
 {
 mountPath: 'string'
 name: 'string'
 readOnly: bool
 }
]
}
ipAddress: {
 autoGeneratedDomainNameLabelScope: 'string'
 dnsNameLabel: 'string'
 ip: 'string'
 ports: [
 {
 port: int
 protocol: 'string'
 }
]
 type: 'string'
}
osType: 'string'
priority: 'string'
restartPolicy: 'string'
secretReferences: [
 {
 identity: 'string'
 name: 'string'
 secretReferenceUri: 'string'
 }
]
sku: 'string'
subnetIds: [
 {
 id: 'string'
 name: 'string'
 }
]
```

```

 }
]
 volumes: [
 {
 azureFile: {
 readOnly: bool
 shareName: 'string'
 storageAccountKey: 'string'
 storageAccountKeyReference: 'string'
 storageAccountName: 'string'
 }
 emptyDir: any(Azure.Bicep.Types.Concrete.AnyType)
 gitRepo: {
 directory: 'string'
 repository: 'string'
 revision: 'string'
 }
 name: 'string'
 secret: {
 {customized property}: 'string'
 }
 secretReference: {
 {customized property}: 'string'
 }
 }
]
}
tags: {
 {customized property}: 'string'
}
zones: [
 'string'
]
}

```

## Property values

### AzureFileVolume

[\[\] Expand table](#)

Name	Description	Value
readOnly	The flag indicating whether the Azure File shared mounted as a volume is read-only.	bool
shareName	The name of the Azure File share to be mounted as a volume.	string (required)

Name	Description	Value
storageAccountKey	The storage account access key used to access the Azure File share.	string
storageAccountKeyReference	The reference to the storage account access key used to access the Azure File share.	string
storageAccountName	The name of the storage account that contains the Azure File share.	string (required)

## ConfidentialComputeProperties

[+] Expand table

Name	Description	Value
ccePolicy	The base64 encoded confidential compute enforcement policy	string

## Container

[+] Expand table

Name	Description	Value
name	The user-provided name of the container instance.	string (required)
properties	The properties of the container instance.	<a href="#">ContainerProperties</a> (required)

## ContainerExec

[+] Expand table

Name	Description	Value
command	The commands to execute within the container.	string[]

## ContainerGroupDiagnostics

[+] Expand table

Name	Description	Value
logAnalytics	Container group log analytics information.	<a href="#">LogAnalytics</a>

## ContainerGroupIdentity

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

Name	Description	Value
type	The type of identity used for the container group. The type 'SystemAssigned, UserAssigned' includes both an implicitly created identity and a set of user assigned identities. The type 'None' will remove any identities from the container group.	'None' 'SystemAssigned' 'SystemAssigned, UserAssigned' 'UserAssigned'
userAssignedIdentities	The list of user identities associated with the container group.	<a href="#">ContainerGroupIdentityUserAssignedIdentities</a>

## ContainerGroupIdentityUserAssignedIdentities

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

Name	Description	Value
------	-------------	-------

## ContainerGroupPropertiesProperties

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

Name	Description	Value
confidentialComputeProperties	The properties for confidential container group	<a href="#">ConfidentialComputeProperties</a>
containers	The containers within the	<a href="#">Container[]</a> (required)

Name	Description	Value
	container group.	
diagnostics	The diagnostic information for a container group.	<a href="#">ContainerGroupDiagnostics</a>
dnsConfig	The DNS config information for a container group.	<a href="#">DnsConfiguration</a>
encryptionProperties	The encryption properties for a container group.	<a href="#">EncryptionProperties</a>
extensions	extensions used by virtual kubelet	<a href="#">DeploymentExtensionSpec[]</a>
identityAcls	The access control levels of the identities.	<a href="#">IdentityAcls</a>
imageRegistryCredentials	The image registry credentials by which the container group is created from.	<a href="#">ImageRegistryCredential[]</a>
initContainers	The init containers for a container group.	<a href="#">InitContainerDefinition[]</a>
ipAddress	The IP address type of the container group.	<a href="#">IpAddress</a>
osType	The operating system type required by the containers in the container group.	'Linux' 'Windows' (required)
priority	The priority of the container group.	'Regular' 'Spot'
restartPolicy	Restart policy for all containers within the container group. - <code>Always</code> Always restart - <code>OnFailure</code> Restart on failure - <code>Never</code> Never restart	'Always' 'Never' 'OnFailure'
secretReferences	The secret references that will be referenced within the container group.	<a href="#">SecretReference[]</a>
sku	The SKU for a container group.	'Confidential' 'Dedicated' 'NotSpecified' 'Standard'

Name	Description	Value
subnetIds	The subnet resource IDs for a container group.	<a href="#">ContainerGroupSubnetId[]</a>
volumes	The list of volumes that can be mounted by containers in this container group.	<a href="#">Volume[]</a>

## ContainerGroupSubnetId

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

Name	Description	Value
id	Resource ID of virtual network and subnet.	string (required)
name	Friendly name for the subnet.	string

## ContainerHttpGet

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

Name	Description	Value
httpHeaders	The HTTP headers.	<a href="#">HttpHeader[]</a>
path	The path to probe.	string
port	The port number to probe.	int (required)
scheme	The scheme.	'http' 'https'

## ContainerPort

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

Name	Description	Value
port	The port number exposed within the container group.	int (required)
protocol	The protocol associated with the port.	'TCP' 'UDP'

## ContainerProbe

[+] Expand table

Name	Description	Value
exec	The execution command to probe	<a href="#">ContainerExec</a>
failureThreshold	The failure threshold.	int
httpGet	The Http Get settings to probe	<a href="#">ContainerHttpGet</a>
initialDelaySeconds	The initial delay seconds.	int
periodSeconds	The period seconds.	int
successThreshold	The success threshold.	int
timeoutSeconds	The timeout seconds.	int

## ContainerProperties

[+] Expand table

Name	Description	Value
command	The commands to execute within the container instance in exec form.	string[]
environmentVariables	The environment variables to set in the container instance.	<a href="#">EnvironmentVariable[]</a>
image	The name of the image used to create the container instance.	string (required)
livenessProbe	The liveness probe.	<a href="#">ContainerProbe</a>
ports	The exposed ports on the container instance.	<a href="#">ContainerPort[]</a>
readinessProbe	The readiness probe.	<a href="#">ContainerProbe</a>
resources	The resource requirements of the container instance.	<a href="#">ResourceRequirements</a> (required)
securityContext	The container security properties.	<a href="#">SecurityContextDefinition</a>
volumeMounts	The volume mounts available to the container instance.	<a href="#">VolumeMount[]</a>

## DeploymentExtensionSpec

[+] Expand table

Name	Description	Value
name	Name of the extension.	string (required)
properties	Extension specific properties	<a href="#">DeploymentExtensionSpecProperties</a>

## DeploymentExtensionSpecProperties

[+] Expand table

Name	Description	Value
extensionType	Type of extension to be added.	string (required)
protectedSettings	Protected settings for the extension.	any
settings	Settings for the extension.	any
version	Version of the extension being used.	string (required)

## DnsConfiguration

[+] Expand table

Name	Description	Value
nameServers	The DNS servers for the container group.	string[] (required)
options	The DNS options for the container group.	string
searchDomains	The DNS search domains for hostname lookup in the container group.	string

## EncryptionProperties

[+] Expand table

Name	Description	Value
identity	The keyvault managed identity.	string

Name	Description	Value
keyName	The encryption key name.	string (required)
keyVersion	The encryption key version.	string (required)
vaultBaseUrl	The keyvault base url.	string (required)

## EnvironmentVariable

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

Name	Description	Value
name	The name of the environment variable.	string (required)
secureValue	The value of the secure environment variable.	<p>string</p> <p>Constraints:</p> <p>Sensitive value. Pass in as a secure parameter.</p>
secureValueReference	The reference of the secure environment variable.	string
value	The value of the environment variable.	string

## GitRepoVolume

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

Name	Description	Value
directory	Target directory name. Must not contain or start with '..'. If '..' is supplied, the volume directory will be the git repository. Otherwise, if specified, the volume will contain the git repository in the subdirectory with the given name.	string
repository	Repository URL	<p>string</p> <p>(required)</p>
revision	Commit hash for the specified revision.	string

## GpuResource

[\[\] Expand table](#)

Name	Description	Value
count	The count of the GPU resource.	int (required)
sku	The SKU of the GPU resource.	'K80' 'P100' 'V100' (required)

## HttpHeader

[\[\] Expand table](#)

Name	Description	Value
name	The header name.	string
value	The header value.	string

## IdentityAccessControl

[\[\] Expand table](#)

Name	Description	Value
access	The access level of the identity.	'All' 'System' 'User'
identity	An identity.	string

## IdentityAcls

[\[\] Expand table](#)

Name	Description	Value
acls	The access control levels for each identity.	<a href="#">IdentityAccessControl[]</a>
defaultAccess	The default access level.	'All' 'System' 'User'

## ImageRegistryCredential

[+] [Expand table](#)

Name	Description	Value
identity	The identity for the private registry.	string
identityUrl	The identity URL for the private registry.	string
password	The password for the private registry.	string  Constraints: Sensitive value. Pass in as a secure parameter.
passwordReference	The reference for the private registry password.	string
server	The Docker image registry server without a protocol such as "http" and "https".	string (required)
username	The username for the private registry.	string

## InitContainerDefinition

[+] [Expand table](#)

Name	Description	Value
name	The name for the init container.	string (required)
properties	The properties for the init container.	<a href="#">InitContainerPropertiesDefinition</a> (required)

## InitContainerPropertiesDefinition

[+] [Expand table](#)

Name	Description	Value
command	The command to execute within the init container in exec form.	string[]
environmentVariables	The environment variables to set in the init container.	<a href="#">EnvironmentVariable[]</a>
image	The image of the init container.	string

Name	Description	Value
securityContext	The container security properties.	<a href="#">SecurityContextDefinition</a>
volumeMounts	The volume mounts available to the init container.	<a href="#">VolumeMount[]</a>

## IpAddress

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

Name	Description	Value
autoGeneratedDomainNameLabelScope	The value representing the security enum. The 'Unsecure' value is the default value if not selected and means the object's domain name label is not secured against subdomain takeover. The 'TenantReuse' value is the default value if selected and means the object's domain name label can be reused within the same tenant. The 'SubscriptionReuse' value means the object's domain name label can be reused within the same subscription. The 'ResourceGroupReuse' value means the object's domain name label can be reused within the same resource group. The 'NoReuse' value means the object's domain name label cannot be reused within the same resource group, subscription, or tenant.	'Noreuse' 'ResourceGroupReuse' 'SubscriptionReuse' 'TenantReuse' 'Unsecure'
dnsNameLabel	The Dns name label for the IP.	string
ip	The IP exposed to the public internet.	string
ports	The list of ports exposed on the container group.	<a href="#">Port[]</a> (required)

Name	Description	Value
type	Specifies if the IP is exposed to the public internet or private VNET.	'Private' 'Public' (required)

## LogAnalytics

[+] Expand table

Name	Description	Value
logType	The log type to be used.	'ContainerInsights' 'ContainerInstanceLogs'
metadata	Metadata for log analytics.	<a href="#">LogAnalyticsMetadata</a>
workspaceId	The workspace id for log analytics	string (required)
workspaceKey	The workspace key for log analytics	string  Constraints: Sensitive value. Pass in as a secure parameter. (required)
workspaceResourceId	The workspace resource id for log analytics	string  Constraints: Sensitive value. Pass in as a secure parameter.

