Modules

Module 1 - Introduction to Containers

Duration: 105 minutes

Module 1: Table of Contents

Prerequisites

The Linux LOD VM is packaged with all of the required software to perform this lab.

The lab files are required to complete the hands-on exercises and have been pre-downloaded on the Virtual Machines.

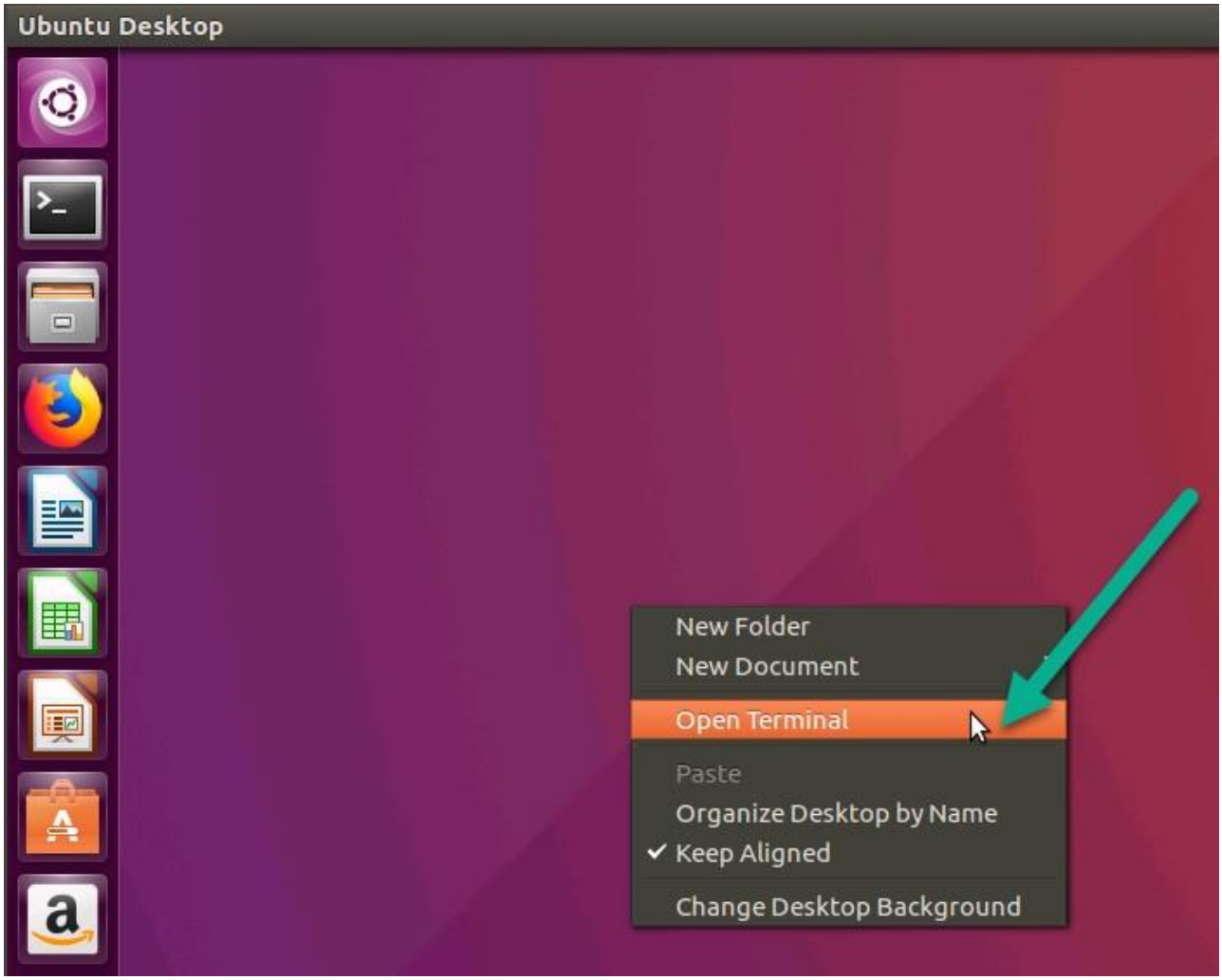
Exercise 1: Running Your First Container

In this exercise, you will launch a fully functional WordPress blog engine using a Linux-based Docker container. You will learn the commands needed to pull the container image and then launch the container using the Docker CLI. Finally, you will observe that by running the container, you don't need to install any of the WordPress dependencies onto your machine; the WordPress engine binaries and all dependencies will be packaged inside of the container.

Running WordPress Blog Engine Container

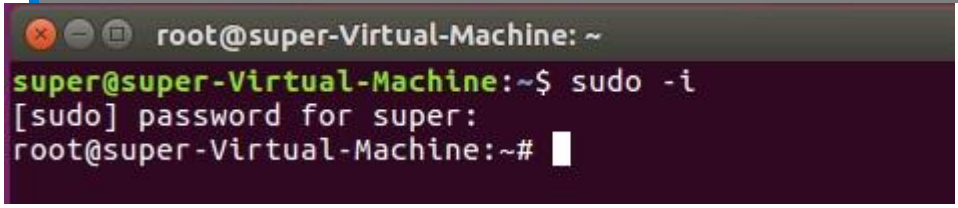
Return to list of exercises - Return to list of modules

1. To open a command line prompt, right click on the desktop and choose **open terminal**.



2. Run "**sudo -i**" to ensure all commands have elevated privileges. You will be prompted for the VM password, type it in, then press enter.

> Note: You should see root@super in your command line as the user.



3. Before we start, notice there is a **Lab 1 Commands Sheet.txt** file that can be used to copy and paste the commands for the lab. Display its content with `cat ~/labs/module1/Lab\ 1\ Commands\ Sheet.txt`

   You can keep this window on the side and open another elevated terminal to follow the below instructions.

```
root@super-Virtual-Machine:~# cat ~/labs/module1/Lab\ 1\ Commands\ Sheet.txt
-----------------------------------
LAB 1 Introduction to Containers
-----------------------------------
sudo -i

Exercise 1:
docker pull tutum/wordpress
docker images
docker run -d -p 80:80 tutum/wordpress
docker ps
docker run -d -p 8080:80 tutum/wordpress
docker run -d -p 9090:80 tutum/wordpress
docker ps
docker run --name mycontainer1 -d -p 8081:80 tutum/wordpress
docker ps


Exercise 2:
docker ps
docker stop <<CONTAINER-ID>>
docker ps -a
docker start <<CONTAINER-ID>>
docker ps
```

4. Type "**docker pull tutum/wordpress**".

> Knowledge:This will tell Docker client to connect to public Docker Registry and download the latest version of the WordPress container image published by tutum (hence the format tutum/wordpress). The container image has been pre-downloaded for you on the VM to save you a few minutes, but you will see each layer that is cached show the text 'Already exists'. docker

```
root@super-Virtual-Machine:~# docker pull tutum/wordpress
Using default tag: latest
latest: Pulling from tutum/wordpress
8387d9ff0016: Already exists
3b52deaaf0ed: Already exists
4bd501fad6de: Already exists
a3ed95caeb02: Already exists
790f0e8363b9: Already exists
11f87572ad81: Already exists
341e06373981: Already exists
709079cecfb8: Already exists
55bf9bbb788a: Already exists
b41f3cfd3d47: Already exists
70789ae370c5: Already exists
43f2fd9a6779: Already exists
6a0b3a1558bd: Already exists
934438c9af31: Already exists
1cfba20318ab: Already exists
de7f3e54c21c: Already exists
596da16c3b16: Already exists
e94007c4319f: Already exists
3c013e645156: Already exists
6eaab85d8b00: Already exists
0e132cb1ce48: Already exists
1eba2a6e1fe2: Already exists
5a56e7332673: Already exists
d4166ef5fefb: Already exists
752b28beb2cc: Already exists
38aa0ae5e379: Already exists
Digest: sha256:2aa05fd3e8543b615fc07a628da066b48e6bf41cceeeb8f4b81e189de6eeda77
Status: Image is up to date for tutum/wordpress:latest
```

5. Run the command "**docker images**" and notice "tutum/wordpress" container image is now available locally for you to use.

```
root@super-Virtual-Machine:~# docker images
REPOSITORY                          TAG                 IMAGE ID            CREATED             SIZE
mcr.microsoft.com/dotnet/core/aspnet   3.1                 a843e0fbe833        2 weeks ago         207MB
nginx                               stable-alpine       aaad4724567b        2 months ago        21.2MB
node                                boron               ab290b853066        8 months ago        884MB
mcr.microsoft.com/mssql/server      2017-CU12-ubuntu    4095d6d460cd        14 months ago       1.32GB
tutum/wordpress                     latest              7e7f97a602ff        3 years ago         477MB
```

6. That's it! You can now run the entire WordPress in a container. To do that run the command `docker run -d -p 80:80 tutum/wordpress`

> Note: Pay close attention to the dash "-" symbol in front of "-p" and "-d" in the command.

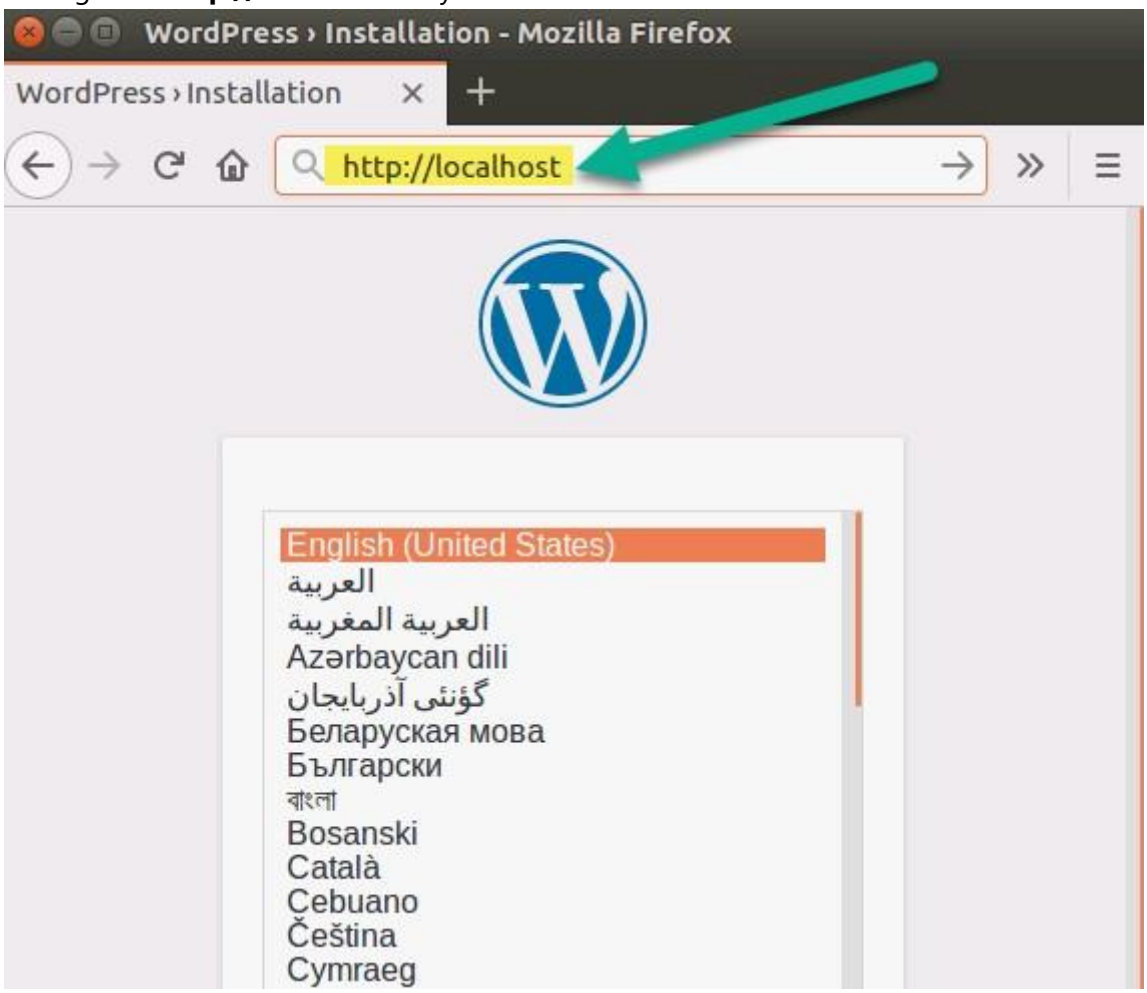
```
root@super-Virtual-Machine:~# docker run -d -p 80:80 tutum/wordpress
84acb95920718929d701f9a9c3ae87e0a154e56fda289a8bf0b90bae4e40f8b9
```

7. Run the following "**docker ps**" to see the running containers.

```
root@super-Virtual-Machine:~# docker ps
CONTAINER ID        IMAGE               COMMAND                  CREATED
STATUS              PORTS                              NAMES
84acb9592071        tutum/wordpress     "/run.sh"                3 minutes ago
Up 3 minutes        0.0.0.0:80->80/tcp, 3306/tcp   vibrant_joliot
```

8. Click on the **Firefox** icon on the left:

9. Navigate to **http://localhost** and you should see WordPress.



10. Let's launch two more containers based on "**tutum/wordpress**" image. Execute following commands (one line at a time)

```
docker run -d -p 8080:80 tutum/wordpress
```

```
docker run -d -p 9090:80 tutum/wordpress
```

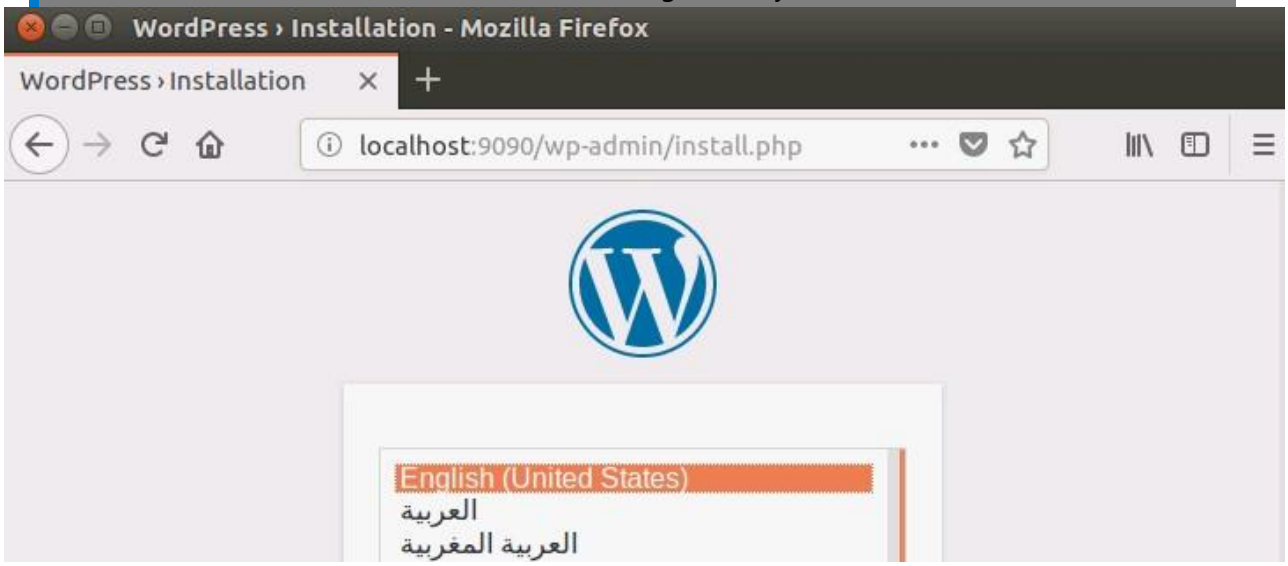
```
root@super-Virtual-Machine:~# docker run -d -p 8080:80 tutum/wordpress
b73631e40f498b4902202a2da2536feb0897ac9b5102d6c802b3d69dcaef42a8
root@super-Virtual-Machine:~# docker run -d -p 9090:80 tutum/wordpress
c272f02ac03d38d0f2b8a8935d1b85c892d2ca91fcfe6945aade707e2f5b6165
```

11. Run "**docker ps**" to see all 3 running containers and their port numbers:

```
root@super-Virtual-Machine:~# docker ps
CONTAINER ID        IMAGE               COMMAND                  CREATED
STATUS              PORTS                                        NAMES
c272f02ac03d        tutum/wordpress     "/run.sh"                2 minutes ago
Up 2 minutes        3306/tcp, 0.0.0.0:9090->80/tcp               priceless_feynman
b73631e40f49        tutum/wordpress     "/run.sh"                2 minutes ago
Up 2 minutes        3306/tcp, 0.0.0.0:8080->80/tcp               dazzling_engelbart
84acb9592071        tutum/wordpress     "/run.sh"                3 hours ago
Up 3 hours          0.0.0.0:80->80/tcp, 3306/tcp                 vibrant_joliot
```

12. Now open a new browser window and navigate to URL (using DNS or IP as before) but with port "**8080**" append to it. You can also try port "**9090**".

