+ -
Helvetica Ne...
▼

Step 3 of 6

Introduction to Containers, Docker, and IBM Cloud Container Registry

Objectives

In this lab, you will:

- Pull an image from Docker Hub
- Run an image as a container using `docker`
- Build an image using a Dockerfile
- Push an image to IBM Cloud Container Registry

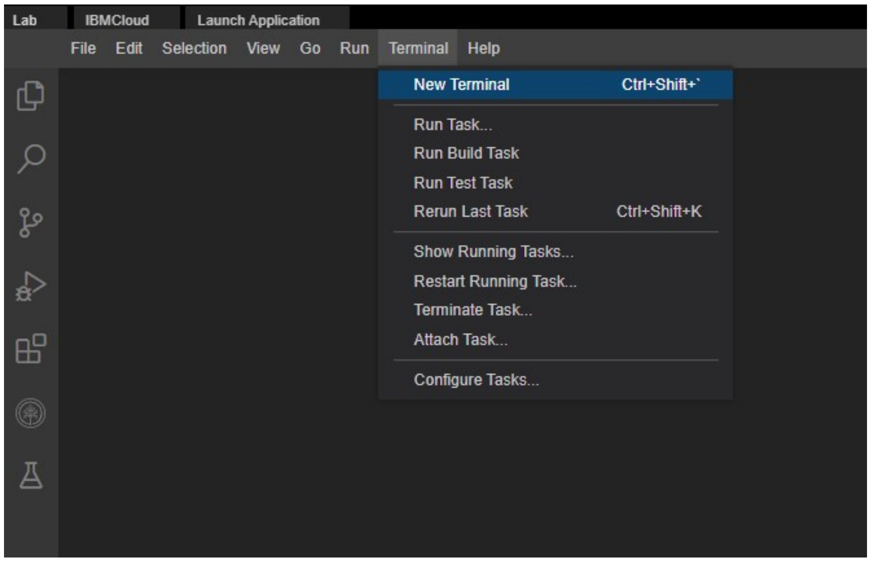
> **Note: Kindly complete the lab in a single session without any break because the lab may go on offline mode and may cause errors. If you face any issues/errors during the lab process, please logout from the lab environment. Then clear your system cache and cookies and try to complete the lab.**

Important: You may already have an IBM Cloud account and may even have a namespace in the IBM Container Registry (ICR). However, in this lab **you will not be using your own IBM Cloud account or your own ICR namespace**. You will be using an IBM Cloud account that has been automatically generated for you for this excercise. The lab environment will *not* have access to any resources within your personal IBM Cloud account, including ICR namespaces and images.

Verify the environment and command line tools

1. Open a terminal window by using the menu in the editor: `Terminal > New Terminal`.

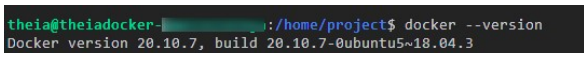
> **Note:If the terminal is already opened, please skip this step.**



2. Verify that `docker` CLI is installed.

```
docker --version
```

You should see the following output, although the version may be different:

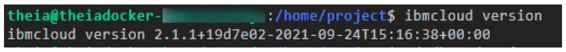


```
theia@theiadocker-[redacted]:/home/project$ docker --version
Docker version 20.10.7, build 20.10.7-0ubuntu5~18.04.3
```

3. Verify that `ibmcloud` CLI is installed.

```
ibmcloud version
```

You should see the following output, although the version may be different:



```
theia@theiadocker-[redacted]:/home/project$ ibmcloud version
ibmcloud version 2.1.1+19d7e02-2021-09-24T15:16:38+00:00
```

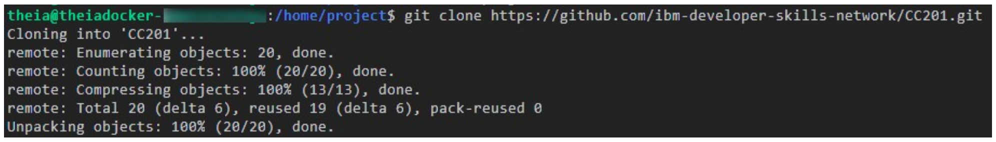
4. Change to your project folder.

Note: If you are already on the '/home/project' folder, please skip this step.

```
cd /home/project
```

5. Clone the git repository that contains the artifacts needed for this lab, if it doesn't already exist.

```
[ ! -d 'CC201' ] && git clone https://github.com/ibm-developer-skills-network/CC201.git
```



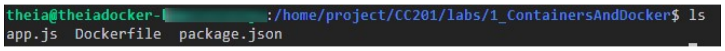
```
theia@theiadocker-[redacted]:/home/project$ git clone https://github.com/ibm-developer-skills-network/CC201.git
Cloning into 'CC201'...
remote: Enumerating objects: 20, done.
remote: Counting objects: 100% (20/20), done.
remote: Compressing objects: 100% (13/13), done.
remote: Total 20 (delta 6), reused 19 (delta 6), pack-reused 0
Unpacking objects: 100% (20/20), done.
```

