Microsoft

AZ-204T00A

Learning Path 05: Implement containerized solutions

Agenda

- Manage container images in Azure Container Registry
- Run container images in Azure Container Instances
- Implement Azure Container Apps

Module 1: Manage Container Images in Azure Container Registry

Learning objectives

- Explain the features and benefits Azure Container Registry offers
- Describe how to use ACR Tasks to automate builds and deployments
- Explain the elements in a Dockerfile
- Build and run an image in the ACR by using Azure CLI

Discover the Azure Container Registry (1 of 2)

Use the Azure Container Registry (ACR) service with your existing container development and deployment pipelines or use Azure Container Registry Tasks to build container images in Azure.

Use cases

Pull images from an Azure container registry to various deployment targets:

- *Scalable orchestration systems* that manage containerized applications across clusters of hosts.
- *Azure services* that support building and running applications at scale.

Azure Container Registry service tiers

Azure Container Registry is available in multiple service tiers.

- Basic
- Standard
- Premium

Discover the Azure Container Registry (2 of 2)

Supported images and artifacts

- Grouped in a repository, each image is a read-only snapshot of a Docker-compatible container.
- Azure container registries can include both Windows and Linux images.
- Azure Container Registry also stores Helm charts and images built to the Open Container Initiative (OCI) Image Format Specification.

Azure Container Registry Tasks

- Use Azure Container Registry Tasks (ACR Tasks) to streamline building, testing, pushing, and deploying images in Azure.

Explore storage capabilities

Every Basic, Standard, and Premium Azure container registry benefits from advanced Azure storage features.

- **Encryption-at-rest:** All container images in your registry are encrypted at rest.
- **Geo-redundant storage:** Azure uses a geo-redundant storage scheme to guard against loss of your container images.
- **Geo-replication:** For scenarios requiring even more high-availability assurance, consider using the geo-replication feature of Premium registries.

Build and manage containers with tasks

ACR Tasks provides cloud-based container image building for platforms including Linux, Windows, and ARM. It can automate OS and framework patching for your Docker containers.

Task scenarios

ACR Tasks supports several scenarios to build and maintain container images and other artifacts:

- Quick task
- Automatically triggered tasks
- Multi-step task

Explore elements of a Dockerfile (1 of 2)

```
# Step 1: Specify the parent image for the new image
FROM ubuntu:18.04

# Step 2: Update OS packages and install additional software
RUN apt -y update &&  apt install -y wget nginx software-properties-common apt-transport-https \
    && wget -q <URL>/ubuntu/18.04/packages-microsoft-prod.deb -O packages-microsoft-prod.deb \
    && dpkg -i packages-microsoft-prod.deb \
    && add-apt-repository universe \
    && apt -y update \
    && apt install -y dotnet-sdk-3.0

# Step 3: Configure Nginx environment
CMD service nginx start

# Step 4: Configure Nginx environment
COPY ./default /etc/nginx/sites-available/default
```

Explore elements of a Dockerfile (2 of 2)

```
# Use the .NET 6 runtime as a base image
FROM mcr.microsoft.com/dotnet/runtime:6.0

# Set the working directory to /app
WORKDIR /app

# Copy the contents of the published app to the container's /app directory
COPY bin/Release/net6.0/publish/ .

# Expose port 80 to the outside world
EXPOSE 80

# Set the command to run when the container starts
CMD ["dotnet", "MyApp.dll"]
```

Exercise: Build and run a container image by using Azure Container Registry Tasks

In this exercise you learn how to ACR Tasks to create a registry and build, push, and run an image in the ACR.

Objectives

- Create an Azure Container Registry
- Build and push image from a Dockerfile
- Verify the results
- Run the image in the ACR
- Clean up resources

Summary and knowledge check

In this module, you learned how to:

- Explain the features and benefits Azure Container Registry offers
- Describe how to use ACR Tasks to automate builds and deployments
- Explain the elements in a Dockerfile
- Build and run an image in the ACR by using Azure CLI

1 Which Azure Container Registry option supports geo-replication to manage a single registry across multiple regions?

2 Which Azure container registry tiers benefit from encryption-at-rest?

Module 2: Run container images in Azure Container Instances

Learning objectives

- 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.

Introduction

- Azure Container Instances (ACI) offers the fastest and simplest way to run a container in Azure.
- No requirement to manage any virtual machines and without having to adopt a higher-level service.