> Note:Notice that you now have three WordPress blog instances running inside separate containers launched within few seconds. Contrast this to instead creating and running WordPress on virtual machine, which could take significantly more time.

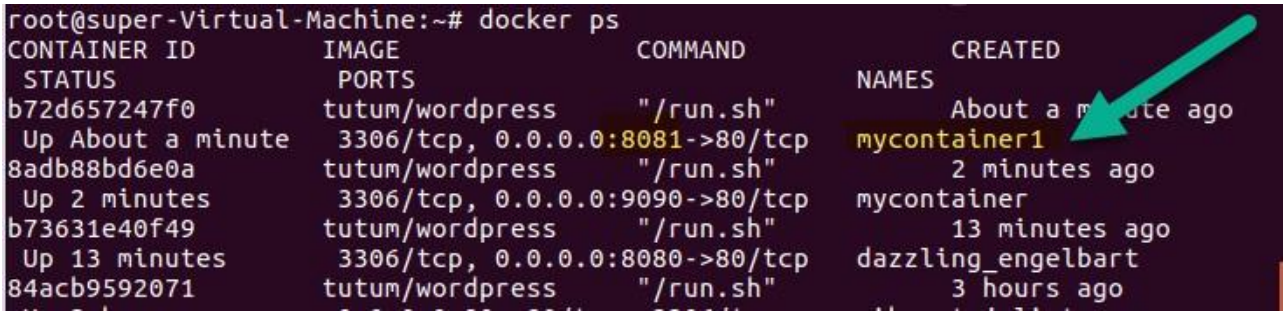


13. If you want to run a container with a name, you can specify the parameter like this: `docker run \--name mycontainer1 -d -p 8081:80 tutum/wordpress`

> Note:Run this on port **8081** so that it does not conflict with one of the previously running containers.

```
root@super-Virtual-Machine:~# docker run --name mycontainer1 -d -p 8081:80 tutum
/wordpress
b72d657247f0388c311f0b22605e1321b164be367c135f1af1ed0a69b6d309c8
```

14. And, now if you run "**docker ps**", you will see that the container has the name you assigned it using the "**--name parameter**".



Congratulations!

You have successfully completed this exercise. Click **Next** to advance to the next exercise.

Exercise 2: Working with the Docker Command Line Interface (CLI)

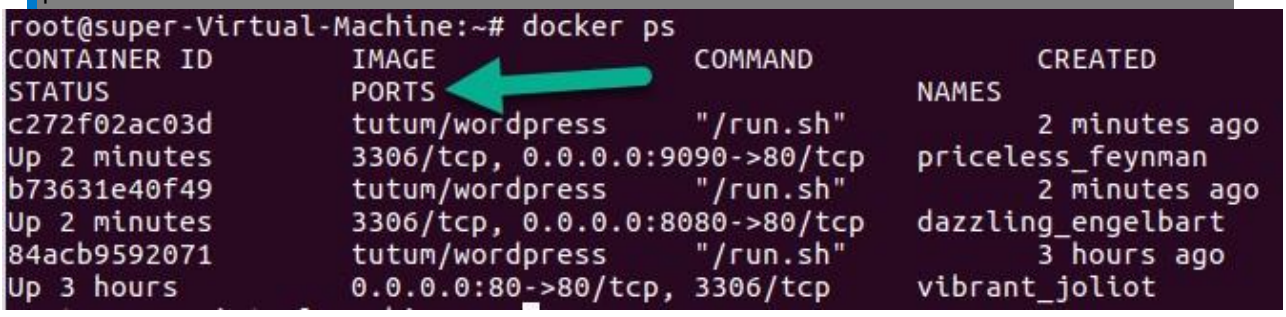
In this exercise, you will learn about common Docker commands needed to work with containers. A comprehensive list of docker commands are available at:
https://docs.docker.com/engine/reference/commandline/docker

Stopping Single Container

Return to list of exercises - Return to list of modules

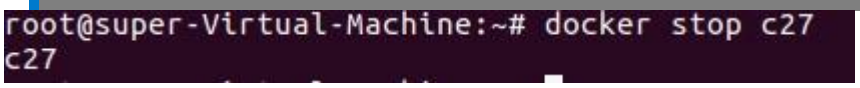
1. First list all the containers currently running by executing "**docker ps**" command. You should see list of all running containers.

> Note:Notice, the list contains multiple containers based on WordPress image that you run in previous exercise.



2. You can stop a running container by using "**docker stop <CONTAINER_ID>**" command. Where CONTAINER_ID is the identifier of a running container.

> Note:You can just use the first couple characters to identity the container ID, such as "**c27**" for the screenshot below.



3. Now run the "**docker ps**" command and notice the listing show one less container running.

```
root@super-Virtual-Machine:~# docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED
STATUS              PORTS                                   NAMES
b73631e40f49        tutum/wordpress     "/run.sh"           5 minutes ago
Up 5 minutes        3306/tcp, 0.0.0.0:8080->80/tcp          dazzling_engelbart
84acb9592071        tutum/wordpress     "/run.sh"           3 hours ago
Up 3 hours          0.0.0.0:80->80/tcp, 3306/tcp            vibrant_joliot
```

4. If you want see the Container ID of the stopped container, and you forgot the Container ID, you can run "**docker ps -a"** to see all containers, even those that are stopped/exited.

```
root@super-Virtual-Machine:~# docker ps -a
CONTAINER ID        IMAGE               COMMAND             CREATED
STATUS                          PORTS                                   NAMES
c272f02ac03d        tutum/wordpress     "/run.sh"           7 minutes ago
Exited (137) About a minute ago                                         priceless_fey
nman
b73631e40f49        tutum/wordpress     "/run.sh"           7 minutes ago
Up 7 minutes                    3306/tcp, 0.0.0.0:8080->80/tcp          dazzling_enge
lbart
2e9e3bddc8fe        tutum/wordpress     "/run.sh"           7 minutes ago
Created                                                                 mystifying_ar
dinghelli
84acb9592071        tutum/wordpress     "/run.sh"           3 hours ago
Up 3 hours                      0.0.0.0:80->80/tcp, 3306/tcp            vibrant_jolio
t
root@super-Virtual-Machine:~#
```

Restart a Container

1. In previous task you issued a Docker command to stop a running container. You can also issue command to start the container which was stopped. All you need is a container ID (same container ID you used earlier to stop a container in previous section), you can also get this using 'docker ps -a'.

2. To start a container run "**docker start <CONTAINER_ID>**".

> Tip: This uses the container identifier you use in previous section to stop the container.

```
root@super-Virtual-Machine:~# docker start c27
c27
```

3. To make sure that container has started successfully run "**docker ps**" command.

> Note:Notice that WordPress container is now started.

```
root@super-Virtual-Machine:~# docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED
STATUS              PORTS                                   NAMES
b72d657247f0        tutum/wordpress     "/run.sh"           6 minutes ago
Up 6 minutes        3306/tcp, 0.0.0.0:8081->80/tcp          mycontainer1
c272f02ac03d        tutum/wordpress     "/run.sh"           18 minutes ago
Up 23 seconds       3306/tcp, 0.0.0.0:9090->80/tcp          priceless_feynman
b73631e40f49        tutum/wordpress     "/run.sh"           19 minutes ago
Up 19 minutes       3306/tcp, 0.0.0.0:8080->80/tcp          dazzling_engelbart
84acb9592071        tutum/wordpress     "/run.sh"           3 hours ago
Up 3 hours          0.0.0.0:80->80/tcp, 3306/tcp            vibrant_joliot
```

Removing a Container

1. Stopping a container does not remove it and that's the reason why you were able to start it again in the previous task.

To delete/remove a container and free the resources you need to issue a different command. Please note that this command does not remove the underlying image but rather the specific container that was based on the image. To remove the image and reclaim its resources, like disk space, you'll will need to issue a different command which is covered under the later section "Removing Container Image". 2. To remove a container, run "**docker rm -f <CONTAINER_ID>**" command.

This uses the container identifier you used in previous section. If you don't have it handy, simply run "docker ps" and copy the container ID from the listing.

```
root@super-Virtual-Machine:~# docker rm -f c27
c27
```

> Note:The "**-f**" switch is used to force the remove operation. It's needed if you are trying to remove a container that is not already stopped.

Stopping All Containers

1. At times you may want to stop all of the running containers and avoid issuing command to stop one container at a time. Run "**docker stop $(docker ps -aq)**" command to stop all running containers. Basically, you are issuing two commands: First the **docker ps** with relevant switches to capture list of container IDs and then passing list of IDs to **docker stop** command.

```
root@super-Virtual-Machine:~# docker stop $(docker ps -aq)
b72d657247f0
8adb88bd6e0a
d2414edc37a0
3a235465566c
1bb0c6c5839e
5269d62ea056
763e22cd0765
b73631e40f49
2e9e3bddc8fe
84acb9592071
```

Removing WordPress Container Image

1. Removing a container image form a local system will let you reclaim its disk space. Please note that this operation is irreversible so proceed with caution. In this task you will remove the WordPress container image as you will not be using it any more. You must stop all containers using the image before you can delete the image, unless you use the force parameter.

2. To remove a container image, you'll need its IMAGE ID. Run command "**docker images**".

```
root@super-Virtual-Machine:~# docker images
REPOSITORY          TAG                        IMAGE ID            CREATED             SIZE
microsoft/dotnet    2.2-aspnetcore-runtime     2f7531819f40        25 hours ago        260MB
nginx               stable-alpine              cafef9fe265b        5 days ago          16MB
node                boron                      753cb37bc49c        8 days ago          884MB
tutum/wordpress     latest                     7e7f97a602ff        2 years ago         477MB
```

3. Run the command "**docker rmi <IMAGE_ID> -f**".

> Note:Notice the command to remove docker container is "**docker rm**" and to remove an image is "**docker rmi**", with an 'i' for the image. Don't confuse these two commands! The **-f** is to force the removal, you cannot remove an image associated with a stopped container unless you use the force parameter.

```
root@super-Virtual-Machine:~# docker rmi 7e7f -f
Untagged: tutum/wordpress:latest
Untagged: tutum/wordpress@sha256:2aa05fd3e8543b615fc07a628da066b48e6bf41cceeeb8f
4b81e189de6eeda77
Deleted: sha256:7e7f97a602ff0c3a30afaaac1e681c72003b4c8a76f8a90696f03e785bf36b90
```

4. Now, run the command "**docker images**".

> Note:Notice that "tutum/wordpress" image is no longer available.

```
root@super-Virtual-Machine:~# docker images
REPOSITORY                          TAG                 IMAGE ID            CREATED             SIZE
mcr.microsoft.com/dotnet/core/aspnet   3.1                 a843e0fbe833        2 weeks ago         207MB
nginx                               stable-alpine       aaad4724567b        2 months ago        21.2MB
node                                boron               ab290b853066        8 months ago        884MB
mcr.microsoft.com/mssql/server      2017-CU12-ubuntu    4095d6d460cd        14 months ago       1.32GB
```

Congratulations!

You have successfully completed this exercise. Click **Next** to advance to the next exercise.

Exercise 3: Building Custom Container Images with Dockerfile: NodeJS, Nginx, and ASP.NET Core 3.x

---

A Dockerfile is essentially a plain text file with Docker commands in it that are used to create a new image. You can think of it as a configuration file with a set of instructions needed to assemble a new image. In this exercise, you will learn the common commands that go into Dockerfiles by creating custom images based on common technologies like NGINX, Node JS and ASP .NET Core.

Return to list of exercises - Return to list of modules

Building and Running Node.JS Application as Container

1. In this task you will create a new image based on the Node.js base image. You will start with a Dockerfile with instructions to copy the files needed to host a custom Node.js application, install necessary software inside the image and expose ports to allow the traffic. Later, you will learn how to build the image using Dockerfile and finally will run and test it out.

> Note:The relevant files related to a node.js application along with the Dockerfile are available inside the directory "labs/module1/nodejs". You can get to that directory by using the "cd" command to navigate.

```
root@super-Virtual-Machine:~# ls
Desktop  labs
root@super-Virtual-Machine:~# cd labs/module1/nodejs/
root@super-Virtual-Machine:~/labs/module1/nodejs#
```

2. On the command prompt type "**ls**" and press Enter.

> Note:Notice the available files include "server.js", "package.json" and "Dockerfile".

```
root@super-Virtual-Machine:~/labs/module1/nodejs# ls
Dockerfile  package.json  server.js
```

3. Let's examine the Dockerfile by typing the command "**nano Dockerfile**" and press Enter.

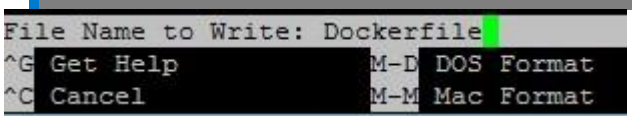
> Note: The file is case sensitive, so make sure the D in Dockerfile is capitalized.
>
> Note:You can use any other text editor (for example vi etc. but instructions are provided for nano text editor).
>
> Note:Notice the structure of Dockerfile.

```
  GNU nano 2.5.3                          File: Dockerfile

# Simple Dockerfile for NodeJS

FROM node:boron

LABEL author="sampleauthor@contoso.com"

# Create app directory
RUN mkdir -p /usr/src/app
WORKDIR /usr/src/app

# Install app dependencies
COPY package.json .
RUN npm install

# Bundle app source
COPY . .

EXPOSE 8080

CMD [ "npm", "start" ]
```

4. Move your cursor using the arrow keys to the line starting with **Label author="sampleauthor@contoso.com"** and change the text from that to the following: **LABEL author="YourEmail@Email.com"**. Once you finish making changes press **CTRL + X** and then press **Y**

when asked for confirmation to retain your changes. Finally, you will be asked for file name to write. For that press **Enter** (without changing the name of the file). This will close the nano text editor.

