100 XP

Introduction

3 minutes

Azure Container Instances (ACI) offers the fastest and simplest way to run a container in Azure, without having to manage any virtual machines and without having to adopt a higher-level service.

After completing this module, you'll be able to:

- Describe the benefits of Azure Container Instances and how resources are grouped
- Deploy a container instance in Azure by using the Azure CLI
- Start and stop containers using policies
- Set environment variables in your container instances
- Mount file shares in your container instances

Next unit: Explore Azure Container Instances

Continue

How are we doing? ☆ ☆ ☆ ☆ ☆

100 XP

Explore Azure Container Instances

3 minutes

Azure Container Instances (ACI) is a great solution for any scenario that can operate in isolated containers, including simple applications, task automation, and build jobs. Here are some of the benefits:

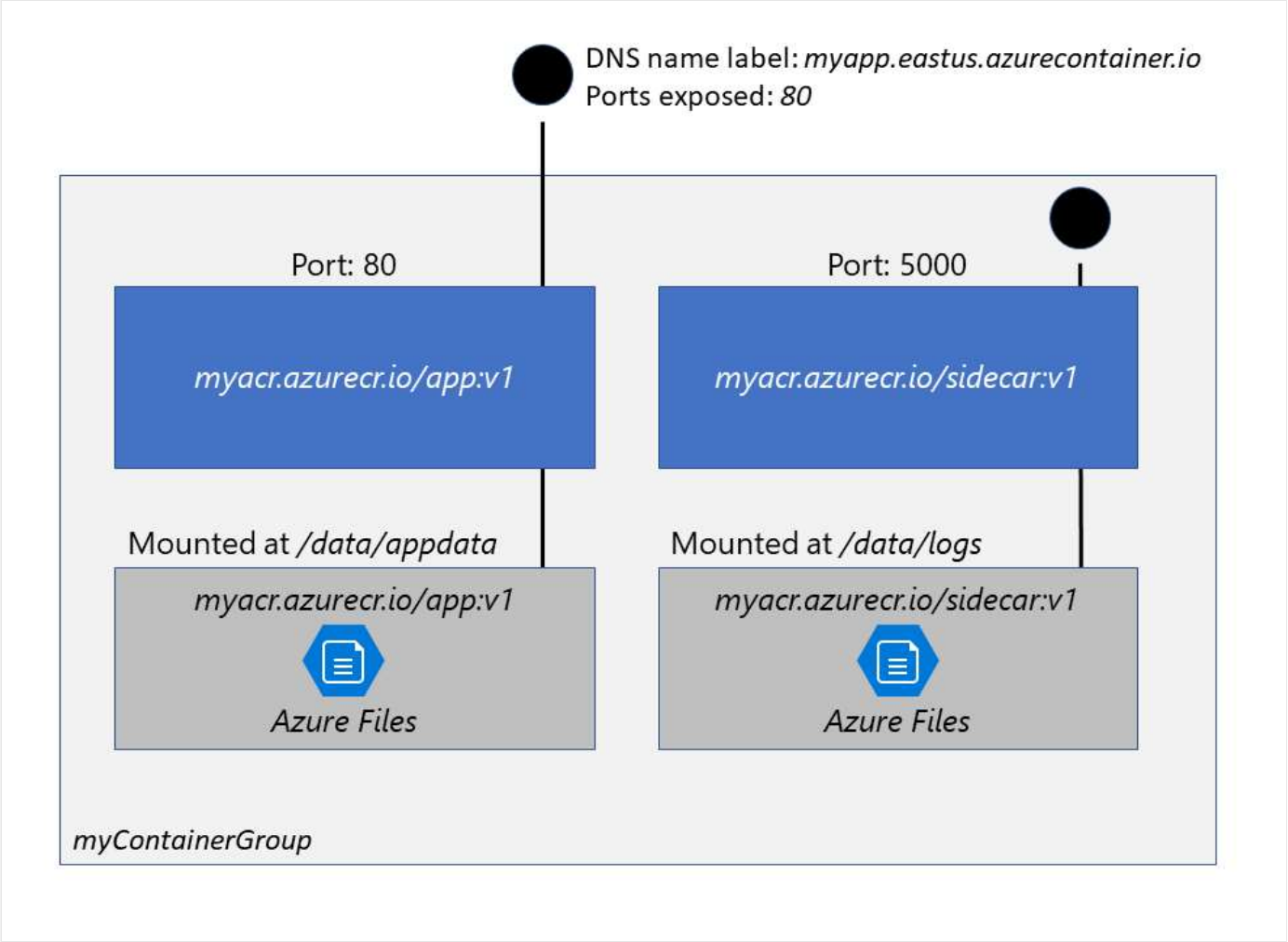
- **Fast startup**: ACI can start containers in Azure in seconds, without the need to provision and manage VMs
- **Container access**: ACI enables exposing your container groups directly to the internet with an IP address and a fully qualified domain name (FQDN)
- **Hypervisor-level security**: Isolate your application as completely as it would be in a VM
- **Customer data**: The ACI service stores the minimum customer data required to ensure your container groups are running as expected
- **Custom sizes**: ACI provides optimum utilization by allowing exact specifications of CPU cores and memory
- **Persistent storage**: Mount Azure Files shares directly to a container to retrieve and persist state
- **Linux and Windows**: Schedule both Windows and Linux containers using the same API.

