

Introduction to Azure Container Instances

Containers offer a standardized and repeatable way to package, deploy, and manage cloud applications. Azure Container Instances let you run a container in Azure without managing virtual machines and without a higher-level service.

Learning objectives

In this module, you'll:

- Run containers in Azure Container Instances.
 - Control what happens when your container exits.
 - Use environment variables to configure your container when it starts.
 - Attach a data volume to persist data when your container exits.
 - Learn some basic ways to troubleshoot issues on your Azure containers.
-

Exercise - Run Azure Container Instances

Create a container in Azure and expose it to the Internet with a fully qualified domain name (FQDN).

❗ Important

You need your own Azure subscription to run this exercise, and you might incur charges. If you don't already have an Azure subscription, create a [free account](#) before you begin.

Why use Azure Container Instances?

Azure Container Instances is useful for scenarios that can operate in isolated containers, including simple applications, task automation, and build jobs. Here are some of the benefits:

- **Fast startup:** Launch containers in seconds.
- **Per second billing:** Incur costs only while the container is running.
- **Hypervisor-level security:** Isolate your application as completely as it would be in a VM.
- **Custom sizes:** Specify exact values for 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).

Create a container

1. Sign in to the [Azure portal](#) with your Azure subscription.
2. Open Azure Cloud Shell from the Azure portal by selecting the Cloud Shell icon. Make sure to select **Bash** as the interactive Cloud Shell environment.



3. Create a new resource group with the name **learn-deploy-aci-rg** so that it's easier to clean up these resources when you're finished with the module. If you choose a different resource group name, remember it for the rest of the exercises in this module. You also need to choose a region in which you want to create the resource group; for example, **East US**.

Azure CLI

```
az group create --name learn-deploy-aci-rg --location eastus
```

Create a container by providing a name, a Docker image, and an Azure resource group to the `az container create` command. You can optionally expose the container to the Internet by specifying a DNS name label. In this example, deploy a container that hosts a small web app. You can also select the location to place the image; we use the **East US** region, but you can change it to a location close to you.

4. Provide a DNS name to expose your container to the Internet. Your DNS name must be unique. For learning purposes, run this command from Cloud Shell to create a Bash variable that holds a unique name:

```
DNS_NAME_LABEL=aci-demo-$RANDOM
```

5. Run the following `az container create` command to start a container instance:

```
az container create \
  --resource-group learn-deploy-aci-rg \
  --name mycontainer \
  --image mcr.microsoft.com/azuredocs/aci-helloworld \
  --ports 80 \
  --dns-name-label $DNS_NAME_LABEL \
  --location eastus
```

`$DNS_NAME_LABEL` specifies your DNS name. The image name, **azuredocs/aci-helloworld**, refers to a container image hosted on Microsoft Container Registry that runs a basic Node.js web application.