Explore Azure Container Instances (1 of 3)

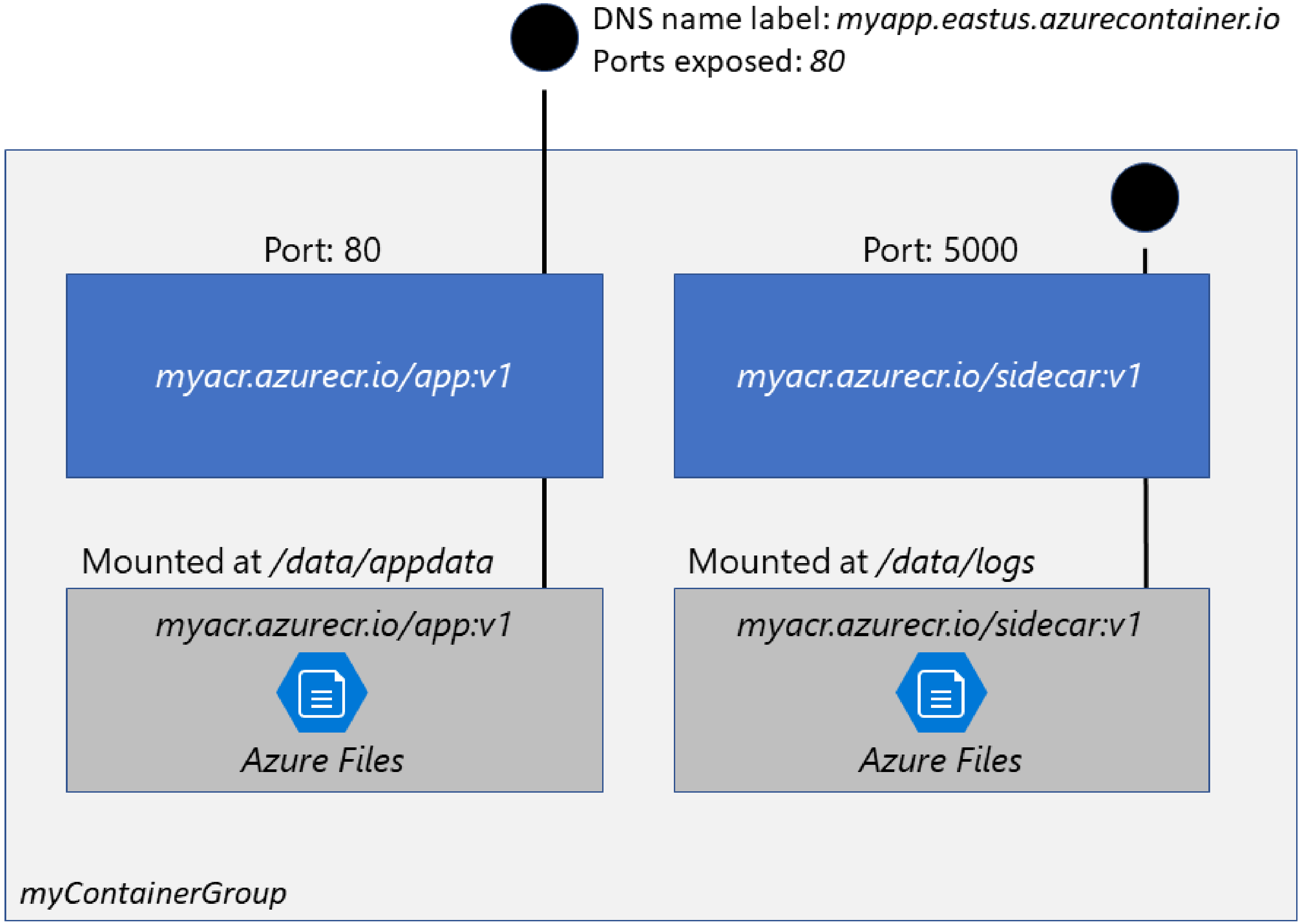
Feature	Description
Fast startup times	Containers can start in seconds without the need to provision and manage VMs
Public IP connectivity and DNS name	Containers can be directly exposed to the internet with an IP address and a fully qualified domain name (FQDN)
Hypervisor-level security	Container applications are as isolated in a container as they would be in a VM
Custom sizes	ACI provides optimum utilization by allowing exact specifications of CPU cores and memory
Persistent storage	Containers support direct mounting of Azure Files shares
Linux and Windows containers	The same API is used to schedule both Linux and Windows containers
Co-scheduled groups	Container Instances supports scheduling of multicontainer groups that share host machine resources
Virtual network deployment	Container Instances can be deployed into an Azure virtual network

Explore Azure Container Instances (2 of 3)

Container groups

The top-level resource in Azure Container Instances is the container group.

