Table of Contents

[Demo Guide: Module 2 – Getting Started with Windows Containers 2](#_Toc8394641)

[Demo 1: Working with Nano Server & Server Core for Container Images 2](#_Toc8394642)

[Demo 2: Building and Running IIS Server Container using Dockerfile 4](#_Toc8394643)

[Demo 3: Building and Running ASP.NET 4.7 Application in a Container Image 7](#_Toc8394644)

[Demo 4: Package ASP.NET Core Web Application as Container 11](#_Toc8394645)

[Demo 5 (optional. Need to install Docker Desktop): Visual Studio 2019 and Docker 13](#_Toc8394646)

Demo Guide: Module 2 – Getting Started with Windows Containers

For all the demos involving Windows Containers, we recommend building all required images (even custom) before showing the demo to have everything cached. Or at least, make sure you have downloaded all required base images beforehand.

Demo 1a: Working with Nano Server & Server Core for Container Images

In this demo, you will provide **walkthrough of Windows Nano and Server container** images and **running containers based on these images**.

Tasks

1. Run a Nano Server
2. Login to the Windows VM on LOD (or ideally, using your own VM). You will need to run the commands in this **section using the PowerShell console as an administrator**. Right click the PowerShell icon on the taskbar and select “Run as Administrator”.

A screenshot of a computer screen

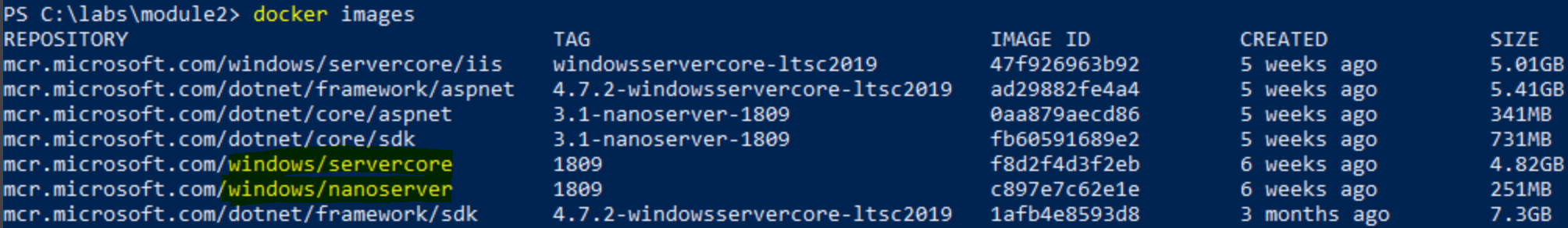
Description automatically generated

1. The PowerShell console is now available to you. Make sure you are inside the windows containers labs directory.
2. You can do that by running the command “**cd C:\labs\module2\**”. This will put you inside the windows containers lab folder where all the necessary files are located.

A screenshot of a cell phone

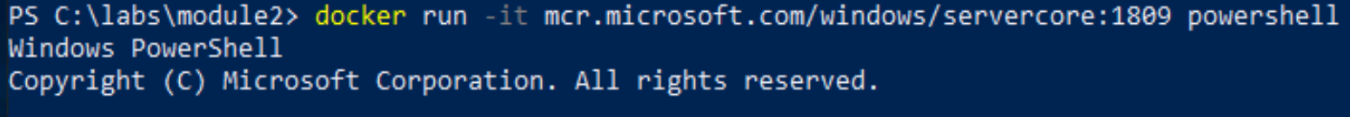
Description automatically generated

1. First, let’s get the list of all the container images available on this Docker host by running the command “**docker images**”.
2. Notice that you already have “windows/servercore” and “windows/nanoserver” images available to you representing “Server Core” and “Nano Server” images.



*NOTE: It’s important to understand that you can always download specific version of “windows/servercore” and “windows/nanoserver” images by using an appropriate tag. It is also worth calling out that the images are pulled from Microsoft Container Registry although they are still discoverable from Docker Hub.*

1. You will now run a container based on “Server Core” image (windows/servercore). Before you do that run the command “**hostname**”. This will reveal the hostname of your virtual machine. Host nan
2. Run the command “**docker run -it mcr.microsoft.com/windows/servercore:1809 powershell**”. Please be patient as it will take a minute or so for this command to work. The **-it** switch provides you with an **interactive session**. The **powershell** is a parameter passed as an argument which basically gives you access to Powershell (command line) running inside the container. Technically, the **-it** switch puts you inside a running container.



1. Run the command “**hostname**”. This time you are running it inside the running container. **Notice that the host name is different from the hostname you get in step 4.** The host name you see inside the container is host name of the container as is based on container id. You may want to run other commands as you wish.
2. Finally, exit the interactive session by typing “**exit**” and pressing Enter. This will take you back to the PowerShell console on the host.
3. Now let’s run another container based on “Nano Server” image (windows/nanoserver). To do that run the command “**docker run -it mcr.microsoft.com/windows/nanoserver:1809 CMD**” (we use CMD because Powershell is not available in nanoserver).
4. Run the command “**hostname**”. Notice that the host name is different from host name you get in step 4. The host name you see inside the container is the host name of the container which is based on container id. You can run other commands as you wish.
5. Finally, exit the interactive session by typing “**exit**” and pressing Enter. This will take you back to the PowerShell console on the host. In this task, you have created and run containers based on Windows Server Core & Nano Server container images that Microsoft provides.

Demo 1b: Windows vs Hyper-V Isolation

In this demo you will **compare container process isolation between Windows Server and Hyper-V Containers**. You will need to run this demo on Windows Server 2019 since Windows 10 only supports Hyper-V Containers. **Recommend using the LOD VM**

