

**Module 5 Lab: Windows Containers**

Template Version: 2.0

**Estimated Time**

90 minutes

**Objectives**

At the end of this lab, you will be able to:

* Run your container locally
* Create a simple Dockerfile for a web app and web API application
* Create a docker-compose.yml file

**Logon Information**

Please use the Azure Pass provided to you for this lab.

* Username: Your personal Outlook email address.
* Password: P@ssw0rd123!

Table of Contents

[Module 5 Lab: Windows Containers 3](#_Toc523416173)

[Exercise 1: Create a Dockerfile 3](#_Toc523416174)

[Exercise 2: Build an Image based on your Dockerfile 6](#_Toc523416175)

[Exercise 3: Running, Stopping, and Starting your Container 7](#_Toc523416176)

[Exercise 4: Tagging your Dockerfile and Push it to the Azure Container Registry (ACR) 9](#_Toc523416177)

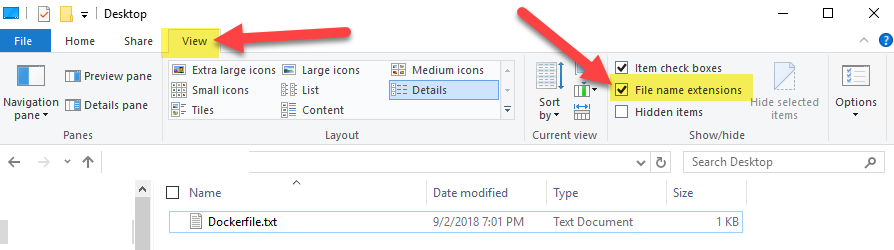
[Exercise 5: End to End 11](#_Toc523416178)

Module 5 Lab: Windows Containers

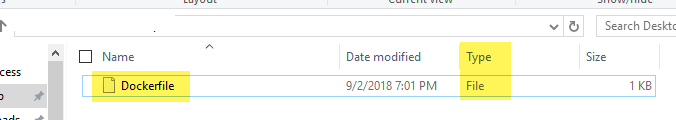
Exercise 1: Create a Dockerfile

Tasks

1. Try it out! Make a Dockerfile for the API layer
2. RDP into your VM and complete all work on the VM.
3. Navigate to this page and use the example under the section titled “Create a Dockerfile for an ASP.NET Core application” as the starter: <https://docs.docker.com/engine/examples/dotnetcore/#prerequisites>
4. Open Notepad and copy the starting example into your Notepad.
5. Save the file as Dockerfile.txt under your C:/labs/day2/mywebapp directory. Then go to the file’s location in Windows Explorer, open the View tab, check “File name extensions”, right click on the Dockerfile.txt file, choose Rename, and remove the .txt extension, say Yes on the popup warning about unusable files.



1. The file should now look like this:



1. Make the following changes:
   1. Change the build-env image to: microsoft/aspnetcore-build:2.0.7-2.1.105
   2. Change the main base image to: microsoft/aspnetcore:2.0.7
   3. Change the ENTRYPOINT’s second parameter (the name of the .dll should change).
2. Save your changes
3. Check your solution
4. The correct Dockerfile is below, check your Dockerfile in comparison.

FROM microsoft/aspnetcore-build:2.0.7-2.1.105 AS build-env

WORKDIR /app

# Copy csproj and restore as distinct layers

COPY \*.csproj ./

RUN dotnet restore

# Copy everything else and build

COPY . ./

RUN dotnet publish -c Release -o out

# Build runtime image

FROM microsoft/aspnetcore:2.0.7

WORKDIR /app

COPY --from=build-env /app/out .

ENTRYPOINT ["dotnet", "mywebapp.dll"]

ENTRYPOINT ["dotnet", "mywebapi.dll"]

1. Make a Dockerfile for the Web App layer.
2. Copy the Dockerfile you just made into **C:/labs/day2/mywebapp** directory.
3. Add the following two lines before the ENTRYPOINT instruction. This will add an environment variable and expose port 9000 so that the backend API can be mapped to that port.