The containers in a container group share a lifecycle, resources, local network, and storage volumes.


DNS name label: *myapp.eastus.azurecontainer.io*
Ports exposed: *80*
Port: 80
Port: 5000
myacr.azurecr.io/app:v1
myacr.azurecr.io/sidecar:v1
Mounted at */data/appdata*
Mounted at */data/logs*
myacr.azurecr.io/app:v1
Azure Files
myacr.azurecr.io/sidecar:v1
Azure Files
myContainerGroup

Explore Azure Container Instances (3 of 3)

Deployment

- There are two common ways to deploy a multi-container group: ARM template or a YAML file.

Resource allocation

- ACI 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.

Networking

- Container groups share an IP address and a port namespace on that IP address.

Storage

- Specify external volumes to mount within a container group.
- 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 small number of container images.

Exercise: Deploy a container instance by using the Azure CLI

In this exercise you learn how to use the Azure CLI in the Azure Cloud Shell to create and run a container in Azure Container Instances.

Objectives

- Create the resource group
- Create a container
- Verify the container is running
- Clean up resources

Run containerized tasks with restart policies (1 of 2)

Overview

With a configurable restart policy, you can specify that your containers are stopped when their processes have completed.

Container restart policy

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

- `Always`
- `Never`
- `OnFailure`

Set environment variables in container instances (1 of 3)

YAML example

- 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 YAML demonstrate the two variable types.

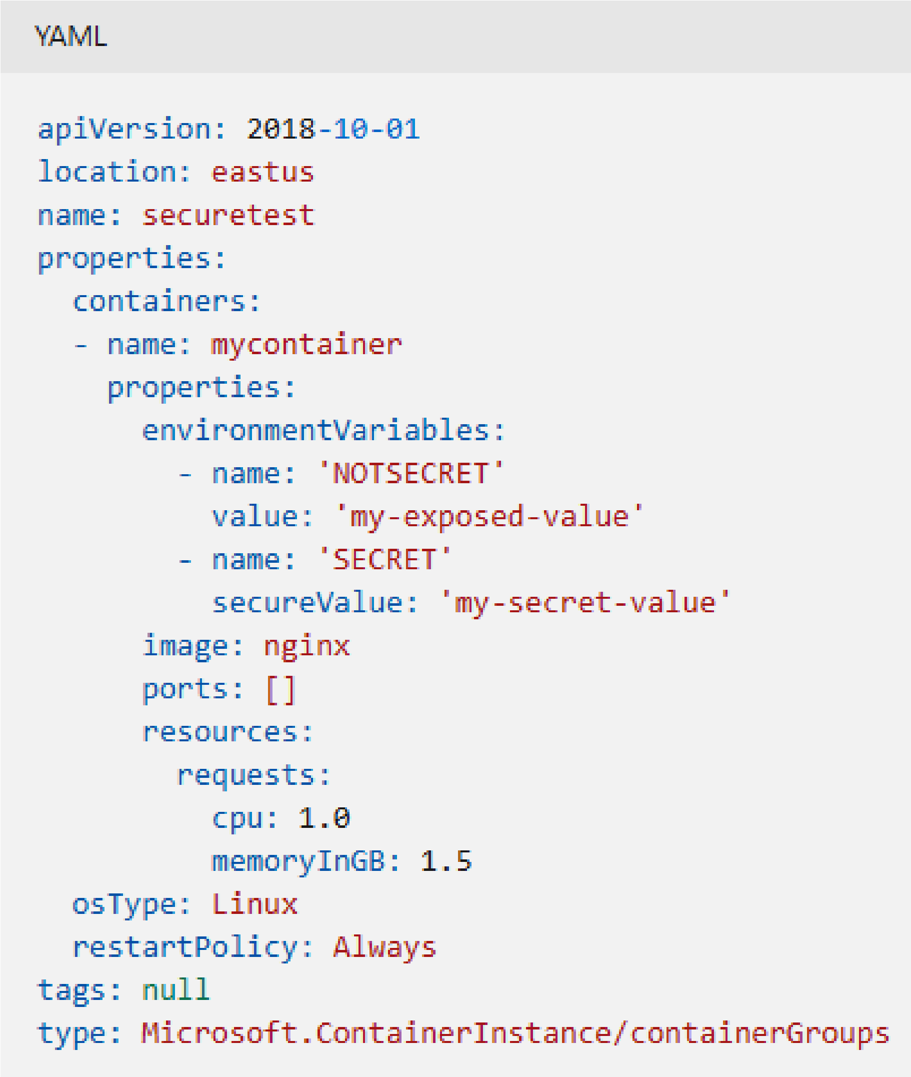
```
az container create \
    --resource-group myResourceGroup \
    --name mycontainer2 \
    --image mcr.microsoft.com/azuredocs/aci-wordcount:latest \
    --restart-policy OnFailure \
    --environment-variables 'NumWords'='5' 'MinLength'='8'
```