## LogAnalyticsMetadata

[+] Expand table

Name	Description	Value

## Microsoft.ContainerInstance/containerGroups

[+] Expand table

Name	Description	Value
identity	The identity of the container group, if configured.	<a href="#">ContainerGroupIdentity</a>
location	The resource location.	string
name	The resource name	string (required)
properties	The container group properties	<a href="#">ContainerGroupPropertiesProperties</a> (required)
tags	Resource tags	Dictionary of tag names and values. See <a href="#">Tags in templates</a>
zones	The zones for the container group.	string[]

## Port

[+] [Expand table](#)

Name	Description	Value
port	The port number.	int (required)
protocol	The protocol associated with the port.	'TCP' 'UDP'

## ResourceLimits

[+] [Expand table](#)

Name	Description	Value
cpu	The CPU limit of this container instance.	int
gpu	The GPU limit of this container instance.	<a href="#">GpuResource</a>
memoryInGB	The memory limit in GB of this container instance.	int

## ResourceRequests

[+] [Expand table](#)

Name	Description	Value
cpu	The CPU request of this container instance.	int (required)
gpu	The GPU request of this container instance.	<a href="#">GpuResource</a>
memoryInGB	The memory request in GB of this container instance.	int (required)

## ResourceRequirements

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

Name	Description	Value
limits	The resource limits of this container instance.	<a href="#">ResourceLimits</a>
requests	The resource requests of this container instance.	<a href="#">ResourceRequests</a> (required)

## ResourceTags

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

Name	Description	Value

## SecretReference

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

Name	Description	Value
identity	The ARM resource id of the managed identity that has access to the secret in the key vault	string (required)
name	The identifier of the secret reference	string (required)
secretReferenceUri	The URI to the secret in key vault	string (required)

## SecretReferenceVolume

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

Name	Description	Value
------	-------------	-------

## SecretVolume

[+] [Expand table](#)

Name	Description	Value
------	-------------	-------

## SecurityContextCapabilitiesDefinition

[+] [Expand table](#)

Name	Description	Value
add	The capabilities to add to the container.	string[]
drop	The capabilities to drop from the container.	string[]

## SecurityContextDefinition

[+] [Expand table](#)

Name	Description	Value
allowPrivilegeEscalation	A boolean value indicating whether the init process can elevate its privileges	bool
capabilities	The capabilities to add or drop from a container.	<a href="#">SecurityContextCapabilitiesDefinition</a>
privileged	The flag to determine if the container permissions is elevated to Privileged.	bool
runAsGroup	Sets the User GID for the container.	int
runAsUser	Sets the User UID for the container.	int
seccompProfile	a base64 encoded string containing the contents of the JSON in the seccomp profile	string

# UserAssignedIdentities

[+] [Expand table](#)

Name	Description	Value

## Volume

[+] [Expand table](#)

Name	Description	Value
azureFile	The Azure File volume.	<a href="#">AzureFileVolume</a>
emptyDir	The empty directory volume.	any
gitRepo	The git repo volume.	<a href="#">GitRepoVolume</a>
name	The name of the volume.	string (required)
secret	The secret volume.	<a href="#">SecretVolume</a>
secretReference	The secret reference volume.	<a href="#">SecretReferenceVolume</a>

## VolumeMount

[+] [Expand table](#)

Name	Description	Value
mountPath	The path within the container where the volume should be mounted. Must not contain colon (:).	string (required)
name	The name of the volume mount.	string (required)
readOnly	The flag indicating whether the volume mount is read-only.	bool

## Quickstart samples

The following quickstart samples deploy this resource type.

[+] [Expand table](#)

Bicep File	Description
<a href="#">Azure Container Instances - BC with SQL Server and IIS</a>	Deploy a single Windows container with a fully featured self-contained Microsoft Dynamics 365 Business Central environment on Azure Container Instances.
<a href="#">Azure Container Instances - container with secrets</a>	Deploy a Linux container that has a secret volume using Azure Container Instances.
<a href="#">Azure Container Instances - Linux container with public IP</a>	Deploy a single Linux container accessible via a public IP using Azure Container Instances.
<a href="#">Azure Container Instances - VNet</a>	Deploy a container instance into an Azure virtual network.
<a href="#">Create a WordPress site</a>	This template creates a WordPress site on Container Instance
<a href="#">Create an on-demand SFTP Server with persistent storage</a>	This template demonstrates an on-demand SFTP server using an Azure Container Instance (ACI).
<a href="#">Create Application Gateway with Certificates</a>	This template shows how to generate Key Vault self-signed certificates, then reference from Application Gateway.
<a href="#">Front Door Standard/Premium with Azure Container Instances</a>	This template creates a Front Door Standard/Premium with a container group.
<a href="#">Front Door with Container Instances and Application Gateway</a>	This template creates a Front Door Standard/Premium with a container group and Application Gateway.
<a href="#">SQL Server availability group on AKS</a>	This creates a new AKS Cluster and then deploys SQL Server availability groups into it using a CNAB package deployed using Duffle and ACI

## Feedback

Was this page helpful?

 Yes

 No

# YAML reference: Azure Container Instances

Article • 08/29/2024

This article covers the syntax and properties for the YAML file supported by Azure Container Instances to configure a [container group](#). Use a YAML file to input the group configuration to the [az container create](#) command in the Azure CLI.

A YAML file is a convenient way to configure a container group for reproducible deployments. It's a concise alternative to using a [Resource Manager template](#) or the Azure Container Instances SDKs to create or update a container group.

## ⓘ Note

This reference applies to YAML files for Azure Container Instances REST API version 2021-10-01.

## Schema

The schema for the YAML file follows, including comments to highlight key properties. For a description of the properties in this schema, see the [Property values](#) section.

YAML

```
name: string # Name of the container group
apiVersion: '2021-10-01'
location: string
tags: {}
identity:
 type: string
 userAssignedIdentities: {}
properties: # Properties of container group
 containers: # Array of container instances in the group
 - name: string # Name of an instance
 properties: # Properties of an instance
 image: string # Container image used to create the instance
 command:
 - string
 ports: # External-facing ports exposed on the instance, must also be set in group ipAddress property
 - protocol: string
 port: integer
 environmentVariables:
 - name: string
 value: string
 secureValue: string
 resources: # Resource requirements of the instance
 requests:
 memoryInGB: number
 cpu: number
 gpu:
 count: integer
 sku: string
 limits:
 memoryInGB: number
 cpu: number
 gpu:
 count: integer
 sku: string
 volumeMounts: # Array of volume mounts for the instance
 - name: string
 mountPath: string
 readOnly: boolean
 livenessProbe:
 exec:
 command:
 - string
 httpGet:
 httpHeaders:
 - name: string
 value: string
 path: string
 port: integer
 scheme: string
 initialDelaySeconds: integer
 periodSeconds: integer
 failureThreshold: integer
 successThreshold: integer
```

```
 timeoutSeconds: integer
 readinessProbe:
 exec:
 command:
 - string
 httpGet:
 httpHeaders:
 - name: string
 value: string
 path: string
 port: integer
 scheme: string
 initialDelaySeconds: integer
 periodSeconds: integer
 failureThreshold: integer
 successThreshold: integer
 timeoutSeconds: integer
 imageRegistryCredentials: # Credentials to pull a private image
 - server: string
 username: string
 password: string
 identity: string
 identityUrl: string
 restartPolicy: string
 ipAddress: # IP address configuration of container group
 ports:
 - protocol: string
 port: integer
 type: string
 ip: string
 dnsNameLabel: string
 dnsNameLabelReusePolicy: string
 osType: string
 volumes: # Array of volumes available to the instances
 - name: string
 azureFile:
 shareName: string
 readOnly: boolean
 storageAccountName: string
 storageAccountKey: string
 emptyDir: {}
 secret: {}
 gitRepo:
 directory: string
 repository: string
 revision: string
 diagnostics:
 logAnalytics:
 workspaceId: string
 workspaceKey: string
 workspaceResourceId: string
 logType: string
 metadata: {}
 subnetIds: # Subnet to deploy the container group into
 - id: string
 name: string
 dnsConfig: # DNS configuration for container group
 nameServers:
 - string
 searchDomains: string
 options: string
 sku: string # SKU for the container group
 encryptionProperties:
 vaultBaseUrl: string
 keyName: string
 keyVersion: string
 initContainers: # Array of init containers in the group
 - name: string
 properties:
 image: string
 command:
 - string
 environmentVariables:
 - name: string
 value: string
 secureValue: string
 volumeMounts:
 - name: string
 mountPath: string
 readOnly: boolean
```

# Property values

The following tables describe the values you need to set in the schema.

## Microsoft.ContainerInstance/containerGroups object

[Expand table](#)

Name	Type	Required	Value
name	string	Yes	The name of the container group.
apiVersion	enum	Yes	2021-10-01 (latest), 2021-09-01, 2021-07-01, 2021-03-01, 2020-11-01, 2019-12-01, 2018-10-01, 2018-09-01, 2018-07-01, 2018-06-01, 2018-04-01
location	string	No	The resource location.
tags	object	No	The resource tags.
identity	object	No	The identity of the container group, if configured. - <a href="#">ContainerGroupIdentity object</a>
properties	object	Yes	<a href="#">ContainerGroupProperties object</a>

## ContainerGroupIdentity object

[Expand table](#)

Name	Type	Required	Value
type	enum	No	The type of identity used for the container group. The type 'SystemAssigned, UserAssigned' includes both an implicitly assigned user assigned identities. The type 'None' removes any identities from the container group. - SystemAssigned, UserAssigned, None
userAssignedIdentities	object	No	The list of user identities associated with the container group. The user identity dictionary key references are Azure Resource IDs in the form: '/subscriptions/{subscriptionId}/resourceGroups/{resourceGroupName}/providers/Microsoft.ManagedIdentity/userAssignedIdentities/{identityName}'

## ContainerGroupProperties object

[Expand table](#)

Name	Type	Required	Value
containers	array	Yes	The containers within the container group. - <a href="#">Container object</a>
imageRegistryCredentials	array	No	The image registry credentials by which the container group is created from. - <a href="#">ImageRegistryCredential object</a>
restartPolicy	enum	No	Restart policy for all containers within the container group. - <a href="#">Always</a> Always restart- <a href="#">OnFailure</a> Restart on failure- <a href="#">Never</a> Never restart. - Always, OnFailure, Never
ipAddress	object	No	The IP address type of the container group. - <a href="#">IpAddress object</a>
osType	enum	Yes	The operating system type required by the containers in the container group. - Windows or Linux
volumes	array	No	The list of volumes that containers in this container group can mount. - <a href="#">Volume object</a>
diagnostics	object	No	The diagnostic information for a container group. - <a href="#">ContainerGroupDiagnostics object</a>
subnetIds	object	No	The subnet information for a container group. - <a href="#">ContainerGroupSubnetIds object</a>
dnsConfig	object	No	The DNS config information for a container group. - <a href="#">DnsConfiguration object</a>
sku	enum	No	The SKU for a container group - Standard or Dedicated
encryptionProperties	object	No	The encryption properties for a container group. - <a href="#">EncryptionProperties object</a>
initContainers	array	No	The init containers for a container group. - <a href="#">InitContainerDefinition object</a>

## Container object

[Expand table](#)

Name	Type	Required	Value
name	string	Yes	The user-provided name of the container instance.
properties	object	Yes	The properties of the container instance. - <a href="#">ContainerProperties object</a>

## ImageRegistryCredential object

[Expand table](#)

Name	Type	Required	Value
server	string	Yes	The Docker image registry server without a protocol such as "http" and "https".
username	string	No	The username for the private registry.
password	string	No	The password for the private registry.
identity	string	No	The resource ID of the user or system-assigned managed identity used to authenticate.
identityUrl	string	No	The identity URL for the private registry.

## IpAddress object

[Expand table](#)

Name	Type	Required	Value
ports	array	Yes	The list of ports exposed on the container group. - <a href="#">Port object</a>
type	enum	Yes	Specifies if the IP is exposed to the public internet or private virtual network. - Public or Private
ip	string	No	The IP exposed to the public internet.
dnsNameLabel	string	No	The Dns name label for the IP.

## Volume object

[Expand table](#)

Name	Type	Required	Value
name	string	Yes	The name of the volume.
azureFile	object	No	The Azure File volume. - <a href="#">AzureFileVolume object</a>
emptyDir	object	No	The empty directory volume.
secret	object	No	The secret volume.
gitRepo	object	No	The git repo volume. - <a href="#">GitRepoVolume object</a>

## ContainerGroupDiagnostics object

[Expand table](#)

Name	Type	Required	Value
logAnalytics	object	No	Container group log analytics information. - <a href="#">LogAnalytics object</a>

## ContainerGroupSubnetIds object

[Expand table](#)

Name	Type	Required	Value
id	string	Yes	The identifier for a subnet.
name	string	No	The name of the subnet.

## DnsConfiguration object

[Expand table](#)

Name	Type	Required	Value
nameServers	array	Yes	The DNS servers for the container group. - string
searchDomains	string	No	The DNS search domains for hostname lookup in the container group.
options	string	No	The DNS options for the container group.

## EncryptionProperties object

[Expand table](#)

Name	Type	Required	Value
vaultBaseUrl	string	Yes	The keyvault base url.
keyName	string	Yes	The encryption key name.
keyVersion	string	Yes	The encryption key version.

## InitContainerDefinition object

[Expand table](#)

Name	Type	Required	Value
name	string	Yes	The name for the init container.
properties	object	Yes	The properties for the init container. - <a href="#">InitContainerPropertiesDefinition object</a>

## ContainerProperties object

[Expand table](#)

Name	Type	Required	Value
image	string	Yes	The name of the image used to create the container instance.
command	array	No	The commands to execute within the container instance in exec form. - string
ports	array	No	The exposed ports on the container instance. - <a href="#">ContainerPort object</a>
environmentVariables	array	No	The environment variables to set in the container instance. - <a href="#">EnvironmentVariable object</a>
resources	object	Yes	The resource requirements of the container instance. - <a href="#">ResourceRequirements object</a>
volumeMounts	array	No	The volume mounts available to the container instance. - <a href="#">VolumeMount object</a>
livenessProbe	object	No	The liveness probe. - <a href="#">ContainerProbe object</a>
readinessProbe	object	No	The readiness probe. - <a href="#">ContainerProbe object</a>

## Port object

[Expand table](#)

Name	Type	Required	Value
protocol	enum	No	The protocol associated with the port. - TCP or UDP
port	integer	Yes	The port number.