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

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

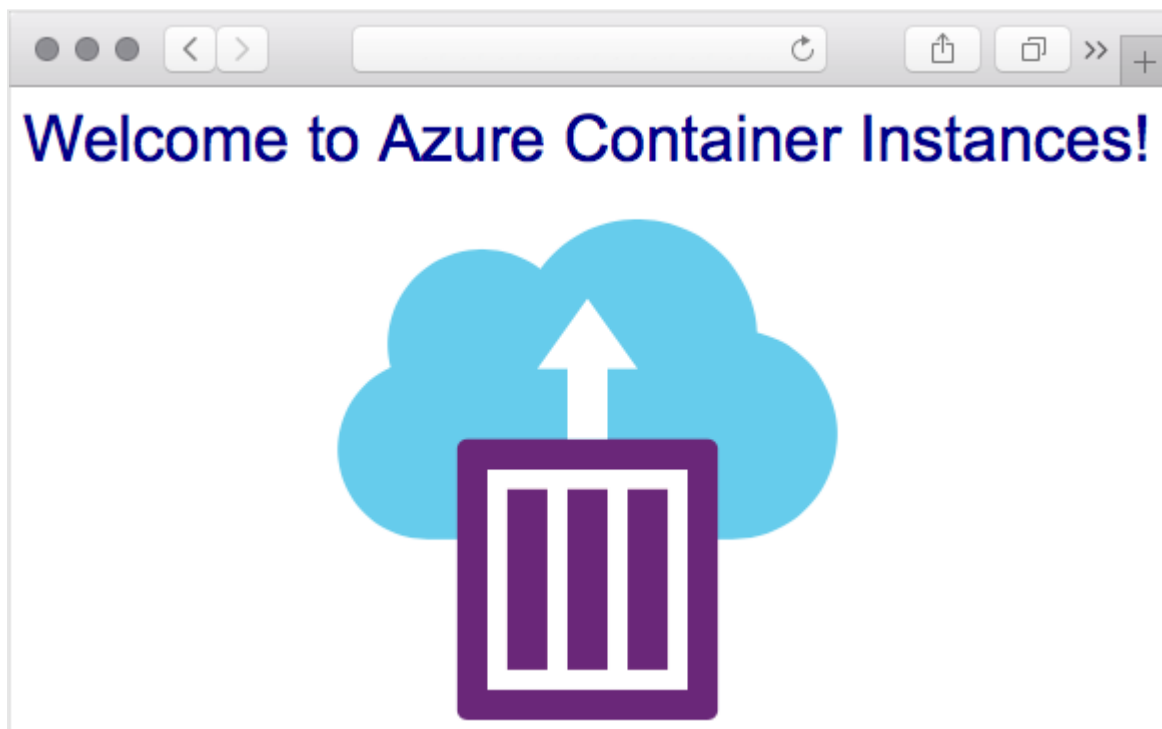
You get your container's fully qualified domain name (FQDN) and its provisioning state. Here's an example:

Output

FQDN	ProvisioningState
aci-demo-0000.eastus.azurecontainer.io	Succeeded

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

7. From a browser, go to your container's FQDN to see it running. Ensure you enter the `http://` prefix in front of the `aci-demo...` string. You should get this welcome page:



Summary

Here, you created an Azure container instance to run a web server and application. You also accessed this application using the FQDN of the container instance.

Exercise - Deploy a container instance by using the Azure CLI

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

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

Prerequisites

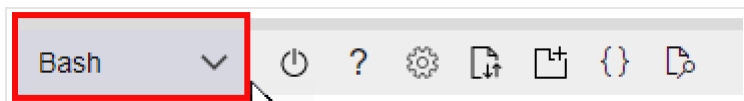
- An **Azure account** with an active subscription. If you don't already have one, you can sign up for a free trial at <https://azure.com/free>

Sign-in to Azure and create the resource group

1. Sign-in to the [Azure portal](#) and open the Cloud Shell.



2. When 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's easier to clean up these resources when you're finished with the module. Replace `<myLocation>` with a region near you.

```
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 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.

```
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 takes a few minutes for the operation to complete.

```
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 previous command, `$DNS_NAME_LABEL` specifies your DNS name. The image name, `mcr.microsoft.com/azuredocs/aci-helloworld`, refers to a Docker image 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.

```
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.

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

7 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.

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

Exercise - Control restart behavior

The ease and speed of deploying containers in Azure Container Instances makes it a great fit for executing run-once tasks like image rendering or building and testing applications.

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.

What are container restart policies?

Azure Container Instances has three restart-policy options:

Restart policy	Description
Always	Containers in the container group are always restarted. This policy makes sense for long-running tasks like a web server. This setting is the default applied when no restart policy is specified at container creation.
Never	Containers in the container group are never restarted. The containers run one time only.
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. This policy works well for containers that run short-lived tasks.

Run a container to completion

To see the restart policy in action, create a container instance from the **azuredocs/aci-wordcount** container image and specify the **OnFailure** restart policy. This container runs a

Python script that analyzes the text of Shakespeare's Hamlet, writes the 10 most common words to standard output, and then exits.

1. Run this `az container create` command to start the container:

```
az container create \
  --resource-group learn-deploy-aci-rg \
  --name mycontainer-restart-demo \
  --image mcr.microsoft.com/azuredocs/aci-wordcount:latest \
  --restart-policy OnFailure \
  --location eastus
```

Azure Container Instances starts the container and then stops it when its process (a 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**.

2. Run `az container show` to check your container's status:

```
az container show \
  --resource-group learn-deploy-aci-rg \
  --name mycontainer-restart-demo \
  --query "containers[0].instanceView.currentState.state"
```

Repeat the command until it reaches the **Terminated** status.