Set environment variables in container instances (2 of 3)

- Provides dynamic configuration of the application or script run by the container.
- ACI supports both Windows and Linux containers to pass secrets as environment variables
- In the example two variables are passed to the container when it is created.

```
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'
```

Set environment variables in container instances (3 of 3)



YAML

```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
```

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

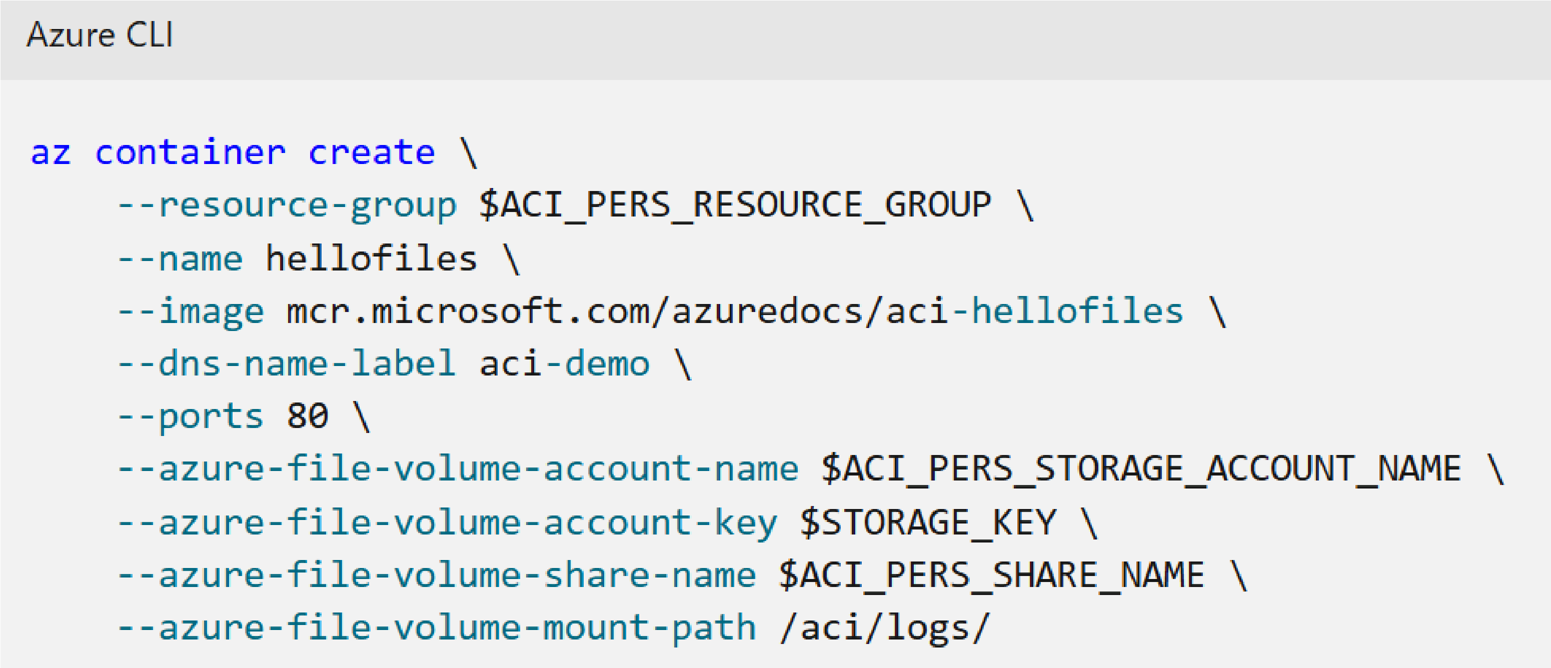
Mount an Azure file share in Azure Container Instances (1 of 3)

Overview

- 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.