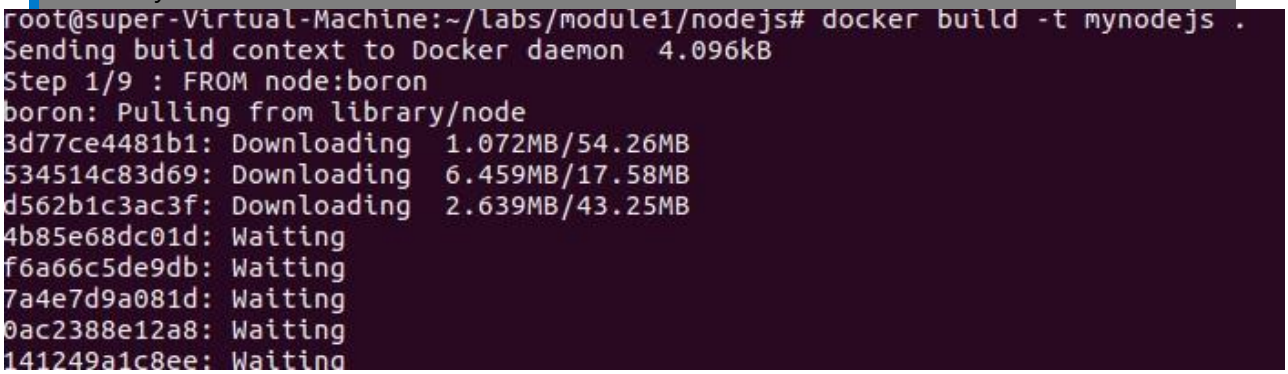
> Note: This will close the nano text editor.



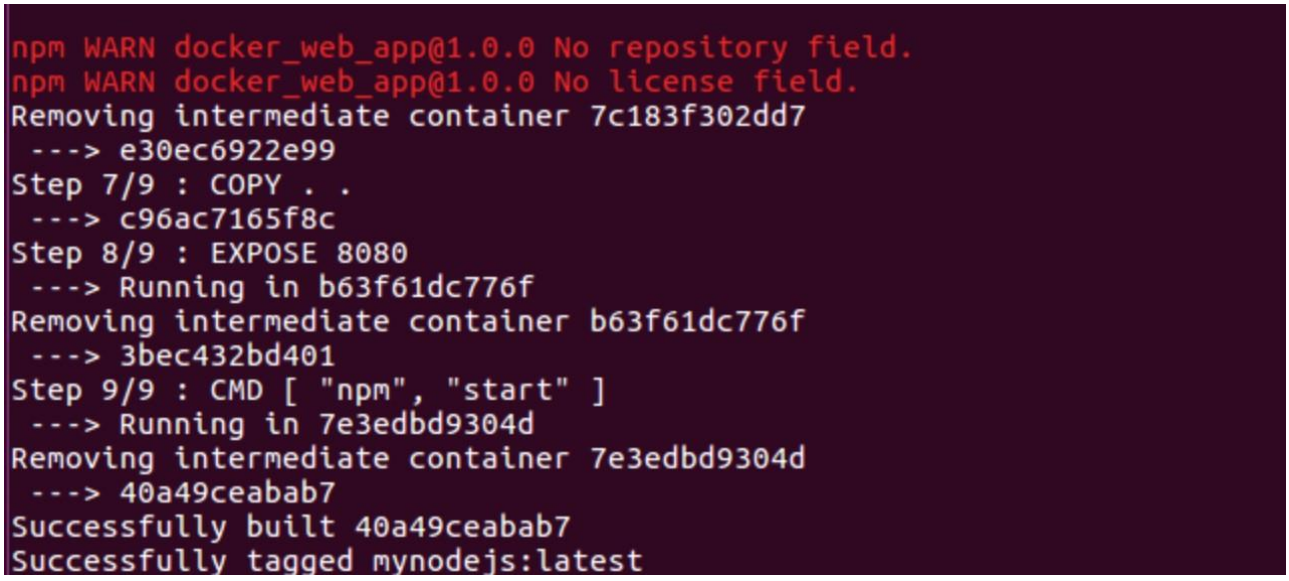
You are now ready to build a new image based on the Dockerfile you just modified.

5. Run the command "**docker build -t mynodejs .**"

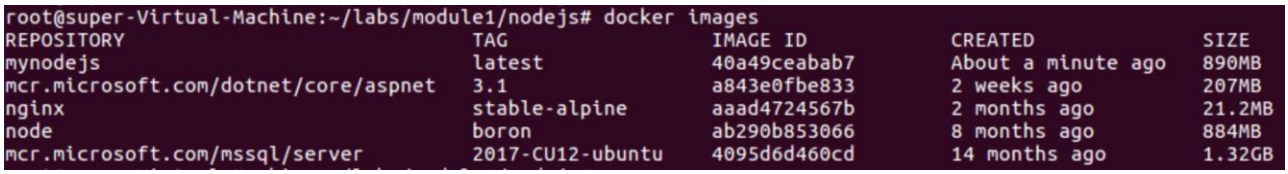
> Alert:Pay close attention to the period that is at the end of command.
>
> Knowledge:Notice how the build command is reading instructions from the Dockerfile staring from the top and executing them one at a time. This will take a few minutes to pull the image down to your VM.



6. When it is complete, you will see a couple of npm warnings, these are expected.



7. Run the command "**docker images**" and notice the new container image appears with the name "**mynodejs**". Also notice the presence of parent image "**node**" that was also pulled from Docker Hub during the build operation (if you were using the provided lab machines, they may have already been cached locally).

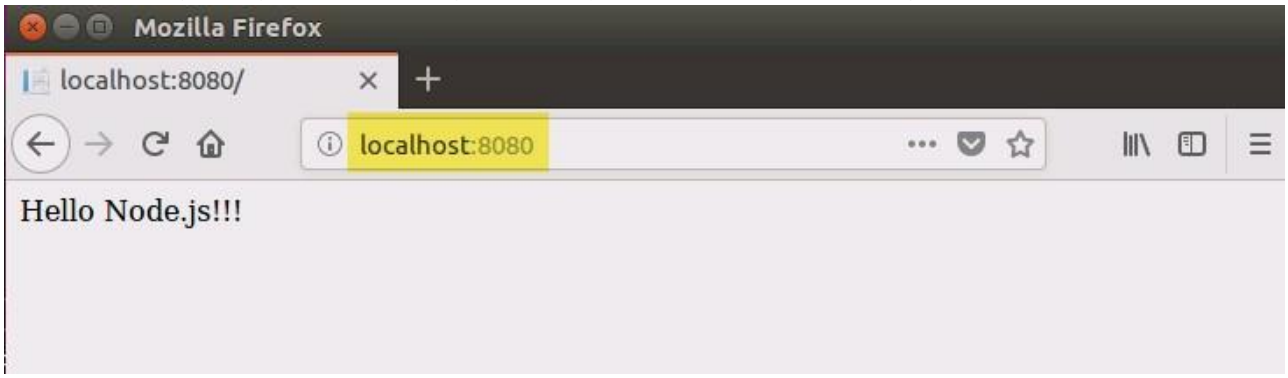


8. Finally, lets create and run a new container based on "**mynodejs**" image. Run command `docker run -d -p 8080:8080 mynodejs`

> Knowledge:The **-d** parameter will run the container in the background, whereas the **-p** parameter publishes the container port to the host.
>
> Note:Here, we are binding the port of the container (port number on right-side of colon) to the port of the host machine (port number on the left-side of the colon).

```
root@super-Virtual-Machine:~/labs/module1/nodejs# docker run -d -p 8080:8080 mynodejs
81c95ab1d59c954a8da7d8aa145782456597e6e3884750519267fb12ab4ff108
```

9. To test the "**mynodejs**" application, go back to your Firefox browser and go to **localhost:8080**.



Building and Running NGINX Container

1. In this task you will create a new image using the NGINX web server base image hosting a simple static html page. You will start with a Dockerfile with instructions to define its base image, then copy the static html file inside the image and then specify the startup command for the image (using CMD instruction).
   Later, you will learn how to build the image using Dockerfile and finally will run and test it out.

   The relevant files including static html file **index.html** along with the Dockerfile are available inside the directory **labs/module1/nginx**.

```
root@super-Virtual-Machine:~/labs/module1/nodejs# cd ..
root@super-Virtual-Machine:~/labs/module1# cd nginx
root@super-Virtual-Machine:~/labs/module1/nginx#
```

2. Type "**ls**" and press Enter. Notice the available files include "**index.html**" and "**Dockerfile**".

```
root@super-Virtual-Machine:~/labs/module1/nginx# ls
Dockerfile  index.html
```

3. Let's examine the Dockerfile by typing the command "**nano Dockerfile**" and press Enter.

> Knowledge:You can use any other text editor (for example, vi, etc.), but instructions are provided for nano text editor).
> Note:Notice the structure of Dockerfile.

```
  GNU nano 2.5.3                          File: Dockerfile

# Simple Dockerfile for NGINX

FROM nginx:stable-alpine

LABEL author="sampleauthor@contoso.com"

COPY index.html /usr/share/nginx/html/index.html

CMD ["nginx", "-g", "daemon off;"]
```

4. Move your cursor using the arrow keys to the line starting with **Label author="sampleauthor@contoso.com"** and change the text from that to the following: **LABEL author="YourEmail@Email.com"**. Once you are finished making changes press **CTRL + X** and then press **Y** when asked for confirmation to retain your changes. Finally, you will be asked for file name to write. For that press **Enter** (without changing the name of the file). This will close the nano text editor.

```
File Name to Write: Dockerfile
^G Get Help          M-D DOS Format
^C Cancel            M-M Mac Format
```

5. You are now ready to build a new container image based on the Dockerfile you just modified. Run the command "**docker build -t mynginx .**"

```
root@super-Virtual-Machine:~/labs/module1/nginx# docker build -t mynginx .
Sending build context to Docker daemon  3.072kB
Step 1/4 : FROM nginx:stable-alpine
 ---> a74ad6df6eeb
Step 2/4 : LABEL author="sampleauthor@contoso.com"
 ---> Running in ec896079f821
Removing intermediate container ec896079f821
 ---> 1117e459df6c
Step 3/4 : COPY index.html /usr/share/nginx/html/index.html
 ---> ac4455381453
Step 4/4 : CMD ["nginx", "-g", "daemon off;"]
 ---> Running in 38007d566bb1
Removing intermediate container 38007d566bb1
 ---> 6c6c6cbb90c4
Successfully built 6c6c6cbb90c4
Successfully tagged mynginx:latest
```

> Note:Notice how the build command is reading instructions from the Docker file starting from the top and executing them one at a time. The image will download much faster as this is a very small image.

6. If you want to see the layers of an image, you can do "**docker history mynginx**" and see the one you just built. You can also try running this command on other images you have on your VM too.

```
root@super-Virtual-Machine:~/labs/module1/nginx# docker history mynginx
IMAGE               CREATED             CREATED BY                                      SIZE
867008461f8e        22 seconds ago      /bin/sh -c #(nop)  CMD ["nginx" "-g" "daemon…   0B
972311bb9adf        23 seconds ago      /bin/sh -c #(nop) COPY file:a572a7c02b4bb63c…   57B
480ffea12e7c        23 seconds ago      /bin/sh -c #(nop)  LABEL author=sampleauthor…   0B
aaad4724567b        2 months ago        /bin/sh -c #(nop)  CMD ["nginx" "-g" "daemon…   0B
<missing>           2 months ago        /bin/sh -c #(nop)  STOPSIGNAL SIGTERM           0B
<missing>           2 months ago        /bin/sh -c #(nop)  EXPOSE 80                    0B
<missing>           2 months ago        /bin/sh -c set -x     && addgroup -g 101 -S …   15.6MB
<missing>           2 months ago        /bin/sh -c #(nop)  ENV PKG_RELEASE=1            0B
<missing>           2 months ago        /bin/sh -c #(nop)  ENV NJS_VERSION=0.3.5        0B
<missing>           2 months ago        /bin/sh -c #(nop)  ENV NGINX_VERSION=1.16.1     0B
<missing>           2 months ago        /bin/sh -c #(nop)  LABEL maintainer=NGINX Do…   0B
<missing>           2 months ago        /bin/sh -c #(nop)  CMD ["/bin/sh"]              0B
<missing>           2 months ago        /bin/sh -c #(nop) ADD file:fe1f09249227e2da2…   5.55MB
```

7. Run the command "**docker images**"

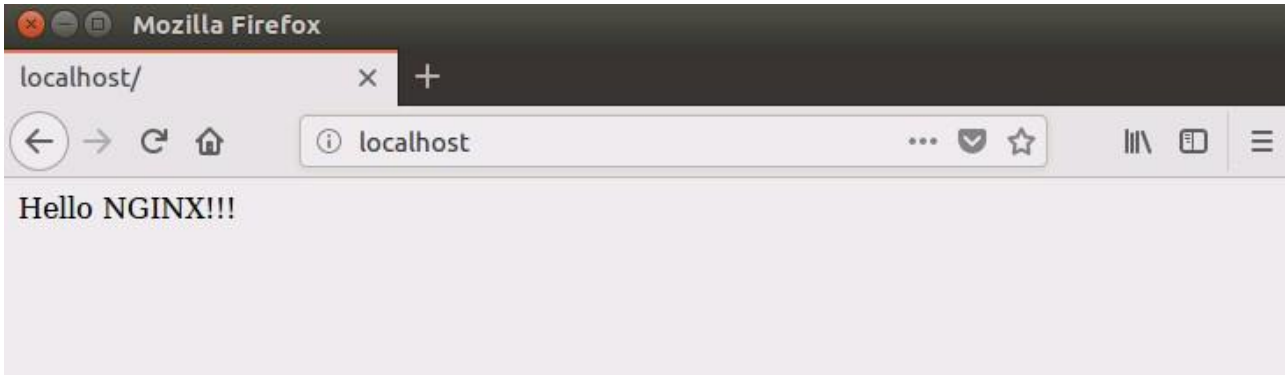
> Note:Notice the new container image appears with the name **mynginx**. Also notice the presence of parent image **nginx** that was pulled from Docker Hub during the build operation. Take a look at the sizes of different images also. This will become important when you build your own custom images to reduce the size for both security and performance.

```
root@super-Virtual-Machine:~/labs/module1/nginx# docker images
REPOSITORY                                TAG                 IMAGE ID            CREATED             SIZE
mynginx                                   latest              867008461f8e        3 minutes ago       21.2MB
mynodejs                                  latest              40a49ceabab7        8 minutes ago       890MB
mcr.microsoft.com/dotnet/core/aspnet      3.1                 a843e0fbe833        2 weeks ago         207MB
nginx                                     stable-alpine       aaad4724567b        2 months ago        21.2MB
node                                      boron               ab290b853066        8 months ago        884MB
mcr.microsoft.com/mssql/server            2017-CU12-ubuntu    4095d6d460cd        14 months ago       1.32GB
```