ENV ASPNETCORE\_URLS <http://+:9000>

EXPOSE 9000

1. In the ENTRYPOINT command, change the .dll to the correct name.
2. Check your solution
3. The correct Dockerfile is below, check your Dockerfile in comparison.

FROM microsoft/aspnetcore-build:2.0.7-2.1.105 AS build-env

WORKDIR /app

# Copy csproj and restore as distinct layers

COPY \*.csproj ./

RUN dotnet restore

# Copy everything else and build

COPY . ./

RUN dotnet publish -c Release -o out

# Build runtime image

FROM microsoft/aspnetcore:2.0.7

WORKDIR /app

COPY --from=build-env /app/out .

# Expose port 9000

ENV ASPNETCORE\_URLS http://+:9000

EXPOSE 9000

ENTRYPOINT ["dotnet", "mywebapi.dll"]

Exercise 1 has been completed

Exercise 2: Build an Image based on your Dockerfile

Tasks

1. Build an Image
2. Open PowerShell and check your current docker images (should be none):

docker images

1. Navigate to your code api project

cd C:\labs\day2\mywebapi

1. Run the following command to build the API image based on the steps in the Dockerfile located in the directory. When the Dockerfile hits the FROM command for an image that isn’t downloaded locally to the machine, it will do a docker pull from the Docker Hub. This will take about 10 minutes. Then the image and its layers will be cached on the machine for future use.

docker build -t demo-webapi .

1. Navigate to your code web project

cd C:\labs\day2\mywebapp

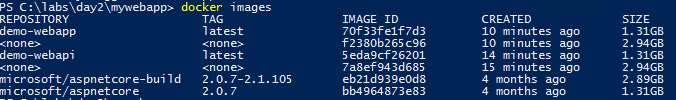
1. Run the following command. Notice that this time when the FROM command is hit, Docker will recognize that the image is already cached on the VM and this step is very quick. This time, it should only take about 1 minute.

docker build -t demo-webapp .

1. Check your docker images:

docker images

1. You should see 6 new images: The two <none> images which are images for the preliminary transitive images used to build the main image.



Exercise 2 has been completed

Exercise 3: Running, Stopping, and Starting your Container

Tasks

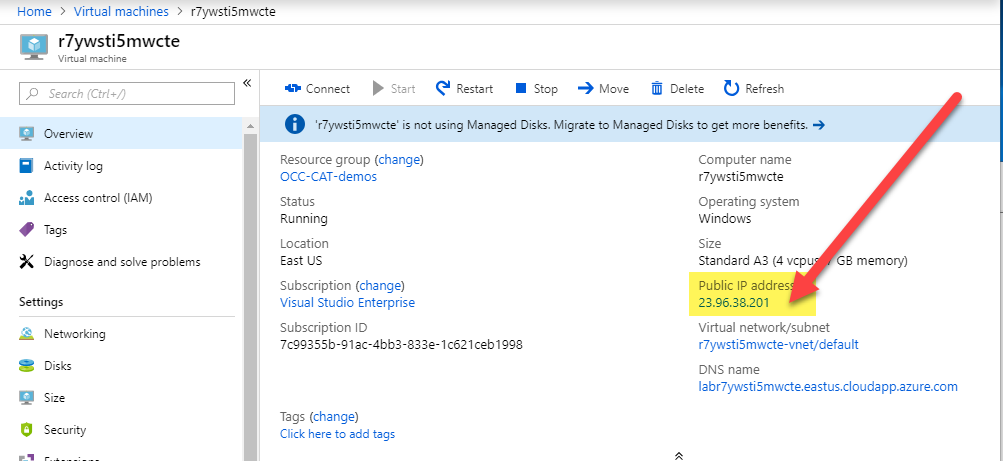
1. Run your Container
2. Run the following:

docker run -d -p 8080:80 demo-webapp

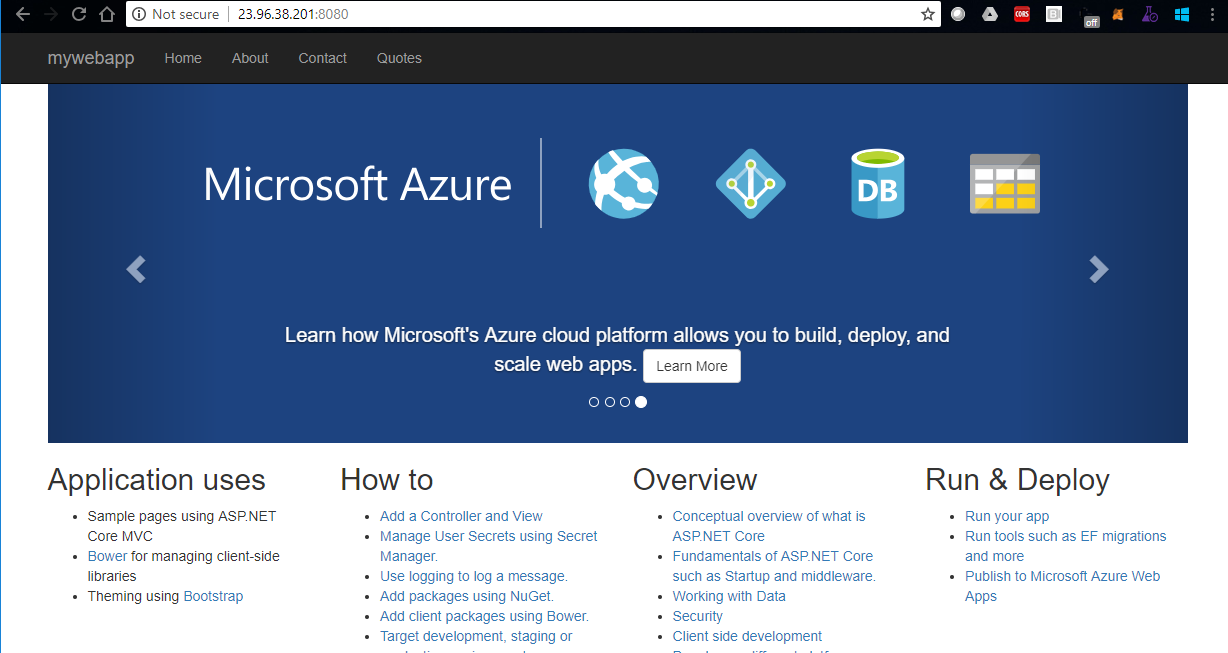
1. See the containers running, you should see one.

docker ps

1. Get the IP address of your VM from the Azure Portal. In Azure click VMs, click on your VM, then on the overview page, copy the IP address.



1. Navigate to <http://IPAddress:8080> (in this ex: <http://23.96.38.201:8080/> )

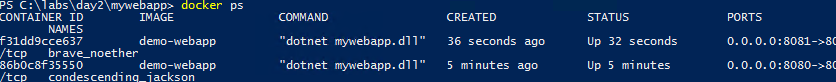


1. Run another copy of your Container on a new port
2. Run the following:

docker run -d -p 8081:80 demo-webapp

1. See the containers running, you should see two.

docker ps



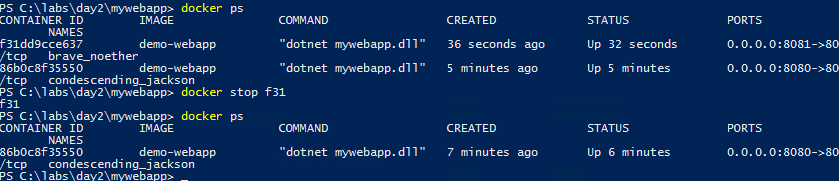
1. Navigate to http://IPAddress:8081
2. Stop one Container
3. Get the Container ID of one of the containers using the “docker ps” command, doesn’t matter which one. Run the following and use it in the command below:

(Note you only need the first 3-4 characters of the container ID, not the full hash)

docker stop <container ID>

1. You should only see one container running now

docker ps



1. Restart the Container
2. Run the following to see all containers, stopped or running.

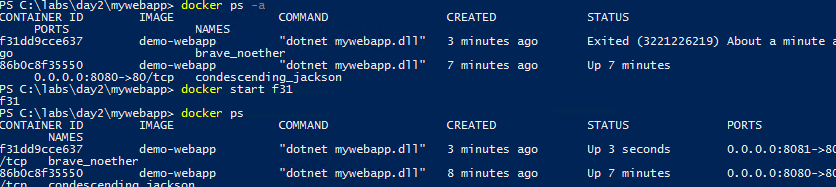
docker ps -a

1. Run the following, use the same ID of the stopped container from the command above.

docker start <container ID or name>

1. Check to make sure you have 2 containers running again:

docker ps



1. Stop all Containers
2. Run the following:

docker stop (docker ps -aq)

Hint! docker stop (docker ps -a -q) is equivalent, but notice you can smush together -a and -q into -aq.

1. Check all containers and see you have 2 exited containers:

docker ps -a

1. Remove a Container
2. Run the following

docker rm (docker ps -aq)

Hint! If you want to Stop AND remove all containers, you can chain commands:

docker stop (docker ps -aq) ; docker rm (docker ps -aq)

1. Check all containers and see you have 0 containers! But all your images are still there, just the running instances of containers are gone.

docker ps -a

1. Remove your Images
2. Let’s see what images we have

docker images

1. Let’s remove the transition images labeled <none> that we no longer need. Run the following and fill in just the first letters of the image ID for the <none> image. You will need to run this command twice with both image IDs as there are two <none> transition images, one for the web app and one for the api layer.

docker rmi <container ID>

Hint: If you image is associated with a container, you cannot delete the image unless you pass in “-f” for force. You would then need to run:

docker rmi -f <container ID>

1. Check your images and both <none> images should be gone.

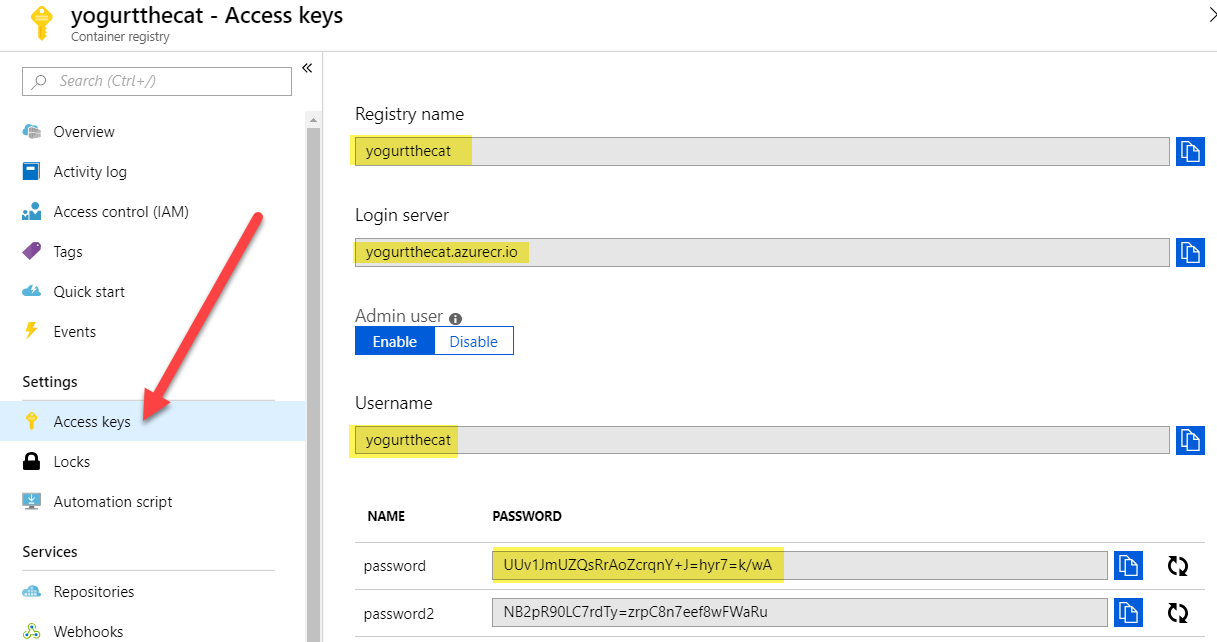
docker images

Exercise 3 has been completed

Exercise 4: Tagging your Dockerfile and Push it to the Azure Container Registry (ACR)