6. Change to the directory for this lab.

```
cd CC201/labs/1_ContainersAndDocker/
```

7. List the contents of this directory to see the artifacts for this lab.

```
ls
```



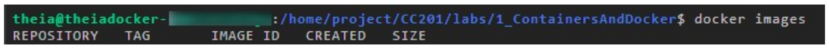
```
theia@theiadocker-[redacted]:/home/project/CC201/labs/1_ContainersAndDocker$ ls
app.js  Dockerfile  package.json
```

Pull an image from Docker Hub and run it as a container

1. Use the `docker` CLI to list your images.

```
docker images
```

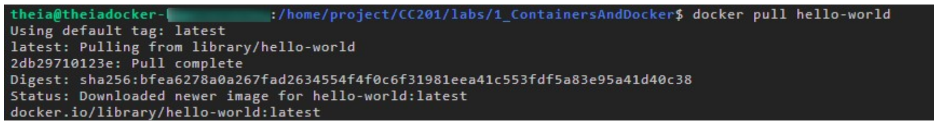
You should see an empty table (with only headings) since you don't have any images yet.



```
theia@theiadocker-[redacted]:/home/project/CC201/labs/1_ContainersAndDocker$ docker images
REPOSITORY   TAG       IMAGE ID   CREATED   SIZE
```

2. Pull your first image from Docker Hub.

```
docker pull hello-world
```

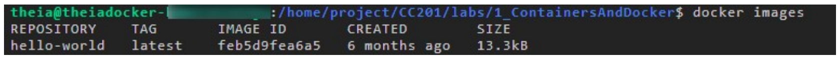


```
theia@theiadocker-[redacted]:/home/project/CC201/labs/1_ContainersAndDocker$ docker pull hello-world
Using default tag: latest
latest: Pulling from library/hello-world
2db29710123e: Pull complete
Digest: sha256:bfea6278a0a267fad2634554f4f0c6f31981eea41c553fdf5a83e95a41d40c38
Status: Downloaded newer image for hello-world:latest
docker.io/library/hello-world:latest
```

3. List images again.

```
docker images
```

You should now see the `hello-world` image present in the table.



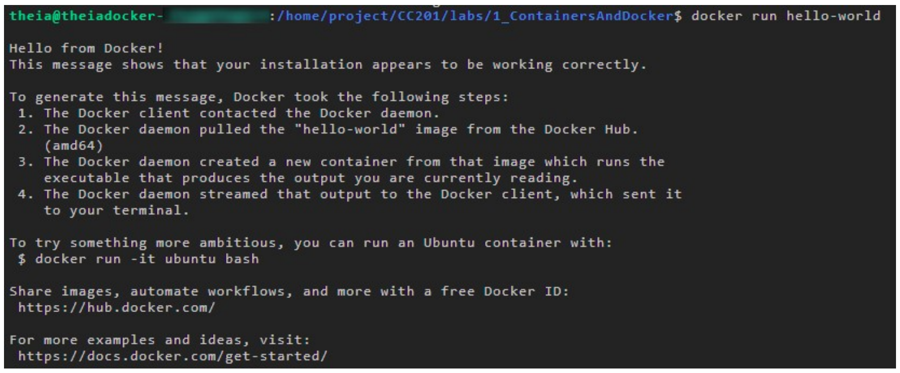
```
theia@theiadocker-[redacted]:/home/project/CC201/labs/1_ContainersAndDocker$ docker images
REPOSITORY    TAG       IMAGE ID       CREATED        SIZE
hello-world   latest    feb5d9fea6a5   6 months ago   13.3kB
```

4. Run the `hello-world` image as a container.

```
docker run hello-world
```

You should see a **'Hello from Docker!'** message.

There will also be an explanation of what Docker did to generate this message.

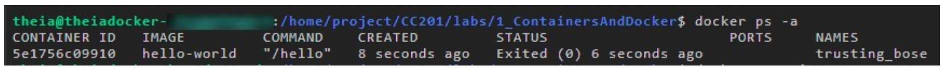


```
theia@theiadocker-[redacted]:/home/project/CC201/labs/1_ContainersAndDocker$ docker run hello-world

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

To generate this message, Docker took the following steps:
 1. The Docker client contacted the Docker daemon.
 2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
    (amd64)
 3. The Docker daemon created a new container from that image which runs the
    executable that produces the output you are currently reading.
 4. The Docker daemon streamed that output to the Docker client, which sent it
    to your terminal.

To try something more ambitious, you can run an Ubuntu container with:
 $ docker run -it ubuntu bash

Share images, automate workflows, and more with a free Docker ID:
 https://hub.docker.com/

For more examples and ideas, visit:
 https://docs.docker.com/get-started/
```

5. List the containers to see that your container ran and exited successfully.

```
docker ps -a
```

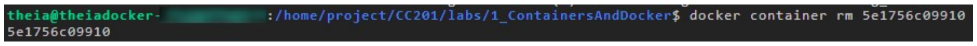
Among other things, for this container you should see a container ID, the image name (`hello-world`), and a status that indicates that the container exited successfully.



```
theia@theiadocker-[redacted]:/home/project/CC201/labs/1_ContainersAndDocker$ docker ps -a
CONTAINER ID   IMAGE         COMMAND    CREATED         STATUS                     PORTS     NAMES
5e1756c09910   hello-world   "/hello"   8 seconds ago   Exited (0) 6 seconds ago             trusting_bose
```

6. Note the CONTAINER ID from the previous output and replace the \ in the command below with this value. This command removes your container.