3. Run `az container logs` to view the container's logs to examine the output:

```
az container logs \
  --resource-group learn-deploy-aci-rg \
  --name mycontainer-restart-demo
```

You get this output:

Output

```
[('the', 990),
 ('and', 702),
 ('of', 628),
 ('to', 610),
 ('I', 544),
 ('you', 495),
 ('a', 453),
 ('my', 441),
 ('in', 399),
 ('HAMLET', 386)]
```

Exercise - Set environment variables

Environment variables allow you to dynamically configure the application or script the container runs. You can use the Azure CLI, PowerShell, or the Azure portal to set variables when you create the container. Secured environment variables prevent sensitive information from displaying in the container's output.

Create an Azure Cosmos DB instance and use environment variables to pass the connection information to an Azure container instance. An application in the container uses the variables to write and read data from Azure Cosmos DB. Create both an environment variable and a secured environment variable so that you can see the difference between them.

Deploy Azure Cosmos DB

1. When you deploy Azure Cosmos DB, you provide a unique database name. For learning purposes, run this command in Cloud Shell to create a Bash variable that holds a unique name:

```
COSMOS_DB_NAME=aci-cosmos-db-$RANDOM
```

2. Run this `az cosmosdb create` command to create your Azure Cosmos DB instance:

```
COSMOS_DB_ENDPOINT=$(az cosmosdb create \
  --resource-group learn-deploy-aci-rg \
  --name $COSMOS_DB_NAME \
  --query documentEndpoint \
  --output tsv)
```

This command can take a few minutes to complete.

`$COSMOS_DB_NAME` specifies your unique database name. The command prints the endpoint address for your database. Here, the command saves this address to the Bash variable `COSMOS_DB_ENDPOINT`.

3. Run `az cosmosdb keys list` to get the Azure Cosmos DB connection key and store it in a Bash variable named `COSMOS_DB_MASTERKEY`:

```
COSMOS_DB_MASTERKEY=$(az cosmosdb keys list \
  --resource-group learn-deploy-aci-rg \
  --name $COSMOS_DB_NAME \
  --query primaryMasterKey \
  --output tsv)
```

Deploy a container that works with your database

Create an Azure container instance that can read from and write records to your Azure Cosmos DB instance.

The two environment variables you created in the last part, `COSMOS_DB_ENDPOINT` and `COSMOS_DB_MASTERKEY`, hold the values you need to connect to the Azure Cosmos DB instance.

1. Run the following `az container create` command to create the container:

```
az container create \
  --resource-group learn-deploy-aci-rg \
  --name aci-demo \
  --image mcr.microsoft.com/azuredocs/azure-vote-front:cosmosdb \
  --ip-address Public \
  --location eastus \
  --environment-variables \
    COSMOS_DB_ENDPOINT=$COSMOS_DB_ENDPOINT \
    COSMOS_DB_MASTERKEY=$COSMOS_DB_MASTERKEY
```

azuredocs/azure-vote-front:cosmosdb refers to a container image that runs a fictitious voting app.

Note the `--environment-variables` argument. This argument specifies environment variables that are passed to the container when the container starts. The container image is configured to look for these environment variables. Pass the name of the Azure Cosmos DB endpoint and its connection key.