## AzureFileVolume object

[Expand table](#)

Name	Type	Required	Value
shareName	string	Yes	The name of the Azure File share to be mounted as a volume.
readOnly	boolean	No	The flag indicating whether the Azure File shared mounted as a volume is read-only.
storageAccountName	string	Yes	The name of the storage account that contains the Azure File share.
storageAccountKey	string	No	The storage account access key used to access the Azure File share.

## GitRepoVolume object

[Expand table](#)

Name	Type	Required	Value
directory	string	No	Target directory name. Must not contain or start with '..'. If '..' is supplied, the volume directory is the git repository. Otherwise, if specified, the volume contains the git repository in the subdirectory with the given name.
repository	string	Yes	Repository URL
revision	string	No	Commit hash for the specified revision.

## LogAnalytics object

[Expand table](#)

Name	Type	Required	Value
workspaceId	string	Yes	The workspace ID for log analytics
workspaceKey	string	Yes	The workspace key for log analytics
workspaceResourceId	string	No	The workspace resource ID for log analytics
logType	enum	No	The log type to be used. - ContainerInsights or ContainerInstanceLogs
metadata	object	No	Metadata for log analytics.

## InitContainerPropertiesDefinition object

[Expand table](#)

Name	Type	Required	Value
image	string	No	The image of the init container.
command	array	No	The command to execute within the init container in exec form. - string
environmentVariables	array	No	The environment variables to set in the init container. - <a href="#">EnvironmentVariable object</a>
volumeMounts	array	No	The volume mounts available to the init container. - <a href="#">VolumeMount object</a>

## ContainerPort object

[Expand table](#)

Name	Type	Required	Value
protocol	enum	No	The protocol associated with the port. - TCP or UDP
port	integer	Yes	The port number exposed within the container group.

## EnvironmentVariable object

[Expand table](#)

Name	Type	Required	Value
name	string	Yes	The name of the environment variable.
value	string	No	The value of the environment variable.
secureValue	string	No	The value of the secure environment variable.

## ResourceRequirements object

[Expand table](#)

Name	Type	Required	Value
requests	object	Yes	The resource requests of this container instance. - <a href="#">ResourceRequests object</a>
limits	object	No	The resource limits of this container instance. - <a href="#">ResourceLimits object</a>

## VolumeMount object

[Expand table](#)

Name	Type	Required	Value
name	string	Yes	The name of the volume mount.
mountPath	string	Yes	The path within the container where the volume should be mounted. Must not contain colon (:).
readOnly	boolean	No	The flag indicating whether the volume mount is read-only.

## ContainerProbe object

[Expand table](#)

Name	Type	Required	Value
exec	object	No	The execution command to probe - <a href="#">ContainerExec object</a>
httpGet	object	No	The Http Get settings to probe - <a href="#">ContainerHttpGet object</a>
initialDelaySeconds	integer	No	The initial delay seconds.
periodSeconds	integer	No	The period seconds.
failureThreshold	integer	No	The failure threshold.
successThreshold	integer	No	The success threshold.
timeoutSeconds	integer	No	The timeout seconds.

## ResourceRequests object

[Expand table](#)

Name	Type	Required	Value
memoryInGB	number	Yes	The memory request in GB of this container instance.

Name	Type	Required	Value
cpu	number	Yes	The CPU request of this container instance.
gpu	object	No	The GPU request of this container instance. - <a href="#">GpuResource object</a>

## ResourceLimits object

[Expand table](#)

Name	Type	Required	Value
memoryInGB	number	No	The memory limit in GB of this container instance.
cpu	number	No	The CPU limit of this container instance.
gpu	object	No	The GPU limit of this container instance. - <a href="#">GpuResource object</a>

## ContainerExec object

[Expand table](#)

Name	Type	Required	Value
command	array	No	The commands to execute within the container. - string

## ContainerHttpGet object

[Expand table](#)

Name	Type	Required	Value
path	string	No	The path to probe.
port	integer	Yes	The port number to probe.
scheme	enum	No	The scheme. - http or https
httpHeaders	object	No	The HTTP headers included in the probe. - <a href="#">HttpHeaders object</a>

## HttpHeaders object

[Expand table](#)

Name	Type	Required	Value
name	string	No	Name of the header.
value	string	No	Value of the header.

### Important

K80 and P100 GPU SKUs are retiring by August 31st, 2023. This is due to the retirement of the underlying VMs used: [NC Series](#) and [NCv2 Series](#). Although V100 SKUs will be available, it is recommended to use Azure Kubernetes Service instead. GPU resources are not fully supported and should not be used for production workloads. Use the following resources to migrate to AKS today: [How to Migrate to AKS](#).

## GpuResource object

[Expand table](#)

Name	Type	Required	Value
count	integer	Yes	The count of the GPU resource.

Name	Type	Required	Value
sku	enum	Yes	The SKU of the GPU resource. - V100

## Next steps

See the tutorial [Deploy a multi-container group using a YAML file](#).

See examples of using a YAML file to deploy container groups in a [virtual network](#) or that [mount an external volume](#).

---

## Feedback

Was this page helpful? [!\[\]\(804fe2980c4c05a62b87f95a3eabd0dc\_img.jpg\) Yes](#) [!\[\]\(36dbf282e076e9d0098c1bf0ec4fcb19\_img.jpg\) No](#)

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

# Azure Container Instance

Reference



Easily run containers on Azure with a single command. Create container groups, get the logs of a container and more.

This connector is available in the following products and regions:

[ ] Expand table

Service	Class	Regions
Copilot Studio	Premium	All <a href="#">Power Automate regions</a>
Logic Apps	Standard	All <a href="#">Logic Apps regions</a>
Power Apps	Premium	All <a href="#">Power Apps regions</a>
Power Automate	Premium	All <a href="#">Power Automate regions</a>

[ ] Expand table

Contact	
Name	Microsoft
URL	<a href="#">Microsoft LogicApps Support</a> <a href="#">Microsoft Power Automate Support</a> <a href="#">Microsoft Power Apps Support</a>

[ ] Expand table

Connector Metadata	
Publisher	Microsoft
Website	<a href="https://azure.microsoft.com/services/container-instances/">https://azure.microsoft.com/services/container-instances/</a>

## Creating a connection

The connector supports the following authentication types:

[ ] Expand table

Default	Parameters for creating connection.	All regions	Not shareable
---------	-------------------------------------	-------------	---------------

## Default

Applicable: All regions

Parameters for creating connection.

This is not shareable connection. If the power app is shared with another user, another user will be prompted to create new connection explicitly.

## Throttling Limits

[ ] Expand table

Name	Calls	Renewal Period
API calls per connection	100	60 seconds

# Actions

[Expand table](#)

Attach to the output of a specific container instance	Attach to the output stream of a specific container instance in a specified resource group and container group.
Create or update a container group	Create or update a container group with specified configurations.
Delete a container group	Delete the specified container group in the specified subscription and resource group. The operation does not delete other resources provided by the user, such as volumes.
Delete container group virtual network association links	Delete container group virtual network association links. The operation does not delete other resources provided by the user.
Executes a command in a specific container instance	Executes a command for a specific container instance in a specified resource group and container group.
Get a list of container groups in a resource group	Get a list of container groups in a specified subscription and resource group. This operation returns properties of each container group including containers, image registry credentials, restart policy, IP address type, OS type, state, and volumes.
Get a list of container groups in a subscription	Get a list of container groups in the specified subscription. This operation returns properties of each container group including containers, image registry credentials, restart policy, IP address type, OS type, state, and volumes.
Get all network dependencies for container group	Gets all the network dependencies for this container group to allow complete control of network setting and configuration. For container groups, this will always be an empty list.
Get cached images	Get the list of cached images on specific OS type for a subscription in a region.
Get capabilities	Get the list of CPU/memory/GPU capabilities for a subscription in a region.
Get current usage	Get the usage for a subscription
Get logs from a container instance	Get the logs for a specified container instance in a specified resource group and container group.
Get properties of a container group	Gets the properties of the specified container group in the specified subscription and resource group. The operation returns the properties of each container group including containers, image registry credentials, restart policy, IP address type, OS type, state, and volumes.
Restart containers in a container group	Restarts all containers in a container group in place. If container image has updates, new image will be downloaded.
Start containers in a container group	Starts all containers in a container group. Compute resources will be allocated and billing will start.
Stop containers in a container group	Stops all containers in a container group. Compute resources will be deallocated and billing will stop.
Update a container group location or tags	Update a container group with the specified tags and/or location.

## Attach to the output of a specific container instance

Operation ID: Containers\_Attach

Attach to the output stream of a specific container instance in a specified resource group and container group.

## Parameters

[Expand table](#)

Name	Key	Required	Type	Description
Subscription Id	subscriptionId	True	string	Subscription credentials which uniquely identify Microsoft Azure subscription. The subscription ID forms part of the URI for every service call.
Resource Group	resourceGroupName	True	string	The name of the resource group.
Container Group Name	containerGroupName	True	string	The name of the container group.

<b>containerName</b>	containerName	True	string	The name of the container instance.
----------------------	---------------	------	--------	-------------------------------------

## Returns

The information for the output stream from container attach.

**Body** ContainerAttachResponse

## Create or update a container group

Operation ID: ContainerGroups\_CreateOrUpdate

Create or update a container group with specified configurations.

## Parameters

[Expand table](#)

Name	Key	Required	Type	Description
<b>Subscription Id</b>	subscriptionId	True	string	Subscription credentials which uniquely identify Microsoft services.
<b>Resource Group</b>	resourceGroupName	True	string	The name of the resource group.
<b>Container Group Name</b>	containerGroupName	True	string	The name of the container group.
<b>id</b>	id		string	The resource id.
<b>name</b>	name		string	The resource name.
<b>type</b>	type		string	The resource type.
<b>Location</b>	location	True	string	The resource location.
<b>Tags</b>	tags		object	The resource tags.
<b>Zone</b>	zones		array of string	The zones for the container group.
<b>principalId</b>	principalId		string	The principal id of the container group identity. This property is required if the type is 'SystemAssigned'.
<b>tenantId</b>	tenantId		string	The tenant id associated with the container group. This property is required if the type is 'SystemAssigned'.
<b>Type</b>	type		string	The type of identity used for the container group. The user assigned identities. The type 'None' will remove a principalId and tenantId.
<b>User Assigned Identities</b>	userAssignedIdentities		object	The list of user identities associated with the container group. /subscriptions/{subscriptionId}/resourceGroups/{resourceGroupName}/providers/Microsoft.ContainerInstance/containerGroups/{containerGroupName}/userAssignedIdentities
<b>provisioningState</b>	provisioningState		string	The provisioning state of the container group. This only applies to the 'UserAssigned' type.
<b>Name</b>	name	True	string	The name of the container.
<b>Image</b>	image	True	string	The name of the image used to create the container. E.g. 'mcr.microsoft.com/azurillabs/testapp:1.0'
<b>Command</b>	command		array of string	The commands to execute within the container. E.g. [ "cmd1", "cmd2" ]
<b>Protocol</b>	protocol		string	The protocol associated with the port.
<b>Number</b>	port	True	integer	The port number exposed within the container group.
<b>Name</b>	name	True	string	The name of the environment variable.
<b>Value</b>	value		string	The value of the environment variable.
<b>secureValue</b>	secureValue		string	The value of the secure environment variable.
<b>restartCount</b>	restartCount		integer	The number of times that the container instance has been restarted.
<b>state</b>	state		string	The state of the container instance.

<b>startTime</b>	startTime	date-time	The date-time when the container instance state starts.	
<b>exitCode</b>	exitCode	integer	The container instance exit codes correspond to those.	
<b>finishTime</b>	finishTime	date-time	The date-time when the container instance state finish.	
<b>detailStatus</b>	detailStatus	string	The human-readable status of the container instance.	
<b>count</b>	count	integer	The count of the event.	
<b>firstTimestamp</b>	firstTimestamp	date-time	The date-time of the earliest logged event.	
<b>lastTimestamp</b>	lastTimestamp	date-time	The date-time of the latest logged event.	
<b>name</b>	name	string	The event name.	
<b>message</b>	message	string	The event message.	
<b>type</b>	type	string	The event type.	
<b>Memory</b>	memoryInGB	True	number	The memory request in GB of this container. E.g. 3.5
<b>CPU</b>	cpu	True	number	The CPU request of this container. E.g. 2.0
<b>Count</b>	count	integer	The count of the GPU resource.	
<b>SKU</b>	sku	string	The SKU of the GPU resource.	
<b>Memory</b>	memoryInGB	double	The memory limit in GB of this container.	
<b>CPU</b>	cpu	double	The CPU limit of this container.	
<b>Name</b>	name	True	string	The name of the volume.
<b>Path</b>	mountPath	True	string	The path within the container where the volume should be mounted.
<b> Readonly</b>	readOnly	boolean	The flag indicating whether the volume mount is read-only.	
