100 XP

Introduction

3 minutes

Azure Container Registry (ACR) is a managed, private Docker registry service based on the open-source Docker Registry 2.0. Create and maintain Azure container registries to store and manage your private Docker container images.

After completing this module, you'll be able 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

Next unit: Discover the Azure Container Registry

Continue

How are we doing? ☆ ☆ ☆ ☆ ☆

100 XP

Discover the Azure Container Registry

3 minutes

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. Build on demand, or fully automate builds with triggers such as source code commits and base image updates.

Use cases

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

- **Scalable orchestration systems** that manage containerized applications across clusters of hosts, including Kubernetes, DC/OS, and Docker Swarm.
- **Azure services** that support building and running applications at scale, including Azure Kubernetes Service (AKS), App Service, Batch, Service Fabric, and others.

Developers can also push to a container registry as part of a container development workflow. For example, target a container registry from a continuous integration and delivery tool such as Azure Pipelines or Jenkins.

Configure ACR Tasks to automatically rebuild application images when their base images are updated, or automate image builds when your team commits code to a Git repository. Create multi-step tasks to automate building, testing, and patching multiple container images in parallel in the cloud.

Azure Container Registry service tiers

Azure Container Registry is available in multiple service tiers. These tiers provide predictable pricing and several options for aligning to the capacity and usage patterns of your private Docker registry in Azure.

Tier	Description

Tier	Description
Basic	A cost-optimized entry point for developers learning about Azure Container Registry. Basic registries have the same programmatic capabilities as Standard and Premium (such as Azure Active Directory authentication integration, image deletion, and webhooks). However, the included storage and image throughput are most appropriate for lower usage scenarios.
Standard	Standard registries offer the same capabilities as Basic, with increased included storage and image throughput. Standard registries should satisfy the needs of most production scenarios.
Premium	Premium registries provide the highest amount of included storage and concurrent operations, enabling high-volume scenarios. In addition to higher image throughput, Premium adds features such as geo-replication for managing a single registry across multiple regions, content trust for image tag signing, and private link with private endpoints to restrict access to the registry.

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. In addition to Docker container images, Azure Container Registry stores related content formats such as Helm charts and images built to the Open Container Initiative (OCI) Image Format Specification .

Automated image builds

Use Azure Container Registry Tasks (ACR Tasks) to streamline building, testing, pushing, and deploying images in Azure. Configure build tasks to automate your container OS and framework patching pipeline, and build images automatically when your team commits code to source control.

Next unit: Explore storage capabilities

Continue >

How are we doing? ☆ ☆ ☆ ☆ ☆

100 XP

Explore storage capabilities

3 minutes

Every Basic, Standard, and Premium Azure container registry benefits from advanced Azure storage features like encryption-at-rest for image data security and geo-redundancy for image data protection.

- **Encryption-at-rest:** All container images in your registry are encrypted at rest. Azure automatically encrypts an image before storing it, and decrypts it on-the-fly when you or your applications and services pull the image.

- **Regional storage:** Azure Container Registry stores data in the region where the registry is created, to help customers meet data residency and compliance requirements. In all regions except Brazil South and Southeast Asia, Azure may also store registry data in a paired region in the same geography. In the Brazil South and Southeast Asia regions, registry data is always confined to the region, to accommodate data residency requirements for those regions.

  If a regional outage occurs, the registry data may become unavailable and is not automatically recovered. Customers who wish to have their registry data stored in multiple regions for better performance across different geographies or who wish to have resiliency in the event of a regional outage should enable geo-replication.

- **Zone redundancy:** A feature of the Premium service tier, zone redundancy uses Azure availability zones to replicate your registry to a minimum of three separate zones in each enabled region.

- **Scalable storage:** Azure Container Registry allows you to create as many repositories, images, layers, or tags as you need, up to the registry storage limit.

  Very high numbers of repositories and tags can impact the performance of your registry. Periodically delete unused repositories, tags, and images as part of your registry maintenance routine. Deleted registry resources like repositories, images, and tags *cannot* be recovered after deletion.

Next unit: Build and manage containers with tasks

Continue >

How are we doing? ☆ ☆ ☆ ☆ ☆

100 XP

Build and manage containers with tasks

3 minutes

ACR Tasks is a suite of features within Azure Container Registry. It provides cloud-based container image building for platforms including Linux, Windows, and Azure Resource Manager, and can automate OS and framework patching for your Docker containers. ACR Tasks enables automated builds triggered by source code updates, updates to a container's base image, or timers.