2. Run the `az container show` command to get your container's public IP address:

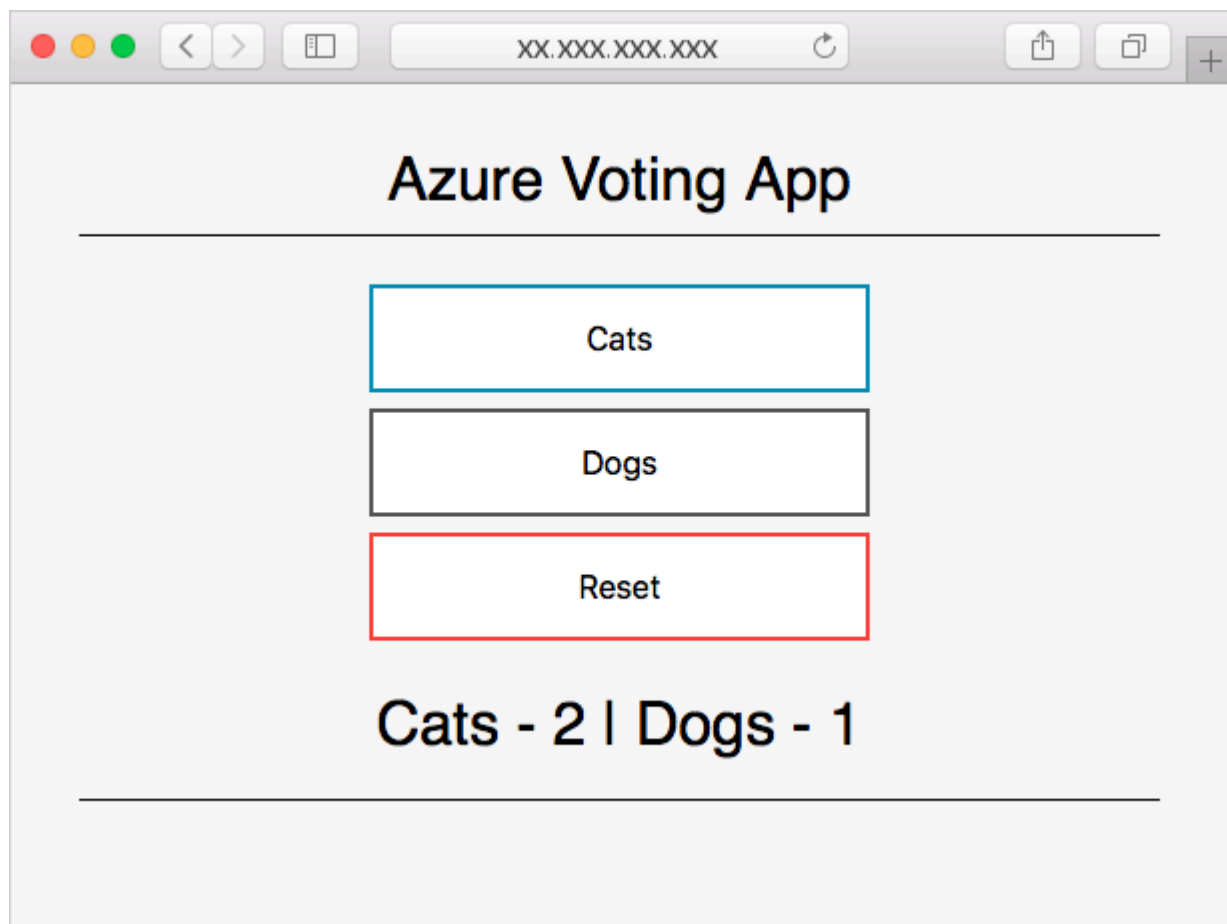
```
az container show \
  --resource-group learn-deploy-aci-rg \
  --name aci-demo \
  --query ipAddress.ip \
  --output tsv
```

3. In a browser, go to your container's IP address.

Important

Sometimes containers take a minute or two to fully start and be able to receive connections. If there's no response when you go to the IP address in your browser, wait a few moments and refresh the page.

Once the app is available, you get this page:



Try casting a vote for cats or dogs. Each vote is stored in your Azure Cosmos DB instance.

Use secured environment variables to hide connection information

In the previous section, you used two environment variables to create your container. By default, these environment variables are accessible through the Azure portal and command-line tools in plain text.

In this section, learn how to prevent sensitive information, such as connection keys, from being displayed in plain text.

1. Let's start by seeing the current behavior in action. Run the following `az container show` command to display your container's environment variables:

```
az container show \
  --resource-group learn-deploy-aci-rg \
  --name aci-demo \
  --query containers[0].environmentVariables
```

You get output with both values in plain text. Here's an example:

JSON

```
[
  {
    "name": "COSMOS_DB_ENDPOINT",
    "secureValue": null,
    "value": "https://aci-cosmos.documents.azure.com:443/"
  },
  {
    "name": "COSMOS_DB_MASTERKEY",
    "secureValue": null,
    "value": "abcdefghijklmnopqrstuvwxyz0123456789=="
  }
]
```

Although these values don't appear to your users through the voting application, it's a good security practice to ensure that sensitive information (such as connection keys) isn't stored in plain text.