8. Finally, create and run a new container based on "**mynginx**" image. Run command "**docker run -d -p 80:80 mynginx**".

```
root@super-Virtual-Machine:~/labs/module1/nginx# docker run -d -p 80:80 mynginx
6a421269b708e8c81197883fb6e7766c8008418cd061d1555d0fc9d0971081a1
```

9. To test the node app, go to your **Firefox** browser and go to **localhost**.



Building and Running ASP.NET Core 3.x Application Inside A Container

---

1. In this task you will build ASP .NET Core 3.x application and then package and run it as a container. Change to the relevant directory **labs/module1/aspnetcore**. First, we need to run **dotnet build**, and **publish** to generate the binaries for our application. This can be done manually or by leveraging a **Dockerfile**. In this example, we will run the commands manually to produce the artifacts in a folder called **published**. The **Dockerfile** will only contain instructions to copy the files from the **published** folder into the image. `dotnet build`

```
root@super-Virtual-Machine:~/labs/module1/aspnetcore# dotnet build

Welcome to .NET Core 3.1!
---------------------
SDK Version: 3.1.100

Telemetry
---------
The .NET Core tools collect usage data in order to help us improve your experience. The data is anonymous. It is collec
ted by Microsoft and shared with the community. You can opt-out of telemetry by setting the DOTNET_CLI_TELEMETRY_OPTOUT
 environment variable to '1' or 'true' using your favorite shell.

Read more about .NET Core CLI Tools telemetry: https://aka.ms/dotnet-cli-telemetry

---------------
Explore documentation: https://aka.ms/dotnet-docs
Report issues and find source on GitHub: https://github.com/dotnet/core
Find out what's new: https://aka.ms/dotnet-whats-new
Learn about the installed HTTPS developer cert: https://aka.ms/aspnet-core-https
Use 'dotnet --help' to see available commands or visit: https://aka.ms/dotnet-cli-docs
Write your first app: https://aka.ms/first-net-core-app
--------------------------------------------------------------------------------
Microsoft (R) Build Engine version 16.4.0+e901037fe for .NET Core
Copyright (C) Microsoft Corporation. All rights reserved.

  Restore completed in 474.93 ms for /root/labs/module1/aspnetcore/mywebapp.csproj.
  mywebapp -> /root/labs/module1/aspnetcore/bin/Debug/netcoreapp3.1/mywebapp.dll
  mywebapp -> /root/labs/module1/aspnetcore/bin/Debug/netcoreapp3.1/mywebapp.Views.dll

Build succeeded.
    0 Warning(s)
    0 Error(s)

Time Elapsed 00:00:10.48
```

`dotnet publish -o published`

```
root@super-Virtual-Machine:~/labs/module1/aspnetcore# dotnet publish -o published
Microsoft (R) Build Engine version 16.4.0+e901037fe for .NET Core
Copyright (C) Microsoft Corporation. All rights reserved.

  Restore completed in 40.96 ms for /root/labs/module1/aspnetcore/mywebapp.csproj.
  mywebapp -> /root/labs/module1/aspnetcore/bin/Debug/netcoreapp3.1/mywebapp.dll
  mywebapp -> /root/labs/module1/aspnetcore/bin/Debug/netcoreapp3.1/mywebapp.Views.dll
  mywebapp -> /root/labs/module1/aspnetcore/published/
root@super-Virtual-Machine:~/labs/module1/aspnetcore# ls
appsettings.Development.json  bin          Dockerfile  mywebapp.csproj  Program.cs  published  Views
appsettings.json              Controllers  Models      obj              Properties  Startup.cs  wwwroot
```

2. Now that the application is ready, you will create your container image. The Dockerfile is provided to you. View the content of Dockerfile by running the **nano Dockerfile** command. To exit the editor press **CTRL+X**..

> Note:The Dockerfile contents should match the screenshot below:

```
  GNU nano 2.5.3                                File: Dockerfile

FROM mcr.microsoft.com/dotnet/core/aspnet:3.1
WORKDIR /app
COPY published ./
ENTRYPOINT ["dotnet", "mywebapp.dll"]
```

3. To create the container image run the command `docker build -t myaspcoreapp:3.1 .`

> Note:Notice the **3.1** tag representing the dotnet core framework version.

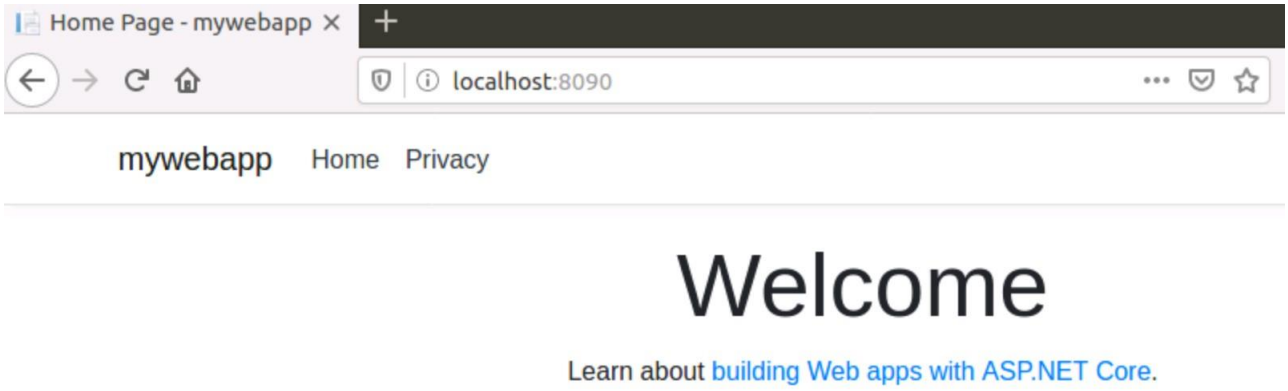
```
root@super-Virtual-Machine:~/labs/module1/aspnetcore# docker build -t myaspnetcoreapp:3.1 .
Sending build context to Docker daemon  10.19MB
Step 1/4 : FROM mcr.microsoft.com/dotnet/core/aspnet:3.1
 ---> a843e0fbe833
Step 2/4 : WORKDIR /app
Removing intermediate container d2d896f5e7f9
 ---> b6bf139c497f
Step 3/4 : COPY published ./
 ---> edbf5428a149
Step 4/4 : ENTRYPOINT ["dotnet", "mywebapp.dll"]
 ---> Running in ae3bf3132366
Removing intermediate container ae3bf3132366
 ---> 59804bf835a3
Successfully built 59804bf835a3
Successfully tagged myaspnetcoreapp:3.1
root@super-Virtual-Machine:~/labs/module1/aspnetcore#
```

4. Launch the container running your application using the command `docker run -d -p 8090:80 myaspcoreapp:3.1`

```
root@super-Virtual-Machine:~/labs/module1/aspnetcore# docker run -d -p 8090:80 myaspnetcoreapp:3.1
1c98748022e0c1b64a4b1fea9a3b7af9e4d735bf1d94074a0d9b497350ca0801
```

> Note:You are now running ASP.NET Core application inside the container listening at port 80 which is mapped to port 8090 on the host.

5. To test the application, go to **localhost:8090** in your **Firefox** browser.



Congratulations!

You have successfully completed this exercise. Click **Next** to advance to the next exercise.

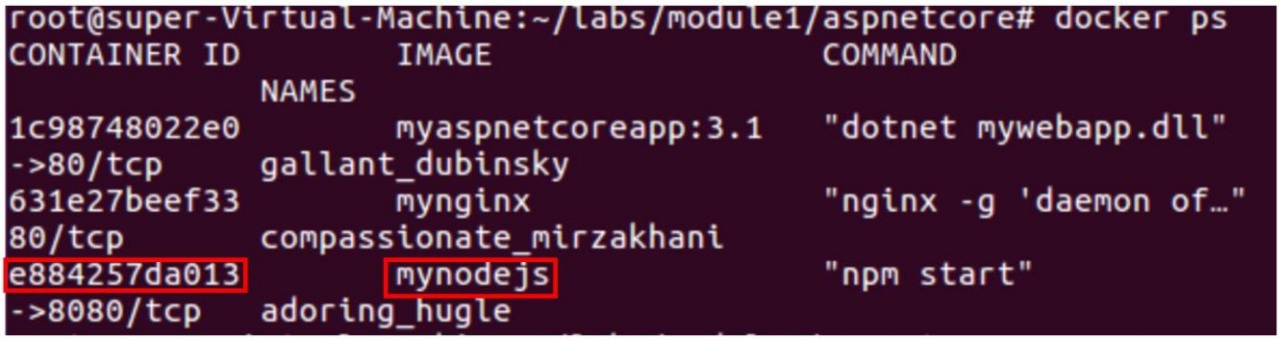
Exercise 4: Interaction with a Running Container

In the previous exercise, you were able to build and run containers based on Dockerfiles. However, there may be situations that warrant interacting with a running container for the purposes of troubleshooting, monitoring etc. You may also want to make changes/updates to a running container and then build a new image based on those changes. In this exercise, you will interact with a running container and then learn to persist your changes as a new image.

Return to list of exercises - Return to list of modules

Interaction with a Running Container

1. On the command line run "**docker ps**" to list all the currently running containers on your virtual machine.



> Note:Notice that multiple containers are running. To establish interactive session a with a running container you will need its **CONTAINER ID** or **NAME**. Let's establish an interactive session to a container based on "**mynodejs**" image. Please note that your **CONTAINER ID** or **NAME** will probably be different. And, unless you specified a name, Docker came up with a random adjective and noun and smushed them together to come up with its own clever name.

2. Run a command "**docker exec -it <CONTAINER_ID_OR_NAME> bash**" on your **mynodejs container**.

> Note: You can run the command **docker exec -it <CONTAINER_ID_OR_NAME> bash** using either the container ID or name.
>
> Knowledge: **docker exec** is used to run a command in a running container. The **it** parameter will invoke an interactive bash shell inside the container.
>
> Note:Notice that a new interactive session is now establish to a running container. Since "**bash**" is the program that was asked to be executed you now have access to full bash shell inside the container.

```
root@super-Virtual-Machine:~/labs/module1/aspnetcore# docker exec -it e8 bash
root@e884257da013:/usr/src/app#
```

3. You can run a command "**ls**" to view the listing of files and directories.

> Note: Notice it has all the files copied by Dockerfile command in previous section.

```
root@e884257da013:/usr/src/app# ls
Dockerfile  node_modules  package.json  server.js
```

> Knowledge: For more information regarding running commands inside docker container please visit: https://docs.docker.com/engine/reference/commandline/exec

Making Changes to a Running Container

While you are interacting and running commands inside a running container, you may also want to make changes/updates to it. Later, you may also create a brand-new image out of these changes. In this task you will make changes to "mynodejs" container image, test them out and finally create a new image (without the need of Dockerfile).

> Note:Please note that this approach of creating container images is generally used to quickly test various changes, but the best practice to create container images is to use a Dockerfile since it is a declarative file that can be kept in source control repositories.

First, you will make updates to **server.js** file. You should have an active session already established from previous exercise (if not then please follow the instructions from the previous section to create an active session now).

Before we can edit the **server.js** file we need to install a text editor. To keep the size of container to a minimum, the **nodejs** container image does not have any extra software installed in the container. This is a common theme when building images and is also the recommend practice.

1. Before installing any software run the command `apt-get update`

   > Warning:Note the dash between "apt" and "-get".

```
root@e884257da013:/usr/src/app# apt-get update
Ign:1 http://deb.debian.org/debian stretch InRelease
Get:2 http://deb.debian.org/debian stretch-updates InRelease [91.0 kB]
Get:3 http://security.debian.org/debian-security stretch/updates InRelease [94.3 kB]
Get:4 http://deb.debian.org/debian stretch Release [118 kB]
Get:5 http://deb.debian.org/debian stretch Release.gpg [2365 B]
Get:6 http://deb.debian.org/debian stretch-updates/main amd64 Packages [27.9 kB]
Get:7 http://security.debian.org/debian-security stretch/updates/main amd64 Packages [513 kB]
Get:8 http://deb.debian.org/debian stretch/main amd64 Packages [7086 kB]
Fetched 7933 kB in 2s (3542 kB/s)
Reading package lists... Done
```

2. To install "**nano**" run a command `apt-get install nano`

```
root@e884257da013:/usr/src/app# apt-get update
Ign:1 http://deb.debian.org/debian stretch InRelease
Get:2 http://deb.debian.org/debian stretch-updates InRelease [91.0 kB]
Get:3 http://security.debian.org/debian-security stretch/updates InRelease [94.3 kB]
Get:4 http://deb.debian.org/debian stretch Release [118 kB]
Get:5 http://deb.debian.org/debian stretch Release.gpg [2365 B]
Get:6 http://deb.debian.org/debian stretch-updates/main amd64 Packages [27.9 kB]
Get:7 http://security.debian.org/debian-security stretch/updates/main amd64 Packages [513 kB]
Get:8 http://deb.debian.org/debian stretch/main amd64 Packages [7086 kB]
Fetched 7933 kB in 2s (3542 kB/s)
Reading package lists... Done
root@e884257da013:/usr/src/app#
root@e884257da013:/usr/src/app# apt-get install nano
Reading package lists... Done
Building dependency tree
Reading state information... Done
Suggested packages:
  spell
The following NEW packages will be installed:
  nano
0 upgraded, 1 newly installed, 0 to remove and 49 not upgraded.
Need to get 485 kB of archives.
After this operation, 2092 kB of additional disk space will be used.
Get:1 http://deb.debian.org/debian stretch/main amd64 nano amd64 2.7.4-1 [485 kB]
Fetched 485 kB in 0s (2573 kB/s)
debconf: delaying package configuration, since apt-utils is not installed
Selecting previously unselected package nano.
(Reading database ... 29980 files and directories currently installed.)
Preparing to unpack .../nano_2.7.4-1_amd64.deb ...
Unpacking nano (2.7.4-1) ...
Setting up nano (2.7.4-1) ...
update-alternatives: using /bin/nano to provide /usr/bin/editor (editor) in auto mode
update-alternatives: using /bin/nano to provide /usr/bin/pico (pico) in auto mode
```

3. After "**nano**" is installed successfully, run the command "**nano server.js**" to open "**server.js**" file for editing.

```
root@e884257da013:/usr/src/app# nano server.js
```

4. Use the arrow keys to go to the line starting with "**res.Send(...**" and update the text from "**Hello Node.js!!!**" to "**Hello Node.js AGAIN!!!**".

> Note:Your final changes should look like following:

```
  GNU nano 2.7.4                File: server.js                Modified

'use strict';

const express = require('express');

// Constants
const PORT = 8080;

// App
const app = express();
app.get('/', function (req, res) {
  res.send('Hello Node.js AGAIN!!!\n');
});

app.listen(PORT);
console.log('Running on http://localhost:' + PORT);
```

5. Once you finish making changes press "**CTRL + X**" and then press "**Y**" when asked for confirmation to retain changes. Finally, you will be asked for file name to write. For that press **enter** (without changing the name of the file). This will close the Nano text editor.