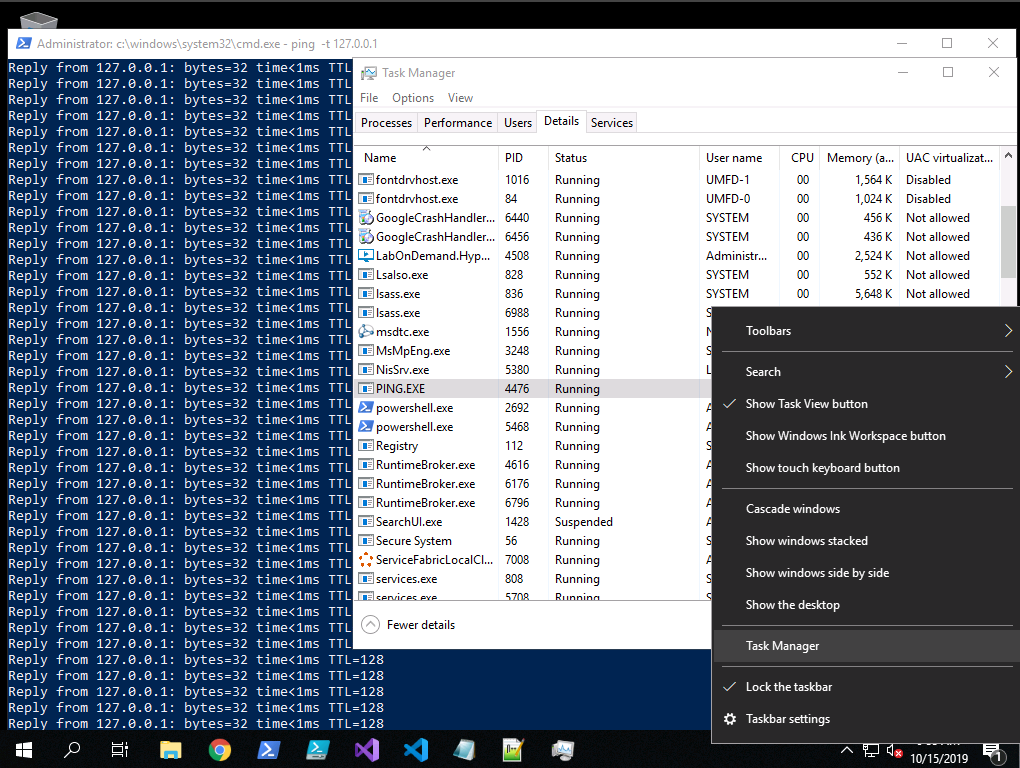
TASK Process Isolation

1. Run the command “**docker run -it mcr.microsoft.com/windows/nanoserver:1809**”
2. The default isolation level for Windows Server is “process”. Start another PowerShell prompt and inspect the running containers
   1. “docker inspect <container id>”

Scroll through JSON until you see the “Isolation Level” Verify is says “process”



1. Inside the container run “ping -t 127.0.0.1”
2. **From the host** start the task manager and show the ***ping*** command displayed on the hosts task manager



Processes running in a Windows Server Container are viewable on the host. **Container processes have access only to other processes running inside the same container and cannot access other container processes or host processes**.

1. From the task manager right-click on “ping.exe” and choose “End Task”. Verify the ping command running in the container stops
2. Exit & stop the container

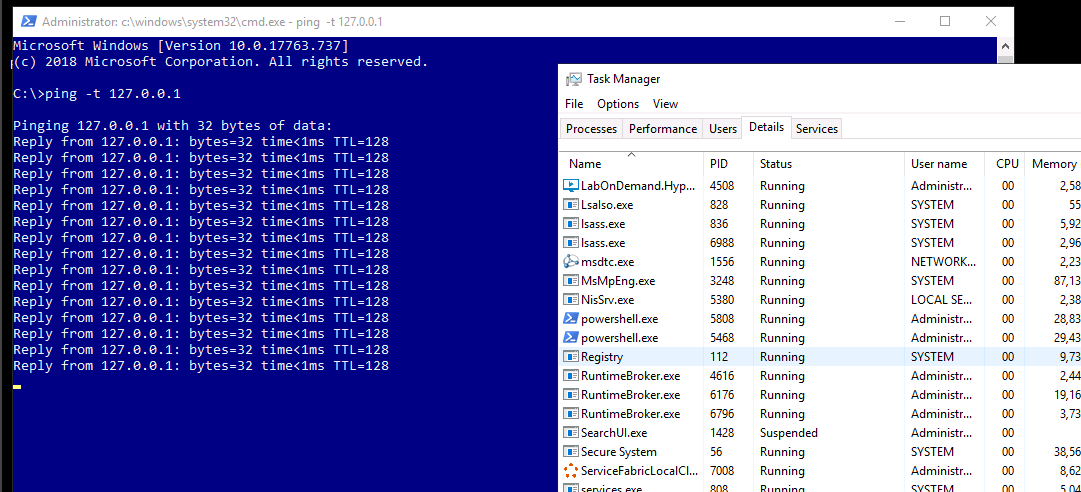
Task – Hyper-V isolation

1. Using PowerShell, start an container with an interactive prompt using HyperV isolation “**docker run -it -–isolation=hyperv mcr.microsoft.com/windows/nanoserver:1809”**
2. The isolation level for this container has been changed to “Hyper-V”. Start another powershell prompt and inspect the running containers
   1. “docker inspect <container id>”

Scroll through JSON until you see the “Isolation Level” Verify is says “hyperv”



1. Inside the container run “ping -t 127.0.0.1”
2. **From the host** start the task manager and show that the ***ping*** command is NOT displayed on the hosts task manager



**Processes running in a Hyper-V Container are NOT viewable on the host**. Much like an application running in a Virtual Machine.

1. Close and stop container

Task: (Optional) LCOW

In this **demo you will show running Linux Containers on a Windows Host (LCOW)**. This is currently an ‘experimental’ feature and while stable, multi-platform capabilities are missing from some docker commands and configurations.

PREP

To prepare the LODs you must enable experimental features and download the Linux kernel BEFORE performing the demo

In a powershell prompt perform the following steps

//Enable Experimental Features in Dockerd daemon.conf

$configfile = @"

{

"experimental": true

}

"@

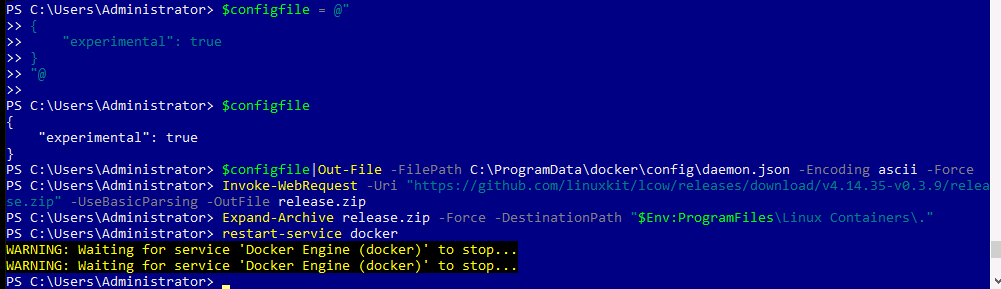
$configfile|Out-File -FilePath C:\ProgramData\docker\config\daemon.json -Encoding ascii –Force

//install LCOW Linux kernel

Invoke-WebRequest -Uri "https://github.com/linuxkit/lcow/releases/download/v4.14.35-v0.3.9/release.zip" -UseBasicParsing -OutFile release.zip

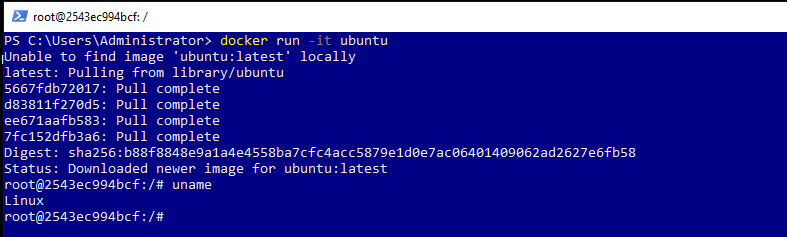
Expand-Archive release.zip -Force -DestinationPath "$Env:ProgramFiles\Linux Containers\."

restart-service docker



1. From a powershell prompt run the following command to start a linux container on a windows host

docker run -it ubuntu



Verify you are in a linux container by issuing the command ‘**uname**’

Linux Hyper-V containers can run side-by-side with Windows Containers (hyperv or process isolation) and can communicate with each other.