Secure environment variables prevent clear text output. To use secure environment variables, use the `--secure-environment-variables` argument instead of the `--environment-variables` argument.

2. Run the following command to create a second container named **aci-demo-secure** that makes use of secured environment variables:

```
--image mcr.microsoft.com/azuredocs/azure-vote-front:cosmosdb \
--ip-address Public \
--location eastus \
--secure-environment-variables \
    COSMOS_DB_ENDPOINT=$COSMOS_DB_ENDPOINT \
    COSMOS_DB_MASTERKEY=$COSMOS_DB_MASTERKEY
```

Note the use of the `--secure-environment-variables` argument.

3. Run the following `az container show` command to display your container's environment variables:

```
az container show \
--resource-group learn-deploy-aci-rg \
--name aci-demo-secure \
--query containers[0].environmentVariables
```

This time, you can see that your environment variables don't appear in plain text:

```
[
  {
    "name": "COSMOS_DB_ENDPOINT",
    "secureValue": null,
    "value": null
  },
  {
    "name": "COSMOS_DB_MASTERKEY",
    "secureValue": null,
    "value": null
  }
]
```

In fact, the values of your environment variables don't appear at all. That's OK because these values refer to sensitive information. Here, all you need to know is that the environment variables exist.

Exercise - Use data volumes

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.

Mount an Azure file share to an Azure container instance so that you can store data and access it later.

Create an Azure file share

Create a storage account and a file share. Later, you can make the file share accessible to an Azure container instance.

1. Your storage account requires a unique name. For learning purposes, run the following command to store a unique name in a Bash variable:

```
STORAGE_ACCOUNT_NAME=mystorageaccount$RANDOM
```

2. Run the following `az storage account create` command to create your storage account:

```
az storage account create \
  --resource-group learn-deploy-aci-rg \
  --name $STORAGE_ACCOUNT_NAME \
  --sku Standard_LRS \
  --location eastus
```

3. Run the following command to place the storage account connection string into an environment variable named `AZURE_STORAGE_CONNECTION_STRING`:

```
export AZURE_STORAGE_CONNECTION_STRING=$(az storage account show-connection-
string \
  --resource-group learn-deploy-aci-rg \
  --name $STORAGE_ACCOUNT_NAME \
  --output tsv)
```

`AZURE_STORAGE_CONNECTION_STRING` is a special environment variable that the Azure CLI understands. The `export` part makes this variable accessible to other CLI commands that you run later.

4. Run this command to create a file share named **aci-share-demo** in the storage account:

```
az storage share create --name aci-share-demo
```

Get storage credentials

To mount an Azure file share as a volume in Azure Container Instances, you need these three values:

- Storage account name
- Share name
- Storage account access key

You already have the first two values. The storage account name is stored in the `STORAGE_ACCOUNT_NAME` Bash variable. You specified **aci-share-demo** as the share name in the previous step. Here, you get the remaining value: the storage account access key.

1. Run the following command to get the storage account key:

```
STORAGE_KEY=$(az storage account keys list \
  --resource-group learn-deploy-aci-rg \
  --account-name $STORAGE_ACCOUNT_NAME \
  --query "[0].value" \
  --output tsv)
```

The result is stored in a Bash variable named `STORAGE_KEY`.

2. As an optional step, print the storage key to the console.

```
echo $STORAGE_KEY
```


Deploy a container and mount the file share

To mount an Azure file share as a volume in a container, you specify the share and volume mount point when you create the container.