6. To save the updates and exit the interactive bash session, run the command "**exit**"

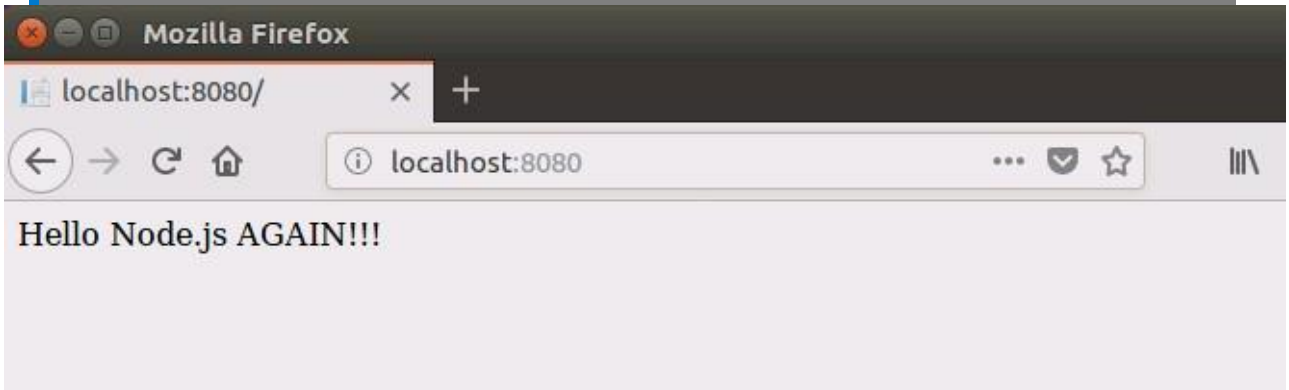
```
root@e884257da013:/usr/src/app# exit
exit
root@super-Virtual-Machine:~/labs/module1/aspnetcore#
```

7. The running container needs to be stopped first and then started again to reflect the changes. Run the command "**docker stop <<CONTAINER ID>>**" to stop the container. Run the command "**docker start <<CONTAINER ID>>**" to start the container.

```
root@super-Virtual-Machine:~/labs/module1/aspnetcore# docker stop e8
e8
root@super-Virtual-Machine:~/labs/module1/aspnetcore# docker start e8
e8
```

8. Finally, to test the update you have made to the container go to **Firefox** and **localhost:8080**.

> Note:Notice the output "**Hello Node.js AGAIN!!!**". This verifies that changes to the container were persisted.



Interaction with a Running Container

In the previous task you have made changes to running container. However, these changes are only available to that container and if you were to remove the container, these changes would be lost. One way to address this is by creating a new container image based on running container that has the changes. This way changes will be available as part of a new container image. This is helpful during dev/test phases, where rapid development and testing requires a quick turn-around time. However, this approach is generally not recommended, as it's hard to manage and scale at the production level. Also, if content is the only piece that needs to be changed and shared, then using **volumes** may be another viable option. Volumes are covered in module three.

1. To create new image run the command "**docker commit <CONTAINER ID> mynodejsv2**".

   > Knowledge:The docker commit command is used to create a new image from a container's changes.
   >
   > Knowledge:If you don't already have it, you can use "**docker ps**" command to get the list of running containers or "**docker ps -a**" to get list of all the containers that are stopped and capture the CONTAINER ID of the container you have updated in previous section.

```
root@super-Virtual-Machine:~/labs/module1/aspnetcore# docker commit e8 mynodejsv2
sha256:4c89d4f34e70e57c797350f17d8482cf50c9d2901f9a5dd83f1131c010c96850
```

2. Now, view the list of all container images by running the command `docker images`

   > Note:Notice the availability of new image with name "mynodejsv2"

```
root@super-Virtual-Machine:~/labs/module1/aspnetcore# docker images
REPOSITORY                  TAG                 IMAGE ID            CREATED              SIZE
mynodejsv2                  latest              4c89d4f34e70        About a minute ago   909MB
myaspnetcoreapp             3.1                 59804bf835a3        32 minutes ago       212MB
```

   > Note:You now have a container image with the changes you made and tested earlier and is ready to be used.

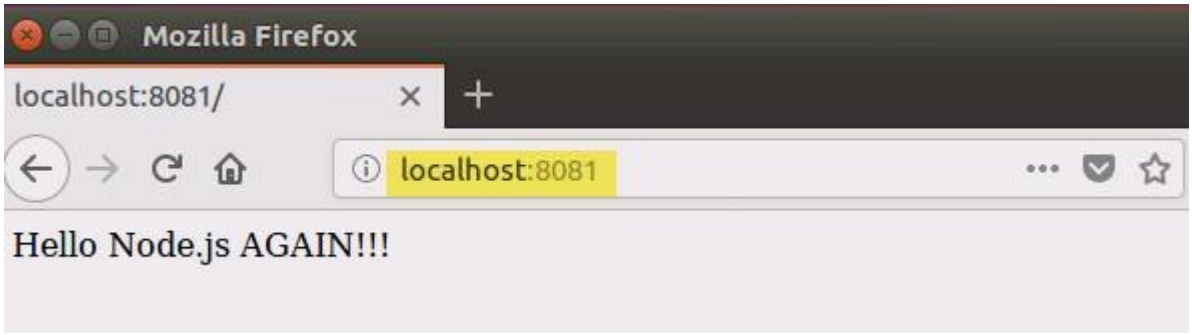
3. To test the new image run a command `docker run -d -p 8081:8080 mynodejsv2`

   > Note:This will create a new container based on the image "**mynodejsv2**".

```
root@super-Virtual-Machine:~/labs/module1/aspnetcore# docker run -d -p 8081:8080 mynodejsv2
c000396cf6b8b4bc34ab7d4eafbd2ce0a952215a16dc17f98cc4c2a6930c5fb1
```

4. Finally, to test the container, go to **localhost:8081** in **Firefox**.

   > Note:Notice the text "**Hello Node.js AGAIN!!!**" is returned from the node.js application. This attest that changes were committed properly to new image and hence available to any container created based on the that image.



Congratulations!

You have successfully completed this exercise. Click **Next** to advance to the next exercise.

Exercise 5: Tagging

In this exercise you will learn the role of tagging in container and how to tag new and existing container images using Docker commands.

Return to list of exercises - Return to list of modules

Tagging Existing Container Image

- In this task you will tag the **mynodejs** container image with **v1**. Recall from the last task that currently this image has the **latest** tag associated with it. You can simply run **docker images** to verify that. When working with container images it becomes important to provide consistent versioning information.

- Tagging provides you with the ability to tag container images properly at the time of building a new image using the **docker build -t imagename:tag .** command. You can then refer to the image (for example inside Dockerfile with **FROM** statement) using a format **image-name:tag**.

- If you don't provide a tag, Docker assumes that you meant **latest** and use it as a default tag for the image. It is not good practice to make images without tagging them. You'd think you could assume latest = most recent image version always? Wrong. Latest is just the tag which is applied to an image by default which does not have a tag. If you push a new image with a tag which is neither empty nor 'latest', :latest will not be affected or created. Latest is also easily overwritten by default if you forget to tag something again in the future. **Careful!!!**

- When you run **docker images** notice the **TAG** column and pay attention to the fact that all of the custom images created in the lab so far have tag value of **latest**.

```
root@super-Virtual-Machine:~/labs/module1/aspnetcore# docker images
REPOSITORY                                TAG                 IMAGE ID            CREATED              SIZE
mynodejsv2                                latest              4c89d4f34e70        About a minute ago   909MB
myaspnetcoreapp                           3.1                 59804bf835a3        32 minutes ago       212MB
mynginx                                   latest              867008461f8e        About an hour ago    21.2MB
mynodejs                                  latest              40a49ceabab7        About an hour ago    890MB
mcr.microsoft.com/dotnet/core/aspnet      3.1                 a843e0fbe833        2 weeks ago          207MB
nginx                                     stable-alpine       aaad4724567b        2 months ago         21.2MB
node                                      boron               ab290b853066        8 months ago         884MB
mcr.microsoft.com/mssql/server            2017-CU12-ubuntu    4095d6d460cd        14 months ago        1.32GB
```

- To understand the importance of tagging take a look at the container image created in the previous section **mynodejsv2**. The **v2** at the very end was appended to provide an indicator that this is the

second version of the image **mynodejs**. The challenge with this scheme is that there is no inherent connection between the **mynodejs** and **mynodejsv2**. With tagging, the same container image will take the format **mynodejs:v2**. This way you are telling everyone that **v2** is different but has relation to the **mynodejs** container image.

- Please note that tags are just strings. So, any string including **v1**, **1.0**, **1.1**, **1.0-beta**, and **banana** all qualify as a valid tag.

- You should always want to follow consistent nomenclature when using tagging to reflect versioning. This is critical because when you start developing and deploying containers into production, you may want to roll back to previous versions in a consistent manner. Not having a well-defined scheme for tagging will make it very difficult particularly when it comes to troubleshooting containers.

> Knowledge: A good example of various tagging scheme chosen by Microsoft with dotnet core framework is available at: https://hub.docker.com/r/microsoft/dotnet/tags

1. To tag an existing docker image, run the command "**docker tag <<IMAGE_ID or IMAGE_NAME>> mynodejs:v1**". Replace the IMAGE ID with the image id of "**mynodejs**" container image. To see the updated tag for "**mynodejs**" image run the command "**docker images**".

```
root@super-Virtual-Machine:~/labs/module1/aspnetcore# docker images
REPOSITORY                          TAG                 IMAGE ID            CREATED             SIZE
mynodejsv2                          latest              4c89d4f34e70        6 minutes ago       909MB
myaspnetcoreapp                     3.1                 59804bf835a3        37 minutes ago      212MB
mynginx                             latest              867008461f8e        About an hour ago   21.2MB
mynodejs                            latest              40a49ceabab7        About an hour ago   890MB
mynodejs                            v1                  40a49ceabab7        About an hour ago   890MB
mcr.microsoft.com/dotnet/core/aspnet 3.1                a843e0fbe833        2 weeks ago         207MB
nginx                               stable-alpine       aaad4724567b        2 months ago        21.2MB
node                                boron               ab290b853066        8 months ago        884MB
mcr.microsoft.com/mssql/server      2017-CU12-ubuntu    4095d6d460cd        14 months ago       1.32GB
```

> Note:Notice how **latest** and **v1** both exist. **V1** is technically newer, and **latest** just signifies the image that did not have a version/tag before and can feel misleading.
> Also, note the Image ID for both are identical. The image and its content / layers are all cached on your machine. The Image ID is content addressable, so the full content of it is hashed through a hashing algorithm and it spits out an ID.
> If the content of any two (or more) images are the same, then the Image ID will be the same, and only one copy of the actual layers are on your machine and pointed to by many different image names/tags.

Tagging New Container Image

Tagging a new image is done at the time when you build a container image. it's a straightforward process that requires you to simply add the **:tag** at the end of container image name.

1. Navigate to the directory **"labs/module1/nginx"** that contains the "**nginx**" files along with Dockerfile. You can use the command `cd ~/labs/module1/nginx`

2. Build a new image by running the command `docker build -t nginxsample:v1 .`

> Note:In this case you're creating a new image based on Dockerfile (covered in earlier exercise on NGINX).

```
root@super-Virtual-Machine:~/labs/module1/nginx# docker build -t nginxsample:v1 .
Sending build context to Docker daemon  3.072kB
Step 1/4 : FROM nginx:stable-alpine
 ---> aaad4724567b
Step 2/4 : LABEL author="sampleauthor@contoso.com"
 ---> Using cache
 ---> 480ffea12e7c
Step 3/4 : COPY index.html /usr/share/nginx/html/index.html
 ---> Using cache
 ---> 972311bb9adf
Step 4/4 : CMD ["nginx", "-g", "daemon off;"]
 ---> Using cache
 ---> 867008461f8e
Successfully built 867008461f8e
Successfully tagged nginxsample:v1
root@super-Virtual-Machine:~/labs/module1/nginx#
```

3. If you run a "**docker images**" command, it will list the new container image with tag "**v1**"

```
root@super-Virtual-Machine:~/labs/module1/nginx# docker images
REPOSITORY                                TAG                 IMAGE ID            CREATED             SIZE
mynodejsv2                                latest              4c89d4f34e70        10 minutes ago      909MB
myaspnetcoreapp                           3.1                 59804bf835a3        41 minutes ago      212MB
mynginx                                   latest              867008461f8e        About an hour ago   21.2MB
nginxsample                               v1                  867008461f8e        About an hour ago   21.2MB
mynodejs                                  latest              40a49ceabab7        About an hour ago   890MB
mynodejs                                  v1                  40a49ceabab7        About an hour ago   890MB
mcr.microsoft.com/dotnet/core/aspnet      3.1                 a843e0fbe833        2 weeks ago         207MB
nginx                                     stable-alpine       aaad4724567b        2 months ago        21.2MB
node                                      boron               ab290b853066        8 months ago        884MB
mcr.microsoft.com/mssql/server            2017-CU12-ubuntu    4095d6d460cd        14 months ago       1.32GB
```

Congratulations!

You have successfully completed this exercise. Click **Next** to advance to the next exercise.

Exercise 6: Building and Running SQL Server 2017 in a Container

---

Microsoft SQL Server is one of the most commonly used database server in the market today. Microsoft has made an investment to ensure that customers moving towards containers have an ability to leverage SQL Server through a container image.

Previously Microsoft had released container images for Microsoft SQL Server 2016 for both Linux and [Windows]((https://hub.docker.com/r/microsoft/mssql-server-windows-express) including Microsoft SQL Server Express Edition and Microsoft SQL Server (180 days public preview of Enterprise Edition). Currenlty, SQL Server 2017 is only available for Linux Containers and allow users to bring their own license key when starting the container https://hub.docker.com/_/microsoft-mssql-server
In this lab, you will work with Microsoft SQL Server 2017 container image to run a custom database that can store user related information. You will first learn how to associate relevant SQL Server database files to SQL Server container image and how to initialize the database with test data. Then, you will connect a Web Application packaged in another container to the database running inside the SQL Server container. In other words, you will end up with a web application running in a container talking to a database hosted in another container. This is very common scenario, so understanding how it works is important. Let's start by running a SQL Server Express container with the custom database.

