# Azure Containers

## Table of Contents

# 1. Introduction

Docker was initially developed for Linux and has expanded to support Windows. Individual Docker images are either Windows-based or Linux-based, but can't be both at the same time.

For example, Microsoft offers Windows and Linux Docker images containing an ASP.NET Core environment that can be used as the basis for containerized ASP.NET Core applications.

Linux computers with Docker installed can only run Linux containers. Windows computers with Docker installed can run both kinds of containers. Windows accomplishes this by using a virtual machine to run a Linux system and uses the virtual Linux system to run Linux containers.

# 2. ACR

## 2.1. Create an ACR — build images

1. Create a resource group if necessary

```
az group create \
  --name [group_name] \
  --location [location]
```

2. Create an ACR of standard scalability and storage using the admin account

```
az acr create \
  --name [registry_name] \
  --resource-group [group_name] \
  --sku standard \
  --admin-enabled true
```

3. Get registry password and username to login to the specified registry

```
az acr credential show \
  --name [registry_name]
```

4. Login to the private ACR (enter registry username and password)

```
docker login [registry_name].azurecr.io
```

5. Create an alias for the image that specifies the repository and tag to be created in the Docker registry

```
docker tag [local_image_name:tag] [registry_name].azurecr.io/[local_image_name:tag]
```

6. Check image with annotated tags

```
docker images
```

7. Upload the specified image to the registry in the ACR

```
docker push [registry_name].azurecr.io/[local_image_name:tag]
```

8. Query the repositories in the registry to confirm the image upload

```
az acr repository list \
  --name [registry_name] ①
```

① you can additionally specify the username (`--username [username]` and password `--password [password]`

9. List the images and their properties as JSON in the specified registry

```
az acr repository show \
  --repository [repository_name] \
  --name [registry_name]
```

10. Cleanup resources if necessary with `az group delete --name [group_name]`

Keep in mind that you'll have to create a **resource group before** you create the registry if no resource suitable group exists.

The container registry name must be unique within Azure and contain between 5 and 50 alphanumeric characters.

You'll see at least two tags for each image in a repository. One tag will be value you specified in the acr build command. The other will be latest. Every time you rebuild an image, ACR automatically creates the latest tag as an alias for the most recent version of the image.

The name of the repo in the registry is equivalent to the name of the local image.

## 2.2. Manage ACR

**Enable the admin account**

```
az acr update -n [registry_name] \
  --admin-enabled true
```

**Get registry password and username to login to the specified registry**

```
az acr credential show \
  --name [registry_name]
```

**Get registry password to login to the specified registry**

```
az acr credential show \
  --name [registry_name] \
  --query passwords[0].value
```

**Get the URL of the login-server**

```
az acr show \
  --name [registry_name] \
  --query loginServer
```

**Login to the login-server with the specified username and password**

```
docker login [registry_name].azurecr.io \
  --username=[username] \
  --password=[password]
```

**Build an image according to the Dockerfile instructions and store it in the registry**

```
az acr build \
  --registry [registry_name] \
  --image [image_name] .
```

**Show tags of the specified repo as text**

```
az acr repository show-tags \
  --name [registry_name] \
  --repository [repository_name] \
  --username [username] \
  --password [password] \
  --output text
```

**Remove the resource group, the container registry, and the container images stored**

```
az group delete \
  --name [group_name]
```

The **login-server URL** for a registry in Azure Container Registry has the form `[registry_name].azurecr.io`.

## 2.3. Enable continuous integration

1. Define a task to automatically build an image from the source code and store it to the specified registry in the ACR

```
az acr task create \
  --registry [registry_name] \
  --name [task_name] \
  --image [image_name] \
  --context [repository_name] \
  --branch master \
  --file Dockerfile \
  --username [username] \
  --password [password]
```

1. Configure CD and create a webhook
    - go to *Container Settings* in the *Azure portal*
    - check *Continuous Deployment* and save changes
    - go to the *Webhooks* page and check the webhook status
2. Deploy the app to use the webhook for automatic image rebuilds

```
az acr build \
  --registry [registry_name] \
  --image [image_name] .
```