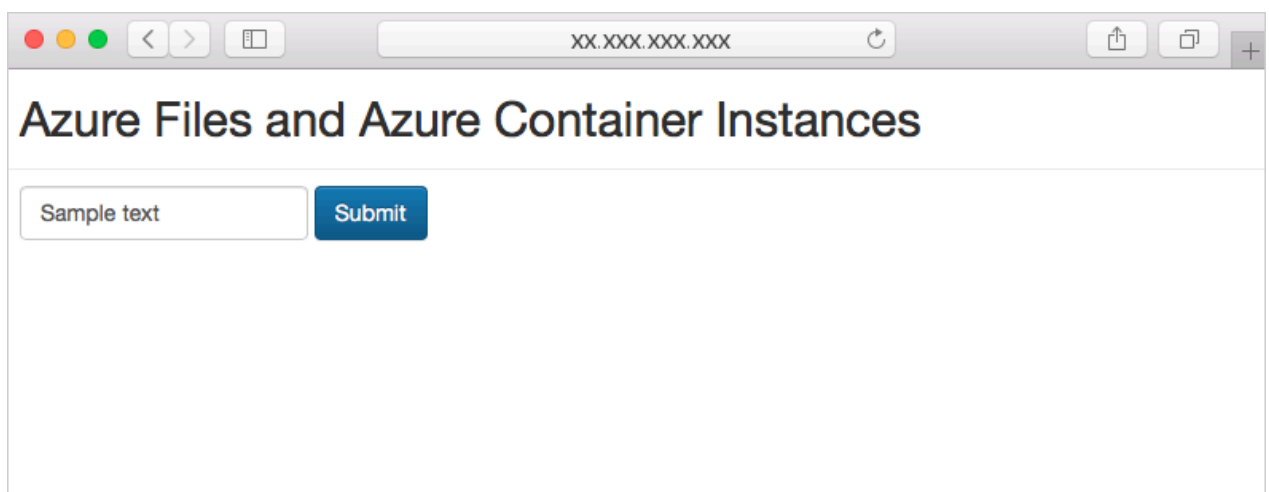
1. Run this `az container create` command to create a container that mounts `/aci/logs/` to your file share:

```
az container create \
  --resource-group learn-deploy-aci-rg \
  --name aci-demo-files \
  --image mcr.microsoft.com/azuredocs/aci-hellofiles \
  --location eastus \
  --ports 80 \
  --ip-address Public \
  --azure-file-volume-account-name $STORAGE_ACCOUNT_NAME \
  --azure-file-volume-account-key $STORAGE_KEY \
  --azure-file-volume-share-name aci-share-demo \
  --azure-file-volume-mount-path /aci/logs/
```

2. Run `az container show` to get your container's public IP address:

```
az container show \
  --resource-group learn-deploy-aci-rg \
  --name aci-demo-files \
  --query ipAddress.ip \
  --output tsv
```

3. From a browser, navigate to your container's IP address. You get this page:



4. Enter some text into the form, and select **Submit**. This action creates a file that contains the text you entered in the Azure file share.
5. Run this `az storage file list` command to display the files that are contained in your file share:

```
az storage file list -s aci-share-demo -o table
```

6. Run `az storage file download` to download a file to your Cloud Shell session. Replace `<filename>` with one of the files that appeared in the previous step:

```
az storage file download -s aci-share-demo -p <filename>
```

7. Run the `cat` command to print the contents of the file.

```
cat <filename>
```

Remember that your data persists when your container exits. You can mount your file share to other container instances to make that data available to them.

Repository location for this sample page : <https://github.com/Azure-Samples/aci-hellofiles/tree/master>

Exercise - Troubleshoot Azure Container Instances

10 minutes

To help you understand basic ways to troubleshoot container instances, perform some basic operations like:

- Pulling container logs
- Viewing container events
- Attaching to a container instance

Get logs from your prior deployed container instance

Run the following `az container logs` command to see the output from the cats and dogs voting app container you created in the prior exercise:

```
az container logs \
  --resource-group learn-deploy-aci-rg \
  --name aci-demo-files
```

You get a similar output to:

Output

```
Checking for script in /app/prestart.sh
Running script /app/prestart.sh
Running inside /app/prestart.sh, you could add migrations to this file, e.g.:
```

```
#!/usr/bin/env bash
```

```
# Let the DB start
sleep 10;
# Run migrations
alembic upgrade head
```

```
...
```

Get container events

The `az container attach` command provides diagnostic information during container startup. Once the container has started, it also writes standard output and standard error streams to your local terminal.