<b>Command</b>	command	array of string	The commands to execute within the container.	
<b>Path</b>	path	string	The path to probe.	
<b>Port</b>	port	integer	The port number to probe.	
<b>Scheme</b>	scheme	string	The scheme.	
<b>name</b>	name	string	The header name.	
<b>value</b>	value	string	The header value.	
<b>Initial Delay</b>	initialDelaySeconds	integer	The initial delay seconds.	
<b>Period</b>	periodSeconds	integer	The period seconds.	
<b>Failure Threshold</b>	failureThreshold	integer	The failure threshold.	
<b>Success Threshold</b>	successThreshold	integer	The success threshold.	
<b>Timeout</b>	timeoutSeconds	integer	The timeout seconds.	
<b>privileged</b>	privileged	boolean	The flag to determine if the container permissions is elevated.	
<b>allowPrivilegeEscalation</b>	allowPrivilegeEscalation	boolean	A boolean value indicating whether the init process can escalate privileges.	
<b>add</b>	add	array of string	The capabilities to add to the container.	
<b>drop</b>	drop	array of string	The capabilities to drop from the container.	
<b>runAsGroup</b>	runAsGroup	integer	Sets the User GID for the container.	

<b>runAsUser</b>	runAsUser	integer	Sets the User UID for the container.
<b>seccompProfile</b>	seccompProfile	string	a base64 encoded string containing the contents of the Docker image's seccomp profile.
<b>Server</b>	server	True	string
<b>User Name</b>	username	string	The username for the private registry.
<b>Password</b>	password	string	The password for the private registry.
<b>Identity</b>	identity	string	The identity for the private registry.
<b>IdentityURL</b>	identityUrl	string	The identity URL for the private registry.
<b>restartPolicy</b>	restartPolicy	string	Restart policy for all containers within the container group.
			<ul style="list-style-type: none"> <li>• <b>Always</b> Always restart</li> <li>• <b>OnFailure</b> Restart on failure</li> <li>• <b>Never</b> Never restart</li> </ul>
<b>Protocol</b>	protocol	string	The protocol associated with the port.
<b>Number</b>	port	True	integer
<b>Type</b>	type	True	string
<b>ip</b>	ip	string	The IP exposed to the public internet.
<b>Dns Name</b>	dnsNameLabel	string	The Dns name label for the IP.
<b>autoGeneratedDomainNameLabelScope</b>	autoGeneratedDomainNameLabelScope	string	The value representing the security enum. The 'Unsecured' value means the domain name label is not secured against subdomain takeover. The 'TenantReuse' value means the object's domain name label can be reused within the same tenant. The 'SubscriptionReuse' value means the object's domain name label can be reused within the same subscription. The 'ResourceGroupReuse' value means the object's domain name label cannot be reused within the same resource group.
<b>fqdn</b>	fqdn	string	The FQDN for the IP.
<b>osType</b>	osType	True	string
<b>Name</b>	name	True	string
<b>Share Name</b>	shareName	True	string
<b>ReadOnly</b>	readOnly	boolean	The flag indicating whether the Azure File shared mount is read-only.
<b>Storage Account Name</b>	storageAccountName	True	string
<b>Storage Account Key</b>	storageAccountKey	string	The storage account access key used to access the Azure Storage account.
<b>Secret Volume</b>	secret	object	The secret volume.
<b>Directory</b>	directory	string	Target directory name. Must not contain or start with '.'. This volume will contain the git repository in the subdirectory.
<b>Repository</b>	repository	True	string
<b>Revision</b>	revision	string	Commit hash for the specified revision.
<b>state</b>	state	string	The state of the container group. Only valid in responses from the API.
<b>Workspace Id</b>	workspaceId	True	string
<b>Workspace Key</b>	workspaceKey	True	string
<b>Log Type</b>	logType	string	The log type to be used.
<b>Metadata</b>	metadata	object	Metadata for log analytics.
<b>workspaceResourceId</b>	workspaceResourceId	string	The workspace resource id for log analytics.
<b>id</b>	id	True	string
<b>name</b>	name	string	Friendly name for the subnet.

<b>Name Servers</b>	nameServers	True	array of string	The DNS servers for the container group.
<b>Search Domains</b>	searchDomains		string	The DNS search domains for hostname lookup in the container group.
<b>Options</b>	options		string	The DNS options for the container group.
<b>SKU</b>	sku		string	The container group SKU.
<b>Vault Url</b>	vaultBaseUrl	True	string	The keyvault base url.
<b>Key Name</b>	keyName	True	string	The encryption key name.
<b>Key Version</b>	keyVersion	True	string	The encryption key version.
<b>Identity</b>	identity		string	The keyvault managed identity.
<b>Name</b>	name	True	string	The name for the init container.
<b>Image</b>	image		string	The image of the init container.
<b>Command</b>	command		array of string	The command to execute within the init container in each pod.
<b>restartCount</b>	restartCount		integer	The number of times that the init container has been restarted.
<b>name</b>	name	True	string	Name of the extension.
<b>extensionType</b>	extensionType	True	string	Type of extension to be added.
<b>version</b>	version	True	string	Version of the extension being used.
<b>settings</b>	settings		object	Settings for the extension.
<b>protectedSettings</b>	protectedSettings		object	Protected settings for the extension.
<b>ccePolicy</b>	ccePolicy		string	The base64 encoded confidential compute enforcement policy.
<b>priority</b>	priority		string	The priority of the container group.

## Returns

A container group.

**ContainerGroup** [ContainerGroup](#)

## Delete a container group

Operation ID: ContainerGroups\_Delete

Delete the specified container group in the specified subscription and resource group. The operation does not delete other resources provided by the user, such as volumes.

## Parameters

[Expand table](#)

Name	Key	Required	Type	Description
<b>Subscription Id</b>	subscriptionId	True	string	Subscription credentials which uniquely identify Microsoft Azure subscription. The subscription ID forms part of the URI for every service call.
<b>Resource Group</b>	resourceGroupName	True	string	The name of the resource group.
<b>Container Group Name</b>	containerGroupName	True	string	The name of the container group.

## Returns

A container group.

**ContainerGroup** ContainerGroup

## Delete container group virtual network association links

Operation ID: SubnetServiceAssociationLink\_Delete

Delete container group virtual network association links. The operation does not delete other resources provided by the user.

### Parameters

 Expand table

Name	Key	Required	Type	Description
Subscription Id	subscriptionId	True	string	Subscription credentials which uniquely identify Microsoft Azure subscription. The subscription ID forms part of the URI for every service call.
Resource Group	resourceGroupName	True	string	The name of the resource group.
Virtual Network Name	virtualNetworkName	True	string	The name of the virtual network.
Subnet Name	subnetName	True	string	The name of the subnet.

## Executes a command in a specific container instance

Operation ID: Containers\_ExecuteCommand

Executes a command for a specific container instance in a specified resource group and container group.

### Parameters

 Expand table

Name	Key	Required	Type	Description
Subscription Id	subscriptionId	True	string	Subscription credentials which uniquely identify Microsoft Azure subscription. The subscription ID forms part of the URI for every service call.
Resource Group	resourceGroupName	True	string	The name of the resource group.
Container Group Name	containerGroupName	True	string	The name of the container group.
containerName	containerName	True	string	The name of the container instance.
command	command		string	The command to be executed.
rows	rows		integer	The row size of the terminal
cols	cols		integer	The column size of the terminal

### Returns

The information for the container exec command.

**Body** ContainerExecResponse

## Get a list of container groups in a resource group

Operation ID: ContainerGroups\_ListByResourceGroup

Get a list of container groups in a specified subscription and resource group. This operation returns properties of each container group including containers, image registry credentials, restart policy, IP address type, OS type, state, and volumes.

## Parameters

[Expand table](#)

Name	Key	Required	Type	Description
Subscription Id	subscriptionId	True	string	Subscription credentials which uniquely identify Microsoft Azure subscription. The subscription ID forms part of the URI for every service call.
Resource Group	resourceGroupName	True	string	The name of the resource group.

## Returns

The container group list response that contains the container group properties.

**Body** [ContainerGroupListResult](#)

## Get a list of container groups in a subscription

Operation ID: ContainerGroups\_List

Get a list of container groups in the specified subscription. This operation returns properties of each container group including containers, image registry credentials, restart policy, IP address type, OS type, state, and volumes.

## Parameters

[Expand table](#)

Name	Key	Required	Type	Description
Subscription Id	subscriptionId	True	string	Subscription credentials which uniquely identify Microsoft Azure subscription. The subscription ID forms part of the URI for every service call.

## Returns

The container group list response that contains the container group properties.

**Body** [ContainerGroupListResult](#)

## Get all network dependencies for container group

Operation ID: ContainerGroups\_GetOutboundNetworkDependenciesEndpoints

Gets all the network dependencies for this container group to allow complete control of network setting and configuration. For container groups, this will always be an empty list.

## Parameters

[Expand table](#)

Name	Key	Required	Type	Description
Subscription Id	subscriptionId	True	string	Subscription credentials which uniquely identify Microsoft Azure subscription. The subscription ID forms part of the URI for every service call.
Resource Group	resourceGroupName	True	string	The name of the resource group.
Container Group Name	containerGroupName	True	string	The name of the container group.

## Returns

Response for network dependencies, always empty list.

Items NetworkDependenciesResponse

## Get cached images

Operation ID: Location\_ListCachedImages

Get the list of cached images on specific OS type for a subscription in a region.

### Parameters

 Expand table

Name	Key	Required	Type	Description
Subscription Id	subscriptionId	True	string	Subscription credentials which uniquely identify Microsoft Azure subscription. The subscription ID forms part of the URI for every service call.
Location	location	True	string	The identifier for the physical azure location.

### Returns

The response containing cached images.

Body CachedImagesListResult

## Get capabilities

Operation ID: Location\_ListCapabilities

Get the list of CPU/memory/GPU capabilities for a subscription in a region.

### Parameters

 Expand table

Name	Key	Required	Type	Description
Subscription Id	subscriptionId	True	string	Subscription credentials which uniquely identify Microsoft Azure subscription. The subscription ID forms part of the URI for every service call.
Location	location	True	string	The identifier for the physical azure location.

### Returns

The response containing list of capabilities.

Body CapabilitiesListResult

## Get current usage

Operation ID: Location\_ListUsage

Get the usage for a subscription

### Parameters

 Expand table

Name	Key	Required	Type	Description

<b>Subscription Id</b>	subscriptionId	True	string	Subscription credentials which uniquely identify Microsoft Azure subscription. The subscription ID forms part of the URI for every service call.
<b>Location</b>	location	True	string	The identifier for the physical azure location.

## Returns

The response containing the usage data

**Body** [UsageListResult](#)

## Get logs from a container instance

Operation ID: ContainerLogs\_List

Get the logs for a specified container instance in a specified resource group and container group.

## Parameters

[Expand table](#)

Name	Key	Required	Type	Description
<b>Subscription Id</b>	subscriptionId	True	string	Subscription credentials which uniquely identify Microsoft Azure subscription. The subscription ID forms part of the URI for every service call.
<b>Resource Group</b>	resourceGroupName	True	string	The name of the resource group.
<b>Container Group Name</b>	containerGroupName	True	string	The name of the container group.
<b>Container Name</b>	containerName	True	string	The name of the container instance.
<b>Tail Lines</b>	tail		integer	The number of lines to show from the tail of the container instance log. If not provided, all available logs are shown up to 4mb.

## Returns

The logs.

**Body** [Logs](#)

## Get properties of a container group

Operation ID: ContainerGroups\_Get

Gets the properties of the specified container group in the specified subscription and resource group. The operation returns the properties of each container group including containers, image registry credentials, restart policy, IP address type, OS type, state, and volumes.

## Parameters

[Expand table](#)

Name	Key	Required	Type	Description
<b>Subscription Id</b>	subscriptionId	True	string	Subscription credentials which uniquely identify Microsoft Azure subscription. The subscription ID forms part of the URI for every service call.
<b>Resource Group</b>	resourceGroupName	True	string	The name of the resource group.
<b>Container Group Name</b>	containerGroupName	True	string	The name of the container group.

## Returns

A container group.

**ContainerGroup** ContainerGroup

## Restart containers in a container group

Operation ID: ContainerGroups\_Restart

Restarts all containers in a container group in place. If container image has updates, new image will be downloaded.

### Parameters

[Expand table](#)

Name	Key	Required	Type	Description
Subscription Id	subscriptionId	True	string	Subscription credentials which uniquely identify Microsoft Azure subscription. The subscription ID forms part of the URI for every service call.
Resource Group	resourceGroupName	True	string	The name of the resource group.
Container Group Name	containerGroupName	True	string	The name of the container group.

## Start containers in a container group

Operation ID: ContainerGroups\_Start

Starts all containers in a container group. Compute resources will be allocated and billing will start.