Demo 2: Building and Running IIS Server Container using Dockerfile

In the demo, you will show how to install IIS Web Server (Web Server Role) on a Windows Server Container and build a Container Image. Later, you will run a container with IIS running inside it. IIS Server is a popular Web Server released by Microsoft. Considering the strong footprint of IIS within the enterprises Microsoft supports IIS on Windows Container. IIS Server can be installed on windows server core.

**Tasks**

1. Build and Run IIS Server Image
2. Make sure you have a PowerShell Console open as an administrator (if you have followed previous task you should already be running a Console). Also, change the current directory to “**iis**” by running the command “cd” to get to “labs\module2\iis\”
3. The iis folder contains the Dockerfile with instructions to install IIS Server (Web Server Role) on the Windows Server Core base image. Open the Docker file by running the command “**cat Dockerfile**” and press enter.
4. Describe the structure of this file.

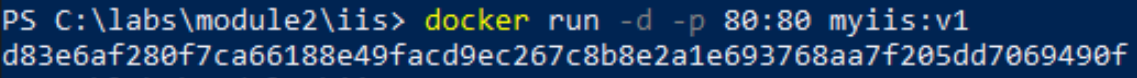
* The **FROM** instruction points to the **mcr.microsoft.com/windows/servercore** to be used as a base image for the new container image.
* The **RUN** instruction executes PowerShell to install Windows Feature "Web Server" (IIS Server).
* The next command is the **ADD** instruction which copies the **ServiceMonitor.exe** utility to the container image. The **ServiceMonitor.exe** is a utility that monitors **w3svc** service inside the container, if the service fails, the exe fails, so Docker knows the container is unhealthy. The **ServiceMonitor.exe** is developed and released by Microsoft: <https://github.com/microsoft/iis-docker/tree/master/windowsservercore-ltsc2019>
* The **EXPOSE** instruction does not actually publish the port. It functions as a type of documentation between the person who builds the image and the person who runs the container, about which ports are intended to be published.
* Finally, the **ENTRYPOINT** instruction makes sure that monitoring of **w3svc** begins immediately as soon as container starts running. This is what will keep the container in running state.

1. To build the new image with IIS installed on it run the command “**docker build -t myiis:v1 .**”. This command builds a new container image with name “**myiis**” and tag “**v1**”. That tag conveniently tells everyone about the information of version of the image. Please note that the STEP 3/6 of the build process performs the installation of Web-Server (IIS Server) and may take few minutes. Eventually you should see the results like following.

A screenshot of a cell phone

Description automatically generated

1. Now you can a run a new container based on “**myiis:v1”** image by using command (please type this one in instead of copy and pasting it, tends to have issues with copy pasting)  
   “**docker run -d –p 80:80 myiis:v1**”.



1. The container ID is shown after the run command (“154” in the above screenshot), or by using “docker ps”. To get the IP address of the container, run the following command:

docker inspect <container id> | FINDSTR "IPAddress"

A close up of a sign

Description automatically generated

1. Open any web browser of your choice and browse to the IP address from the previous step.

A screenshot of a cell phone

Description automatically generated

*Note: In the Windows world, the loopback address might not work depending on the kernel version (due to of a limitation in the default NAT network stack), only the external request routing would work.*

*In the case of the Lab On Demand VM pre-provisioned for the workshop, we can also open the browser and go to http://localhost to see the site. If your demo VM allows, feel free to demonstrate both approaches.*

Demo 3: Building and Running ASP.NET 4.7 Application in a Container Image

* In this demo, you will provide walkthrough of how to package an existing **ASP.NET 4.7 web**
* **application** into a container.
* It's important to understand that Microsoft supports both the latest .NET frameworks like .NET Core, ASP.NET Core etc. and more legacy .NET Frameworks like .NET 3.5, .NET 4.5 and ASP.NET 4.5 on Windows Containers.
* Most customers today have heavy investments on some of the legacy Microsoft technologies, and Microsoft wants to make sure that they can still move towards application containerization, not only for new applications but also for legacy applications.

*NOTE: You can find more comprehensive list of application frameworks*

*supported by Microsoft on Windows Containers at:*

[*https://docs.microsoft.com/en-us/virtualization/windowscontainers/samples#Application-Frameworks*](https://docs.microsoft.com/en-us/virtualization/windowscontainers/samples#Application-Frameworks)

Tasks

1. Build and Run ASP.NET 4.7 MVC Application
2. Make sure you have a PowerShell Console open as an administrator (if you have followed previous task you should already be running a Console). Also, change the current directory to “aspnet4.7” by running the command “cd” to get to “\labs\module2\aspnet4.7\”.

A screenshot of a cell phone

Description automatically generated

1. Before proceeding further let’s stop and remove all the running containers from previous task. Run the command “**docker rm -f (docker ps -aq)**”

A close up of a logo

Description automatically generated

1. Let’s examine the Dockerfile. Open it in Notepad by running the command “**cat Dockerfile**”. See the text below for an explanation of the commands in the DockerFile image.

A screenshot of a cell phone

Description automatically generated

The first noticeable statements are the two **FROM** which are used in what is called, a **multi-staged build** process.

It allows us to **create two Docker images** with a single **docker build** command. This is a good time to introduce multi-staged build.

Explain each part of the Dockerfile: [https://docs.docker.com/develop/develop-images/**multistage-build**/](https://docs.docker.com/develop/develop-images/multistage-build/)).

For more information about multi-staged build: <https://www.youtube.com/watch?v=gdoXtFpXvik>

* The first image is built on top of the **.NET 4.7 SDK base image** containing all utilities necessary to build your application: **mcr.microsoft.com/dotnet/framework/sdk:4.7.2-windowsservercore-ltsc2019**.
* This resulting image will be much bigger than what is required to simply run your application. This build stage will produce all application artifacts that will be picked up when building the second image. They will be copied in the folder **/app/WebAppLegacy** of the first image. Also note that this first image is identified as **build**
* The second image is built on top the .NET 4.7 runtime base image and only contains what is needed to run the application: **cr.microsoft.com/dotnet/framework/aspnet:4.7.2-windowsservercore-ltsc2019**.
* It uses the output from the first image to build its own runtime image **COPY --from=build /app/WebAppLegacy/. ./**.
* Having an image as small as possible is beneficial to reduce the attack surface (less tools equals less opportunities to exploit in an attack) and they will be much faster to download at deployment time.

1. Build a new image with web application packaged inside it by running the command “**docker build –t aspnetapp:v4.7 .”** The build process to create a new container image will take few minutes. You will see the progress of all the steps as they are happening, so you will know the overall progress.

*Notice the tag* ***\*\*v4.7\*\**** *that indicates the version number of ASP.NET framework. This use of tag is optional but recommended.*

*Note: At the end of the build process, feel free to look at the produced images with `***docker images***`. You will see the two images that have been built. the SDK image (that is not named) and the image that is actually going to be hosting our application:* ***\*\*aspnetapp:v4.7\*\*****. Note that we can automatically remove the SDK image once the runtime image is built. For that, use the* ***\*\*--rm\*\**** *parameter in the `docker build` command*

1. To run a container with the ASP.NET 4.5 web application based on new container image run the command: “**docker run -d -p 80:80 aspnetapp:v4.7**”

A picture containing indoor, clock

Description automatically generated

1. Run the following to get the IP address (remember the container ID comes from the previous run step):

**docker inspect <container id> | FINDSTR "IPAddress"**

A close up of a screen

Description automatically generated

1. Open web browser of your choice and browse to the IP address from the previous step. This will take you to the Home page of the ASP.NET 4.7 Web Application. It may take few seconds for the home page to load for the first time since IIS Application Pool takes a hit for serving the first page. Subsequent navigation to other pages will be much faster.