Tasks

1. Tag your Dockerfile with a new version + push to ACR
2. In the Azure Portal, click All Resources, and then find your Container Registry (hint: Sort by Type). Click on the name of your registry. Click Access Keys and gather the Login Server, Username, and Password fields into a Notepad.



1. In PowerShell, run this command (Note: this command **is case sensitive** so please make sure your registry name is exact):

docker login registry-name.azurecr.io -u username -p password

1. Now let’s tag our images. We must rename the image to push to ACR:

docker tag demo-webapi username.azurecr.io/demo-webapi

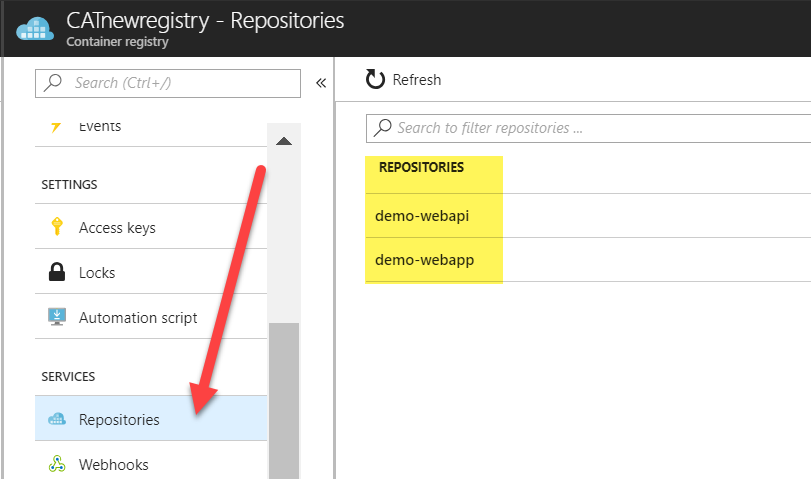
docker tag demo-webapp username.azurecr.io/demo-webapp

1. And now we push the images (if you get an authentication error, check your upper/lower case on the login and image name and make sure they match the Registry Name in Azure):

docker push username.azurecr.io/demo-webapi

docker push username.azurecr.io/demo-webapp

1. In the Azure Portal, click on the Repositories tab and see that both of your images are there:



1. Remove the image from your local VM
2. Remove the username.azurecr.io/demo-webapp and username.azurecr.io/demo-webpi images from your local machine

docker rmi -f <image ID or name>

1. Check that they are no longer in your image list:

docker images

1. Pull the image from your Azure Container Registry
2. Run the following:

docker pull username.azurecr.io/demo-webapp

docker pull username.azurecr.io/demo-webapi

1. Verify the images are back on your local machine. Notice it did not take long to download because you have the image layers cached already in the “docker-webapp” and “docker-webapi” images already on your machine.

docker images

Exercise 4 has been completed

Exercise 5: End to End

This exercise will test what you just learned, you can reference back to other parts of the guide as needed to complete these steps.

Tasks

1. Make a change to your code base to a visible part of your front end that you can see-- Change a “.cshtml” file under mywebapp > Views > Home.
2. Build a new image for the mywebapp project using the Dockerfile and the “docker build” command. Make sure you tag the image with a new version in the build command, for example, name it demo-webapp:v2.

docker build -t initialsnewregistry.azurecr.io/demo-webapp:v2 .

docker build -t initialsnewregistry.azurecr.io/demo-webapi:v2 .

1. Run your new Container based on the v2 images.
2. Verify you can see your changes are there in the browser and that the new image tagged v2 contains the new version of the code.
3. Push the new version of your image to ACR.
4. Check in ACR > Repos and verify that your new v2 tagged image is inside your repo.

Exercise 5 has been completed