### Parameters

[Expand table](#)

Name	Key	Required	Type	Description
Subscription Id	subscriptionId	True	string	Subscription credentials which uniquely identify Microsoft Azure subscription. The subscription ID forms part of the URI for every service call.
Resource Group	resourceGroupName	True	string	The name of the resource group.
Container Group Name	containerGroupName	True	string	The name of the container group.

## Stop containers in a container group

Operation ID: ContainerGroups\_Stop

Stops all containers in a container group. Compute resources will be deallocated and billing will stop.

### Parameters

[Expand table](#)

Name	Key	Required	Type	Description
Subscription Id	subscriptionId	True	string	Subscription credentials which uniquely identify Microsoft Azure subscription. The subscription ID forms part of the URI for every service call.
Resource Group	resourceGroupName	True	string	The name of the resource group.
Container Group Name	containerGroupName	True	string	The name of the container group.

## Update a container group location or tags

Operation ID: ContainerGroups\_Update

Update a container group with the specified tags and/or location.

## Parameters

[Expand table](#)

Name	Key	Required	Type	Description
Subscription Id	subscriptionId	True	string	Subscription credentials which uniquely identify Microsoft Azure subscription. The subscription ID forms part of the URI for every service call.
Resource Group	resourceGroupName	True	string	The name of the resource group.
Container Group Name	containerGroupName	True	string	The name of the container group.
id	id		string	The resource id.
name	name		string	The resource name.
type	type		string	The resource type.
Location	location		string	The resource location.
Tags	tags		object	The resource tags.
zones	zones		array of string	The zones for the container group.

## Returns

A container group.

**ContainerGroup** [ContainerGroup](#)

## Definitions

### Container

A container instance.

[Expand table](#)

Name	Path	Type	Description
Name	name	string	The name of the container.
properties	properties	ContainerProperties	The container properties.

### ContainerProperties

The container properties.

[Expand table](#)

Name	Path	Type	Description
Image	image	string	The name of the image used to create the container. E.g. nginx:latest
Command	command	array of string	The commands to execute within the container. E.g. [ "echo", "hello aci" ]
Ports	ports	array of ContainerPort	The exposed ports on the container instance.

<b>Environment Variables</b>	environmentVariables	array of EnvironmentVariable	The environment variables to set in the container instance.
<b>restartCount</b>	instanceView.restartCount	integer	The number of times that the container instance has been restarted.
<b>currentState</b>	instanceView.currentState	ContainerState	The container instance state.
<b>previousState</b>	instanceView.previousState	ContainerState	The container instance state.
<b>events</b>	instanceView.events	array of Event	The events of the container instance.
<b>resources</b>	resources	ResourceRequirements	The resource requirements.
<b>Volume Mounts</b>	volumeMounts	array of VolumeMount	The volume mounts available to the container instance.
<b>Liveness Probe</b>	livenessProbe	ContainerProbe	The container probe, for liveness
<b>Readiness Probe</b>	readinessProbe	ContainerProbe	The container probe, for liveness
<b>securityContext</b>	securityContext	SecurityContextDefinition	The security context for the container.

## SecurityContextDefinition

The security context for the container.

[Expand table](#)

Name	Path	Type	Description
<b>privileged</b>	privileged	boolean	The flag to determine if the container permissions is elevated to Privileged.
<b>allowPrivilegeEscalation</b>	allowPrivilegeEscalation	boolean	A boolean value indicating whether the init process can elevate its privileges
<b>capabilities</b>	capabilities	SecurityContextCapabilitiesDefinition	The capabilities to add or drop from a container.
<b>runAsGroup</b>	runAsGroup	integer	Sets the User GID for the container.
<b>runAsUser</b>	runAsUser	integer	Sets the User UID for the container.
<b>seccompProfile</b>	seccompProfile	string	a base64 encoded string containing the contents of the JSON in the seccomp profile

## SecurityContextCapabilitiesDefinition

The capabilities to add or drop from a container.

[Expand table](#)

Name	Path	Type	Description
<b>add</b>	add	array of string	The capabilities to add to the container.
<b>drop</b>	drop	array of string	The capabilities to drop from the container.

## ContainerState

The container instance state.

[Expand table](#)

Name	Path	Type	Description
<b>state</b>	state	string	The state of the container instance.
<b>startTime</b>	startTime	date-time	The date-time when the container instance state started.

<code>exitCode</code>	exitCode	integer	The container instance exit codes correspond to those from the <code>docker run</code> command.
<code>finishTime</code>	finishTime	date-time	The date-time when the container instance state finished.
<code>detailStatus</code>	detailStatus	string	The human-readable status of the container instance state.

## Event

A container group or container instance event.

[Expand table](#)

Name	Path	Type	Description
<code>count</code>	count	integer	The count of the event.
<code>firstTimestamp</code>	firstTimestamp	date-time	The date-time of the earliest logged event.
<code>lastTimestamp</code>	lastTimestamp	date-time	The date-time of the latest logged event.
<code>name</code>	name	string	The event name.
<code>message</code>	message	string	The event message.
<code>type</code>	type	string	The event type.

## ResourceRequirements

The resource requirements.

[Expand table](#)

Name	Path	Type	Description
<code>Resource Requests</code>	requests	ResourceRequests	The resource requests.
<code>Resource Limits</code>	limits	ResourceLimits	The resource limits.

## ResourceRequests

The resource requests.

[Expand table](#)

Name	Path	Type	Description
<code>Memory</code>	memoryInGB	number	The memory request in GB of this container. E.g. 3.5
<code>CPU</code>	cpu	number	The CPU request of this container. E.g. 2.0
<code>GPU</code>	gpu	GpuResource	The GPU resource.

## ResourceLimits

The resource limits.

[Expand table](#)

Name	Path	Type	Description
<code>Memory</code>	memoryInGB	double	The memory limit in GB of this container.
<code>CPU</code>	cpu	double	The CPU limit of this container.
<code>GPU</code>	gpu	GpuResource	The GPU resource.

## GpuResource

The GPU resource.

[Expand table](#)

Name	Path	Type	Description
Count	count	integer	The count of the GPU resource.
SKU	sku	string	The SKU of the GPU resource.

## AzureFileVolume

The properties of the Azure File volume. Azure File shares are mounted as volumes.

[Expand table](#)

Name	Path	Type	Description
Share Name	shareName	string	The name of the Azure File share to be mounted as a volume.
ReadOnly	readOnly	boolean	The flag indicating whether the Azure File shared mounted as a volume is read-only.
Storage Account Name	storageAccountName	string	The name of the storage account that contains the Azure File share.
Storage Account Key	storageAccountKey	string	The storage account access key used to access the Azure File share.

## EmptyDirVolume

The empty directory volume.

## SecretVolume

The secret volume.

[Expand table](#)

Name	Path	Type	Description
Secret Volume		object	The secret volume.

## GitRepoVolume

Represents a volume that is populated with the contents of a git repository

[Expand table](#)

Name	Path	Type	Description
Directory	directory	string	Target directory name. Must not contain or start with '..'. If '.' is supplied, the volume directory will be the git repository. Otherwise, if specified, the volume will contain the git repository in the subdirectory with the given name.
Repository	repository	string	Repository URL
Revision	revision	string	Commit hash for the specified revision.

## Volume

The properties of the volume.

[Expand table](#)

Name	Path	Type	Description
------	------	------	-------------

<b>Name</b>	name	string	The name of the volume.
<b>Azure File Volume</b>	azureFile	AzureFileVolume	The properties of the Azure File volume. Azure File shares are mounted as volumes.
<b>Empty Directory Volume</b>	emptyDir	EmptyDirVolume	The empty directory volume.
<b>Secret Volume</b>	secret	SecretVolume	The secret volume.
<b>Git Repo Volume</b>	gitRepo	GitRepoVolume	Represents a volume that is populated with the contents of a git repository

## VolumeMount

The properties of the volume mount.

[Expand table](#)

Name	Path	Type	Description
<b>Name</b>	name	string	The name of the volume.
<b>Path</b>	mountPath	string	The path within the container where the volume should be mounted. Must not contain colon (:).
<b>ReadOnly</b>	readOnly	boolean	The flag indicating whether the volume mount is read-only.

## ContainerExec

The container execution command, for liveness or readiness probe

[Expand table](#)

Name	Path	Type	Description
<b>Command</b>	command	array of string	The commands to execute within the container.

## ContainerHttpGet

The container Http Get settings, for liveness or readiness probe

[Expand table](#)

Name	Path	Type	Description
<b>Path</b>	path	string	The path to probe.
<b>Port</b>	port	integer	The port number to probe.
<b>Scheme</b>	scheme	string	The scheme.
<b>httpHeaders</b>	httpHeaders	array of HttpHeaders	The HTTP headers.

## ContainerProbe

The container probe, for liveness

[Expand table](#)

Name	Path	Type	Description
<b>Exec</b>	exec	ContainerExec	The container execution command, for liveness or readiness probe
<b>HTTP GET</b>	httpGet	ContainerHttpGet	The container Http Get settings, for liveness or readiness probe
<b>Initial Delay</b>	initialDelaySeconds	integer	The initial delay seconds.
<b>Period</b>	periodSeconds	integer	The period seconds.

<b>Failure Threshold</b>	failureThreshold	integer	The failure threshold.
<b>Success Threshold</b>	successThreshold	integer	The success threshold.
<b>Timeout</b>	timeoutSeconds	integer	The timeout seconds.

## ContainerGroup

A container group.

[Expand table](#)

Name	Path	Type	Description
<b>id</b>	id	string	The resource id.
<b>name</b>	name	string	The resource name.
<b>type</b>	type	string	The resource type.
<b>Location</b>	location	string	The resource location.
<b>Tags</b>	tags	object	The resource tags.
<b>Zone</b>	zones	array of string	The zones for the container group.
<b>Managed Identity</b>	identity	ContainerGroupIdentity	Identity for the container group.
<b>properties</b>	properties	ContainerGroupProperties	The container group properties

## ContainerGroupProperties

The container group properties

[Expand table](#)

Name	Path	Type	Description
<b>provisioningState</b>	provisioningState	string	The provisioning state of the container group. This only appears in the response.
<b>containers</b>	containers	array of Container	The containers within the container group.
<b>imageRegistryCredentials</b>	imageRegistryCredentials	array of ImageRegistryCredential	The image registry credentials by which the container group is created from.
<b>restartPolicy</b>	restartPolicy	string	Restart policy for all containers within the container group. <ul style="list-style-type: none"> <li>• <b>Always</b> Always restart</li> <li>• <b>OnFailure</b> Restart on failure</li> <li>• <b>Never</b> Never restart</li> </ul>
<b>IP Address</b>	ipAddress	IpAddress	IP address for the container group.
<b>osType</b>	osType	string	The operating system type required by the containers in the container group.
<b>volumes</b>	volumes	array of Volume	The list of volumes that can be mounted by containers in this container group.
<b>events</b>	instanceView.events	array of Event	The events of this container group.
<b>state</b>	instanceView.state	string	The state of the container group. Only valid in response.
<b>diagnostics</b>	diagnostics	ContainerGroupDiagnostics	Container group diagnostic information.
<b>subnetIds</b>	subnetIds	array of ContainerGroupSubnetId	The subnet resource IDs for a container group.

<b>DNS Configuration</b>	dnsConfig	DnsConfiguration	DNS configuration for the container group.
<b>SKU</b>	sku	ContainerGroupSku	The container group SKU.
<b>Encryption</b>	encryptionProperties	EncryptionProperties	The container group encryption properties.
<b>initContainers</b>	initContainers	array of InitContainerDefinition	The init containers for a container group.
<b>extensions</b>	extensions	array of DeploymentExtensionSpec	extensions used by virtual kubelet
<b>confidentialComputeProperties</b>	confidentialComputeProperties	ConfidentialComputeProperties	The properties for confidential container group
<b>priority</b>	priority	string	The priority of the container group.

## DeploymentExtensionSpec

Extension sidecars to be added to the deployment.

[Expand table](#)

Name	Path	Type	Description
<b>name</b>	name	string	Name of the extension.
<b>extensionType</b>	properties.extensionType	string	Type of extension to be added.
<b>version</b>	properties.version	string	Version of the extension being used.
<b>settings</b>	properties.settings	object	Settings for the extension.
<b>protectedSettings</b>	properties.protectedSettings	object	Protected settings for the extension.

## ContainerGroupIdentity

Identity for the container group.

[Expand table](#)

Name	Path	Type	Description
<b>principalId</b>	principalId	string	The principal id of the container group identity. This property will only be provided for a system assigned identity.
<b>tenantId</b>	tenantId	string	The tenant id associated with the container group. This property will only be provided for a system assigned identity.
<b>Type</b>	type	string	The type of identity used for the container group. The type 'SystemAssigned, UserAssigned' includes both an implicitly assigned identities. The type 'None' will remove any identities from the container group.