A screenshot of a cell phone

Description automatically generated

Demo 4: Package ASP.NET Core Web Application as Container

In the previous demo you built container images using some of the more mature technologies and products released by Microsoft.

In this task you will build container that will **run ASP.NET Core Web Application**.

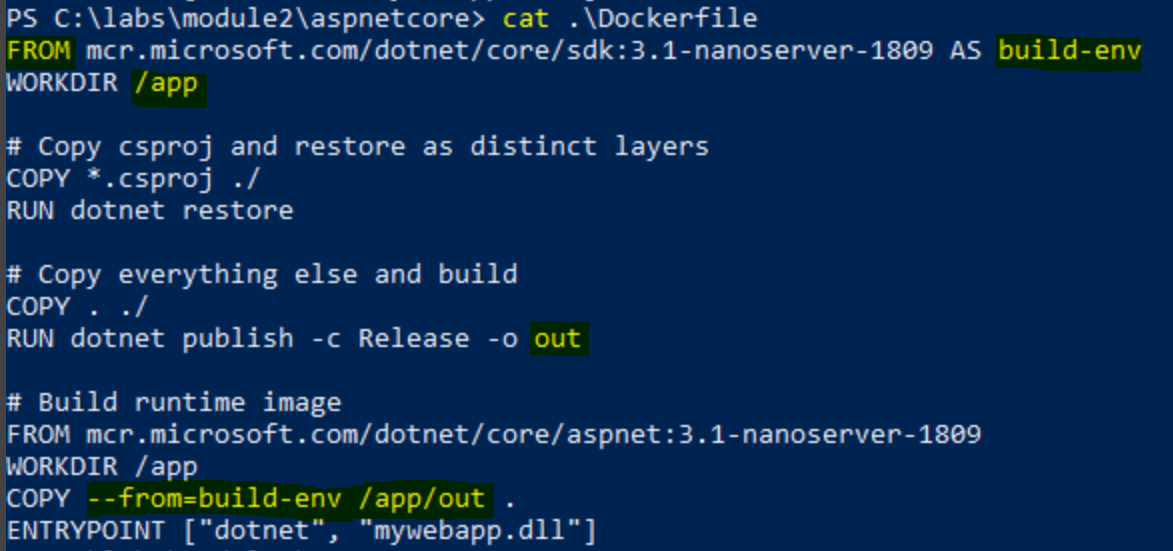
ASP.NET Core is a significant step forward for Microsoft to allow ASP.NET to run cross platform including MacOS, Linux and Windows. ASP.NET sits on top of .NET Core so this cross-platform support will of course include .NET Framework.

*NOTE*: To understand when to use .NET Core and when to use .NET Framework please read article: <https://docs.microsoft.com/en-us/dotnet/articles/standard/choosing-core-framework-server>

In this task, you will package a simple ASP.NET Core MVC application into a container image using Dockerfile. Finally, you will run container hosting the ASP.NET Core application using the `docker run` command.

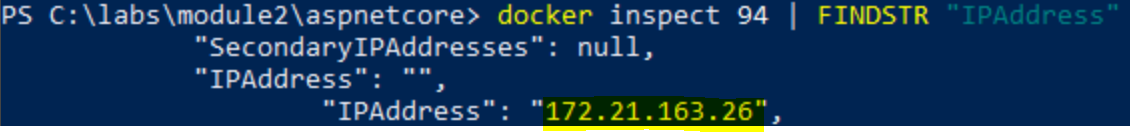
Tasks

1. Build an ASP.NET Core Application
2. Change to the relevant directory by using “cd” to go to the path: module2\aspnetcore.
3. You are provided with a Dockerfile. View the content of Dockerfile by running a command “**cat .\Dockerfile**”. The Dockerfile should look like below Unlike in module 1, the dotnet build/publish is within the Dockerfile itself:

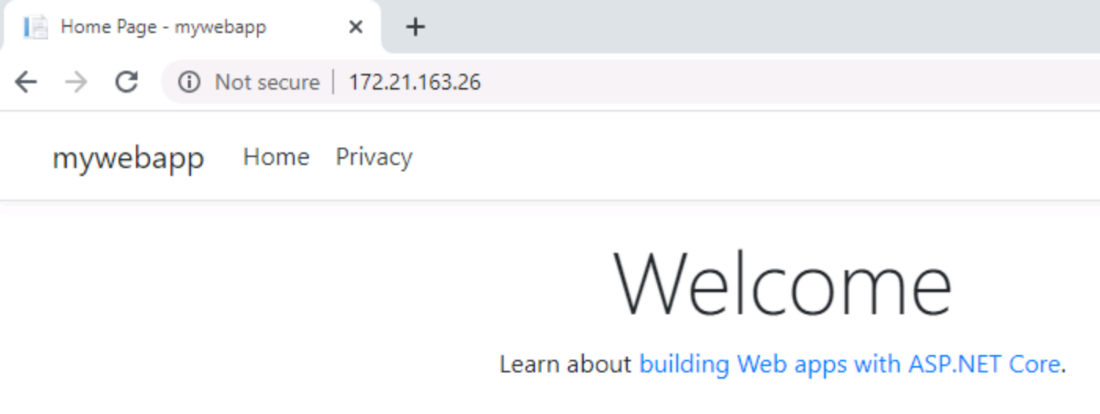


1. You are all set to build ASP.NET Core web application container image. Run the command   
   “**docker build -t aspnetcoreapp:3.1 .”**
2. Finally, to run the container by executing command “**docker run -d -p 8080:80 aspnetcoreapp:3.1**”
3. You are now running ASP.NET Core application inside the container listening at port 80 which is mapped to port 8080 on the host. Get the IP Address of the container you just ran:

“**docker inspect <container id> | FINDSTR "IPAddress**"



