Introduction to Containers, Docker, and IBM Cloud Container Registry

Skills Network

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

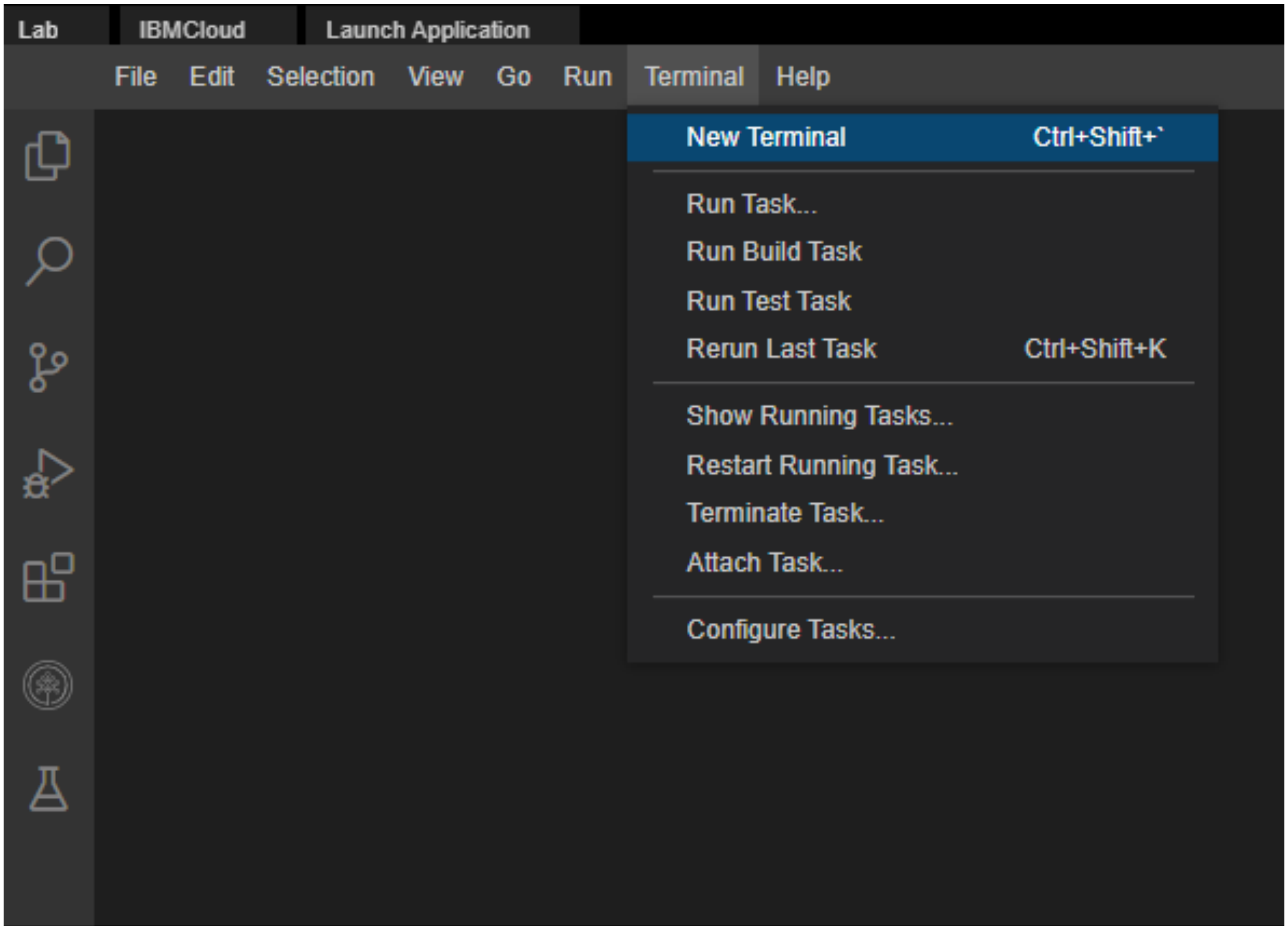
> **Note: Kindly complete the lab in a single session without any break because the lab may go on offline mode and may cause errors. If you face any issues/errors during the lab process, please logout from the lab environment. Then clear your system cache and cookies and try to complete the lab.**

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`.

