Microsoft LATAM Partner Bootcamp

Cloud App Dev

Azure Container Instances & Azure Container Registry

January 2018

Contents

Cloud App Dev Bootcamp: ACI & ACR

Requirements

- Microsoft Azure (tree/trial or paid subscription)
- Local machine or a virtual machine (VM), configured with:
    - Docker Engine
        - Option 1 – Install in your own local machine or VM
            - https://docs.docker.com/engine/installation/
        - Option 2 – Deploy a VM with Docker on Ubuntu (does not include Azure CLI)
            - https://azuremarketplace.microsoft.com/en-us/marketplace/apps/CanonicalandMSOpenTech.DockerOnUbuntuServer1404LTS
    - Git (VM from "Option 2" in previous step has already installed git)
        - https://git-scm.com/book/en/v2/Getting-Started-Installing-Git
    - Azure CLI (version 2.0.23 or later)
        - Install Azure CLI
            - https://docs.microsoft.com/en-us/cli/azure/install-azure-cli?view=azure-cli-latest
        - If you have more than 1 Azure subscription, please learn how to manage multiple subscriptions
            - https://docs.microsoft.com/en-us/cli/azure/manage-azure-subscriptions-azure-cli?view=azure-cli-latest

Exercise 1: Create container for deployment to Azure Container Instances

Overview

Azure Container Instances enables deployment of Docker containers onto Azure infrastructure without provisioning any virtual machines or adopting any higher-level service. In this exercise, you build a small web application in Node.js and package it in a container that can be run using Azure Container Instances.

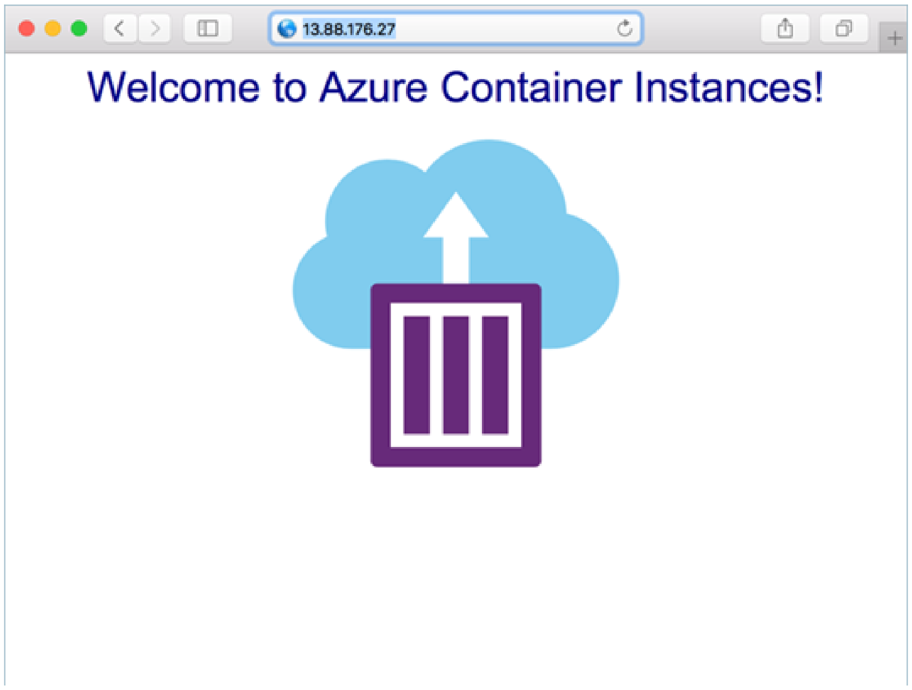
In this exercise, part one of the hands-on-lab, you:

- Clone application source code from GitHub
- Create a container image from application source
- Test the image in a local Docker environment

In subsequent exercises, you upload your image to an Azure Container Registry, and then deploy it to Azure Container Instances.