1. Navigate to the IP address from the previous step:



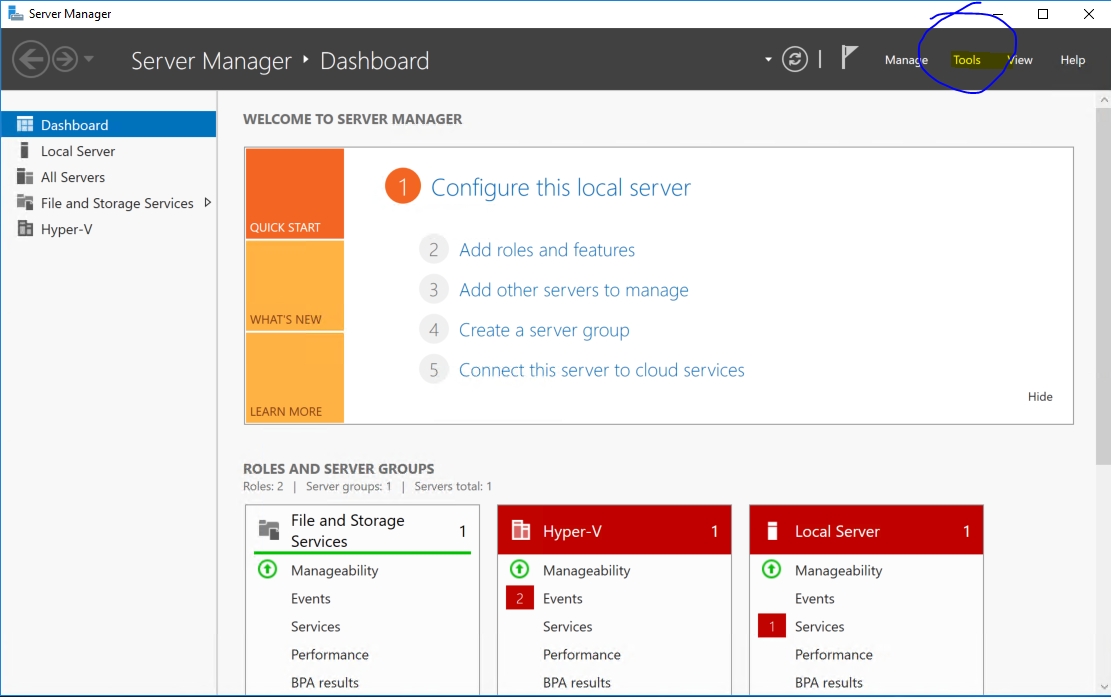
Demo 5 (optional. Need to install Docker Desktop): Visual Studio 2019 and Docker

In this demo, you will show:

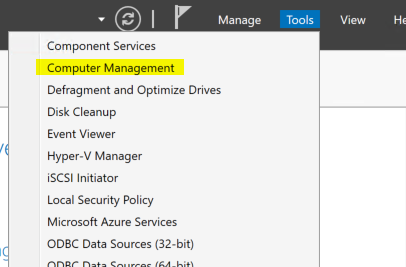
* The Visual Studio 2019 Docker Support. Running a Container in Visual Studio.

Tasks

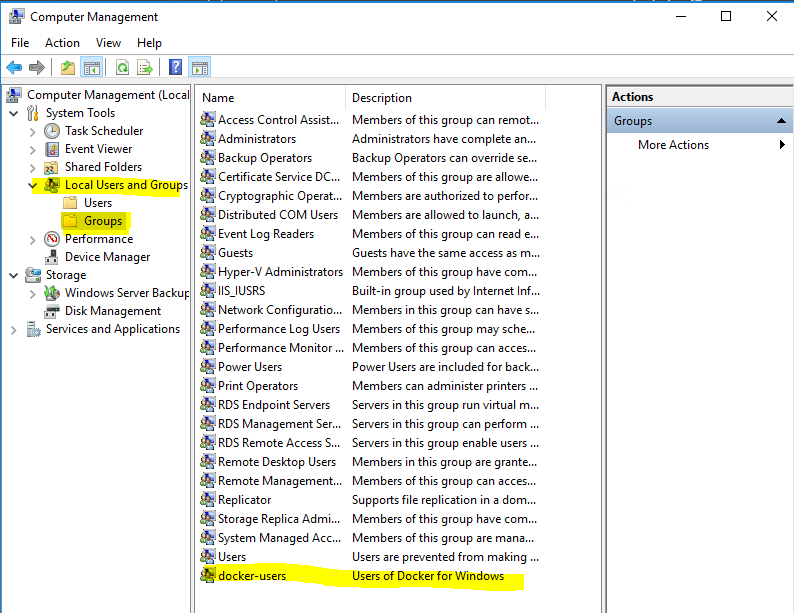
1. Visual Studio Docker Support
2. Warning – Do this before you do your workshop and test it out on your VM. DO NOT do step 2 during a workshop demo, do it ahead of time.
3. Try to go to the Start menu and type in Docker. If you click on it and try to run Docker.. if you get an error that says “**Docker Desktop – Access Denied** You are not allowed to use Docker. You must be in the “docker-users” group” then please continue to step to fix this. If Docker starts with no issue then please skip step 2, and continue on to step 3.
4. This is how you add your user to the “docker-users” group (this is a workaround until this is fixed and will make it so your Docker Desktop will run properly).
   1. Go into Server Manager.



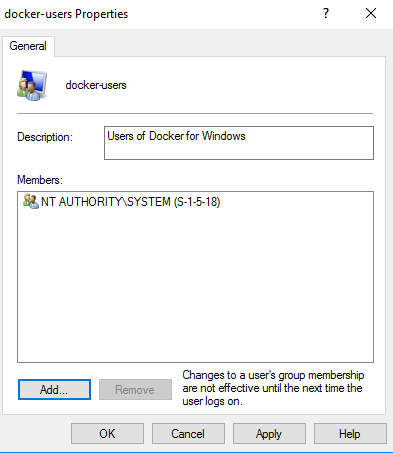
* 1. On the top right (see highlighted with blue circle in screenshot below) click Tools > Computer Management.



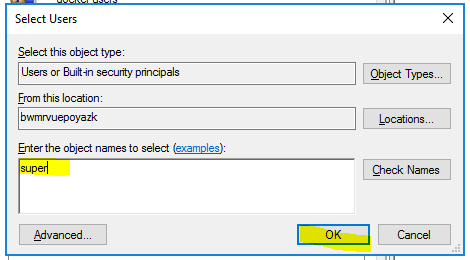
* 1. On the left menu, hit Local Users and Groups. Click on Groups. Double click on **docker-users.**



* 1. A modal will pop-up. Click on the add button (this is outlined in blue in the screenshot below).