Limitations

- You can only mount Azure Files shares to Linux containers.
- Azure file share volume mount requires the Linux container run as root.



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/
```

Mount an Azure file share in Azure Container Instances (2 of 3)

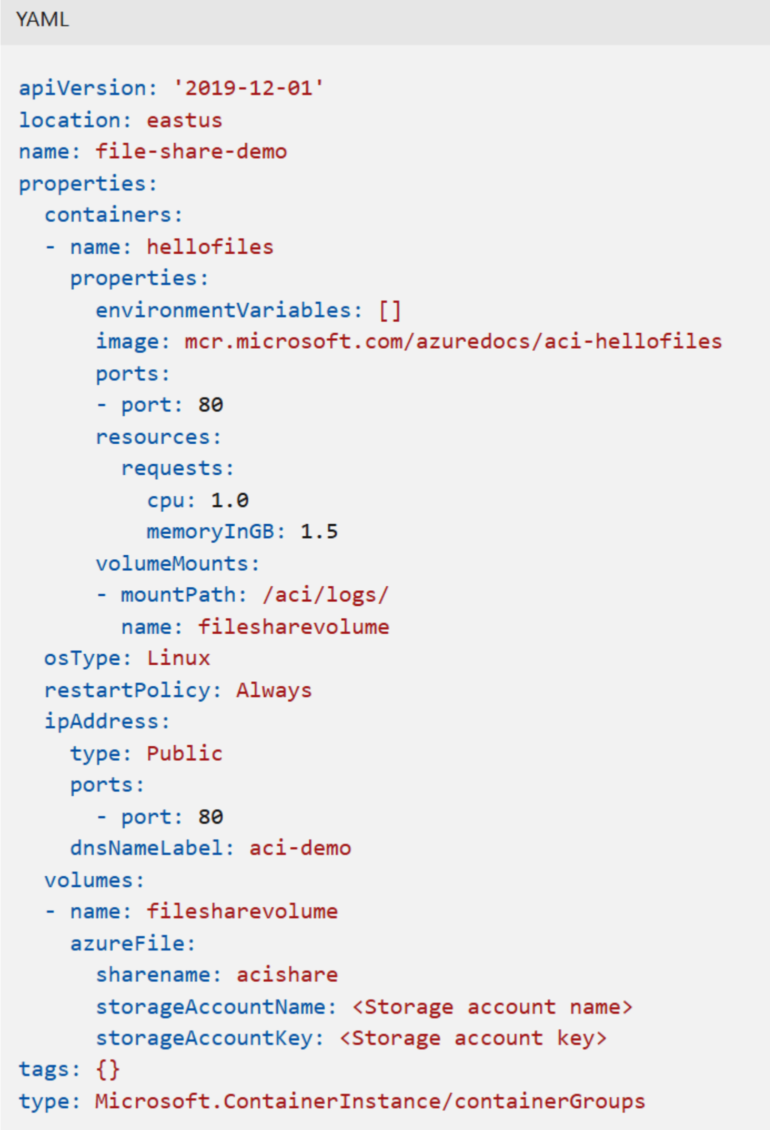
Deploy container and mount volume - YAML

- You can also deploy a container group and mount a volume in a container with the Azure CLI and a YAML template.

Mount multiple volumes

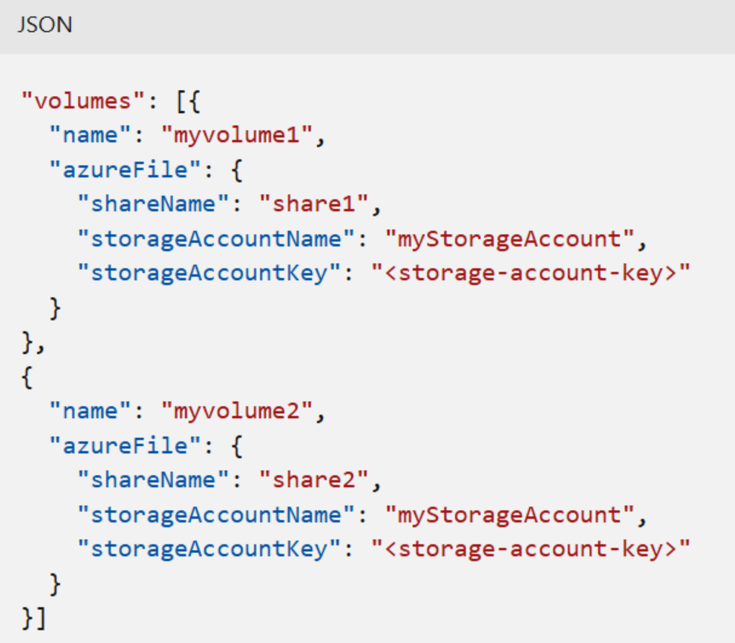
- To mount multiple volumes in a container instance, you must deploy using an Azure Resource Manager template or a YAML file.
- To use a template or YAML file, provide the share details and define the volumes by populating the volumes array in the properties section of the template.

Mount an Azure file share in Azure Container Instances (3 of 3)



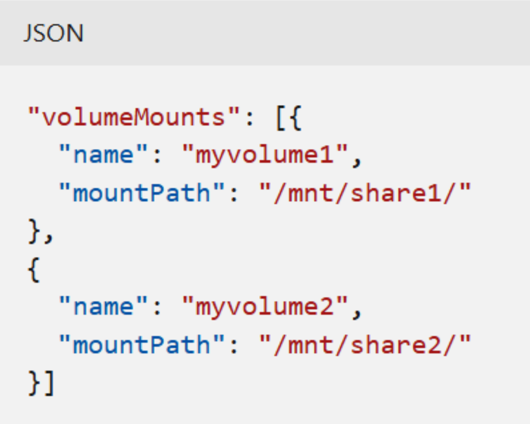
YAML