Task 1: Get application code

The sample in this hands-on-lab includes a simple web application built in Node.js. The app serves a static HTML page and looks like this:



Use git to download the sample:

```
git clone https://github.com/Azure-Samples/aci-helloworld.git
```

Task 2: Build the container image

The Dockerfile provided in the sample repo shows how the container is built. It starts from an official Node.js image based on Alpine Linux, a small distribution that is well suited to use with containers. It then copies the application files into the container, installs dependencies using the Node Package Manager, and finally starts the application.

```
FROM node:8.9.3-alpine
RUN mkdir -p /usr/src/app
COPY ./app/* /usr/src/app/
WORKDIR /usr/src/app
RUN npm install
CMD node /usr/src/app/index.js
```

Use the *docker build* command to create the container image, tagging it as *aci-tutorial-app*:

```
docker build ./aci-helloworld -t aci-tutorial-app
```

Output from the *docker build* command is similar to the following (truncated for readability):

```
Sending build context to Docker daemon  119.3kB
Step 1/6 : FROM node:8.9.3-alpine
8.9.3-alpine: Pulling from library/node
88286f41530e: Pull complete
84f3a4bf8410: Pull complete
d0d9b2214720: Pull complete
Digest:
sha256:c73277ccc763752b42bb2400d1aaecb4e3d32e3a9dbedd0e49885c71bea07354
Status: Downloaded newer image for node:8.9.3-alpine
 ---> 90f5ee24bee2
...
Step 6/6 : CMD node /usr/src/app/index.js
 ---> Running in f4a1ea099eec
 ---> 6edad76d09e9
Removing intermediate container f4a1ea099eec
Successfully built 6edad76d09e9
Successfully tagged aci-tutorial-app:latest
```

Use the *docker images* command to see the built image:

```
docker images
```

Output:

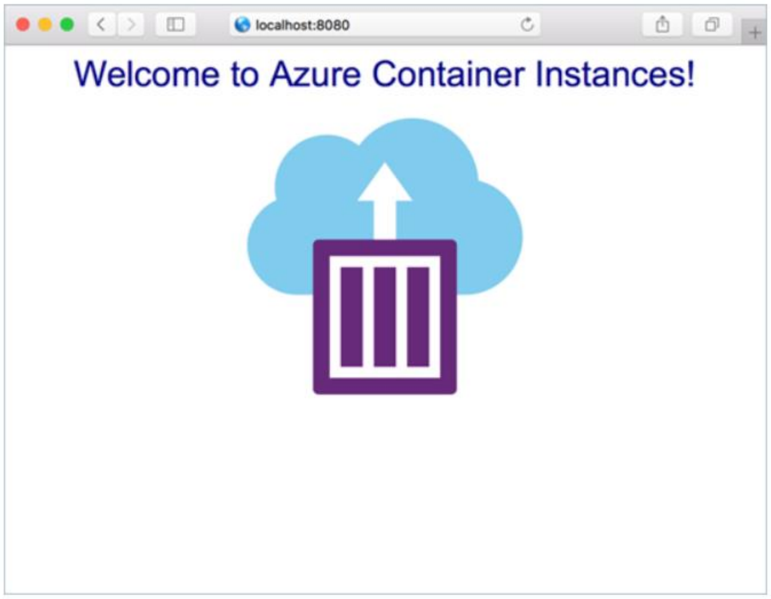
```
REPOSITORY           TAG            IMAGE ID         CREATED           SIZE
aci-tutorial-app     latest         5c745774dfa9     39 seconds ago    68.1 MB
```

Task 3: Run the container locally

Before you try deploying the container to Azure Container Instances, run it locally to confirm that it works. The -d switch lets the container run in the background, while -p allows you to map an arbitrary port on your compute to port 80 in the container.

```
docker run -d -p 8080:80 aci-tutorial-app
```

To confirm that the container is running, please open the browser to http://localhost:8080 in the local machine if you have GUI access). If you are working in a virtual machine, please open the browser to <VirtualMachine-IP>:8080 in another PC (make sure to configure access from outside the VM), or type *curl localhost:8080* in the console (terminal) of the virtual machine (you can also install and use lynx in the console).



Exercise 2: Deploy and use Azure Container Registry

Overview

In the previous exercise, a container image was created for a simple web application written in Node.js. In this exercise, you push the image to an Azure Container Registry. If you have not created the container image, return to Exercise 1.

The Azure Container Registry is an Azure-based, private registry for Docker container images. This exercise walks you through deploying an Azure Container Registry instance and pushing a container image to it.

In this exercise, part two of the hands-on-labs, you:

- Deploy an Azure Container Registry instance
- Tag a container image for your Azure container registry
- Upload the image to your registry

In the next exercise, the final one, you deploy the container from your private registry to Azure Container Instances.

Task 1: Deploy Azure Container Registry

Login into your Azure account, typing the *az login* command and following the instructions.

```
az login
```

You will receive the following message:

```
To sign in, use a web browser to open the page https://aka.ms/devicelogin and
enter the code XXXXXX to authenticate.
```

When deploying an Azure Container Registry, you first need a resource group. An Azure resource group is a logical collection into which Azure resources are deployed and managed.