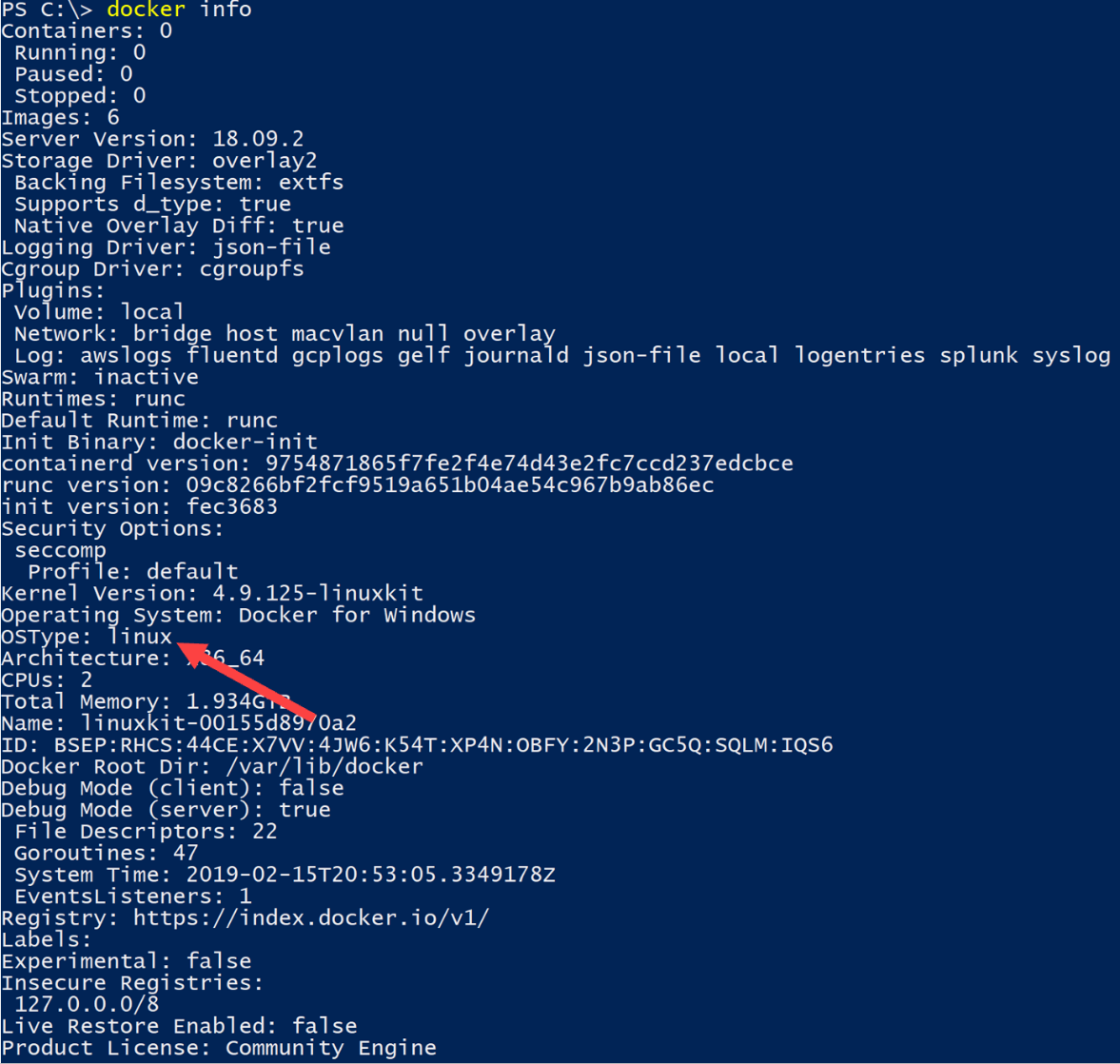


* 1. Another modal will pop-up. Add your username that you used to login to the VM in the last textbox that says: “Enter the object names to select”, then hit OK. In this example, the VM username used was “super”.