## 2.4. Create a replicated region for ACR

1. Replicate the specified registry to another region

    ```
    az acr replication create \
      --registry [registry_name] \
      --location [replicated_region]
    ```

2. Show all container image replicas created

    ```
    az acr replication list \
      --registry [registry_name] \
      --output table
    ```

# 3. ACI

## 3.1. Create an ACI — run containers

1. Create an ACI, which loads the image from the ACR, and run it in Azure

```
az container create \
  --resource-group [group_name] \
  --name [instance_name] \
  --image [registry_name].azurecr.io/[image_name:latest] \
  --dns-name-label [dns_name] \
  --registry-username [username] \
  --registry-password [password]
```

2. Check running containers

```
docker ps
```

3. Query the IP address of the instance to find the fully qualified domain name of the instance

```
az container show \
  --resource-group [group_name] \
  --name [instance_name] \
  --query ipAddress.fqdn
```

The instance will be **allocated a public IP address**. You access the instance with this IP address. You can **optionally specify a DNS name** if you prefer to reference the instance through a more user-friendly label.

The default port is 80 and the port protocol is TCP.

## 3.2. Manage containers

### 3.2.1. Container lifecycle

**Deploy a container inside the specified resource group with the specified properties**

```
az container create \
  --resource-group [group_name] \
  --name [instance_name] \
  --image [full_image_name]\
  --dns-name-label [dns_name] \
  --cpu [1] \
  --memory [1] \
  --ip-address Public \
  --location [location] \
  --image-registry-login-server [login_server] \
  --registry-username [username] \
  --registry-password [password]
```

**Start the specified container**

```
az container start \
  --resource-group [group_name] \
  --name [instance_name]
```

**Stop the specified container**

```
az container stop \
  --resource-group [group_name] \
  --name [instance_name
```

**Restart the specified container**

```
az container restart \
  --resource-group [group_name] \
  --name [instance_name]
```

**Delete the specified container**

```
az container delete \
  --resource-group [group_name] \
  --name [instance_name]
```

### 3.2.2. Container information

**List containers**

```
az container list
```

**List all containers in a resource group**

```
az container list \
  --resource-group [group_name]
```

**List specified information of all containers in a resource group**

```
az container list \
  --resource-group [group_name] \
  --query value[].[name,provisioningState]
```

**Show the IP address and provisioning state of the specified container table-formatted**

```
az container show \
  --resource-group [group_name] \
  --name [instance_name] \
  --query "{FQDN:ipAddress.fqdn,ProvisioningState:provisioningState}" --out table
```

**Query the IP address of the specified container instance**

```
az container show \
  --resource-group [group_name] \
  --name [instance_name] \
  --query ipAddress.ip
```

**Show all details of the specified container (JSON)**

```
az container show \
  --resource-group [group_name] \
  --name [instance_name]
```

**List specified information of the specified container in a resource group**

```
az container show \
  --resource-group [group_name] \
  --name [instance_name] \
  --query value[].[name,provisioningState]
```

### 3.2.3. Container stats

**Show the logs of the specified container (JSON)**

```
az container logs \
  --resource-group [group_name] \
  --name [instance_name]
```

**Attach the local standard out and standard error streams to that of the container**

```
az container attach \
  --resource-group [group_name] \
  --name [instance_name]
```

The container details is shown as JSON, specifying e.g. the operating system, the image, resource requests, etc.

# 4. AKS

## 4.1. Deploy an ACI with Kubernetes as orchestrator

1. `az group create --name [group_name] --location [location]` — create a resource group if necessary
2. `az aks create --resource-group [group_name] --name [cluster_name] --node-count 1 --enable-addons monitoring --generate-ssh-keys` — create an AKS cluster
3. `az acs kubernetes install-cli` — install the Kubernetes CLI tool to manage the Kubernetes cluster **kubectl**
4. `az aks get-credentials --resource-group [group_name] --name [cluster_name]` Kubernetes cluster
5. `kubectl get nodes` — verify the connection to your cluster (status must be ready)
6. `kubectl apply -f [kubernetes_yaml_file_name].yaml`— deploy the app and specify the name of your YAML manifest
7. `kubectl get service [service_name] --watch` — monitor the deployment process (stop the watch process when the EXTERNAL-IP address changes from pending to an actual public IP address)
8. `az group delete --name [group_name] --yes --no-wait` — delete the resource group and all resources it contains when the cluster is no longer needed

You can change any of the default AKS cluster properties after your cluster has been created.