Create a resource group with the *az group create* command. In this example, a resource group named myResourceGroup is created in the eastus region.

```
az group create --name myResourceGroup --location eastus
```

Create an Azure container registry with the *az acr create* command. The container registry name must be unique within Azure, and must contain 5-50 alphanumeric characters. Replace <acrName> with a unique name for your registry:

```
az acr create --resource-group myResourceGroup --name <acrName> -
-sku Basic --admin-enabled true
```

For example, to create an Azure container registry named mycontainerregistryCR007:

```
az acr create --resource-group myResourceGroup --name
mycontainerregistryCR007 --sku Basic --admin-enabled true
```

Throughout the rest of this hands-on-lab, we use <acrName> as a placeholder for the container registry name that you chose.

Task 2: Container registry login

You must log in to your Azure Container Registry instance before pushing images to it. Use the *az acr login* command to complete the operation. You must provide the unique name you provided for the container registry when you created it, and also the username & password (go to the Azure Portal to get those values).

```
az acr login --name <acrName> -u <myAdminName> -p <myPassword1>
```

The command returns a Login Succeeded message once completed. If not, you can also log in with *docker login*. The following example passes the ID and password of the container registry admin:

```
docker login <acrName>.azurecr.io -u <myAdminName> -p
<myPassword1>
```

You might see a security warning recommending the use of the --password-stdin parameter. While its use is outside the scope of this article, we recommend following this best practice. For more information, see the *docker login* command reference online.

Task 3: Tag container image

To deploy a container image from a private registry, you must tag the image with the loginServer name of the registry.

To see a list of current images, use the *docker images* command.

```
docker images
```

Output:

```
REPOSITORY          TAG           IMAGE ID        CREATED          SIZE
aci-tutorial-app    latest        5c745774dfa9    39 seconds ago   68.1 MB
```

To get the loginServer name, run the *az acr show* command. Replace <acrName> with the name of your container registry.

```
az acr show --name <acrName> --query loginServer --output table
```

Example output:

```
Result
------------------------
mycontainerregistryCR007.azurecr.io
```

Tag the aci-tutorial-app image with the loginServer of your container registry. Also, add :v1 to the end of the image name. This tag indicates the image version number. Replace <acrLoginServer> with the result of the *az acr show* command you just executed.

```
docker tag aci-tutorial-app <acrLoginServer>/aci-tutorial-app:v1
```

Once tagged, run docker images to verify the operation.

```
docker images
```

Output:

```
REPOSITORY                                          TAG       IMAGE ID       CREATED          SIZE
aci-tutorial-app                                    latest    5c745774dfa9   39 seconds ago   68.1 MB
mycontainerregistry082.azurecr.io/aci-tutorial-app  v1        a9dace4e1a17   7 minutes ago    68.1 MB
```

Task 4: Push image to Azure Container Registry

Push the aci-tutorial-app image to the registry with the *docker push* command. Replace <acrLoginServer> with the full login server name you obtain in the earlier step.

```
docker push <acrLoginServer>/aci-tutorial-app:v1
```

The push operation should take a few seconds to a few minutes depending on your internet connection, and output is similar to the following:

```
The push refers to a repository [mycontainerregistryCR007.azurecr.io/aci-
tutorial-app]
3db9cac20d49: Pushed
13f653351004: Pushed
4cd158165f4d: Pushed
d8fbd47558a8: Pushed
44ab46125c35: Pushed
5bef08742407: Pushed
v1: digest:
sha256:ed67fff971da47175856505585dcd92d1270c3b37543e8afd46014d328f05715 size:
1576
```

Task 5: List images in Azure Container Registry

To return a list of images that have been pushed to your Azure Container registry, use the *az acr repository list* command.

```
az acr repository list --name <acrName> -u <myAdminName> -p
<myPassword1> --output table
```

Output:

```
Result
----------------
aci-tutorial-app
```

And then to see the tags for a specific image, use the following command.

```
az acr repository show-tags --name <acrName> --repository aci-
tutorial-app -u <myAdminName> -p <myPassword1> --output table
```

Output:

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

Exercise 3: Deploy a container to Azure Container Instances

Overview

This is the final exercise. Earlier in the hands-on-lab, a container image was created and pushed to an Azure Container Registry. This exercise completes the hands-on-lab by deploying the container to Azure Container Instances.