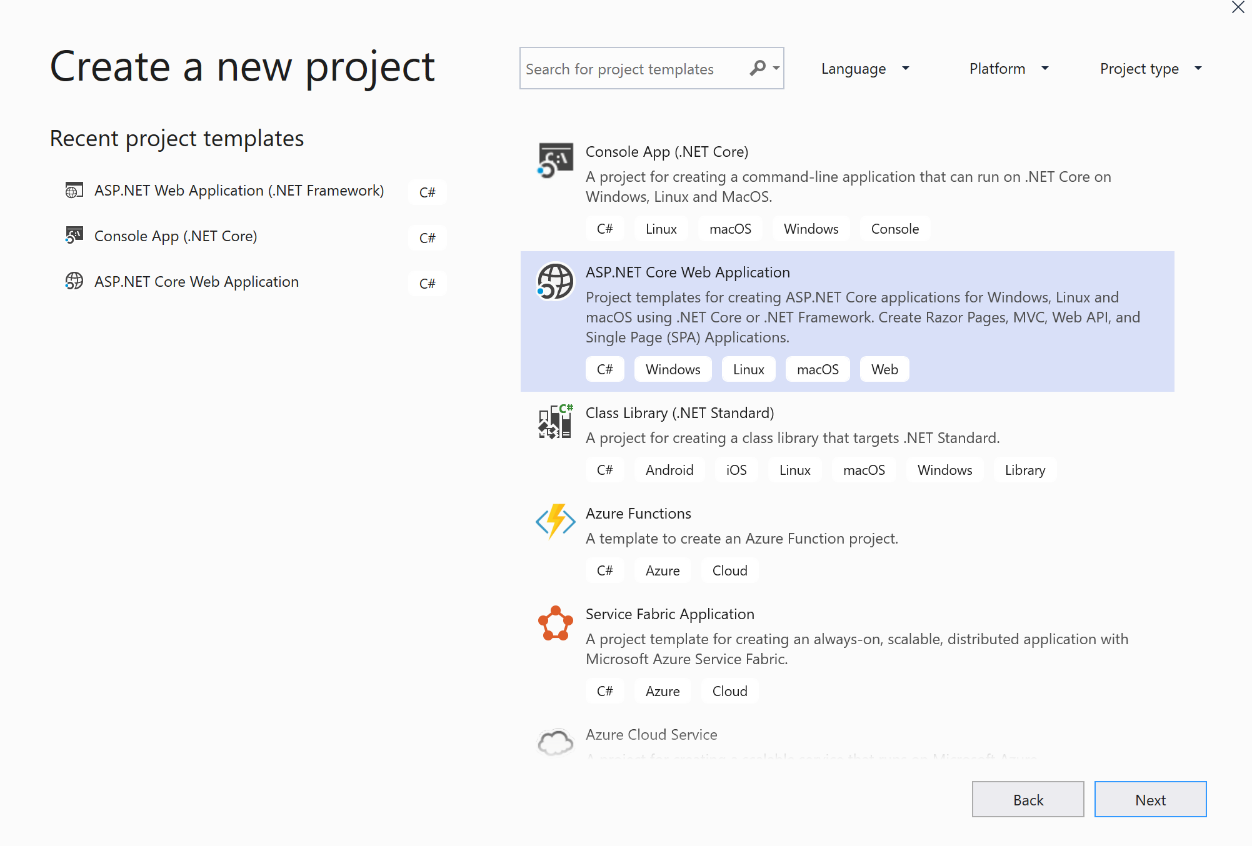


* 1. Click OK and close these Windows

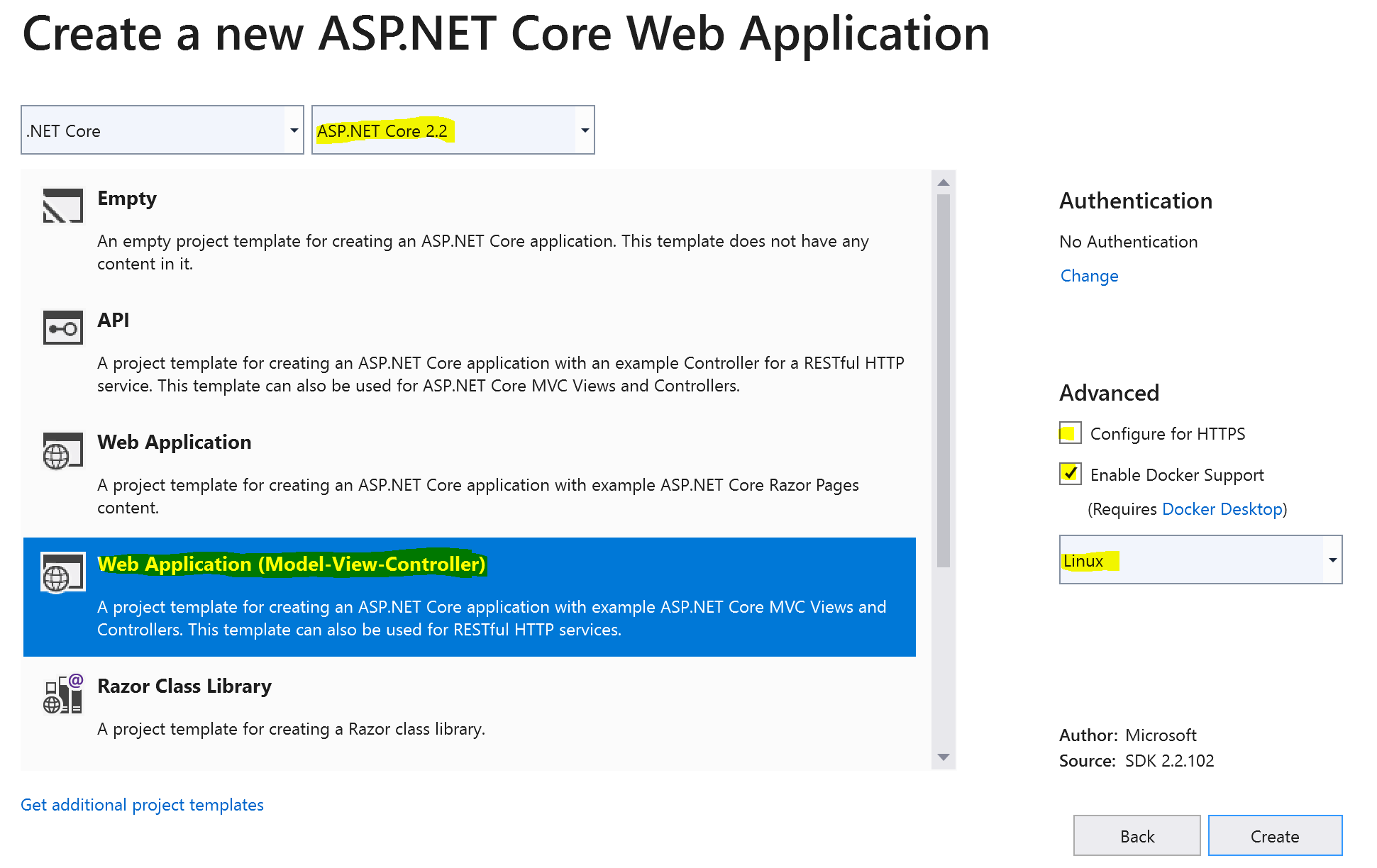
1. Make sure you have Docker Desktop on your machine and that it is set to **Linux mode**. You can right click on the Docker whale on the task bar and if it says “Switch to Windows mode” it means you are in Linux mode now. If it says “Switch to Linux mode” then click on that and wait for it to swap (the Docker icon will be moving while it is switching, and it will stabilize and say Docker is ready once it is done). Another option is to run the command **docker info** and check that **OSType = Linux**



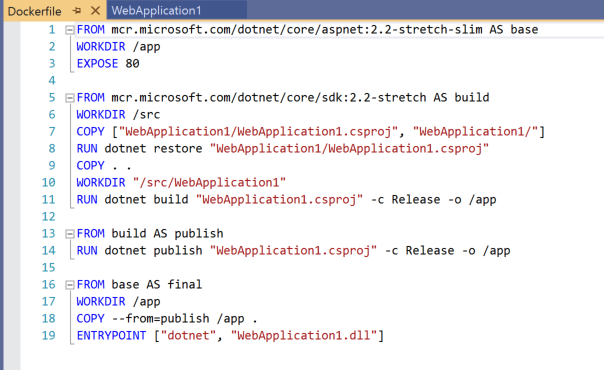
1. Open an instance of Visual Studio 2019.
2. Create a new ASP.NET Web Core Application.



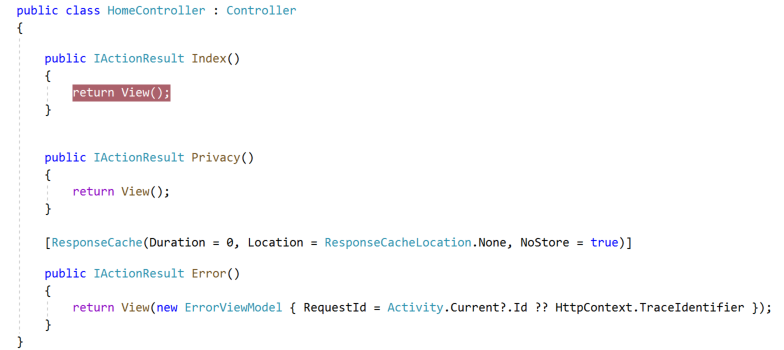
1. Choose MVC, check the box for Enable Docker Support, and Choose Linux for the Operating System (OS). Uncheck **Configure HTTPS**. The following screenshots show an example for ASP.NET Core 3.1 but feel free to try more recent versions in your demo.



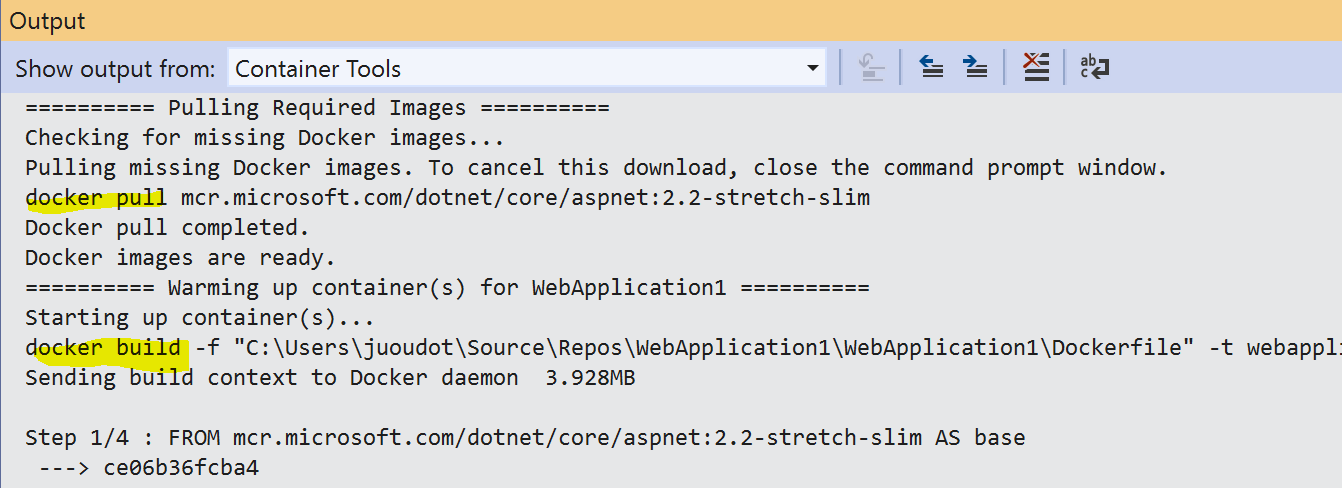
1. Hit **Create**.
2. Show the **Dockerfile** that was added to the Solution.