---

*Intro video*

1. `az acs create --resource-group [group_name] --name [cluster_name] --dns-prefix [prefix] --generate-ssh-key --orchestrator-type kubernetes` — create a new container service
2. `az acs kubernetes install-cli` — install the Kubernetes CLI tool to manage the Kubernetes cluster **kubectl**
3. `az acs kubernetes get-credentials --resource-group [group_name] --name [cluster_name]` — get the credentials to configure kubectl to connect to your
4. `az ad sp create-for-rbac --role=Contributor --scopes /subscriptions/[subscription_id` — create a service principal
5. adjust content of yaml file to match service principal info (Azure client id, Azure client key, Azure tenant id, Azure subscription id, Azure resource group)
6. `kubectl create -f examples/aci-connector.yaml`— set up the ACI connector
7. create a Kubernetes manifest file

**Example Kubernetes manifest file**

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: azure-vote-back
spec:
  replicas: 1
  selector:
    matchLabels:
      app: azure-vote-back
    containers:
      - name: azure-vote-back
        image: mcr.microsoft.com/oss/bitnami/redis:6.0.8
        imagePullPolicy: Always
        env:
        - name: ALLOW_EMPTY_PASSWORD
          value: "yes"
        - name: AZURE_CLIENT_ID
         value: id
        - name: AZURE_CLIENT_KEY
         value: key
        - name: AZURE_TENANT_ID
          value: id
        - name: AZURE_SUBSCRIPTION_ID
          value: id
        - name: AZURE_RESOURCE_GROUP
          value: group
        resources:
          requests:
            cpu: 100m
            memory: 128Mi
          limits:
            cpu: 250m
            memory: 256Mi
        ports:
        - containerPort: 6379
          name: redis
```

# 5. Manage Azure Account

- `az login` — sign in to the Azure CLI
- `az logout` — Log out to remove access to Azure subscriptions
- `az account list -o table` — list Azure accounts table-formatted
- `az version` — find the version and dependent libraries that are installed
- `az upgrade` — upgrade to the latest version

## 5.1. Environment variables