<b>User Assigned Identities</b>	userAssignedIdentities	object	The list of user identities associated with the container group. The user identity dictionary key references will be of the form '/subscriptions/{subscriptionId}/resourceGroups/{resourceGroupName}/providers/Microsoft.ManagedIdentity/userAssignedIdentities/{identityName}'.

## ImageRegistryCredential

Image registry credential.

[Expand table](#)

Name	Path	Type	Description
<b>Server</b>	server	string	The Docker image registry server without a protocol such as "myacr.azurecr.io".
<b>User Name</b>	username	string	The username for the private registry.
<b>Password</b>	password	string	The password for the private registry.
<b>Identity</b>	identity	string	The identity for the private registry.

<b>IdentityUrl</b>	identityUrl	string	The identity URL for the private registry.
--------------------	-------------	--------	--------------------------------------------

## ContainerGroupDiagnostics

Container group diagnostic information.

[Expand table](#)

Name	Path	Type	Description
<b>Log Analytics</b>	logAnalytics	LogAnalytics	Container group log analytics information.

## LogAnalytics

Container group log analytics information.

[Expand table](#)

Name	Path	Type	Description
<b>Workspace Id</b>	workspaceId	string	The workspace id for log analytics
<b>Workspace Key</b>	workspaceKey	string	The workspace key for log analytics
<b>Log Type</b>	logType	string	The log type to be used.
<b>Metadata</b>	metadata	object	Metadata for log analytics.
<b>workspaceResourceId</b>	workspaceResourceId	string	The workspace resource id for log analytics

## ContainerGroupSubnetId

Container group subnet information.

[Expand table](#)

Name	Path	Type	Description
<b>id</b>	id	string	Resource ID of virtual network and subnet.
<b>name</b>	name	string	Friendly name for the subnet.

## IpAddress

IP address for the container group.

[Expand table](#)

Name	Path	Type	Description
<b>Ports</b>	ports	array of Port	The list of ports exposed on the container group.
<b>Type</b>	type	string	Specifies if the IP is exposed to the public internet.
<b>ip</b>	ip	string	The IP exposed to the public internet.
<b>Dns Name</b>	dnsNameLabel	string	The Dns name label for the IP.
<b>autoGeneratedDomainNameLabelScope</b>	autoGeneratedDomainNameLabelScope	string	The value representing the security enum. The 'Unsecure' value is the default value if not selected and means the object's domain name label is not secured against subdomain takeover. The 'TenantReuse' value is the default value if selected and means the object's domain name label can be reused within the same tenant. The

			'SubscriptionReuse' value means the object's domain name label can be reused within the same subscription. The 'ResourceGroupReuse' value means the object's domain name label can be reused within the same resource group. The 'NoReuse' value means the object's domain name label cannot be reused within the same resource group, subscription, or tenant.
<b>fqdn</b>	fqdn	string	The FQDN for the IP.

## Port

The port exposed on the container group.

[Expand table](#)

Name	Path	Type	Description
<b>Protocol</b>	protocol	string	The protocol associated with the port.
<b>Number</b>	port	integer	The port number.

## ContainerPort

The port exposed on the container instance.

[Expand table](#)

Name	Path	Type	Description
<b>Protocol</b>	protocol	string	The protocol associated with the port.
<b>Number</b>	port	integer	The port number exposed within the container group.

## EnvironmentVariable

The environment variable to set within the container instance.

[Expand table](#)

Name	Path	Type	Description
<b>Name</b>	name	string	The name of the environment variable.
<b>Value</b>	value	string	The value of the environment variable.
<b>secureValue</b>	secureValue	string	The value of the secure environment variable.

## UsageListResult

The response containing the usage data

[Expand table](#)

Name	Path	Type	Description
<b>value</b>	value	array of Usage	The usage data.

## Usage

A single usage result

[Expand table](#)

Name	Path	Type	Description
id	id	string	Id of the usage result
unit	unit	string	Unit of the usage result
currentValue	currentValue	integer	The current usage of the resource
limit	limit	integer	The maximum permitted usage of the resource.
value	name.value	string	The name of the resource
localizedValue	name.localizedValue	string	The localized name of the resource

## ContainerGroupListResult

The container group list response that contains the container group properties.

[Expand table](#)

Name	Path	Type	Description
value	value	array of ContainerGroup	The list of container groups.
nextLink	nextLink	string	The URI to fetch the next page of container groups.

## Logs

The logs.

[Expand table](#)

Name	Path	Type	Description
content	content	string	The content of the log.

## ContainerExecResponse

The information for the container exec command.

[Expand table](#)

Name	Path	Type	Description
webSocketUri	webSocketUri	string	The uri for the exec websocket.
password	password	string	The password to start the exec command.

## ContainerAttachResponse

The information for the output stream from container attach.

[Expand table](#)

Name	Path	Type	Description
webSocketUri	webSocketUri	string	The uri for the output stream from the attach.
password	password	string	The password to the output stream from the attach. Send as an Authorization header value when connecting to the websocketUri.

## HttpHeader

The HTTP header.

[Expand table](#)

Name	Path	Type	Description
name	name	string	The header name.
value	value	string	The header value.

## DnsConfiguration

DNS configuration for the container group.

[Expand table](#)

Name	Path	Type	Description
Name Servers	nameServers	array of string	The DNS servers for the container group.
Search Domains	searchDomains	string	The DNS search domains for hostname lookup in the container group.
Options	options	string	The DNS options for the container group.

## CachedImagesListResult

The response containing cached images.

[Expand table](#)

Name	Path	Type	Description
value	value	array of cachedImages	The list of cached images.
nextLink	nextLink	string	The URI to fetch the next page of cached images.

## cachedImages

The cached image and OS type.

[Expand table](#)

Name	Path	Type	Description
osType	osType	string	The OS type of the cached image.
image	image	string	The cached image name.

## CapabilitiesListResult

The response containing list of capabilities.

[Expand table](#)

Name	Path	Type	Description
value	value	array of Capabilities	The list of capabilities.
nextLink	nextLink	string	The URI to fetch the next page of capabilities.

## Capabilities

The regional capabilities.

[Expand table](#)

Name	Path	Type	Description
resourceType	resourceType	string	The resource type that this capability describes.
osType	osType	string	The OS type that this capability describes.
location	location	string	The resource location.
ipAddressType	ipAddressType	string	The ip address type that this capability describes.
gpu	gpu	string	The GPU sku that this capability describes.
maxMemoryInGB	capabilities.maxMemoryInGB	number	The maximum allowed memory request in GB.
maxCpu	capabilities.maxCpu	number	The maximum allowed CPU request in cores.
maxGpuCount	capabilities.maxGpuCount	number	The maximum allowed GPU count.

## ContainerGroupSku

The container group SKU.

The container group SKU.

**SKU** [string](#)

## EncryptionProperties

The container group encryption properties.

[Expand table](#)

Name	Path	Type	Description
Vault Url	vaultBaseUrl	string	The keyvault base url.
Key Name	keyName	string	The encryption key name.
Key Version	keyVersion	string	The encryption key version.
Identity	identity	string	The keyvault managed identity.

## InitContainerDefinition

The init container definition.

[Expand table](#)

Name	Path	Type	Description
Name	name	string	The name for the init container.
properties	properties	InitContainerPropertiesDefinition	The init container definition properties.

## InitContainerPropertiesDefinition

The init container definition properties.

[Expand table](#)

Name	Path	Type	Description
Image	image	string	The image of the init container.
Command	command	array of string	The command to execute within the init container in exec form.

<b>Environment Variables</b>	environmentVariables	array of EnvironmentVariable	The environment variables to set in the init container.
<b>restartCount</b>	instanceView.restartCount	integer	The number of times that the init container has been restarted.
<b>currentState</b>	instanceView.currentState	ContainerState	The container instance state.
<b>previousState</b>	instanceView.previousState	ContainerState	The container instance state.
<b>events</b>	instanceView.events	array of Event	The events of the init container.
<b>Volume Mounts</b>	volumeMounts	array of VolumeMount	The volume mounts available to the init container.
<b>securityContext</b>	securityContext	SecurityContextDefinition	The security context for the container.

## NetworkDependenciesResponse

Response for network dependencies, always empty list.

[Expand table](#)

Name	Path	Type	Description
<b>Items</b>		string	

## ConfidentialComputeProperties

The properties for confidential container group

[Expand table](#)

Name	Path	Type	Description
<b>ccePolicy</b>	ccePolicy	string	The base64 encoded confidential compute enforcement policy

# Container Instances monitoring data reference

Article • 08/29/2024

This article contains all the monitoring reference information for this service.

See [Monitor Container Instances](#) for details on the data you can collect for Container Instances and how to use it.

## Metrics

This section lists all the automatically collected platform metrics for this service. These metrics are also part of the global list of [all platform metrics supported in Azure Monitor](#).

For information on metric retention, see [Azure Monitor Metrics overview](#).

### Supported metrics for Microsoft.ContainerInstance/containerGroups

The following table lists the metrics available for the Microsoft.ContainerInstance/containerGroups resource type.

- All columns might not be present in every table.
- Some columns might be beyond the viewing area of the page. Select **Expand table** to view all available columns.

#### Table headings

- **Category** - The metrics group or classification.
- **Metric** - The metric display name as it appears in the Azure portal.
- **Name in REST API** - The metric name as referred to in the [REST API](#).
- **Unit** - Unit of measure.
- **Aggregation** - The default [aggregation](#) type. Valid values: Average (Avg), Minimum (Min), Maximum (Max), Total (Sum), Count.
- **Dimensions** - [Dimensions](#) available for the metric.
- **Time Grains** - [Intervals](#) at which the metric is sampled. For example, `PT1M` indicates that the metric is sampled every minute, `PT30M` every 30 minutes, `PT1H` every hour, and so on.
- **DS Export** - Whether the metric is exportable to Azure Monitor Logs via diagnostic settings. For information on exporting metrics, see [Create diagnostic settings in Azure Monitor](#).

[ ] [Expand table](#)

Metric	Name in REST API	Unit	Aggregation	Dimensions	Time Grains	DS Export
CPU Usage	<code>CpuUsage</code>	Count	Maximum, Minimum, Average	<code>containerName</code>	PT1M, PT5M, PT15M, PT30M, PT1H, PT6H, PT12H	Yes

Metric	Name in REST API	Unit	Aggregation	Dimensions	Time Grains	DS Export
Memory Usage Total memory usage in byte.	MemoryUsage	Bytes	Maximum, Minimum, Average	containerName	PT1M, PT5M, PT15M, PT30M, PT1H, PT6H, PT12H	Yes
Network Bytes Received Per Second The network bytes received per second.	NetworkBytesReceivedPerSecond	Bytes	Maximum, Minimum, Average	<none>	PT1M, PT5M, PT15M, PT30M, PT1H, PT6H, PT12H	Yes
Network Bytes Transmitted Per Second The network bytes transmitted per second.	NetworkBytesTransmittedPerSecond	Bytes	Maximum, Minimum, Average	<none>	PT1M, PT5M, PT15M, PT30M, PT1H, PT6H, PT12H	Yes

## Supported metrics for Microsoft.ContainerInstance/containerScaleSets

The following table lists the metrics available for the Microsoft.ContainerInstance/containerScaleSets resource type.

- All columns might not be present in every table.
- Some columns might be beyond the viewing area of the page. Select **Expand table** to view all available columns.

### Table headings

- **Category** - The metrics group or classification.
- **Metric** - The metric display name as it appears in the Azure portal.
- **Name in REST API** - The metric name as referred to in the [REST API](#).
- **Unit** - Unit of measure.
- **Aggregation** - The default [aggregation](#) type. Valid values: Average (Avg), Minimum (Min), Maximum (Max), Total (Sum), Count.
- **Dimensions** - [Dimensions](#) available for the metric.
- **Time Grains** - [Intervals](#) at which the metric is sampled. For example, `PT1M` indicates that the metric is sampled every minute, `PT30M` every 30 minutes, `PT1H` every hour, and so on.
- **DS Export**- Whether the metric is exportable to Azure Monitor Logs via diagnostic settings. For information on exporting metrics, see [Create diagnostic settings in Azure Monitor](#).

[Expand table](#)

Metric	Name in REST API	Unit	Aggregation	Dimensions	Time Grains	DS Export
Percentage CPU	CpuPercentage	Percent	Average, Minimum, Maximum	containerName	PT1M	Yes
Average of the CPU percentages consumed by individual Container Groups in this Scale Set						
CPU usage	CpuUsage	MilliCores	Total, Average, Minimum, Maximum	containerName	PT1M	Yes
Average of the CPU utilizations in millicores consumed by Container Groups in this Scale Set						
Memory percentage	MemoryPercentage	Percent	Average, Minimum, Maximum	containerName	PT1M	Yes
Average of the memory percentages consumed ((usedMemory/allocatedMemory) * 100) by Container Groups in this Scale Set						
Memory usage	MemoryUsage	Bytes	Total, Average, Minimum, Maximum	containerName	PT1M	Yes
Total memory used by all the Container Groups in this Scale Set						

## Metric dimensions

For information about what metric dimensions are, see [Multi-dimensional metrics](#).