Run `az container attach` to attach to your container:

```
az container attach \
  --resource-group learn-deploy-aci-rg \
  --name aci-demo-files
```

You get a similar output to:

Output

```
Container 'aci-demo-files' is in state 'Running'...
(count: 1) (last timestamp: 2021-09-21 23:48:14+00:00) pulling image
"mcr.microsoft.com/azuredocs/azure-vote-front"
(count: 1) (last timestamp: 2021-09-21 23:49:09+00:00) Successfully pulled image
"mcr.microsoft.com/azuredocs/azure-vote-front"
(count: 1) (last timestamp: 2021-09-21 23:49:12+00:00) Created container
(count: 1) (last timestamp: 2021-09-21 23:49:13+00:00) Started container
```

```
Start streaming logs:
Checking for script in /app/prestart.sh
Running script /app/prestart.sh
...
```

Tip

Enter `Ctrl+C` to disconnect from your attached container.

Execute a command in your container

As you diagnose and troubleshoot issues, you may need to run commands directly on your running container.

1. To see commands in your container, run the following `az container exec` command to start an interactive session on your container:

```
az container exec \  
  --resource-group learn-deploy-aci-rg \  
  --name aci-demo-files \  
  --exec-command /bin/sh
```

At this point, you're effectively working inside of the container.

2. Run the `ls` command to display the contents of the working directory.

```
# ls  
__pycache__  config_file.cfg  main.py  prestart.sh  static  templates  
uwsgi.ini
```

3. You can explore the system further if you wish. When you're done, run the `exit` command to stop the interactive session.

Monitor CPU and memory usage on your container

See how to monitor CPU and memory usage on your container.

1. Run the following `az container show` command to get the ID of your Azure container instance and store the ID in a Bash variable:

```
CONTAINER_ID=$(az container show \  
  --resource-group learn-deploy-aci-rg \  
  --name aci-demo-files \  
  --query id \  
  --output tsv)
```

2. Run the `az monitor metrics list` command to retrieve CPU usage information:

```
az monitor metrics list \  
  --resource $CONTAINER_ID \  
  --metrics CPUUsage \  
  --output table
```

Note the `--metrics` argument. Here, **CPUUsage** specifies to retrieve CPU usage.

You see text similar to the following output:

Output

Timestamp	Name	Average
-----	-----	-----
2021-09-21 23:39:00	CPU Usage	
2021-09-21 23:40:00	CPU Usage	
2021-09-21 23:41:00	CPU Usage	
2021-09-21 23:42:00	CPU Usage	
2021-09-21 23:43:00	CPU Usage	0.375
2021-09-21 23:44:00	CPU Usage	0.875
2021-09-21 23:45:00	CPU Usage	1
2021-09-21 23:46:00	CPU Usage	3.625
2021-09-21 23:47:00	CPU Usage	1.5
2021-09-21 23:48:00	CPU Usage	2.75
2021-09-21 23:49:00	CPU Usage	1.625
2021-09-21 23:50:00	CPU Usage	0.625
2021-09-21 23:51:00	CPU Usage	0.5
2021-09-21 23:52:00	CPU Usage	0.5
2021-09-21 23:53:00	CPU Usage	0.5

3. Run this `az monitor metrics list` command to retrieve memory usage information:

```
az monitor metrics list \  
  --resource $CONTAINER_ID \  
  --metrics MemoryUsage \  
  --output table
```

Here, you specified **MemoryUsage** for the `--metrics` argument to retrieve memory usage information.