```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
```



JSON

```json
"volumes": [{
  "name": "myvolume1",
  "azureFile": {
    "shareName": "share1",
    "storageAccountName": "myStorageAccount",
    "storageAccountKey": "<storage-account-key>"
  }
},
{
  "name": "myvolume2",
  "azureFile": {
    "shareName": "share2",
    "storageAccountName": "myStorageAccount",
    "storageAccountKey": "<storage-account-key>"
  }
}]
```



JSON

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

Summary and knowledge check

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

1. What method is recommended when deploying a multi-container group that includes only containers?

2. What is the purpose of a restart policy in Azure Container Instances?

Module 3: Implement Azure Container Apps

Learning objectives

- Describe the features benefits of Azure Container Apps
- Deploy container app in Azure by using the Azure CLI
- Utilize Azure Container Apps built-in authentication and authorization
- Create revisions and implement app secrets

Introduction

Azure Container Apps provides the flexibility you need with a serverless container service built for microservice applications and robust autoscaling capabilities without the overhead of managing complex infrastructure.

Explore Azure Container Apps

Azure Container Apps enables you to run microservices and containerized applications on a serverless platform that runs on top of Azure Kubernetes Service.

- Supports dynamic scaling based on any KEDA-supported scaler
- Container apps are deployed to a single Container Apps environment, which acts as a secure boundary around groups of container apps.
- Independently develop, upgrade, version, and scale core areas of functionality in an overall system.
- Native Distributed Application Runtime (Dapr) integration

Exercise: Deploy a container app

In this exercise you create a secure Container Apps environment and deploy container app.

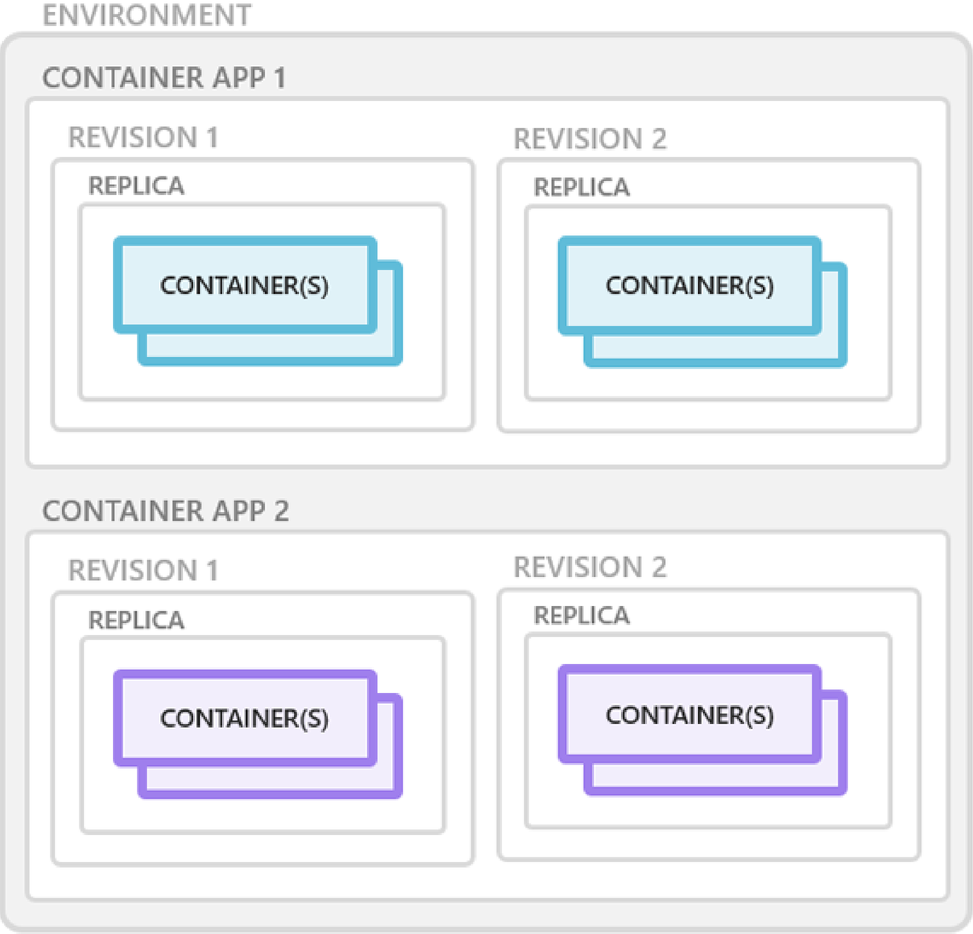
Objectives

- Prepare your environment
- Create an Azure Container Apps environment
- Create a container app
- Verify deployment
- Clean up resources