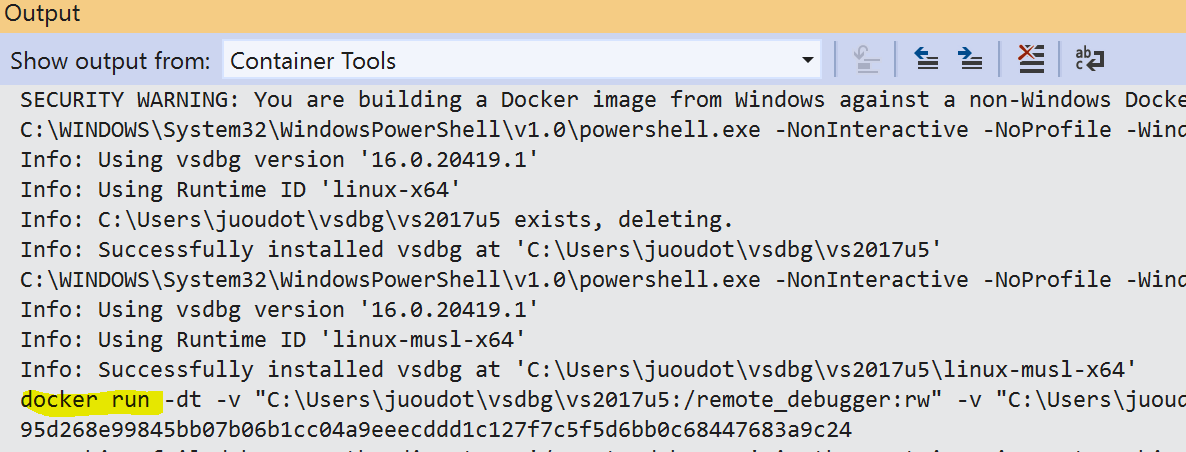


1. Open the **HomeController** and add a breakpoint in the **Index** method so that you can show the debugging containers from Visual Studio.

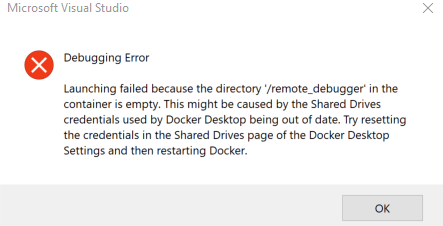


1. Hit Run on Visual Studio (note that it says **Docker** but if you click on the drop down, you can run this application in IIS express or in self-hosted mode with Kestrel). Show the Output window as it runs. Try to highlight and explain the **docker build** and **docker run** commands.

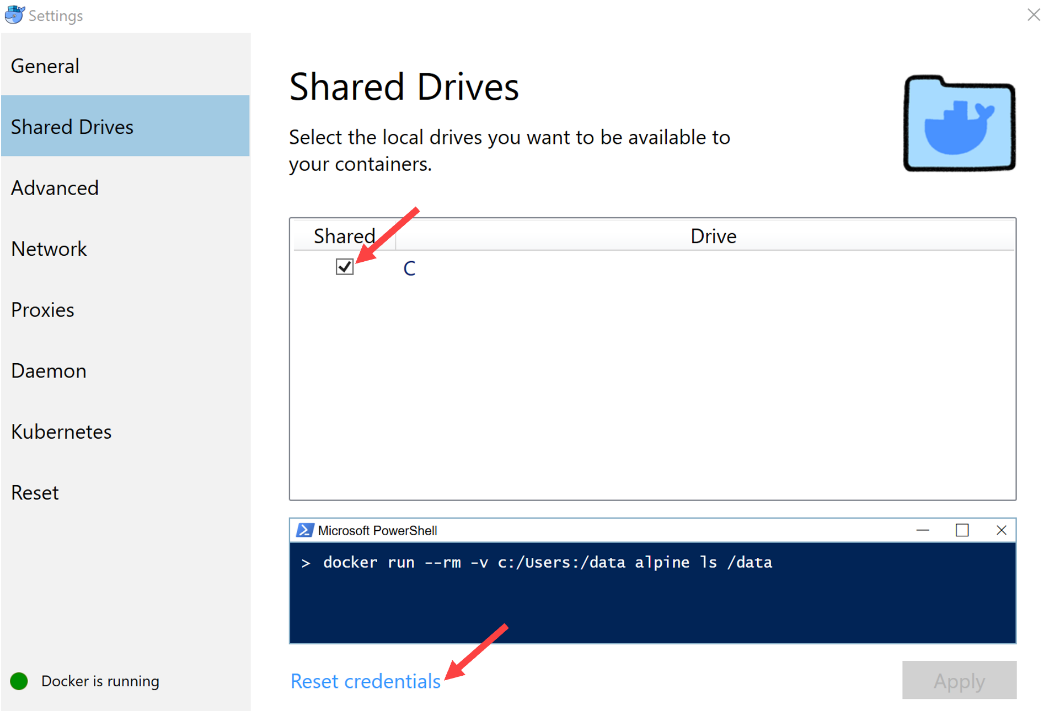




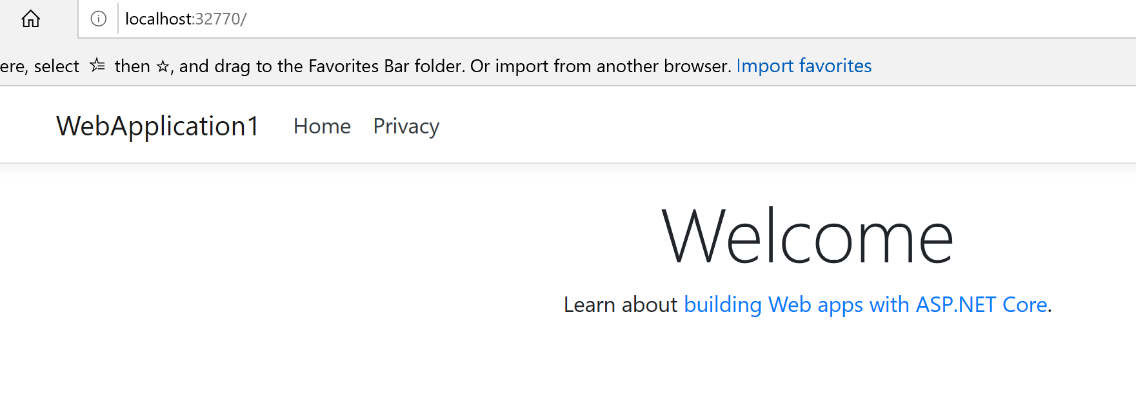
1. If it works, you should see your browser opening the ASP .NET Core default page and the breakpoint should be reached. Otherwise you might see the below error



1. Open the Settings page of Docker Desktop and open **Shared Drives** - **Shared** and click on **Reset credentials**.Once it is done, make sure that Shared is still checked, go back into Visual Studio and try to run again.



1. Call out that Visual Studio now allows to debug just like any regular application. Then hit F5 and show the Core application in your browser.



1. Open a Powershell window and show that behind the scene, you have a running container, as expected.

***docker ps***



1. You can also show the output of ***docker images*** that created the web application image and automatically removed the previous stages (it used the --rm parameter in the ***docker build*** command).