Return to list of exercises - Return to list of modules

SQL Server 2017 Container Image

1. Make sure you have a Terminal opened, and that you are logged in as root. Also, change the current directory to **labs\module1\sqlserver2017** by using the command `cd ~/labs/module1/sqlserver2017`
2. Before proceeding further, let's remove all the containers from previous tasks. Run the command `docker rm $(docker ps -aq) -f`
3. Look at the **Dockerfile** describing how to package the database `cat Dockerfile`

```
FROM mcr.microsoft.com/mssql/server:2017-CU12-ubuntu

# Create db directory
RUN mkdir -p /usr/src/db
WORKDIR /usr/src/db

# Install ping that will be used to keep the container up
RUN apt-get update && apt-get install -y iputils-ping

COPY . .

# Grant permissions for the import-data and entrypoint scripts to be executable
RUN chmod +x /usr/src/db/import-data.sh
RUN chmod +x /usr/src/db/entrypoint.sh

CMD ./entrypoint.sh
```

From the Microsoft SQL Server image, we copy the local files to the container. These local files are composed of:

Users.csv - contains the test data

```
root@super-Virtual-Machine:~/labs/module1/sqlserver2017# cat Users.csv
1,Woody Allen
2,Theodore Roosevelt
3,Mark Twain
```

setup.sql - the SQL commands to create a database named **LabData** and the **Users** table

```
root@super-Virtual-Machine:~/labs/module1/sqlserver2017# cat setup.sql
CREATE DATABASE LabData;
GO

USE LabData;
GO

CREATE TABLE Users (ID int, UserName nvarchar(max));
```

entrypoint.sh - used as an entry point in the **Dockerfile**. It will start the database server and run **import-data.sh**

```
root@super-Virtual-Machine:~/labs/module1/sqlserver2017# cat entrypoint.sh
#start SQL Server, start the script to create the DB and import the data
/opt/mssql/bin/sqlservr & /usr/src/db/import-data.sh
```

import-data.sh - will wait for the server to start and will trigger the database creation, the data import and the ***ping*** command to keep the database alive. Feel free to look at each file we just described to have a better understanding of their role.

```
root@super-Virtual-Machine:~/labs/module1/sqlserver2017# cat import-data.sh
#wait for the SQL Server to come up
sleep 10

#run the setup script to create the DB and the schema in the DB
/opt/mssql-tools/bin/sqlcmd -S localhost -U sa -P P@ssw0rd123! -d master -i setup.sql

#import the data from the csv file
/opt/mssql-tools/bin/bcp LabData.dbo.Users in "/usr/src/db/Users.csv" -c -t',' -S localhost -U sa -P P@ssw0rd123!

ping localhost
```

4. Run the command to build our SQL Server container image `docker build -t mysqlserver .`

```
root@super-Virtual-Machine:~/labs/module1/sqlserver2017# docker build -t mysqlserver .
Sending build context to Docker daemon  9.216kB
Step 1/8 : FROM mcr.microsoft.com/mssql/server:2017-CU12-ubuntu
 ---> 4095d6d460cd
Step 2/8 : RUN mkdir -p /usr/src/db
 ---> Running in 6fa5d0df6ff2
Removing intermediate container 6fa5d0df6ff2
 ---> a2e18c12de2d
Step 3/8 : WORKDIR /usr/src/db
Removing intermediate container 1561bdd14017
 ---> 8ead0fc12808
Step 4/8 : RUN apt-get update && apt-get install -y iputils-ping
 ---> Running in e79fad55c291
Get:1 http://security.ubuntu.com/ubuntu xenial-security InRelease [109 kB]
Get:2 http://archive.ubuntu.com/ubuntu xenial InRelease [247 kB]
Get:3 https://packages.microsoft.com/ubuntu/16.04/prod xenial InRelease [3226 B]
```

5. Once built, run the start your with the following command (note that we explicitly name our container) `docker run -e ACCEPT_EULA=Y -e SA_PASSWORD=P@ssw0rd123! -d -p 1433:1433 --name mydb mysqlserver`

> Note: The docker run command has various parameters. The following table provides a description for parameters specific to SQL server. See the the docker hub for an exhaustive list of parameters.

Parameter	Description
SA_PASSWORD	The system administrator (userid = 'sa') password used to connect to SQL Server once the container is running. The password in this case is provided in plain text for brevity. However, best practice is to use secrets in Docker: https://docs.docker.com/engine/reference/commandline/secret
ACCEPT_EULA	Confirms acceptance of the end user licensing agreement found here.
-e	Flag that is used to pass environment variables to the container. In this particular case password and license eula are passed as environment variable.

```
root@super-Virtual-Machine:~/labs/module1/sqlserver2017# docker run -e ACCEPT_EULA=Y -e SA_PASSWORD=P@ssw0rd123! -d -p
1433:1433 --name mydb mysqlserver
75c97a2f657dc410828ad72dad81f0975fe095b3d5d6704f1bff58d31a9d5cce
```

6. You can follow the database initialization with the command ***docker logs mydb -f*** until you see the ***ping*** command starting. Once it started, you can interrupt ***docker logs*** by hitting ***CTRL + C***

```
root@super-Virtual-Machine:~/labs/module1/sqlserver2017# docker logs mydb -f
2019-04-19 19:59:12.30 Server      Setup step is copying system data file 'C:\templatedata\master.mdf' to '/var/opt/mss
ql/data/master.mdf'.
2019-04-19 19:59:12.38 Server      Did not find an existing master data file /var/opt/mssql/data/master.mdf, copying th
e missing default master and other system database files. If you have moved the database location, but not moved the da
tabase files, startup may fail. To repair: shutdown SQL Server, move the master database to configured location, and re
start.
2019-04-19 19:59:12.39 Server      Setup step is copying system data file 'C:\templatedata\mastlog.ldf' to '/var/opt/ms
sql/data/mastlog.ldf'.
2019-04-19 19:59:12.40 Server      Setup step is copying system data file 'C:\templatedata\model.mdf' to '/var/opt/mssq
l/data/model.mdf'.
2019-04-19 19:59:12.41 Server      Setup step is copying system data file 'C:\templatedata\modellog.ldf' to '/var/opt/m
ssql/data/modellog.ldf'.
2019-04-19 19:59:12.42 Server      Setup step is copying system data file 'C:\templatedata\msdbdata.mdf' to '/var/opt/m
ssql/data/msdbdata.mdf'.
2019-04-19 19:59:12.44 Server      Setup step is copying system data file 'C:\templatedata\msdblog.ldf' to '/var/opt/ms
sql/data/msdblog.ldf'.
2019-04-19 19:59:12.56 Server      Microsoft SQL Server 2017 (RTM-CU12) (KB4464082) - 14.0.3045.24 (X64)
        Oct 18 2018 23:11:05
        Copyright (C) 2017 Microsoft Corporation
        Developer Edition (64-bit) on Linux (Ubuntu 16.04.5 LTS)
```

...

```
2019-04-19 19:59:16.64 spid20s     The default language (LCID 0) has been set for engine and full-text services.
2019-04-19 19:59:19.40 spid51      Starting up database 'LabData'.
2019-04-19 19:59:19.64 spid51      Parallel redo is started for database 'LabData' with worker pool size [2].
2019-04-19 19:59:19.65 spid51      Parallel redo is shutdown for database 'LabData' with worker pool size [2].
Changed database context to 'LabData'.

Starting copy...

3 rows copied.
Network packet size (bytes): 4096
Clock Time (ms.) Total     : 24     Average : (125.0 rows per sec.)
PING localhost (127.0.0.1) 56(84) bytes of data.
64 bytes from localhost (127.0.0.1): icmp_seq=1 ttl=64 time=0.024 ms
64 bytes from localhost (127.0.0.1): icmp_seq=2 ttl=64 time=0.040 ms
64 bytes from localhost (127.0.0.1): icmp_seq=3 ttl=64 time=0.039 ms
64 bytes from localhost (127.0.0.1): icmp_seq=4 ttl=64 time=0.041 ms
64 bytes from localhost (127.0.0.1): icmp_seq=5 ttl=64 time=0.041 ms
64 bytes from localhost (127.0.0.1): icmp_seq=6 ttl=64 time=0.042 ms
64 bytes from localhost (127.0.0.1): icmp_seq=7 ttl=64 time=0.042 ms
64 bytes from localhost (127.0.0.1): icmp_seq=8 ttl=64 time=0.037 ms
64 bytes from localhost (127.0.0.1): icmp_seq=9 ttl=64 time=0.039 ms
64 bytes from localhost (127.0.0.1): icmp_seq=10 ttl=64 time=0.044 ms
64 bytes from localhost (127.0.0.1): icmp_seq=11 ttl=64 time=0.038 ms
64 bytes from localhost (127.0.0.1): icmp_seq=12 ttl=64 time=0.039 ms
64 bytes from localhost (127.0.0.1): icmp_seq=13 ttl=64 time=0.037 ms
64 bytes from localhost (127.0.0.1): icmp_seq=14 ttl=64 time=0.042 ms
^C
root@super-Virtual-Machine:~/labs/module1/sqlserver2017#
```

> Note: **LabData** is listed as a new database and that we imported 3 rows (coming from **Users.csv**).

7. Run the following command to open an interactive session within the database container with **sqlcmd**. **Sqlcmd** is a basic command-line utility provided by Microsoft https://docs.microsoft.com/enus/sql/relationaldatabases/scripting/sqlcmd-use-the-utility for ad hoc, interactive execution of TransactSQL statements and scripts `docker exec -it mydb /opt/mssql-tools/bin/sqlcmd -S localhost -U sa -P P@ssw0rd123!`

```
root@super-Virtual-Machine:~/labs/module1/sqlserver2017# docker exec -it mydb /opt/mssql-tools/bin/sqlcmd -S localhost
-U sa -P P@ssw0rd123!
1>
```

8. Let's begin by listing down all the databases available by running the command

```
SELECT name FROM master.dbo.sysdatabases
GO
```

```
![](content/mod1image75.png)
```

> Note: **LabData** is listed at the very bottom.

1. Now let's check that the users we had in **Users.csv** have been properly ingested into the database at initialization

```
USE LabData
SELECT * FROM Users
GO
```

```
![](content/mod1image76.png)
```

1. Now let's exit the **sqlcmd** session. Note that the database will still be running in the background `exit`

Connect Application to the SQL Server Container

1. Now we will connect an ASP .NET Core Application to our SQL Server and show that it can see the data stored in the database. Change the current directory to **labs/module1/aspnetcorewithsqlserver** by using the command `cd ~/labs/module1/aspnetcorewithsqlserver`

2. First, we need to know what is the IP address of the container running the SQL Server so that the front end web application. Run the following command and note down the IP Address `docker inspect mydb`

```
root@super-Virtual-Machine:~/labs/module1/sqlserver2017# cd ~/labs/module1/aspnetcorewithsqlserver/
root@super-Virtual-Machine:~/labs/module1/aspnetcorewithsqlserver# docker inspect mydb
[
    {
        "Id": "057a628c6ea764681ef297c4ab381f0c578acdc67af07da81b3115166041370d",
        "Created": "2020-01-13T23:16:07.251181711Z",
        "Path": "/bin/sh",
        "Args": [
            "-c",
            "./entrypoint.sh"
        ],
```

...

```
"Networks": {
    "bridge": {
        "IPAMConfig": null,
        "Links": null,
        "Aliases": null,
        "NetworkID": "d68485cbf5b6ccfa0e1add646b3a64555fdaf8a2bdd03ee68e94f9e9a5eb1efd",
        "EndpointID": "afab1f8bf3ea6de996774afe985fed72908009e787a3966526c799ff571dc0ed",
        "Gateway": "172.17.0.1",
        "IPAddress": "172.17.0.2",
        "IPPrefixLen": 16,
        "IPv6Gateway": "",
        "GlobalIPv6Address": "",
        "GlobalIPv6PrefixLen": 0,
        "MacAddress": "02:42:ac:11:00:02",
        "DriverOpts": null
    }
}
```

3. Now we need to update the connection string used by the web app to connect to the SQL Server back end. Open the **Startup.cs** and replace **localhost** in the connection string by the IP address we just copied. Once finish making changes press **CTRL + X** and then press **Y** when asked for confirmation to retain your changes. Finally, you will be asked for file name to write. For that press **Enter** (without changing the name of the file). This will close the nano text editor. `nano Startup.cs`

```
 GNU nano 2.5.3                File: Startup.cs                Modified

using CoreAppWithSQLServer.Data;
using Microsoft.AspNetCore.Builder;
using Microsoft.AspNetCore.Hosting;
using Microsoft.Extensions.Hosting;
using Microsoft.Extensions.Configuration;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.EntityFrameworkCore;
namespace CoreAppWithSQLServer
{
    public class Startup
    {
        public Startup(IConfiguration configuration)
        {
            Configuration = configuration;
        }

        public IConfiguration Configuration { get; }

        // This method gets called by the runtime. Use this method to add services to the container.
        public void ConfigureServices(IServiceCollection services)
        {
            services.AddControllersWithViews();

            services.AddDbContext<DatabaseContext>(options =>
                options.UseSqlServer("Data Source=172.17.0.2,1433;Initial Catalog=LabData;User ID=sa;Password=P@ssw0rd
        }
```

4. We are now ready to build the ASP .NET Core web application. Run `dotnet build` to build it.

```
root@super-Virtual-Machine:~/labs/module1/aspnetcorewithsqlserver# dotnet build
Microsoft (R) Build Engine version 16.4.0+e901037fe for .NET Core
Copyright (C) Microsoft Corporation. All rights reserved.

  Restore completed in 5.25 sec for /root/labs/module1/aspnetcorewithsqlserver/CoreAppWithSQLServer.csproj.
  CoreAppWithSQLServer -> /root/labs/module1/aspnetcorewithsqlserver/bin/Debug/netcoreapp3.1/CoreAppWithSQLServer.dll
  CoreAppWithSQLServer -> /root/labs/module1/aspnetcorewithsqlserver/bin/Debug/netcoreapp3.1/CoreAppWithSQLServer.Views
.dll

Build succeeded.
    0 Warning(s)
    0 Error(s)

Time Elapsed 00:00:11.97
```

5. Then publish the artifacts in a **published** folder with `dotnet publish -o published`

```
root@super-Virtual-Machine:~/labs/module1/aspnetcorewithsqlserver# dotnet publish -o published
Microsoft (R) Build Engine version 16.4.0+e901037fe for .NET Core
Copyright (C) Microsoft Corporation. All rights reserved.

  Restore completed in 51.14 ms for /root/labs/module1/aspnetcorewithsqlserver/CoreAppWithSQLServer.csproj.
  CoreAppWithSQLServer -> /root/labs/module1/aspnetcorewithsqlserver/bin/Debug/netcoreapp3.1/CoreAppWithSQLServer.dll
  CoreAppWithSQLServer -> /root/labs/module1/aspnetcorewithsqlserver/bin/Debug/netcoreapp3.1/CoreAppWithSQLServer.Views
.dll
  CoreAppWithSQLServer -> /root/labs/module1/aspnetcorewithsqlserver/published/
```