Explore containers in Azure Container Apps

- Containers for an Azure Container App are grouped together in pods inside revision snapshots.
- Can define multiple containers in a single container app to implement the sidecar pattern.
- Deploy images hosted on private registries by providing credentials in the Container Apps configuration.

Containers for an Azure Container App are grouped together in pods inside revision snapshots.



Manage revisions and secrets in Azure Container Apps

Revisions

- Azure Container Apps implements container app versioning by creating revisions.
- Control which revisions are active, and the external traffic that is routed to each active revision.
- The `az containerapp update` command can modify environment variables, compute resources, scale parameters, and deploy a different image.
- If the update includes revision-scope changes, a new revision is generated.

Secrets

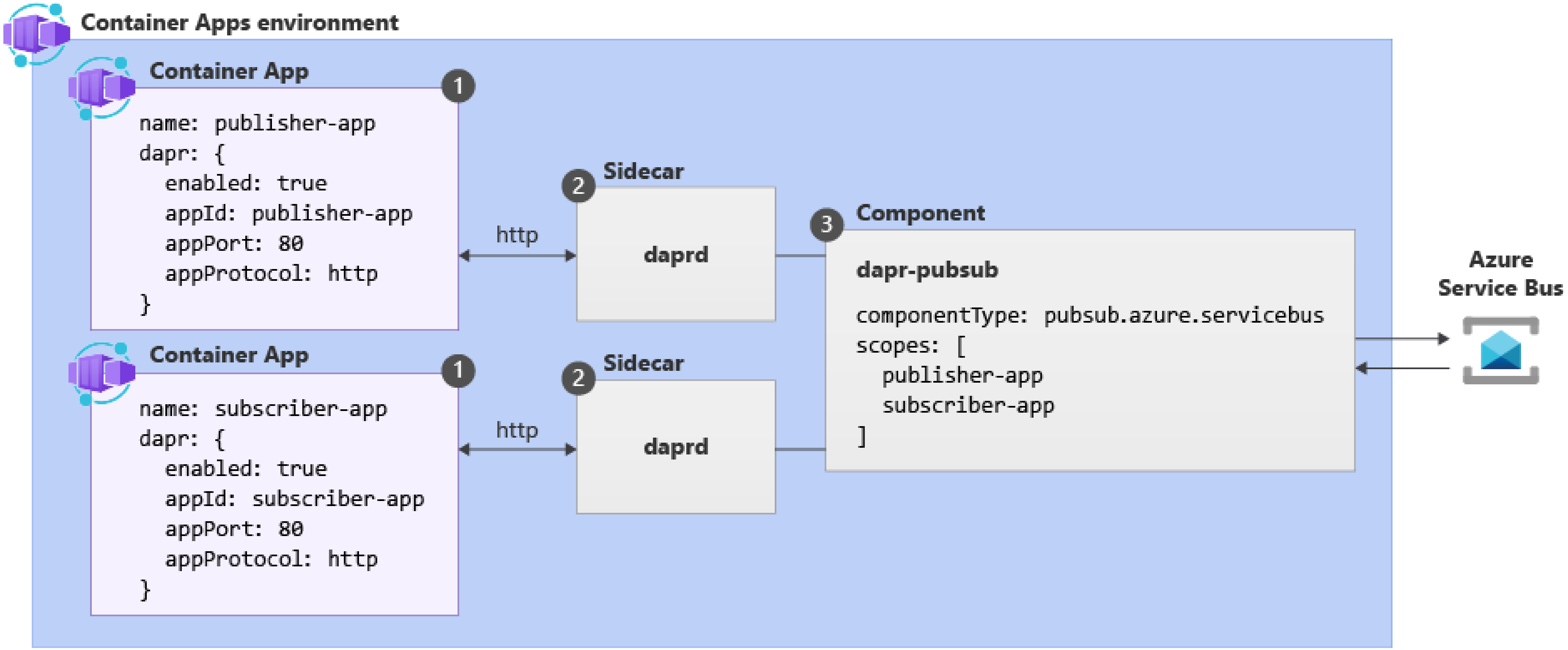
- Secrets are defined at the application level, secured values are available to container apps.
- Each application revision can reference one or more secrets.
- When you create a container app, secrets are defined using the `--secrets` parameter.

Explore Dapr integration with Azure Container Apps

The Distributed Application Runtime (Dapr) provides capabilities for enabling application intercommunication, whether through messaging via pub/sub or reliable and secure service-to-service calls.

Dapr API	Description
Service-to-service invocation	Discover services and perform reliable, direct service-to-service calls with automatic mTLS authentication and encryption.
State management	Provides state management capabilities for transactions and CRUD operations.
Pub/sub	Allows publisher and subscriber container apps to intercommunicate via an intermediary message broker.
Bindings	Trigger your applications based on events.
Actors	Dapr actors are message-driven, single-threaded, units of work designed to quickly scale.
Observability	Send tracing information to an Application Insights backend.
Secrets	Access secrets from your application code or reference secure values in your Dapr components.

Dapr core concepts