In this exercise, you:

- Deploy the container from the Azure Container Registry using the Azure CLI
- View the application in the browser
- View the container logs

Task 1: Deploy the container using the Azure CLI

The Azure CLI enables deployment of a container to Azure Container Instances in a single command. Since the container image is hosted in the private Azure Container Registry, you must include the credentials required to access it. Obtain the credentials with the following Azure CLI commands.

Container registry login server (update with your registry name):

```
az acr show --name <acrName> --query loginServer
```

Container registry password:

```
az acr credential show --name <acrName> --query
"passwords[0].value"
```

To deploy your container image from the container registry with a resource request of 1 CPU core and 1 GB of memory, run the following command. Replace <acrLoginServer> and <acrPassword> with the values you obtained from the previous two commands.

```
az container create --resource-group myResourceGroup --name aci-
tutorial-app --image <acrLoginServer>/aci-tutorial-app:v1 --cpu 1
--memory 1 --registry-password <acrPassword> --ip-address public
--ports 80
```

Within a few seconds, you should receive an initial response from Azure Resource Manager. To view the state of the deployment, use az container show:

```
az container show --resource-group myResourceGroup --name aci-
tutorial-app --query instanceView.state
```

Repeat the *az container show* command until the state changes from Pending to Running, which should take under a minute. When the container is Running, proceed to the next step.

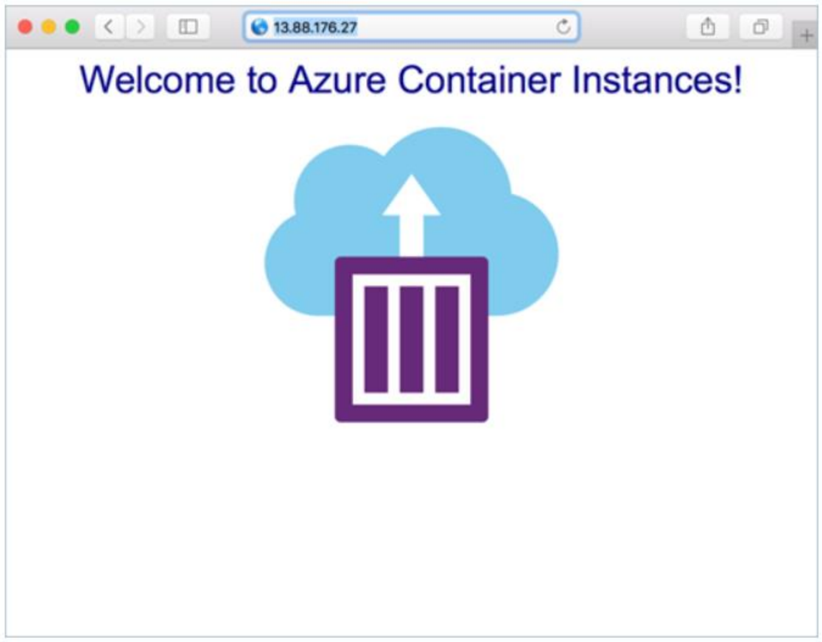
Task 2: View the application and container logs

Once the deployment succeeds, display the container's public IP address with the *az container show* command:

```
az container show --resource-group myResourceGroup --name aci-
tutorial-app --query ipAddress.ip
```

Example output: `"13.88.176.27"`

To see the running application, navigate to the public IP address in your favorite browser.



Welcome to Azure Container Instances!

You can also view the log output of the container:

```
az container logs --resource-group myResourceGroup --name aci-tutorial-app
```

Example output:

```
listening on port 80
::ffff:10.240.0.4 - - [21/Jul/2017:06:00:02 +0000] "GET / HTTP/1.1" 200 1663
"-" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_12_5) AppleWebKit/537.36
(KHTML, like Gecko) Chrome/59.0.3071.115 Safari/537.36"
::ffff:10.240.0.4 - - [21/Jul/2017:06:00:02 +0000] "GET /favicon.ico HTTP/1.1"
404 150 "http://13.88.176.27/" "Mozilla/5.0 (Macintosh; Intel Mac OS X
10_12_5) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/59.0.3071.115
Safari/537.36"
```

Task 3: Clean up resources

If you no longer need any of the resources you created in this hands-on-lab, you can execute the *az group delete* command to remove the resource group and all resources it contains. This command deletes the container registry you created, as well as the running container, and all related resources in the resource group.

```
az group delete --name myResourceGroup
```