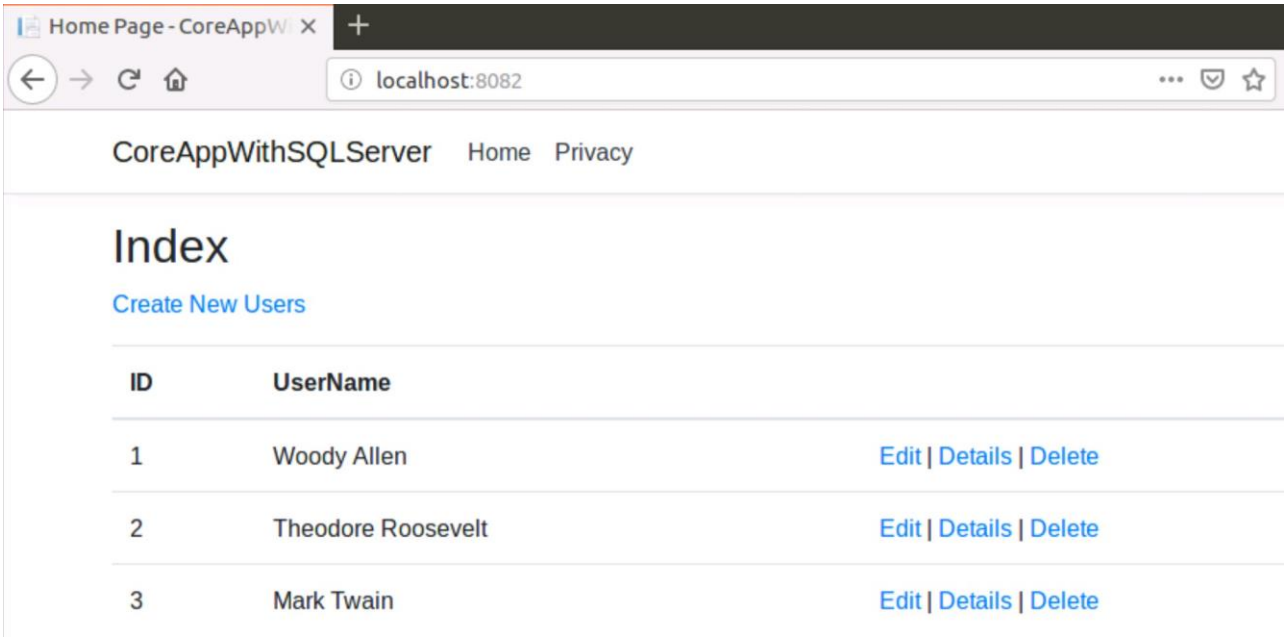
6. Now, we are ready to build our container that we will tag with an explicit **withsql** string `docker build -t myaspcoreapp:3.1-withsql .`

```
root@super-Virtual-Machine:~/labs/module1/aspnetcorewithsqlserver# docker build -t myaspcoreapp:3.1-withsql .
Sending build context to Docker daemon  26.06MB
Step 1/4 : FROM mcr.microsoft.com/dotnet/core/aspnet:3.1
 ---> a843e0fbe833
Step 2/4 : WORKDIR /app
 ---> Using cache
 ---> b6bf139c497f
Step 3/4 : COPY published ./
 ---> 1462fba5d228
Step 4/4 : ENTRYPOINT ["dotnet", "CoreAppWithSQLServer.dll"]
 ---> Running in 1580a45867ed
Removing intermediate container 1580a45867ed
 ---> 0f2cc984726b
Successfully built 0f2cc984726b
Successfully tagged myaspcoreapp:3.1-withsql
```

7. Finally, run the container and expose the web app on port **8082** `docker run -d -p 8082:80 myaspcoreapp:3.1-withsql`

```
root@super-Virtual-Machine:~/labs/module1/aspnetcorewithsqlserver# docker run -d -p 8082:80 myaspcoreapp:3.1-withsql
85c1e08449695ed5da78f04e5373398884cdbfda5b4251eb6640d1488816ced6
```

8. Open a browser and naviguate to **localhost:8082**. The web app should display the list of users that we ingested in the database.



9. Click on **Create New Users** and add a new user name.

> Note: You could remove the front end container and re-run it, you would still see the new user that you have created since it is stored in the SQL Server container.

10. We have reached the end of the lab, let's remove all the containers to leave the environment in a clean state. Run the command `docker rm $(docker ps -aq) -f`

Congratulations!

You have successfully completed this module. Click **Next** to advance to the next module.

Module 5 - Working with Kubernetes MiniKube

Duration

45 minutes

Module 5: Table of Contents

Exercise 1: Setup and Configure MiniKube on a Virtual Machine

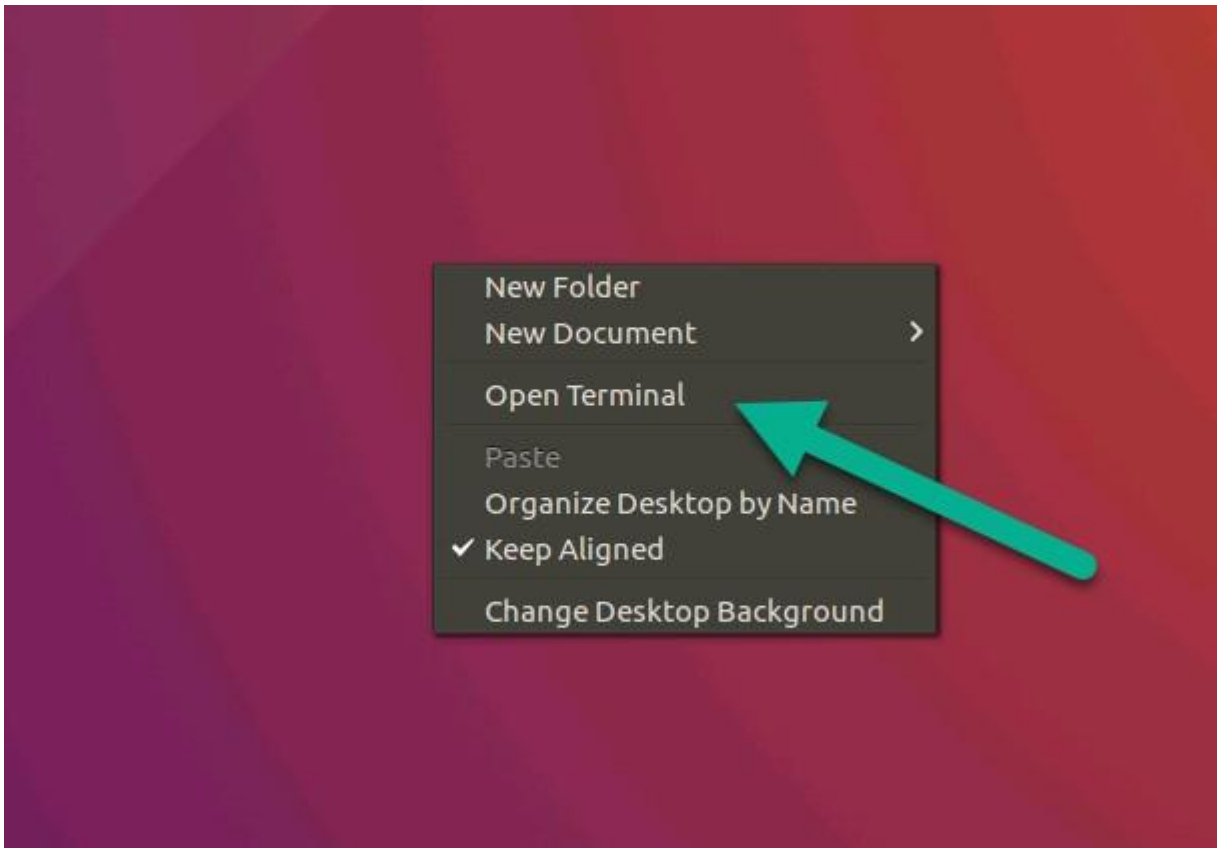
In this exercise you will setup and configure MiniKube on an Ubuntu VM. MiniKube is a tool that makes it easy to run Kubernetes locally and runs a single-node Kubernetes cluster.

> Note:Note that the virtual machine you are using supports nested virtualization. If you want to use your enterprise or MyVisualStudio (MSDN) account in the future to provision a VM, you must use a Standard_D2s_v3 or higher on Azure. The Azure Passes given during this workshop do not provide nested virtualization VMs in Azure.

Return to list of exercises - Return to list of modules

Install Minikube

1. Sign into your LOD Ubuntu VM.
2. Open a **terminal** by right clicking anywhere on the Desktop.

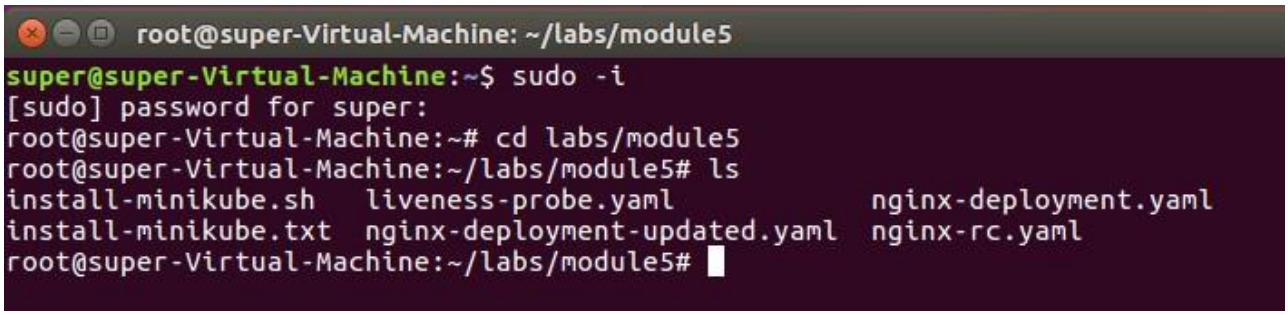


3. Go into root and type in **@lab.VirtualMachine(UbuntuBase).Password** for the password
   `sudo -i`

4. Run the following commands:

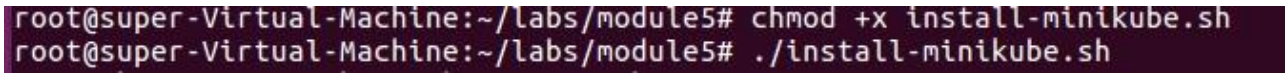
```
rm /var/lib/apt/lists/lock

rm /var/cache/apt/archives/lock

rm /var/lib/dpkg/lock
```

5. Navigate to the lab files by running: `cd labs/module5`



6. You are now to install Minikube. Run the following two commands (the first command gives execute permission to your script, the second script runs it):

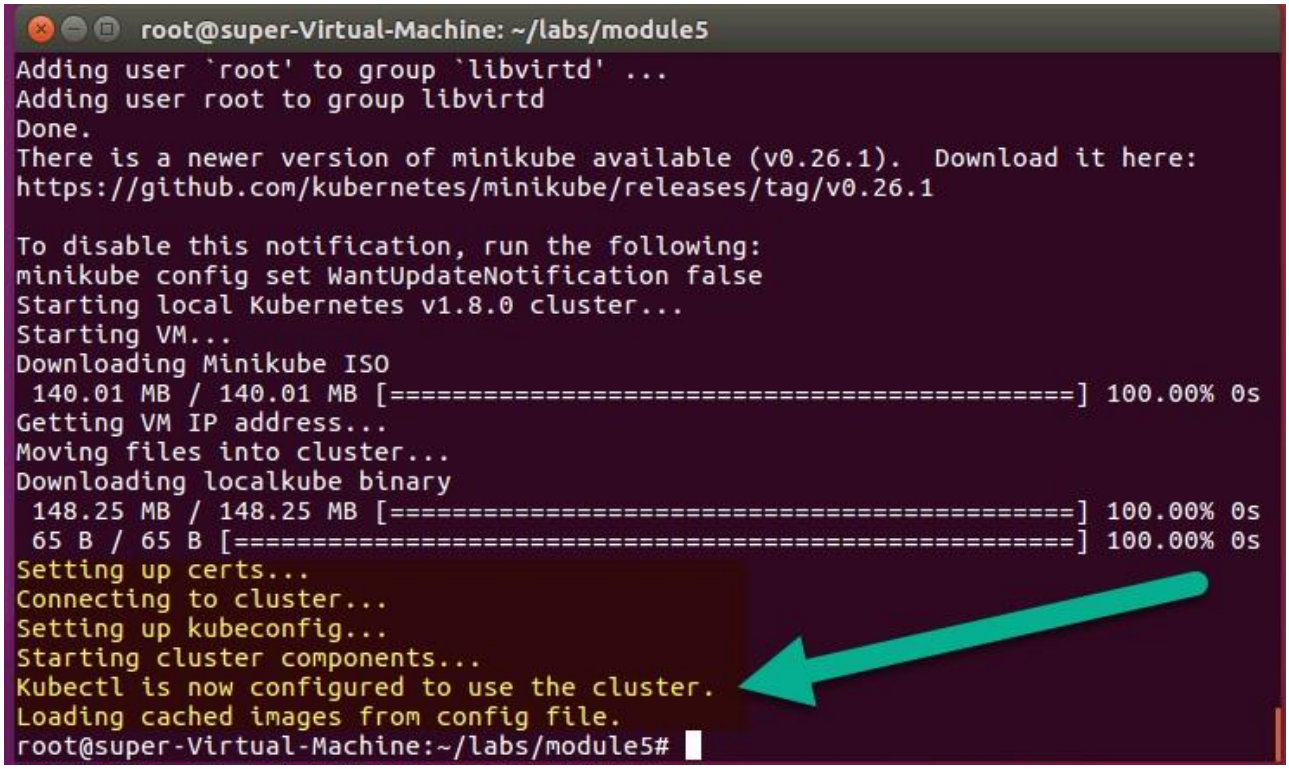
```
chmod +x install-minikube.sh
./install-minikube.sh
```



7. Read over the contents of the bash file you just ran while it is installing (the install will take at ~15 minutes. Feel free to start the Module 5 - Orchestrator Lab on the Windows VM in the meantime):

```
sudo -i

sudo apt-get -y update
sudo apt-get -y upgrade

#Install kubectl (latest)
curl -LO https://storage.googleapis.com/kubernetes-release/release/$(curl -s https://storage.googleapis.com/kubernetes-release/release/stable.txt)/bin/linux/amd64/kubectl

#Make the kubectl binary executable.
chmod +x ./kubectl

#Move the binary in to your PATH.
sudo mv ./kubectl /usr/local/bin/kubectl

#Install minikube. Make sure to check for latest version (current version is 0.24.1)
curl -Lo minikube https://storage.googleapis.com/minikube/releases/v0.24.1/minikube-linux-amd64 && chmod +x minikube && sudo mv minikube /usr/local/bin/

#Install kvm2
curl -LO https://storage.googleapis.com/minikube/releases/latest/docker-machine-driver-kvm2 && chmod +x docker-machine-driver-kvm2 && sudo mv docker-machine-driver-kvm2 /usr/bin/

#Install Install libvirt and qemu-kvm and libvirt-bin
sudo apt install -y qemu-kvm libvirt-bin

#Add group libvirtd
sudo addgroup libvirtd
#Add current user to libvirtd group
sudo adduser $USER libvirtd

#Run minikube
minikube start --vm-driver kvm2
```

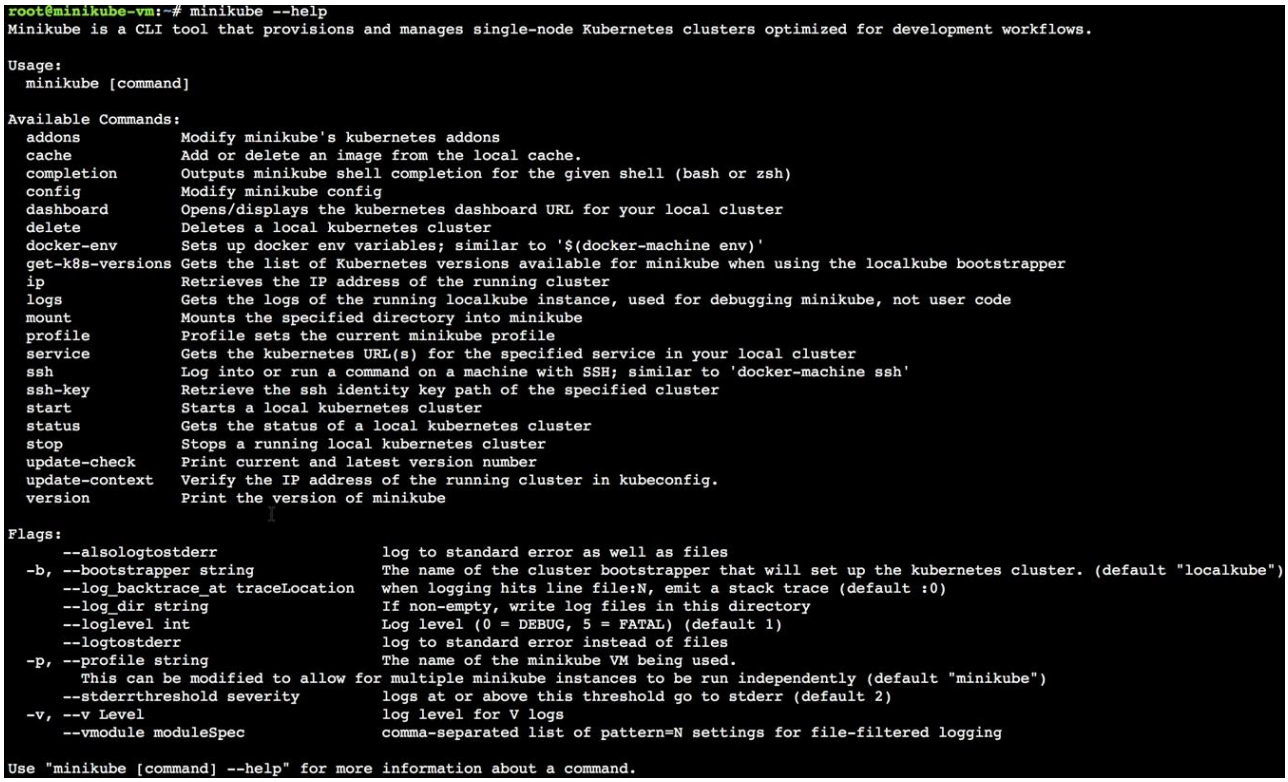
> Note: After minikube installation is completed, the last command in the text file will start the minikube with a single node Kubernetes cluster.