This service has the following dimensions associated with its metrics.

[Expand table](#)

Dimension Name	Description
containerName	The name of the container. The name must be between 1 and 63 characters long. It can contain only lowercase letters numbers, and dashes. Dashes can't begin or end the name, and dashes can't be consecutive. The name must be unique in its resource group.

## Resource logs

This section lists the types of resource logs you can collect for this service. The section pulls from the list of [all resource logs category types supported in Azure Monitor](#).

## Supported resource logs for Microsoft.ContainerInstance/containerGroups

[Expand table](#)

Category	Category display name	Log table	Supports basic log plan	Supports ingestion-time transformation	Example queries	Costs to export
ContainerEvent	Container events	ContainerEvent Container Event Customer Logs.	No	No		Yes
ContainerInstanceLog	Standard output logs	ContainerInstanceLog Container Instance Customer Logs.	No	No		Yes

## Azure Monitor Logs tables

This section lists the Azure Monitor Logs tables relevant to this service, which are available for query by Log Analytics using Kusto queries. The tables contain resource log data and possibly more depending on what is collected and routed to them.

Container Instances has two table schemas, a legacy schema for Log Analytics and a new schema that supports diagnostic settings. The diagnostic settings feature is in public preview in the Azure portal. You can use either or both schemas at the same time.

### Legacy Log Analytics tables

The following \_CL tables represent the legacy Log Analytics integration. Users provide the Log Analytics workspace ID and key in the Container Group payload.

! Note

Some of the columns in the following list exist only as part of the schema, and don't have any data emitted in logs. These columns are denoted with a description of 'Empty'.

#### ContainerInstanceLog\_CL

[ ] Expand table

Column	Type	Description
Computer	string	Empty
ContainerGroup_s	string	The name of the container group associated with the record
ContainerID_s	string	A unique identifier for the container associated with the record
ContainerImage_s	string	The name of the container image associated with the record
Location_s	string	The location of the resource associated with the record
Message	string	If applicable, the message from the container
OSType_s	string	The name of the operating system the container is based on

Column	Type	Description
RawData	string	Empty
ResourceGroup	string	Name of the resource group that the record is associated with
Source_s	string	Name of the logging component, "LoggingAgent"
SubscriptionId	string	A unique identifier for the subscription that the record is associated with
TimeGenerated	datetime	Timestamp when the event was generated by the Azure service processing the request corresponding the event
Type	string	The name of the table
_ResourceId	string	A unique identifier for the resource that the record is associated with
_SubscriptionId	string	A unique identifier for the subscription that the record is associated with

## ContainerEvent\_CL

[Expand table](#)

Column	Type	Description
Computer	string	Empty
ContainerGroupInstanceId_g	string	A unique identifier for the container group associated with the record
ContainerGroup_s	string	The name of the container group associated with the record
ContainerName_s	string	The name of the container associated with the record
Count_d	real	How many times the event has occurred since the last poll
FirstTimestamp_t	datetime	The timestamp of the first time the event occurred
Location_s	string	The location of the resource associated with the record
Message	string	If applicable, the message from the container
OSType_s	string	The name of the operating system the container is based on
RawData	string	Empty
Reason_s	string	Empty
ResourceGroup	string	The name of the resource group that the record is associated with
SubscriptionId	string	A unique identifier for the subscription that the record is associated with
TimeGenerated	datetime	Timestamp when the event was generated by the Azure service processing the request corresponding the event
Type	string	The name of the table
_ResourceId	string	A unique identifier for the resource that the record is associated with
_SubscriptionId	string	A unique identifier for the subscription that the record is associated with

## Azure Monitor Log Analytics tables

The newer tables require use of a diagnostic setting to route information to Log Analytics. The diagnostic settings feature for Container Instances in the Azure portal is in public preview. The table names are similar, but without the \_CL, and some columns are different.

Once this feature is enabled for a subscription, diagnostic settings can be applied to a container group. Applying diagnostic settings causes a container group to restart.

For example, here's how you can use `New-AzDiagnosticSetting` command to apply a diagnostic settings object to a container group.

Azure PowerShell

```
$log = @()
$log += New-AzDiagnosticSettingLogSettingsObject -Enabled $true -Category ContainerInstanceLog
-RetentionPolicyDay 7 -RetentionPolicyEnabled $true

New-AzDiagnosticSetting -Name test-setting -ResourceId <container-group-resource-id> -
WorkspaceId <log-analytics-workspace-id> -Log $log
```

## Container Instances

Microsoft.ContainerInstance/containerGroups

- [ContainerInstanceLog](#)
- [ContainerEvent](#)

## Activity log

The linked table lists the operations that can be recorded in the activity log for this service. These operations are a subset of [all the possible resource provider operations in the activity log](#).

For more information on the schema of activity log entries, see [Activity Log schema](#).

The following table lists a subset of the operations that Azure Container Instances may record in the Activity log. For the complete listing, see [Microsoft.ContainerInstance resource provider operations](#).

[Expand table](#)

Operation	Description
Microsoft.ContainerInstance/register/action	Registers the subscription for the container instance resource provider and enables the creation of container groups.
Microsoft.ContainerInstance/containerGroupProfiles/read	Get all container group profiles.
Microsoft.ContainerInstance/containerGroupProfiles/write	Create or update a specific container group profile.
Microsoft.ContainerInstance/containerGroupProfiles/delete	Delete the specific container group profile.

Operation	Description
Microsoft.ContainerInstance/containerGroups/read	Get all container groups.
Microsoft.ContainerInstance/containerGroups/write	Create or update a specific container group.
Microsoft.ContainerInstance/containerGroups/delete	Delete the specific container group.
Microsoft.ContainerInstance/containerGroups/restart/action	Restarts a specific container group. This log only captures customer-initiated restarts, not restarts initiated by Azure Container Instances infrastructure.
Microsoft.ContainerInstance/containerGroups/stop/action	Stops a specific container group. Compute resources are deallocated and billing stops.
Microsoft.ContainerInstance/containerGroups/start/action	Starts a specific container group.
Microsoft.ContainerInstance/containerGroups/containers/exec/action	Exec into a specific container.
Microsoft.ContainerInstance/containerGroups/containers/attach/action	Attach to the output stream of a container.
Microsoft.ContainerInstance/containerGroups/containers/buildlogs/read	Get build logs for a specific container.
Microsoft.ContainerInstance/containerGroups/containers/logs/read	Get logs for a specific container.
Microsoft.ContainerInstance/containerGroups/detectors/read	List Container Group Detectors
Microsoft.ContainerInstance/containerGroups/operationResults/read	Get async operation result
Microsoft.ContainerInstance/containerGroups/outboundNetworkDependenciesEndpoints/read	List Container Group Detectors
Microsoft.ContainerInstance/containerGroups/providers/Microsoft.Insights/diagnosticSettings/read	Gets the diagnostic setting for the container group.
Microsoft.ContainerInstance/containerGroups/providers/Microsoft.Insights/diagnosticSettings/write	Creates or updates the diagnostic setting for the container group.
Microsoft.ContainerInstance/containerGroups/providers/Microsoft.Insights/metricDefinitions/read	Gets the available metrics for container group.
Microsoft.ContainerInstance/locations/deleteVirtualNetworkOrSubnets/action	Notifies Microsoft.ContainerInstance that virtual network or subnet is being deleted.
Microsoft.ContainerInstance/locations/cachedImages/read	Gets the cached images for the subscription in a region.

Operation	Description
Microsoft.ContainerInstance/locations/capabilities/read	Get the capabilities for a region.
Microsoft.ContainerInstance/locations/operationResults/read	Get async operation result
Microsoft.ContainerInstance/locations/operations/read	List the operations for Azure Container Instance service.
Microsoft.ContainerInstance/locations/usages/read	Get the usage for a specific region.
Microsoft.ContainerInstance/operations/read	List the operations for Azure Container Instance service.
Microsoft.ContainerInstance/serviceassociationlinks/delete	Delete the service association link created by Azure Container Instance resource provider on a subnet.

## Related content

- See [Monitor Container Instances](#) for a description of monitoring Container Instances.
- See [Monitor Azure resources with Azure Monitor](#) for details on monitoring Azure resources.

---

## Feedback

Was this page helpful?



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

# Azure Policy built-in definitions for Azure Container Instances

Article • 08/29/2024

This page is an index of [Azure Policy](#) built-in policy definitions for Azure Container Instances. For more Azure Policy built-ins for other services, see [Azure Policy built-in definitions](#).

The name of each built-in policy definition links to the policy definition in the Azure portal. Use the link in the **Version** column to view the source on the [Azure Policy GitHub repo](#).

## Azure Container Instances

[Expand table](#)

Name (Azure portal)	Description	Effect(s)	Version (GitHub)
<a href="#">[Preview]: Container Instances should be Zone Aligned</a>	Container Instances can be configured to be Zone Aligned or not. They are considered Zone Aligned if they have only one entry in their zones array. This policy ensures that they are configured to operate within a single availability zone.	Audit, Deny, Disabled	<a href="#">1.0.0-preview</a>
<a href="#">Azure Container Instance container group should deploy into a virtual network</a>	Secure communication between your containers with Azure Virtual Networks. When you specify a virtual network, resources within the virtual network can securely and privately communicate with each other.	Audit, Disabled, Deny	<a href="#">2.0.0</a>
<a href="#">Azure Container Instance container group should use customer-managed key for encryption</a>	Secure your containers with greater flexibility using customer-managed keys. When you specify a customer-managed key, that key is used to protect and control access to the key that encrypts your data. Using customer-managed keys provides additional capabilities to control rotation of the key encryption key or cryptographically erase data.	Audit, Disabled, Deny	<a href="#">1.0.0</a>
<a href="#">Configure diagnostic settings for container groups to Log Analytics workspace</a>	Deploys the diagnostic settings for Container Instance to stream resource logs to a Log Analytics workspace when any container instance which is missing this diagnostic settings is created or updated.	DeployIfNotExists, AuditIfNotExists, Disabled	<a href="#">1.0.0</a>
<a href="#">Configure diagnostics for container group to log analytics workspace</a>	Appends the specified log analytics workspaceId and workspaceKey when any container group which is missing these fields is created or updated. Does not modify the fields of container groups created before this policy was applied until those resource groups are changed.	Append, Disabled	<a href="#">1.0.0</a>
<a href="#">Enable logging by category group for Container instances</a>	Resource logs should be enabled to track activities and events that take place on your	DeployIfNotExists, AuditIfNotExists,	<a href="#">1.0.0</a>

Name	Description	Effect(s)	Version (GitHub)
(Azure portal) <a href="#">(microsoft.containerinstance/containergroups) to Event Hub ↗</a>	resources and give you visibility and insights into any changes that occur. This policy deploys a diagnostic setting using a category group to route logs to an Event Hub for Container instances (microsoft.containerinstance/containergroups).	Disabled	
Enable logging by category group for Container instances <a href="#">(microsoft.containerinstance/containergroups) to Log Analytics ↗</a>	Resource logs should be enabled to track activities and events that take place on your resources and give you visibility and insights into any changes that occur. This policy deploys a diagnostic setting using a category group to route logs to a Log Analytics workspace for Container instances (microsoft.containerinstance/containergroups).	DeployIfNotExists, AuditIfNotExists, Disabled	1.0.0 ↗
Enable logging by category group for Container instances <a href="#">(microsoft.containerinstance/containergroups) to Storage ↗</a>	Resource logs should be enabled to track activities and events that take place on your resources and give you visibility and insights into any changes that occur. This policy deploys a diagnostic setting using a category group to route logs to a Storage Account for Container instances (microsoft.containerinstance/containergroups).	DeployIfNotExists, AuditIfNotExists, Disabled	1.0.0 ↗

## Next steps

- See the built-ins on the [Azure Policy GitHub repo ↗](#).
- Review the [Azure Policy definition structure](#).
- Review [Understanding policy effects](#).

---

## Feedback

Was this page helpful?

 Yes
 No

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

# Frequently asked questions about Azure Container Instances

FAQ

This article addresses frequently asked questions about Azure Container Instances.

## Deployment

### How large can my container image be?

The maximum size for a deployable container image on Azure Container Instances is 15 GB. You might be able to deploy larger images depending on the exact availability at the moment you deploy, but larger image size isn't guaranteed.

The size of your container image impacts how long it takes to deploy, so generally you want to keep your container images as small as possible.

### How can I speed up the deployment of my container?

Because one of the main determinants of deployment times is the image size, look for ways to reduce the size. Remove layers you don't need, or reduce the size of layers in the image (by picking a lighter base OS image). For example, if you're running Linux containers, consider using Alpine as your base image rather than a full Ubuntu Server. Similarly, for Windows containers, use a Nano Server base image if possible.

You should also check the list of precached images in Azure Container Images, available via the [List Cached Images API](#). You might be able to switch out an image layer for one of the precached images.

See more [detailed guidance](#) on reducing container startup time.