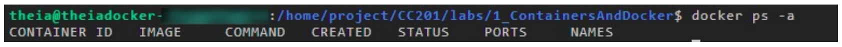
```
docker container rm <container_id>
```



```
theia@theiadocker-[redacted]:/home/project/CC201/labs/1_ContainersAndDocker$ docker container rm 5e1756c09910
5e1756c09910
```

7. Verify that that the container has been removed. Run the following command.

```
docker ps -a
```



```
theia@theiadocker-[redacted]:/home/project/CC201/labs/1_ContainersAndDocker$ docker ps -a
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES
```

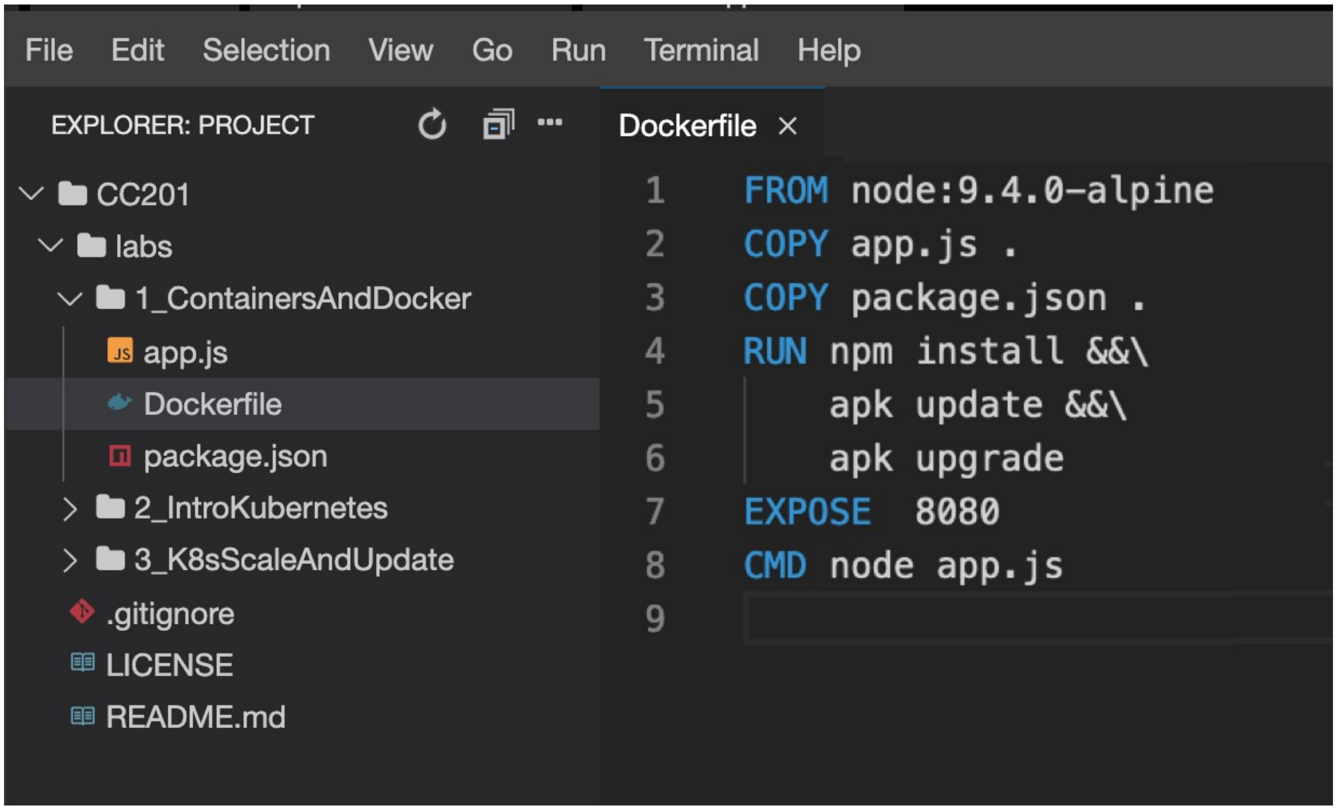
Congratulations on pulling an image from Docker Hub and running your first container! Now let's try and build our own image.

Build an image using a Dockerfile

1. The current working directory contains a simple Node.js application that we will run in a container. The app will print a hello message along with the hostname. The following files are needed to run the app in a container:

- app.js is the main application, which simply replies with a hello world message.
- package.json defines the dependencies of the application.
- Dockerfile defines the instructions Docker uses to build the image.

1. Use the Explorer to view the files needed for this app. Click the Explorer icon (it looks like a sheet of paper) on the left side of the window, and then navigate to the directory for this lab: `CC201 > labs > 1_ContainersAndDocker`. Click `Dockerfile` to view the commands required to build an image.