```
name: publisher-app
dapr: {
  enabled: true
  appId: publisher-app
  appPort: 80
  appProtocol: http
}
```

```
name: subscriber-app
dapr: {
  enabled: true
  appId: subscriber-app
  appPort: 80
  appProtocol: http
}
```

dapr-pubsub

```
componentType: pubsub.azure.servicebus
scopes: [
  publisher-app
  subscriber-app
]
```

Summary and knowledge check

In this module, you learned how to:

- Describe the features benefits of Azure Container Apps
- Deploy container app in Azure by using the Azure CLI
- Utilize Azure Container Apps built-in authentication and authorization
- Create revisions and implement app secrets

1 What is a revision in Azure Container Apps?

Discussion and lab

Group discussion questions

- What factors should you consider when deciding between Azure Container Instances and Azure Container Apps as a deployment target?
- Describe the architecture of an Azure Container App environment. How could some of your apps work within that architecture?
- Describe at least two elements of a Dockerfile. What tools would you use to create a container image?

Lab 05: Deploy compute workloads by using images and containers

In this lab, you will explore how to create and deploy containers to the Azure Container Registry using a .NET application and docker files. And also deploy a containerized solution to Azure Container Apps.

http://aka.ms/az204labs

- Exercise 1: Create a Docker container image and deploy it to Azure Container Registry
- Exercise 2: Deploy an Azure container instance
- Exercise 3: Create a secure Container Apps environment and deploy container app

End of presentation