Task scenarios

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

- **Quick task** - Build and push a single container image to a container registry on-demand, in Azure, without needing a local Docker Engine installation. Think `docker build, docker push` in the cloud.
- **Automatically triggered tasks** - Enable one or more *triggers* to build an image:
  - Trigger on source code update
  - Trigger on base image update
  - Trigger on a schedule
- **Multi-step task** - Extend the single image build-and-push capability of ACR Tasks with multi-step, multi-container-based workflows.

Each ACR Task has an associated source code context - the location of a set of source files used to build a container image or other artifact. Example contexts include a Git repository or a local filesystem.

Quick task

Before you commit your first line of code, ACR Tasks's quick task feature can provide an integrated development experience by offloading your container image builds to Azure. With quick tasks, you can verify your automated build definitions and catch potential problems prior to committing your code.

Using the familiar `docker build` format, the az acr build command in the Azure CLI takes a context (the set of files to build), sends it to ACR Tasks and, by default, pushes the built image to its registry upon completion.

Trigger task on source code update

Trigger a container image build or multi-step task when code is committed, or a pull request is made or updated, to a public or private Git repository in GitHub or Azure DevOps. For example, configure a build task with the Azure CLI command `az acr task create` by specifying a Git repository and optionally a branch and Dockerfile. When your team updates code in the repository, an ACR Tasks-created webhook triggers a build of the container image defined in the repo.

Trigger on base image update

You can set up an ACR task to track a dependency on a base image when it builds an application image. When the updated base image is pushed to your registry, or a base image is updated in a public repo such as in Docker Hub, ACR Tasks can automatically build any application images based on it.

Schedule a task

Optionally schedule a task by setting up one or more timer triggers when you create or update the task. Scheduling a task is useful for running container workloads on a defined schedule, or running maintenance operations or tests on images pushed regularly to your registry.

Multi-step tasks

Multi-step tasks, defined in a YAML file specify individual build and push operations for container images or other artifacts. They can also define the execution of one or more containers, with each step using the container as its execution environment. For example, you can create a multi-step task that automates the following:

1. Build a web application image
2. Run the web application container
3. Build a web application test image

4. Run the web application test container, which performs tests against the running application container
5. If the tests pass, build a Helm chart archive package
6. Perform a `helm upgrade` using the new Helm chart archive package

Next unit: Explore elements of a Dockerfile

Continue >

How are we doing? ☆ ☆ ☆ ☆ ☆

100 XP

Explore elements of a Dockerfile

3 minutes

If you want to create a custom container you will need to understand the elements of a Dockerfile. A Dockerfile is a text file that contains the instructions we use to build and run a Docker image. The following aspects of the image are defined:

- The base or parent image we use to create the new image
- Commands to update the base OS and install additional software
- Build artifacts to include, such as a developed application
- Services to expose, such a storage and network configuration
- Command to run when the container is launched

Let's map these aspects to an example Dockerfile. Suppose we're creating a Docker image for an ASP.NET Core website. The Dockerfile may look like the following example.

Bash Copy

```
# 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-trans-
port-https \
    && wget -q https://packages.microsoft.com/config/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

# STEP 5: Configure work directory
WORKDIR /app
```

```
# STEP 6: Copy website code to container
COPY ./website/. .

# STEP 7: Configure network requirements
EXPOSE 80:8080

# STEP 8: Define the entry point of the process that runs in the container
ENTRYPOINT ["dotnet", "website.dll"]
```

We're not going to cover the Dockerfile file specification here or the detail of each command in our above example. However, notice that there are several commands in this file that allow us to manipulate the structure of the image.

Each of these steps creates a cached container image as we build the final container image. These temporary images are layered on top of the previous and presented as single image once all steps complete.

Finally, notice the last step, step 8. The `ENTRYPOINT` in the file indicates which process will execute once we run a container from an image.

Additional resources

- Dockerfile reference
  - https://docs.docker.com/engine/reference/builder/

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

Continue >

How are we doing? ☆ ☆ ☆ ☆ ☆

100 XP

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

10 minutes

In this exercise you will use ACR Tasks to perform the following actions:

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

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 start the Cloud Shell

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

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

Create an Azure Container Registry

1. Create a resource group for the registry, replace `<myLocation>` in the command below with a location near you.

Bash Copy

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

2. Create a basic container registry. The registry name must be unique within Azure, and contain 5-50 alphanumeric characters. Replace `<myContainerRegistry>` in the command below with a unique value.

Bash Copy

```
az acr create --resource-group az204-acr-rg \
    --name <myContainerRegistry> --sku Basic
```

> **Note**
>
> The command above creates a *Basic* registry, a cost-optimized option for developers learning about Azure Container Registry.