### What Windows base OS images are supported?

#### ⓘ Note

Due to issues with backward compatibility after the Windows updates in 2020, the following image versions include the minimum version number that we recommend you use in your base image. Current deployments using older image versions aren't impacted, but new deployments should adhere to the following base images. After June 14, 2021, ACI will no longer support deployments using older version numbers.

#### ⓘ Note

Confidential containers on Azure Container Instances currently doesn't support Windows containers.

## Windows Server 2016 base images

### Important

From now through 31 December 2022, you can continue to deploy Windows Server 2016 container groups on Azure Container Instances. After this date, Windows Server 2016 images will no longer be supported. See [How do I migrate my Windows Server 2016 container groups to Windows Server 2019 images?](#) for instructions on how to transition your workloads.

- [Nano Server](#): `sac2016`, `10.0.14393.3568` or newer
- [Windows Server Core](#): `ltsc2016`, `10.0.14393.3568` or newer

### Note

Windows images based on Semi-Annual Channel release 1709 or 1803 aren't supported.

## Windows Server 2019 and client base images

- [Nano Server](#): `1809`, `10.0.17763.1040` or newer
- [Windows Server Core](#): `ltsc2019`, `1809`, `10.0.17763.1040` or newer
- [Windows](#): `1809`, `10.0.17763.1040` or newer

## What .NET or .NET Core image layer should I use in my container?

Use the smallest image that satisfies your requirements. For Linux, you could use a *runtime-alpine* .NET Core image, which has been supported since the release of .NET Core 2.1. For Windows, if you're using the full .NET Framework, then you need to use a Windows Server Core image (runtime-only image, such as `4.7.2-windowsservercore-ltsc2016`). Runtime-only images are smaller but don't support workloads that require the .NET SDK.

### Note

ACI can't pull images from non OCI-compliant registries.

## What types of container registries are compatible with ACI?

ACI supports image pulls from ACR and other non-Microsoft container registries such as DockerHub. ACI supports image pulls from ACR and other non-Microsoft OCI compatible container registries such as DockerHub with an endpoint that is publicly exposed to the internet.

## How do I migrate my Windows Server 2016 container groups to Windows Server 2019 images?

1. Identify what Windows base image you're currently using.

If you're pulling directly from Microsoft Container Registry (MCR), then that image name is your base image.

If you're working with a private registry, you'll need to look at your Dockerfile to identify the base image, which will be stated after the '['FROM' line](#)'.

2. Select the new base image you want to use from Windows Server 2019. The following examples show commonly used Windows Server 2016 images on Azure Container Instances and our recommendations for replacement Windows Server 2019 images.

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

Windows Server 2016 Image	Recommended Windows Server 2019 Images
mcr.microsoft.com/windows/servercore/iis	mcr.microsoft.com/windows/servercore/iis:windowsservercore-ltsc2019
mcr.microsoft.com/windows/servercore:ltsc2016	mcr.microsoft.com/windows/servercore:ltsc2019

To learn more, read about [image discovery](#).

 **Note**

If you would like assistance selecting your new base image, create an Azure Support Ticket.

3. Follow the [Update containers in Azure Container Instances how-to guide](#) to update your ACI container group to use your new base image.

If you're using MCR for your container registry, you can pass the MCR image name directly into the [container group image parameter](#).

If you're using a private container registry, follow the steps in [Upgrade containers to a new version of the Windows operating system](#). Make sure the container group's [image registry parameters](#) are updated if you've changed them.

## Availability and quotas

### How many cores and memory should I allocate for my containers or the container group?

This really depends on your workload. Start small and test performance to see how your containers do. [Monitor CPU and memory resource usage](#), and then add cores or memory based on the kind of processes that you deploy in the container.

Make sure also to check the [resource availability](#) for the region you're deploying in for the upper bounds on CPU cores and memory available per container group.

 **Note**

A small amount of a container group's resources is used by the service's underlying infrastructure. Your containers are able to access most but not all of the resources allocated to the group. For this reason, plan a small resource buffer when requesting resources for containers in the group.

## What underlying infrastructure does ACI run on?

Azure Container Instances aims to be a serverless containers-on-demand service, so we want you to be focused on developing your containers, and not worry about the infrastructure! For those that are curious or wanting to do comparisons on performance, ACI runs on sets of Azure VMs of various SKUs, primarily from the F and the D series. We expect this to change in the future as we continue to develop and optimize the service.

## I want to deploy thousands of cores on ACI - can I get my quota increased?

Yes (sometimes). See the [quotas and limits](#) article for current quotas and which limits can be increased by request.

## Can I deploy with more than 4 cores and 16 GB of RAM?

Not yet. Currently, these values are the maximums for a container group. Contact Azure Support with specific requirements or requests.

## When will ACI be in a specific region?

Current region availability is published [here](#). If you have a requirement for a specific region, contact Azure Support.

## Features and scenarios

### How do I scale a container group?

Currently, scaling isn't available for containers or container groups. If you need to run more instances, use our API to automate and create more requests for container group creation to the service.

### What features are available to instances running in a custom virtual network?

You can [deploy container groups in an Azure virtual network](#) of your choice, and delegate private IPs to the container groups to route traffic within the virtual network across your Azure resources. For networking scenarios and limitations with Azure Container Instances, see [Virtual network scenarios and resources](#).

### Does the ACI service reserve ports for service functionality?

Yes, the ACI service does reserve the following ports for service functionality: 22, 1025-1027, 3389-3399, 9999, 19000, 19080, 19390, 19100, 20000-30000, 49152-65534. Avoid using these ports in your container

group definition.

## Can you take a dependency on the container group's IP address?

Container group IP addresses are subject to change after being created or deleted. We recommend that your application code doesn't take a dependency on the container group's IP address. We also suggest using [NAT Gateway](#) or [Application Gateway](#) if you would like to maintain a static IP address.

## Pricing

### When does the meter start running?

Container group duration is calculated from the time that we start to pull your first container's image (for a new deployment) or your container group is restarted (if already deployed), until the container group is stopped. See details at [Container Instances pricing ↗](#).

### Do I stop being charged when my containers are stopped?

Meters stop running once your entire container group is stopped. As long as a container in your container group is running, we hold the resources in case you want to start the containers up again.

## Confidential containers on Azure Container Instances

### What is Confidential computing and how is it being applied to Azure Container Instances?

Confidential computing is an industry term defined by the Confidential Computing Consortium (CCC) - a foundation dedicated to defining and accelerating the adoption of confidential computing. The CCC defines confidential computing as: The protection of data in use by performing computations in a hardware-based Trusted Execution Environment (TEE). ACI Confidential Containers introduce hardware-based protection, code integrity, and verification of the Trusted Execution Environment (TEE). Confidential containers apply the latest in confidential computing hardware enabling customers to deploy their existing applications without any modifications while taking advantage of the hardware-based data protection. Code integrity, and verification of the TEE are achieved through the attestation of a confidential computing enforcement policy, which is attached to the container group at deployment time. If any of the properties of the container group differ from those of the confidential computing enforcement policy, the environment fails to launch ensuring that the TEE isn't compromised.

### When should I use Confidential containers on Azure Container Instances?

Confidential containers can be used for a wide variety of elastic workloads but are especially a great fit for workloads that require strong data protection guarantees. Some examples of these workloads include

machine learning workloads that utilize data sets that include personal data or with algorithms that are considered intellectual property. Healthcare customers can use it for analyzing patient data and researching. Financial services customers may use it for credit analysis risk calculation, and portfolio balancing.

## How do I generate a confidential computing enforcement policy?

Confidential computing enforcement policies can be generated using the confcom extension with the Azure CLI. For more information, see [confcom extension](#).

## Are there any features that aren't supported on confidential containers on Azure Container Instances?

GPU based ACI container deployments and Windows containers aren't supported with confidential containers.

## What regions are confidential containers on Azure Container Instances available?

Current region availability for confidential containers is published [here](#).

## Is there an additional cost for confidential container on Azure Container Instances?

Confidential containers on Azure Container Instances do have an additional cost compared to standard SKU container groups. For more information, see [pricing page](#).

## Spot containers on Azure Container Instances (Preview)

### What are ACI Spot containers?

ACI Spot containers are a new feature that allows customers to run interruptible, containerized workloads on unused Azure capacity at up to 70% discounted prices vs regular-priority ACI containers.

### When should I use ACI Spot containers?

ACI Spot containers may be preempted when Azure has insufficient surplus capacity and customers are billed for per-second memory/core usage. With ACI Spot Containers, you can now run your containerized workloads such as batch processing, Monte Carlo simulations, dev/test workloads and parallelizable offline workloads that can tolerate interruptions on Azure at a fraction of the cost of traditional ACI pricing. This offering is targeted at customers who want to run interruptible workloads with no strict availability requirement.

## Are there any features that aren't supported on ACI spot containers?

GPU based ACI container deployments, availability zones, support for ACI deployments with Public IP and ACI deployments behind custom virtual network with Private IP aren't supported with Spot containers.

## What is the default quota for ACI Spot containers?

All customers get a default quota of 10 vCPU cores and 10 container groups.

## How do I file quota requests for ACI Spot containers?

Customers can file support request to increase capacity for Spot containers by selecting the issue type as "Services and Subscription limits(quotas)" and new quota type as "StandardSpotCores" added for ACI Spot containers offering when you're requested to fill in the requested details.

## What regions are ACI Spot containers available?

Azure Container Instances(ACI) Spot containers are only available in select regions during Public Preview. See [Resource and region availability](#) for more information.

## Is there an additional cost for ACI Spot containers?

ACI Spot containers are offered at discounted price and offer up to a 70% discount on top of regular priority ACI containers. The discounts would vary per month in each region. For more information, see the [pricing page](#).

## Next steps

- [Learn more](#) about Azure Container Instances.
- [Troubleshoot common issues](#) in Azure Container Instances.

---

## Feedback

Was this page helpful?



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

# Support and troubleshooting for Azure Container Instances

Article • 08/29/2024

Here are suggestions for where you can get help when developing your Azure Container Instances solutions.

## Self help troubleshooting



See a list of [common issues in Azure Container instances](#) and how to resolve them.

## Post a question on Microsoft Q&A



For quick and reliable answers on your technical product questions from Microsoft Engineers, Azure Most Valuable Professionals (MVPs), or our expert community, engage with us on [Microsoft Q&A](#), Azure's preferred destination for community support.

If you can't find an answer to your problem using search, submit a new question to Microsoft Q&A using the tag [azure-container-instances](#).

## Create an Azure support request



Explore the range of [Azure support options and choose the plan](#) that best fits, whether you're a developer just starting your cloud journey or a large organization deploying business-critical, strategic applications. Azure customers can create and manage support requests in the Azure portal.

- If you already have an Azure Support Plan, [open a support request here](#).
- To sign up for a new Azure Support Plan, [compare support plans](#) and select the plan that works for you.

## Check the Stack Overflow forum



The [azure-container-instances](#) tag on [Stack Overflow](#) is used for asking general questions about how the platform works and how you may use it to accomplish certain tasks.

## Create a GitHub issue



If you need help with the language and tools used to develop and manage Azure Container Instances, open an issue in its repository on GitHub.

[ ] [Expand table](#)

Library	GitHub issues URL
Azure PowerShell	<a href="https://github.com/Azure/azure-powershell/issues">https://github.com/Azure/azure-powershell/issues</a>
Azure CLI	<a href="https://github.com/Azure/azure-cli/issues">https://github.com/Azure/azure-cli/issues</a>
Azure REST API	<a href="https://github.com/Azure/azure-rest-api-specs/issues">https://github.com/Azure/azure-rest-api-specs/issues</a>
Azure SDK for Java	<a href="https://github.com/Azure/azure-sdk-for-java/issues">https://github.com/Azure/azure-sdk-for-java/issues</a>
Azure SDK for Python	<a href="https://github.com/Azure/azure-sdk-for-python/issues">https://github.com/Azure/azure-sdk-for-python/issues</a>
Azure SDK for .NET	<a href="https://github.com/Azure/azure-sdk-for-net/issues">https://github.com/Azure/azure-sdk-for-net/issues</a>
Azure SDK for JavaScript	<a href="https://github.com/Azure/azure-sdk-for-js/issues">https://github.com/Azure/azure-sdk-for-js/issues</a>
Jenkins	<a href="https://github.com/Azure/jenkins/issues">https://github.com/Azure/jenkins/issues</a>
Terraform	<a href="https://github.com/Azure/terraform/issues">https://github.com/Azure/terraform/issues</a>
Ansible	<a href="https://github.com/Azure/Ansible/issues">https://github.com/Azure/Ansible/issues</a>

## Stay informed of updates and new releases



Learn about important product updates, roadmap, and announcements in [Azure Updates](#).

News and information about Azure Container Instances is shared at the [Azure blog](#).

# Next steps

Learn more about [Azure Container Instances](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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