8. You can always find out more about minikube commands using the help switch, try running it now

> Note:Do not worry about the screen color in the screenshots being different from your Ubuntu VM!):

```
minikube --help
```



You are now going to create a simple deployment based on ***nginx*** container image and expose it using a service.

9. Create a deployment and name it ***nginx***

```
kubectl run nginx --image=nginx --port=80
```

10. Expose the deployment using a service `kubectl expose deployment nginx --type=NodePort`

11. Check that your deployment as well as your service are ready `kubectl get deployment`

```
root@super-Virtual-Machine:~/labs/module5# kubectl get deployment
NAME    DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
nginx   1         1         1            1           28m
```

`kubectl get service`

```
root@super-Virtual-Machine:~/labs/module5# kubectl get service
NAME         TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)        AGE
kubernetes   ClusterIP   10.96.0.1       <none>        443/TCP        42m
nginx        NodePort    10.100.106.12   <none>        80:30891/TCP   28m
```

12. Access the nginx default web page using the curl command. `curl $(minikube service nginx --url)`

```
root@minikube-vm:~# curl $(minikube service nginx --url)
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
    body {
        width: 35em;
        margin: 0 auto;
        font-family: Tahoma, Verdana, Arial, sans-serif;
    }
</style>
</head>
<body>
<h1>Welcome to nginx!</h1>
<p>If you see this page, the nginx web server is successfully installed and
working. Further configuration is required.</p>

<p>For online documentation and support please refer to
<a href="http://nginx.org/">nginx.org</a>.<br/>
Commercial support is available at
<a href="http://nginx.com/">nginx.com</a>.</p>

<p><em>Thank you for using nginx.</em></p>
</body>
</html>
```

13. Clean up the deployment and service with the following commands
`kubectl delete service nginx`
`kubectl delete deployment nginx`

Congratulations!

You have successfully completed this exercise. Click **Next** to advance to the next exercise.

Exercise 2: Working with Replica Sets, Deployments and Health Probes

Return to list of exercises - Return to list of modules

Working with the health probe

In this task you will create a new pod and enable a health probe. To test the probe, pod will run a single container that is going to explicitly fail the health probe request after every 5 probes.

1. You should still be in the **/labs/module5** folder, if not, navigate to there.

2. Create the pod using the yaml file:
   `kubectl apply -f liveness-probe.yaml`

3. Check the status of the newly created pod. It may take few seconds for the container to be up and running. `kubectl get pods`

   > Note: The ***STATUS*** column shows Running and ***RESTARTS*** column have the value zero. That's expected because container is just started, and the health probe has not failed yet.

```
NAME             READY     STATUS    RESTARTS   AGE
liveness-probe   1/1       Running   0          3s
```

4. After 3-4 minutes if you view the status of pods again you should see the RESTARTS column with the value 1 (or higher depending on how long you have waited to check the status of the pod)

```
NAME             READY     STATUS    RESTARTS   AGE
liveness-probe   1/1       Running   1          2m
```

   > Note: If you wait for few more minutes and check the status of pod again you should see the value of RESTARTS column changes to a higher number.

5. Behind the scenes, every time a container fails the health probe it will be restarted again. To get bit more information about the failing health probe run the following command: `kubectl describe po liveness-probe`

   > Note: This describes the pod in detail along with the events that are happening including the failed health probes

```
Events:
  Type     Reason                 Age                From               Message
  ----     ------                 ----               ----               -------
  Normal   Scheduled              5m                 default-scheduler  Successfully assigned liveness-probe to minikube
  Normal   SuccessfulMountVolume  5m                 kubelet, minikube  MountVolume.SetUp succeeded for volume "default-token-nmg4m"
  Normal   Pulled                 1m (x3 over 5m)    kubelet, minikube  Successfully pulled image "rbinrais/healthcheck"
  Normal   Created                1m (x3 over 5m)    kubelet, minikube  Created container
  Normal   Started                1m (x3 over 5m)    kubelet, minikube  Started container
  Warning  Unhealthy              32s (x9 over 4m)   kubelet, minikube  Liveness probe failed: HTTP probe failed with statuscode: 500
  Normal   Pulling                2s (x4 over 5m)    kubelet, minikube  pulling image "rbinrais/healthcheck"
  Normal   Killing                2s (x3 over 3m)    kubelet, minikube  Killing container with id docker://liveness-probe:Container failed
```

6. Eventually after failing the health probes multiple times in a short interval container will be put under ***CrashLoopBackOff*** status.

```
NAME             READY     STATUS             RESTARTS   AGE
liveness-probe   0/1       CrashLoopBackOff   6          12m
```

7. You can view the logs from the container that is terminated by using the command: `kubectl logs liveness-probe --previous`

> Note: The sample docker container application is basic so very limited information is available in logs but typically for production ready applications its recommended to write more detailed messages to the logs.

8. Finally, remove the pod
   `kubectl delete pod liveness-probe`

Return to list of exercises - Return to list of modules

Working with Replica Set

---

1. In this task you will first create a replica with predefined labels assigned to pods. Later you will change the labels for a pod and observe the behavior of replica set.

> Knowledge: A **Replica Set** ensures how many replicas of a pod should be running. It can be considered as a replacement of replication controller. The key difference between the replica set and the replication controller is, the replication controller only supports equality-based selector whereas the replica set supports set-based selector.
>
> Note: The ***nginx-rc.yaml*** file is available inside the ***/labs/module5*** subfolder and contains definition of replica set. If you review the content of file you will notice that it will maintain 3 pods with each running nginx container. Pods are also labeled ***app=webapp***.

   To create the replica set and pods run the following command in the **labs/module5** directory.

   ```
   kubectl create -f nginx-rc.yaml
   ```

2. Let's look at the pods along with their labels. `kubectl get pods --show-labels`

```
NAME                  READY     STATUS    RESTARTS   AGE       LABELS
nginx-replica-2ggkk   1/1       Running   0          2m        app=webapp
nginx-replica-kqdmn   1/1       Running   0          2m        app=webapp
nginx-replica-s5r8j   1/1       Running   0          2m        app=webapp
```

   You can also list all the replica sets that are available by using the command:
   `kubectl get replicaset`

```
NAME            DESIRED   CURRENT   READY     AGE
nginx-replica   3         3         3         36m
```

> Note:Notice we have three pods running. If you delete one of them, replica set will ensure that total pods count remain three and it will do that by creating a new pod.

3. First delete one of the pods

> Note: Get name from this command: `kubectl get pods \--show-labels`

   ```
   kubectl delete pod <POD_NAME>
   ```

4. Now, check the pods again. Notice you still have three pods running and one of them is terminating.

   ```
   kubectl get pods --show-labels
   ```

   ```
   NAME                    READY     STATUS        RESTARTS   AGE       LABELS
   nginx-replica-2ggkk     0/1       Terminating   0          10m       app=webapp
   nginx-replica-f7xsm     1/1       Running       0          3s        app=webapp
   nginx-replica-kqdmn     1/1       Running       0          10m       app=webapp
   nginx-replica-s5r8j     1/1       Running       0          10m       app=webapp
   ```

5. Another factor that plays an important role in determining pods relationship with replica set is the labels. Currently ***app=webapp*** is the selector used by replica set to determine the pods under its watch. If you change the label of a pod from ***app=webapp*** to say ***app=debugging*** then replica set will effectively remove it from its watch and create another pod with the label ***app=webapp***. For replica set its job is to maintain the total count of pods to three as per the definition provided in the yaml file.

   ```
   kubectl label pod <POD_NAME> app=debugging --overwrite=true
   ```

6. View the pods again and notice that there are four pods running. Replica set created an additional pod immediately after it noticed pod count was less than three. `kubectl get pods --show-labels`

   ```
   NAME                    READY     STATUS    RESTARTS   AGE       LABELS
   nginx-replica-f7xsm     1/1       Running   0          16m       app=debugging
   nginx-replica-kqdmn     1/1       Running   0          27m       app=webapp
   nginx-replica-s5r8j     1/1       Running   0          27m       app=webapp
   nginx-replica-v4vcb     1/1       Running   0          1m        app=webapp
   ```

7. Replica set is essentially using selector (defined in the yaml file) to which pods to observe. In this case its label ***app*** matching value ***webapp***. You can also get all the pods with ***app=webapp*** label using the following command.

   ```
   kubectl get pods --show-labels -l app=webapp
   ```

8. Finally remove the replica set using the following command.

   ```
   kubectl delete replicaset nginx-replica
   ```

9. As part of the deletion process replica set will remove all the pods that it had created. You can see that by listing the pods and looking at the ***STATUS*** column which shows ***Terminating***. `kubectl get pods`

   ```
   NAME                    READY     STATUS        RESTARTS   AGE
   nginx-replica-f7xsm     1/1       Running       0          29m
   nginx-replica-kqdmn     0/1       Terminating   0          40m
   nginx-replica-s5r8j     0/1       Terminating   0          40m
   nginx-replica-v4vcb     0/1       Terminating   0          14m
   ```

   Eventually pods will be removed. However, if you list the pods again the pod with label ***app=debugging*** is still ***Running***. `kubectl get pods --show-labels`

   ```
   NAME                    READY     STATUS    RESTARTS   AGE       LABELS
   nginx-replica-f7xsm     1/1       Running   0          34m       app=debugging
   ```

   Since you have change the label this pod is no longer manage by the replica set. In cases like these you can bulk remove pods based on labels.

```
kubectl delete pods -l app=debugging
```

Return to list of exercises - Return to list of modules

Working with Deployments

1. In this task you will begin by performing a deployment based on specific version of the nginx container image (v 1.7.9). Later you will leverage a RollingUpdate strategy for deployment to update pods running nginx container image from v1.7.9 to container image 1.8.

   > Note: The **nginx-deployment.yaml** file is available inside the **/labs/module5** subfolder and contains definition of deployment. If you review the content of file you will notice that it will maintain 2 pods with each running nginx container image v1.7.9. Pods are also labeled **app=nginx**.

   Run the following command from the **labs/module5** directory:

   ```
   kubectl create -f nginx-deployment.yaml
   ```

2. Notice the deployment status by running the command: `kubectl get deployment`

   ```
   NAME               DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
   nginx-deployment   2         2         2            2           3m
   ```

3. If you list the pods you should see the out similar to following:

   ```
   kubectl get pods --show-labels
   ```

   ```
   NAME                                READY     STATUS    RESTARTS   AGE       LABELS
   nginx-deployment-6d8f46cfb7-qhnpz   1/1       Running   0          5m        app=nginx,pod-template-hash=2849027963
   nginx-deployment-6d8f46cfb7-vb45t   1/1       Running   0          5m        app=nginx,pod-template-hash=2849027963
   ```

   > Note: Notice the **LABELS** column and presence of **pod-template-hash** label. This label is used by the deployment during the update process.

4. You are now going to update the deployment. You are going to update nginx container image from **v1.7.9** to **v1.8**. Before you do that first check the existing definition of the deployment:

   ```
   kubectl describe deployment nginx-deployment
   ```

```
Name:                   nginx-deployment
Namespace:              default
CreationTimestamp:      Sat, 20 Jan 2018 17:13:59 +0000
Labels:                 app=nginx
Annotations:            deployment.kubernetes.io/revision=1
Selector:               app=nginx
Replicas:               2 desired | 2 updated | 2 total | 2 available | 0 unavailable
StrategyType:           RollingUpdate
MinReadySeconds:        0
RollingUpdateStrategy:  25% max unavailable, 25% max surge
Pod Template:
  Labels:  app=nginx
  Containers:
   nginx:
    Image:        nginx:1.7.9
    Port:         80/TCP
    Environment:  <none>
    Mounts:       <none>
  Volumes:        <none>
Conditions:
  Type           Status  Reason
  ----           ------  ------
  Available      True    MinimumReplicasAvailable
  Progressing    True    NewReplicaSetAvailable
```

> Note: Notice the line **Image: nginx:1.7.9** which confirms that the current deployment is using **1.7.9** version of **nginx** image.

5. Perform the update using the command below.

```
kubectl apply -f nginx-deployment-updated.yaml
```

> Note: If you review the content of **nginx-deployment-updated.yaml** file and compare it with original **nginx-deployment.yaml** the only difference is the image tag which is changed from **1.7.9** to **1.8**.

6. If you immediately (after step 5) run the command to list all the pods you should see output like following:

```
kubectl get pods --show-labels
```

```
NAME                                READY     STATUS        RESTARTS   AGE       LABELS
nginx-deployment-6d8f46cfb7-qhnpz   0/1       Terminating   0          14m       app=nginx,pod-template-hash=2849027963
nginx-deployment-6d8f46cfb7-vb45t   0/1       Terminating   0          14m       app=nginx,pod-template-hash=2849027963
nginx-deployment-784794c74c-h7sv6   1/1       Running       0          7s        app=nginx,pod-template-hash=3403507307
nginx-deployment-784794c74c-pbsqm   1/1       Running       0          9s        app=nginx,pod-template-hash=3403507307
```

> Note: Notice that the deployment strategy of rolling update ensures that the old pods (nginx **v1.7.9**) are terminated only after new pods (nginx image **v1.8**) are in a running state. Also notice that the label **pod-template-hash** values are different for old and new pods. This is because the pod definition (due to change of image tag) is not same for both deployments.

7. You can also look at the new deployment details and make sure that correct nginx image (**v1.8**) is used.

```
kubectl describe deployment nginx-deployment
```

Congratulations!

You have successfully completed this lab. To mark the lab as complete, click **End**.