```
1 ACR_NAME=[registry_name] ①
```

① referenced as `$ACR_NAME`

# 6. Glossary

**Registry**

A registry is a web service to which Docker can connect to upload and download container images. The most well-known registry is Docker Hub, which is a public registry.

**Repository**

A registry is organized as a series of repositories. Each repository contains multiple Docker images that share a common name and generally the same purpose and functionality. These images normally have different versions identified with a tag.

When you download and run an image, you must specify the registry, repository, and version tag for the image. Tags are text labels, and you can use your version numbering system (v1.0, v1.1, v1.2, v2.0, and so on).

**Base image**

The process of identifying a suitable base image usually starts with a search on Docker Hub for a ready-made image that **already contains an application framework** and all the utilities and tools of a Linux distribution like Ubuntu or Alpine.

**Dockerfile**

A Dockerfile is a plain text file containing all the commands needed to build an image. Dockerfiles are written in a minimal scripting language designed for **building and configuring images**, and documents the operations required to build an image starting with a base image. By convention, applications meant to be packaged as Docker images typically have a Dockerfile located in the **root of their source code**.

The `docker build` command creates a new image by running a Dockerfile. The `-f` flag indicates the name of the Dockerfile to use. The `-t` flag specifies the name of the image to be created. The final parameter, `.`, provides the build context for the source files for the *COPY* command.

The PORTS field indicates port 80 in the image was mapped to port 8080 on your computer.

**Resource Group**

An Azure resource group is a logical container into which Azure resources are deployed and managed.

**Azure Container Registry (ACR)**

Azure Container Registry is a registry hosting service provided by Azure. Each Azure Container Registry resource you create is a separate registry with a unique URL. These registries are private: they require authentication to push or pull images.

Container Registry is organized around repositories that contain one or more images. All images stored in a container registry can be signed and are encrypted at rest. In addition to storing and hosting images, you can also use Container Registry to build images.

Container Registry also lets you automate tasks such as redeploying an app when an image is rebuilt.

The Premium SKU of Container Registry includes 500 GiB of storage.

ACRs are highly scalable. They can be replicated to store images near where they're likely to be deployed.

**Azure Registry authentication**

the recommended authentication method is Azure service principal. Access to a registry with an Azure Active Directory identity is role-based, and identities can be assigned one of three roles: **reader** (pull access only), **contributor** (push and pull access), or **owner** (pull, push, and assign roles to other users).

The **admin account** is included with each registry; it is disabled by default.

When the admin is enabled ( --admin-enabled true`), ACR enables the registry name as the username and the admin access key as the password.

Only use the registry admin account for early testing and exploration, and do not share the username and password. Disable the admin account and use only role-based access with Azure Active Directory identities to maximize the security of your registry.

**Geo-replication**

Geo-replication enables an Azure container registry to function as a single registry, serving several regions with multi-master regional registries.

A geo-replicated registry provides the following benefits:

- Single registry/image/tag names can be used across multiple regions
- Network-close registry access from regional deployments
- No additional egress fees, as images are pulled from a local, replicated registry in the same region as your container host
- Single management of a registry across multiple regions

**Webhooks**

Azure App Service supports continuous deployment of a web app using webhooks. A webhook is a service offered by Azure Container Registry. Services and applications can subscribe to the webhook to receive notifications about updates to Docker images in the registry.

**Container Registry task**

Tasks are configured to monitor registries and trigger rebuilds each time the source code changes automatically. If the build finishes successfully, Container Registry can store the image in the repository. If your web app is set up for continuous integration in App Service, it receives a notification via the webhook and updates the app.

**Azure Container Instance (ACI)**

Azure Container Instances is a great solution for any scenario that can operate in isolated containers, including simple applications, task automation, and build jobs.

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 places over HTTPS, using TLS to secure client connections.

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

**Azure Kubernetes Service (AKS)**

An AKS cluster is a cloud hosted Kubernetes cluster. Azure Kubernetes Service simplifies deploying a managed Kubernetes cluster in Azure by offloading much of the complexity and operational overhead to Azure.

As a hosted Kubernetes service, Azure handles critical tasks for you, like health monitoring and maintenance. AKS environment is enabled with features such as automated updates, self-healing, and easy scaling.

You can use Resource Manager templates to automate cluster creation. With these templates, you specify features such as advanced networking, Azure Active Directory (AD) integration, and monitoring.

Configure basic information about the cluster:

- The Kubernetes cluster name
- The version of Kubernetes to install
- A DNS prefix to make the master node publicly accessible
- The initial node pool size

The Kubernetes cluster master node is managed by Azure and is free. You manage the agent nodes in the cluster and only pay for the node VMs, storage, and networking resources consumed in your cluster.

The initial node pool size defaults to two nodes, however it's recommended that at least three nodes are used for a production environment.

In production and cloud deployments, the preferred configuration is a high-availability deployment with three to five replicated control planes, instead of a single control plane.

AKS supports:

- Identity and security management.
- Integrated logging and monitoring.
- Auto cluster node and pod scaling.
- Cluster node upgrades.
- Static and dynamic storage volumes.
- GPU enabled nodes.

- Cluster deployment into an existing virtual network.
- Ingress with HTTP application routing.
- Docker file image format.
- Private container registry.

AKS also supports all the popular development and management tools such as Helm, Draft, Kubernetes extension for Visual Studio Code and Visual Studio Kubernetes Tools.

For information on the deployment center and DevOps Spaces go to Introduction to Azure Kubernetes Service.

# 7. References

## 7.1. MS modules

- Build a containerized web application with Docker
- Build and store container images with Azure Container Registry
- Deploy and run a containerized web app with Azure App Service
- Introduction to Azure Kubernetes Service

## 7.2. MS Docs

- Quickstart: Create a private container registry using the Azure CLI
- Quickstart: Deploy an Azure Kubernetes Service cluster using the Azure CLI