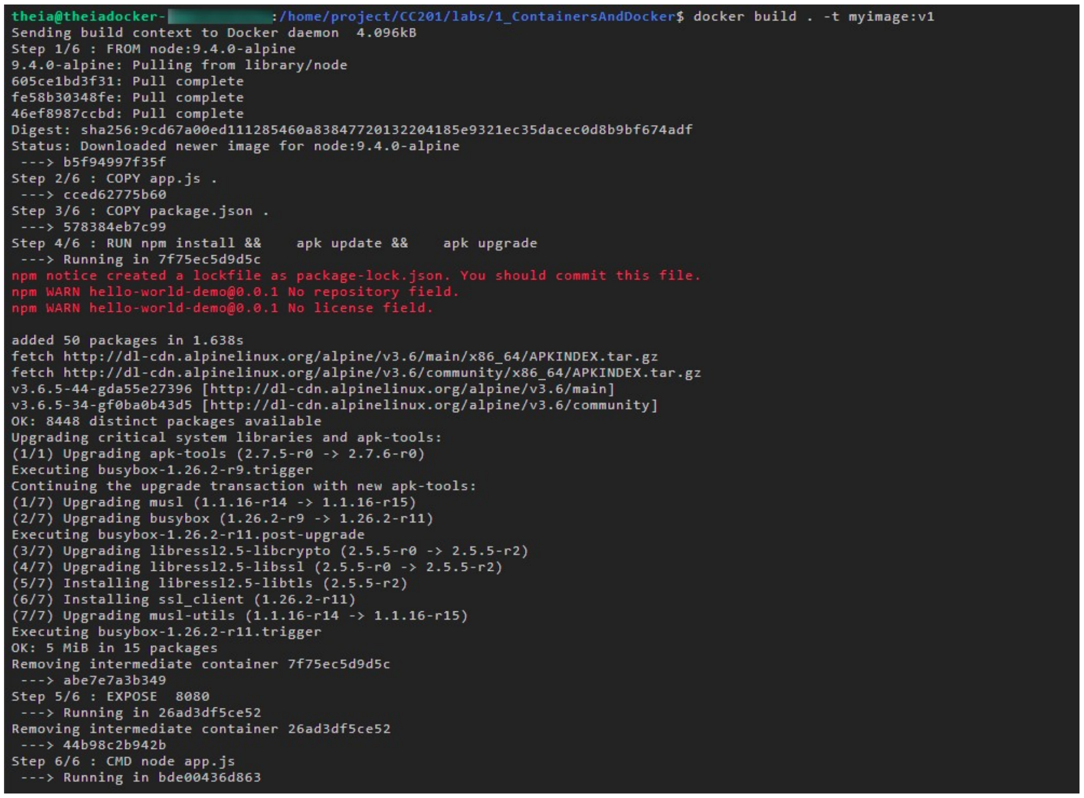
```
FROM node:9.4.0-alpine
COPY app.js .
COPY package.json .
RUN npm install &&\
    apk update &&\
    apk upgrade
EXPOSE  8080
CMD node app.js

```

You can refresh your understanding of the commands mentioned in the Dockerfile below:
The FROM instruction initializes a new build stage and specifies the base image that subsequent instructions will build upon.
The COPY command enables us to copy files to our image.
The RUN instruction executes commands.
The EXPOSE instruction exposes a particular port with a specified protocol inside a Docker Container.
The CMD instruction provides a default for executing a container, or in other words, an executable that should run in your container.

3. Run the following command to build the image:

```
docker build . -t myimage:v1
```

As seen in the module videos, the output creates a new layer for each instruction in the Dockerfile.