For scenarios where you need full container orchestration, including service discovery across multiple containers, automatic scaling, and coordinated application upgrades, we recommend Azure Kubernetes Service (AKS).

Container groups

The top-level resource in Azure Container Instances is the *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 instance.

Deployment

There 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 additional 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.

Resource allocation

Azure Container Instances allocates resources such as CPUs, memory, and optionally GPUs (preview) to a 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 instances, each requesting 1 CPU, then the container group is allocated 2 CPUs.

Networking

Container groups share an IP address and a port namespace on that IP address. To enable external clients to reach a container within the group, you must expose the port on the IP address and from the container. Because containers within the group share a port namespace, port mapping isn't supported. Containers within a group can reach each other via localhost on the ports that they have exposed, even if those ports aren't exposed externally on the group's IP address.

Storage

You can specify external volumes to mount within a container group. You can map those volumes into specific paths within the individual containers in a group. Supported volumes include:

- Azure file share
- Secret
- Empty directory
- Cloned git repo

Common scenarios

Multi-container groups are useful in cases where you want to divide a single functional task into a small number of container images. These images can then be delivered by different teams and have separate resource requirements.

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 unit: Exercise: Deploy a container instance by using the Azure CLI

Continue >

How are we doing? ☆ ☆ ☆ ☆ ☆

100 XP

Exercise: Deploy a container instance by using the Azure CLI

10 minutes

In this exercise you'll learn how to perform the following actions:

- Create a resource group for the container
- Create a container
- Verify the container is running

Prerequisites

- An **Azure account** with an active subscription. If you don't already have one, you can sign up for a free trial at https://azure.com/free

Login to Azure and create the resource group

1. Login to the Azure portal and open the Cloud Shell.

2. After the shell opens be sure to select the **Bash** environment.

3. Create a new resource group with the name **az204-aci-rg** so that it will be easier to clean up these resources when you are finished with the module. Replace `<myLocation>` with a region near you.

Bash Copy

```
az group create --name az204-aci-rg --location <myLocation>
```

Create a container

You create a container by providing a name, a Docker image, and an Azure resource group to the `az container create` command. You will expose the container to the Internet by specifying a DNS name label.

1. Create a DNS name to expose your container to the Internet. Your DNS name must be unique, run this command from Cloud Shell to create a variable that holds a unique name.

   Bash Copy

   ```
   DNS_NAME_LABEL=aci-example-$RANDOM
   ```

2. Run the following `az container create` command to start a container instance. Be sure to replace the `<myLocation>` with the region you specified earlier. It will take a few minutes for the operation to complete.

   Bash Copy

   ```
   az container create --resource-group az204-aci-rg \
       --name mycontainer \
       --image mcr.microsoft.com/azuredocs/aci-helloworld \
       --ports 80 \
       --dns-name-label $DNS_NAME_LABEL --location <myLocation> \
   ```

   In the commands above, `$DNS_NAME_LABEL` specifies your DNS name. The image name, `mcr.microsoft.com/azuredocs/aci-helloworld`, refers to a Docker image hosted on Docker Hub that runs a basic Node.js web application.

Verify the container is running

1. When the `az container create` command completes, run `az container show` to check its status.

   Bash Copy

```
az container show --resource-group az204-aci-rg \
    --name mycontainer \
    --query "{FQDN:ipAddress.fqdn,ProvisioningState:provisioningState}" \
    --out table \
```

You see your container's fully qualified domain name (FQDN) and its provisioning state. Here's an example.

Copy

```
FQDN                                    ProvisioningState
--------------------------------------  -------------------
aci-wt.eastus.azurecontainer.io         Succeeded
```

> **Note**
>
> If your container is in the **Creating** state, wait a few moments and run the command again until you see the **Succeeded** state.

2. From a browser, navigate to your container's FQDN to see it running. You may get a warning that the site isn't safe.

Clean up resources

When no longer needed, you can use the `az group delete` command to remove the resource group, the container registry, and the container images stored there.