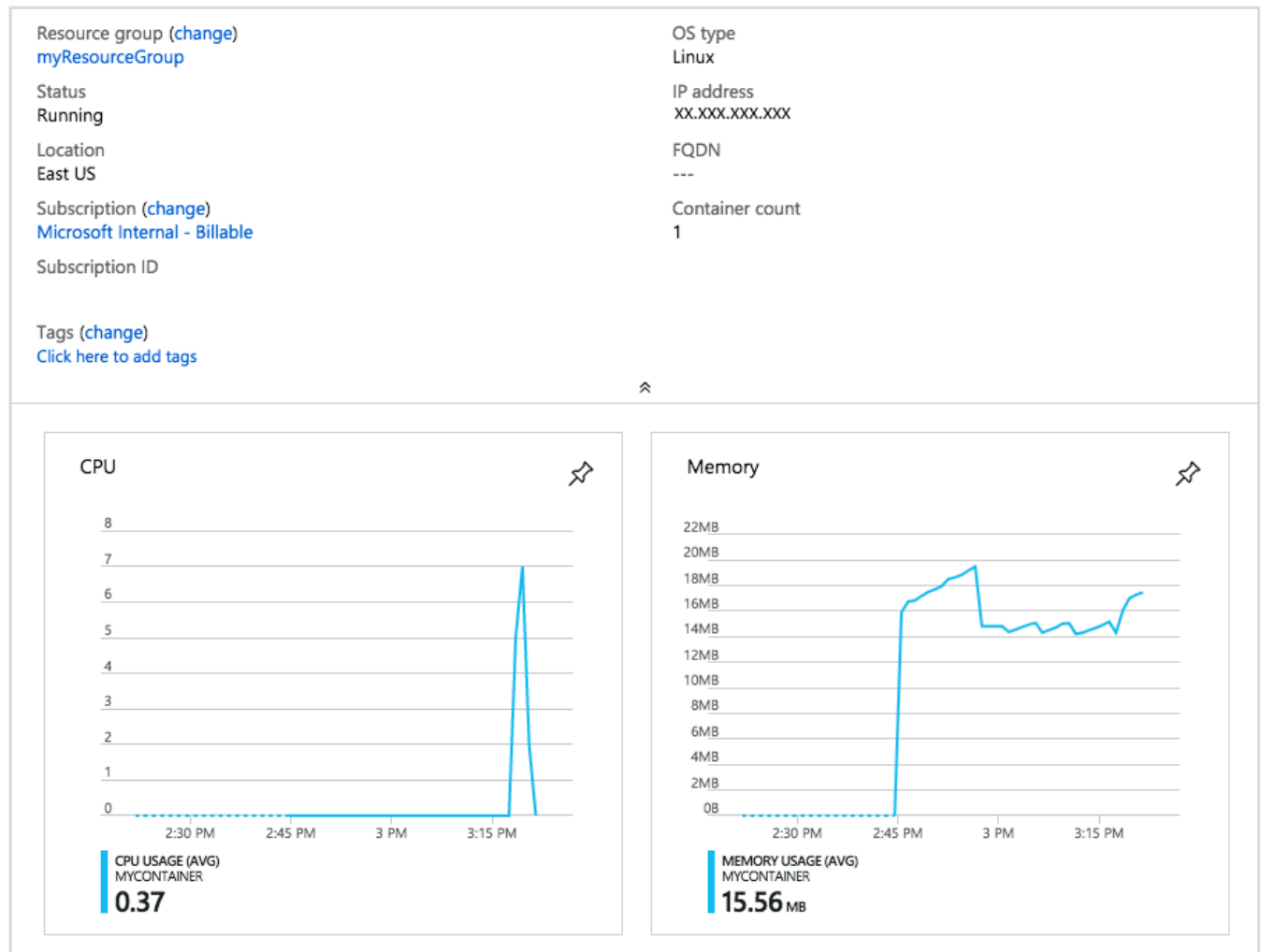
You see text similar to the following output:

Output

Timestamp	Name	Average
-----	-----	-----
2021-09-21 23:43:00	Memory Usage	
2021-09-21 23:44:00	Memory Usage	0.0
2021-09-21 23:45:00	Memory Usage	15917056.0
2021-09-21 23:46:00	Memory Usage	16744448.0
2021-09-21 23:47:00	Memory Usage	16842752.0
2021-09-21 23:48:00	Memory Usage	17190912.0
2021-09-21 23:49:00	Memory Usage	17506304.0
2021-09-21 23:50:00	Memory Usage	17702912.0

2021-09-21 23:51:00	Memory Usage	17965056.0
2021-09-21 23:52:00	Memory Usage	18509824.0
2021-09-21 23:53:00	Memory Usage	18649088.0
2021-09-21 23:54:00	Memory Usage	18845696.0
2021-09-21 23:55:00	Memory Usage	19181568.0

In the Azure portal, Azure Container Instances CPU and memory usage information look like this:



Clean up resources

In this module, you created resources using your Azure subscription. You should clean up these resources so that you aren't still charged for them.

1. In the Azure **home** page, select **All resources**.
2. Find the **learn-deploy-aci-rg** resource group, or whatever resource group name you used, and select it.
3. In the **Overview** tab of the resource group, select **Delete resource group**.

4. A new dialog box opens. Enter the name of the resource group again, and select **Delete**.
All of the resources we created in this module are deleted.
-