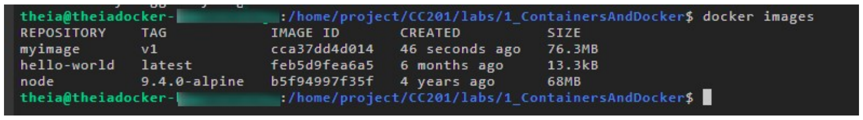
```
theia@theiadocker-[redacted]:/home/project/CC201/labs/1_ContainersAndDocker$ docker build . -t myimage:v1
Sending build context to Docker daemon  4.096kB
Step 1/6 : FROM node:9.4.0-alpine
9.4.0-alpine: Pulling from library/node
605ce1bd3f31: Pull complete
fe58b30348fe: Pull complete
46ef8987ccbd: Pull complete
Digest: sha256:9cd67a00ed111285460a83847720132204185e9321ec35dacec0d8b9bf674adf
Status: Downloaded newer image for node:9.4.0-alpine
 ---> b5f94997f35f
Step 2/6 : COPY app.js .
 ---> cced62775b60
Step 3/6 : COPY package.json .
 ---> 578384eb7c99
Step 4/6 : RUN npm install &&     apk update &&     apk upgrade
 ---> Running in 7f75ec5d9d5c
npm notice created a lockfile as package-lock.json. You should commit this file.
npm WARN hello-world-demo@0.0.1 No repository field.
npm WARN hello-world-demo@0.0.1 No license field.

added 50 packages in 1.638s
fetch http://dl-cdn.alpinelinux.org/alpine/v3.6/main/x86_64/APKINDEX.tar.gz
fetch http://dl-cdn.alpinelinux.org/alpine/v3.6/community/x86_64/APKINDEX.tar.gz
v3.6.5-44-gda55e27396 [http://dl-cdn.alpinelinux.org/alpine/v3.6/main]
v3.6.5-34-gf0ba0b43d5 [http://dl-cdn.alpinelinux.org/alpine/v3.6/community]
OK: 8448 distinct packages available
Upgrading critical system libraries and apk-tools:
(1/1) Upgrading apk-tools (2.7.5-r0 -> 2.7.6-r0)
Executing busybox-1.26.2-r9.trigger
Continuing the upgrade transaction with new apk-tools:
(1/7) Upgrading musl (1.1.16-r14 -> 1.1.16-r15)
(2/7) Upgrading busybox (1.26.2-r9 -> 1.26.2-r11)
Executing busybox-1.26.2-r11.post-upgrade
(3/7) Upgrading libressl2.5-libcrypto (2.5.5-r0 -> 2.5.5-r2)
(4/7) Upgrading libressl2.5-libssl (2.5.5-r0 -> 2.5.5-r2)
(5/7) Installing libressl2.5-libtls (2.5.5-r2)
(6/7) Installing ssl_client (1.26.2-r11)
(7/7) Upgrading musl-utils (1.1.16-r14 -> 1.1.16-r15)
Executing busybox-1.26.2-r11.trigger
OK: 5 MiB in 15 packages
Removing intermediate container 7f75ec5d9d5c
 ---> abe7e7a3b349
Step 5/6 : EXPOSE  8080
 ---> Running in 26ad3df5ce52
Removing intermediate container 26ad3df5ce52
 ---> 44b98c2b942b
Step 6/6 : CMD node app.js
 ---> Running in bde00436d863
```

4. List images to see your image tagged `myimage:v1` in the table.

```
docker images
```

Note that compared to the `hello-world` image, this image has a different image ID. This means that the two images consist of different layers -- in other words, they're not the same image.

You should also see a `node` image in the images output. This is because the `docker build` command pulled `node:9.4.0-alpine` to use it as the base image for the image you built.



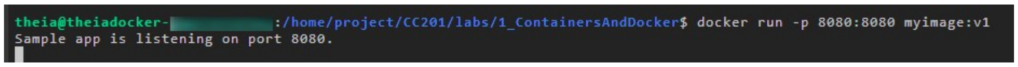
```
theia@theiadocker-[redacted]:/home/project/CC201/labs/1_ContainersAndDocker$ docker images
REPOSITORY    TAG            IMAGE ID       CREATED          SIZE
myimage       v1             cca37dd4d014   46 seconds ago   76.3MB
hello-world   latest         feb5d9fea6a5   6 months ago     13.3kB
node          9.4.0-alpine   b5f94997f35f   4 years ago      68MB
theia@theiadocker-[redacted]:/home/project/CC201/labs/1_ContainersAndDocker$
```

Run the image as a container

1. Now that your image is built, run it as a container with the following command:

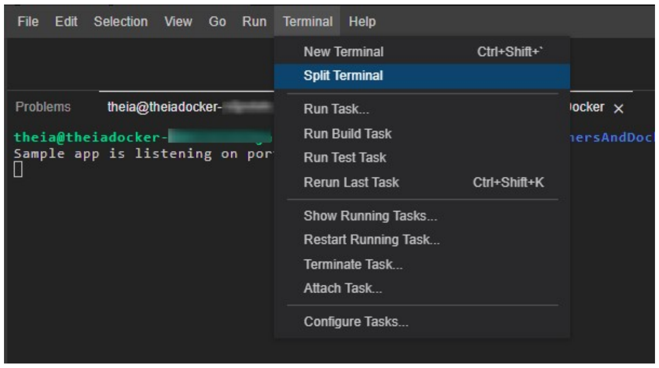
```
docker run -p 8080:8080 myimage:v1
```

The output should indicate that your application is listening on port 8080. This command will continue running until it exits, since the container runs a web app that continually listens for requests. To query the app, we need to open another terminal window.



```
theia@theiadocker-[redacted]:/home/project/CC201/labs/1_ContainersAndDocker$ docker run -p 8080:8080 myimage:v1
Sample app is listening on port 8080.
```

2. To split the terminal, click `Terminal > Split Terminal`.

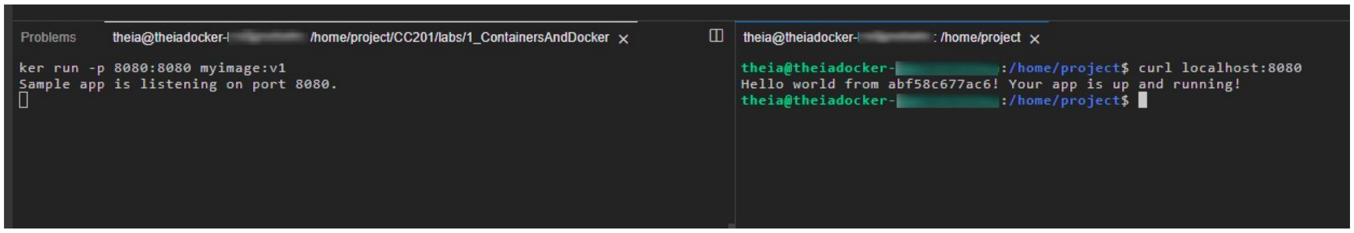


```
theia@theiadocker-[redacted]
Sample app is listening on por
```