Bash Copy

```
az group delete --name az204-aci-rg --no-wait
```

Next unit: Run containerized tasks with restart policies

Continue >

How are we doing? ☆ ☆ ☆ ☆ ☆

100 XP

Run containerized tasks with restart policies

3 minutes

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 have completed. Because container instances are billed by the second, you're charged only for the compute resources used while the container executing your task is running.

Container restart policy

When you create a container group in Azure Container Instances, you can specify one of three restart policy settings.

Restart policy	Description
`Always`	Containers in the container group are always restarted. This is the **default** 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.

Specify a restart policy

Specify the `--restart-policy` parameter when you call `az container create`.

Bash Copy

```
az container create \
    --resource-group myResourceGroup \
    --name mycontainer \
    --image mycontainerimage \
    --restart-policy OnFailure
```

Run to completion

Azure Container Instances starts the container, and then stops it when its application, or script, exits. When Azure Container Instances stops a container whose restart policy is `Never` or `OnFailure`, the container's status is set to **Terminated**.

Next unit: Set environment variables in container instances

Continue >

How are we doing? ☆ ☆ ☆ ☆ ☆

100 XP

Set environment variables in container instances

3 minutes

Setting environment variables in your container instances allows you to provide dynamic configuration of the application or script run by the container. This is similar to the `--env` command-line argument to `docker run`.

If you need to pass secrets as environment variables, Azure Container Instances supports secure values for both Windows and Linux containers.

In the example below two variables are passed to the container when it is created. The example below is assuming you are running the CLI in a Bash shell or Cloud Shell, if you use the Windows Command Prompt, specify the variables with double-quotes, such as `--environment-variables "NumWords"="5" "MinLength"="8"`.

Bash Copy

```
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'\
```

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.

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 Copy

```yaml
apiVersion: 2018-10-01
location: eastus
name: securetest
properties:
  containers:
  - name: mycontainer
    properties:
      environmentVariables:
        - name: 'NOTSECRET'
          value: 'my-exposed-value'
        - name: 'SECRET'
          secureValue: 'my-secret-value'
      image: nginx
      ports: []
      resources:
        requests:
          cpu: 1.0
          memoryInGB: 1.5
  osType: Linux
  restartPolicy: Always
tags: null
type: Microsoft.ContainerInstance/containerGroups
```

You would run the following command to deploy the container group with YAML:

Bash Copy

```bash
az container create --resource-group myResourceGroup \
    --file secure-env.yaml \
```

Next unit: Mount an Azure file share in Azure Container Instances

Continue >

How are we doing? ☆ ☆ ☆ ☆ ☆

100 XP

Mount an Azure file share in Azure Container Instances

3 minutes

By default, Azure Container Instances are stateless. If the container 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 unit, Azure Container Instances can mount an Azure file share created with Azure Files. Azure Files offers fully managed file shares in the cloud 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

- You can only mount Azure Files shares to Linux containers.
- Azure file share volume mount requires the Linux container run as *root*.
- Azure File share volume mounts are limited to CIFS support.

Deploy container and mount volume

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`. Below is an example of the command:

Azure CLI Copy

```
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.

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 the 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. An example YAML file is show below.

YAML Copy

```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
```

Mount multiple volumes

To mount multiple volumes in a container instance, you must deploy using an Azure Resource Manager template or a YAML file. 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 template.

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:

JSON Copy

```
"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 Copy

```
"volumeMounts": [{
  "name": "myvolume1",
  "mountPath": "/mnt/share1/"
},
{
  "name": "myvolume2",
  "mountPath": "/mnt/share2/"
}]
```

Next unit: Knowledge check

Continue >

How are we doing? ☆ ☆ ☆ ☆ ☆

200 XP

Knowledge check

3 minutes

Check your knowledge

1. Which of the methods below is recommended when deploying a multi-container group that includes only containers?

- ○ Azure Resource Management template
- ○ YAML file

✔ That's correct. Due to the YAML format's more concise nature, a YAML file is recommended when your deployment includes only container instances.

- ◉ `az container create` command

✘ That's incorrect. This command is not specific to this scenario.

Next unit: Summary

Continue

How are we doing? ☆ ☆ ☆ ☆ ☆

100 XP

Summary

3 minutes

In this module, you learned how to:

- Describe the benefits of Azure Container Instances and how resources are grouped
- Deploy a container instance in Azure by using the Azure CLI
- Start and stop containers using policies
- Set environment variables in your container instances
- Mount file shares in your container instances

Module complete:

Unlock achievement

How are we doing? ☆ ☆ ☆ ☆ ☆