Build and push image from a Dockerfile

Now use Azure Container Registry to build and push an image based on a local Dockerfile.

1. Create, or navigate, to a local directory and then use the command below to create the Dockerfile. The Dockerfile will contain a single line that references the `hello-world` image hosted at the Microsoft Container Registry.

Bash Copy

```
echo FROM mcr.microsoft.com/hello-world > Dockerfile
```

2. Run the `az acr build` command, which builds the image and, after the image is successfully built, pushes it to your registry. Replace `<myContainerRegistry>` with the name you used earlier.

Azure CLI Copy

```
az acr build --image sample/hello-world:v1  \
    --registry <myContainerRegistry> \
    --file Dockerfile .
```

The command above will generate a lot of output, below is shortened sample of that output showing the last few lines with the final results. You can see in the `repository` field the `sample/hello-word` image is listed.

```
- image:
    registry: <myContainerRegistry>.azurecr.io
    repository: sample/hello-world
    tag: v1
    digest:
sha256:92c7f9c92844bbbb5d0a101b22f7c2a7949e40f8ea90c8b3bc396879d95e899a
  runtime-dependency:
    registry: mcr.microsoft.com
    repository: hello-world
    tag: latest
    digest:
sha256:92c7f9c92844bbbb5d0a101b22f7c2a7949e40f8ea90c8b3bc396879d95e899a
  git: {}


Run ID: cf1 was successful after 11s
```

Verify the results

1. Use the `az acr repository list` command to list the repositories in your registry. Replace `<myContainerRegistry>` with the name you used earlier.

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

Output:

```
Result
----------------
sample/hello-world
```

2. Use the `az acr repository show-tags` command to list the tags on the **sample/hello-world** repository. Replace `<myContainerRegistry>` with the name you used earlier.

Bash Copy

```
az acr repository show-tags --name <myContainerRegistry> \
    --repository sample/hello-world --output table
```

Output:

Copy

```
Result
--------
v1
```

Run the image in the ACR

1. Run the sample/hello-world:v1 container image from your container registry by using the `az acr run` command. The following example uses `$Registry` to specify the registry where you run the command. Replace `<myContainerRegistry>` with the name you used earlier.

Bash Copy

```
az acr run --registry <myContainerRegistry> \
    --cmd '$Registry/sample/hello-world:v1' /dev/null
```

The `cmd` parameter in this example runs the container in its default configuration, but `cmd` supports additional `docker run` parameters or even other `docker` commands.

Below is shortened sample of the output:

Copy

```
Packing source code into tar to upload...
Uploading archived source code from
'/tmp/run_archive_ebf74da7fcb04683867b129e2ccad5e1.tar.gz'...
Sending context (1.855 KiB) to registry: mycontainerre...
Queued a run with ID: cab
Waiting for an agent...
2019/03/19 19:01:53 Using acb_vol_60e9a538-b466-475f-9565-80c5b93eaa15 as the
home volume
2019/03/19 19:01:53 Creating Docker network: acb_default_network, driver:
'bridge'
2019/03/19 19:01:53 Successfully set up Docker network: acb_default_network
2019/03/19 19:01:53 Setting up Docker configuration...
```

```
2019/03/19 19:01:54 Successfully set up Docker configuration
2019/03/19 19:01:54 Logging in to registry: mycontainerregistry008.azurecr.io
2019/03/19 19:01:55 Successfully logged into mycontainerregistry008.azurecr.io
2019/03/19 19:01:55 Executing step ID: acb_step_0. Working directory: '',
Network: 'acb_default_network'
2019/03/19 19:01:55 Launching container with name: acb_step_0

Hello from Docker!
This message shows that your installation appears to be working correctly.

2019/03/19 19:01:56 Successfully executed container: acb_step_0
2019/03/19 19:01:56 Step ID: acb_step_0 marked as successful (elapsed time in
seconds: 0.843801)

Run ID: cab was successful after 6s
```

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-acr-rg --no-wait
```

Next unit: Knowledge check

Continue >

How are we doing? ☆ ☆ ☆ ☆ ☆

200 XP

Knowledge check

3 minutes

Check your knowledge

1. Which of the following Azure Container Registry support geo-replication to manage a single registry across multiple regions?

- ○ Basic
- ○ Standard
- ◉ Premium

✔ That's correct. The premium tier adds geo-replication as a feature.

Check your answers

How are we doing? ☆ ☆ ☆ ☆ ☆

100 XP

Summary

3 minutes

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

Module complete:

Unlock achievement

How are we doing? ☆ ☆ ☆ ☆ ☆