3. In the second terminal window, use the `curl` command to ping the application.

```
curl localhost:8080
```

The output should indicate that **'Your app is up and running!'.**

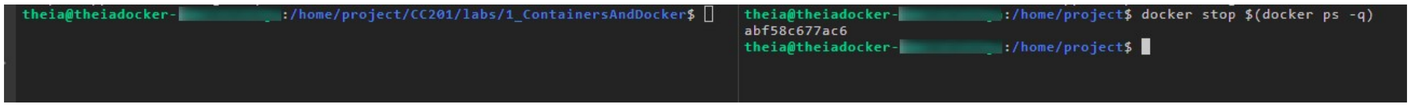


```
Problems   theia@theiadocker-[redacted] /home/project/CC201/labs/1_ContainersAndDocker ×
ker run -p 8080:8080 myimage:v1
Sample app is listening on port 8080.
```

```
theia@theiadocker-[redacted] : /home/project ×
theia@theiadocker-[redacted]:/home/project$ curl localhost:8080
Hello world from abf58c677ac6! Your app is up and running!
theia@theiadocker-[redacted]:/home/project$
```

4. In the second terminal window, stop the container. The following command uses `docker ps -q` to pass in the list of all running containers:

```
docker stop $(docker ps -q)
```

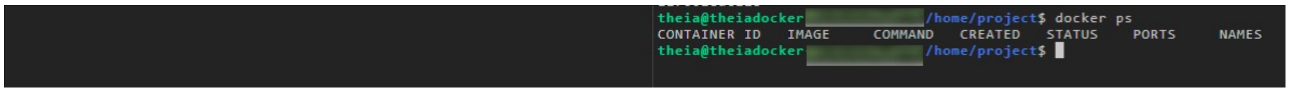


```
theia@theiadocker-[redacted]:/home/project/CC201/labs/1_ContainersAndDocker$
```

```
theia@theiadocker-[redacted]:/home/project$ docker stop $(docker ps -q)
abf58c677ac6
theia@theiadocker-[redacted]:/home/project$
```

5. In the second terminal window, check if the container has stopped by running the following command.

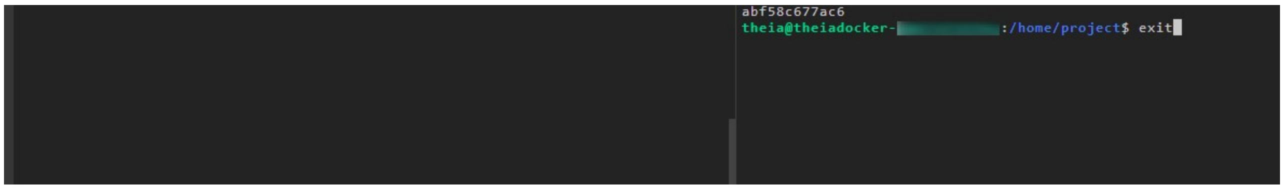
```
docker ps
```



```
theia@theiadocker[redacted]/home/project$ docker ps
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES
theia@theiadocker[redacted]/home/project$
```

6. Close the second terminal window, as it is no longer needed.

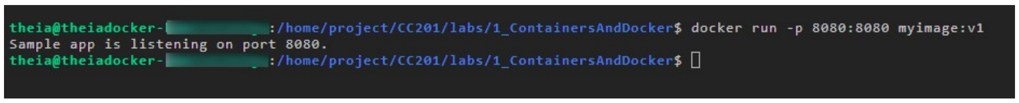
```
exit
```



```
abf58c677ac6
theia@theiadocker-[redacted]:/home/project$ exit
```

In the original terminal window, the `docker run` command has exited and you are able to type commands in that terminal window again.

Note: If you face any issues in typing further commands in the terminal, press Enter.

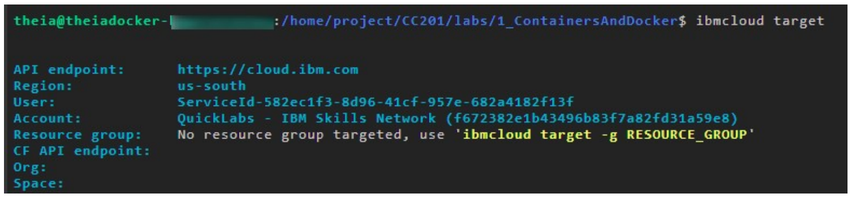


```
theia@theiadocker-[redacted]:/home/project/CC201/labs/1_ContainersAndDocker$ docker run -p 8080:8080 myimage:v1
Sample app is listening on port 8080.
theia@theiadocker-[redacted]:/home/project/CC201/labs/1_ContainersAndDocker$
```

Push the image to IBM Cloud Container Registry

1. The environment should have already logged you into the IBM Cloud account that has been automatically generated for you by the Skills Network Labs environment. The following command will give you information about the account you're targeting:

```
ibmcloud target
```



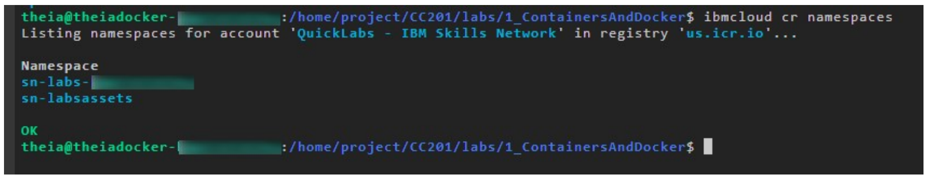
```
theia@theiadocker-[redacted]:/home/project/CC201/labs/1_ContainersAndDocker$ ibmcloud target


API endpoint:      https://cloud.ibm.com
Region:            us-south
User:              ServiceId-582ec1f3-8d96-41cf-957e-682a4182f13f
Account:           QuickLabs - IBM Skills Network (f672382e1b43496b83f7a82fd31a59e8)
Resource group:    No resource group targeted, use 'ibmcloud target -g RESOURCE_GROUP'
CF API endpoint:
Org:
Space:
```

2. The environment also created an IBM Cloud Container Registry (ICR) namespace for you. Since Container Registry is multi-tenant, namespaces are used to divide the registry among several users. Use the following command to see the namespaces you have access to:

```
ibmcloud cr namespaces
```

You should see two namespaces listed starting with `sn-labs`:

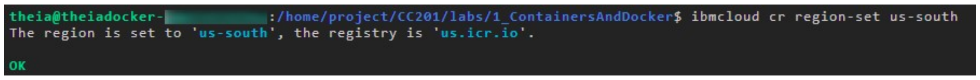
- The first one with your username is a namespace just for you. You have full *read* and *write* access to this namespace.
- The second namespace, which is a shared namespace, provides you with only Read Access



```
theia@theiadocker-[redacted]:/home/project/CC201/labs/1_ContainersAndDocker$ ibmcloud cr namespaces
Listing namespaces for account 'QuickLabs - IBM Skills Network' in registry 'us.icr.io'...

Namespace
sn-labs-[redacted]
sn-labsassets

OK
theia@theiadocker-[redacted]:/home/project/CC201/labs/1_ContainersAndDocker$
```

3. Ensure that you are targeting the region appropriate to your cloud account, for instance `us-south` region where these namespaces reside as you saw in the output of the `ibmcloud target` command.

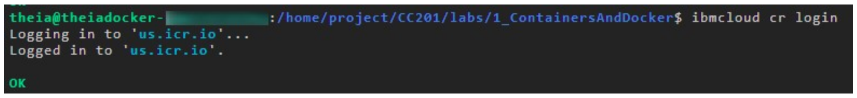
```
ibmcloud cr region-set us-south
```



```
theia@theiadocker-[redacted]:/home/project/CC201/labs/1_ContainersAndDocker$ ibmcloud cr region-set us-south
The region is set to 'us-south', the registry is 'us.icr.io'.

OK
```

4. Log your local Docker daemon into IBM Cloud Container Registry so that you can push to and pull from the registry.

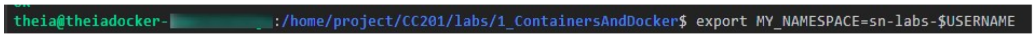
```
ibmcloud cr login
```



```
theia@theiadocker-[redacted]:/home/project/CC201/labs/1_ContainersAndDocker$ ibmcloud cr login
Logging in to 'us.icr.io'...
Logged in to 'us.icr.io'.

OK
```

5. Export your namespace as an environment variable so that it can be used in subsequent commands.

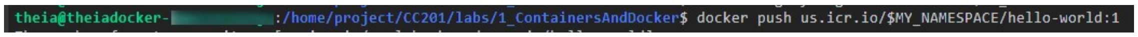
```
export MY_NAMESPACE=sn-labs-$USERNAME
```



```
theia@theiadocker-[redacted]:/home/project/CC201/labs/1_ContainersAndDocker$ export MY_NAMESPACE=sn-labs-$USERNAME
```

6. Tag your image so that it can be pushed to IBM Cloud Container Registry.

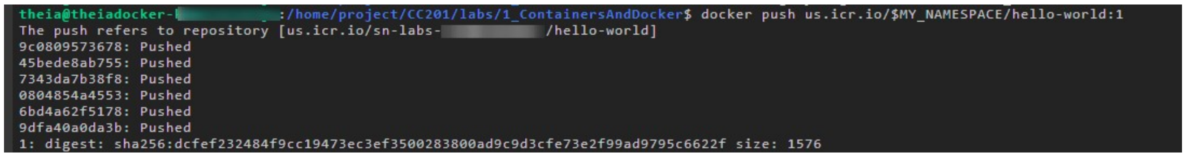
```
docker tag myimage:v1 us.icr.io/$MY_NAMESPACE/hello-world:1
```



```
theia@theiadocker-[redacted]:/home/project/CC201/labs/1_ContainersAndDocker$ docker push us.icr.io/$MY_NAMESPACE/hello-world:1
```

7. Push the newly tagged image to IBM Cloud Container Registry.

```
docker push us.icr.io/$MY_NAMESPACE/hello-world:1
```

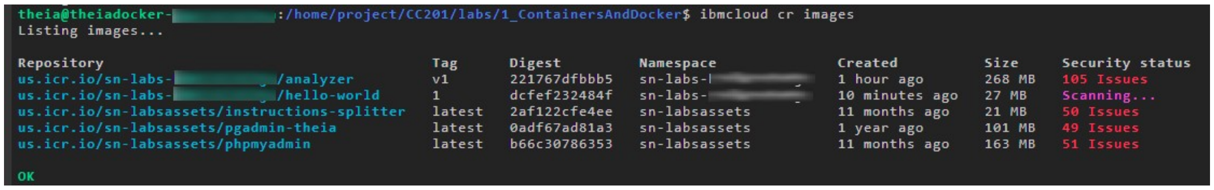


```
theia@theiadocker-[redacted]:/home/project/CC201/labs/1_ContainersAndDocker$ docker push us.icr.io/$MY_NAMESPACE/hello-world:1
The push refers to repository [us.icr.io/sn-labs-[redacted]/hello-world]
9c0809573678: Pushed
45bede8ab755: Pushed
7343da7b38f8: Pushed
0804854a4553: Pushed
6bd4a62f5178: Pushed
9dfa40a0da3b: Pushed
1: digest: sha256:dcfef232484f9cc19473ec3ef3500283800ad9c9d3cfe73e2f99ad9795c6622f size: 1576
```

> **Note:** If you have tried this lab earlier, there might be a possibility that the previous session is still persistent. In such a case, you will see a **'Layer already Exists'** message instead of the **'Pushed'** message in the above output. We recommend you to proceed with the next steps of the lab.

8. Verify that the image was successfully pushed by listing images in Container Registry.

```
ibmcloud cr images
```

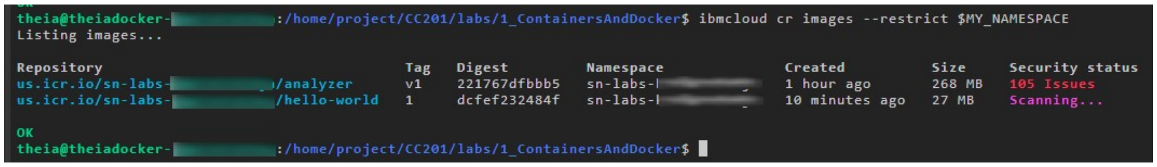


```
theia@theiadocker-[redacted]:/home/project/CC201/labs/1_ContainersAndDocker$ ibmcloud cr images
Listing images...

Repository                                  Tag      Digest         Namespace          Created          Size     Security status
us.icr.io/sn-labs-[redacted]/analyzer       v1       221767dfbbb5   sn-labs-[redacted] 1 hour ago       268 MB   105 Issues
us.icr.io/sn-labs-[redacted]/hello-world    1        dcfef232484f   sn-labs-[redacted] 10 minutes ago   27 MB    Scanning...
us.icr.io/sn-labsassets/instructions-splitter latest 2af122cfe4ee   sn-labsassets      11 months ago    21 MB    50 Issues
us.icr.io/sn-labsassets/pgadmin-theia       latest   0adf67ad81a3   sn-labsassets      1 year ago       101 MB   49 Issues
us.icr.io/sn-labsassets/phpmyadmin          latest   b66c30786353   sn-labsassets      11 months ago    163 MB   51 Issues

OK
```

Optionally, to only view images within a specific namespace.

```
ibmcloud cr images --restrict $MY_NAMESPACE
```

You should see your image name in the output. Recall from the module videos that we discussed Vulnerability Advisor, which scans images in IBM Cloud Container Registry for common vulnerabilities and exposures. In the last column of the output, note that Vulnerability Advisor is either scanning your image or it has provided a security status, depending on how quickly you list the images and how long the scan takes.



```
theia@theiadocker-[redacted]:/home/project/CC201/labs/1_ContainersAndDocker$ ibmcloud cr images --restrict $MY_NAMESPACE
Listing images...

Repository                                 Tag   Digest         Namespace          Created          Size     Security status
us.icr.io/sn-labs-[redacted]/analyzer      v1    221767dfbbb5   sn-labs-[redacted] 1 hour ago       268 MB   105 Issues
us.icr.io/sn-labs-[redacted]/hello-world   1     dcfef232484f   sn-labs-[redacted] 10 minutes ago   27 MB    Scanning...

OK
theia@theiadocker-[redacted]:/home/project/CC201/labs/1_ContainersAndDocker$
```

Congratulations! You have completed the second lab for the first module of this course.

> **Note:** Please delete your project from SN labs environment before signing out to ensure that further labs run correctly. To do the same, click on this link

Changelog

Date	Version	Changed by	Change Description
2022-04-08	1.1	K Sundararajan	Updated Lab instructions
2022-04-19	1.2	K Sundararajan	Updated Lab instructions

| | | | |

Previous Continue