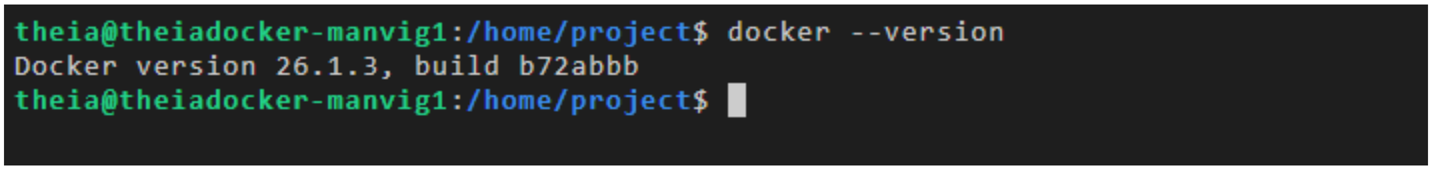
> **Note:If the terminal is already opened, please skip this step.**



2. Verify that `docker` CLI is installed.

```
docker --version
```

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

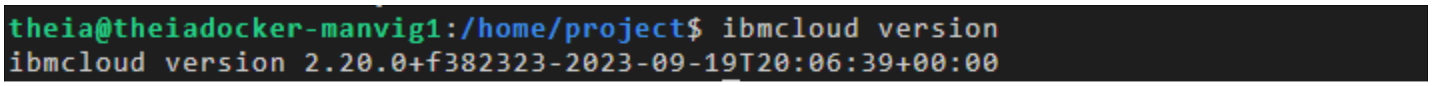


```
theia@theiadocker-manvig1:/home/project$ docker --version
Docker version 26.1.3, build b72abbb
theia@theiadocker-manvig1:/home/project$ 
```

3. Verify that `ibmcloud` CLI is installed.

```
ibmcloud version
```

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



```
theia@theiadocker-manvig1:/home/project$ ibmcloud version
ibmcloud version 2.20.0+f382323-2023-09-19T20:06:39+00:00
```

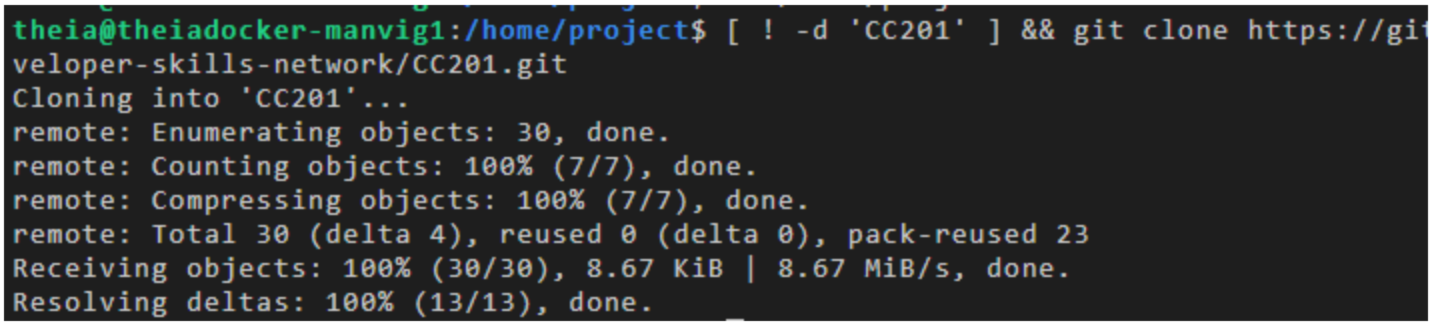
4. Change to your project folder.

   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
```



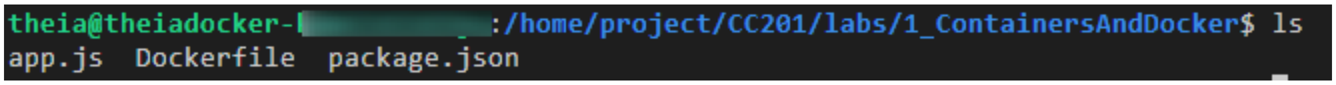
```
theia@theiadocker-manvig1:/home/project$ [ ! -d 'CC201' ] && git clone https://git
veloper-skills-network/CC201.git
Cloning into 'CC201'...
remote: Enumerating objects: 30, done.
remote: Counting objects: 100% (7/7), done.
remote: Compressing objects: 100% (7/7), done.
remote: Total 30 (delta 4), reused 0 (delta 0), pack-reused 23
Receiving objects: 100% (30/30), 8.67 KiB | 8.67 MiB/s, done.
Resolving deltas: 100% (13/13), done.
```

6. Change to the directory for this lab by running the following command. `cd` will change the working/current directory to the directory with the name specified, in this case **CC201/labs/1_ContainersAndDcoker**.

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

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

   ```
   ls
   ```



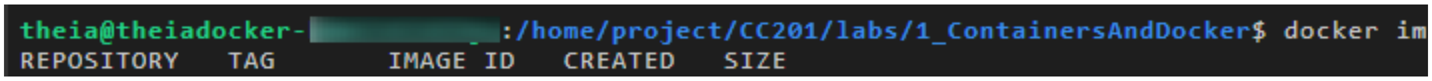
```
theia@theiadocker-[redacted]:/home/project/CC201/labs/1_ContainersAndDocker$ ls
app.js  Dockerfile  package.json
```

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

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

   ```
   docker images
   ```

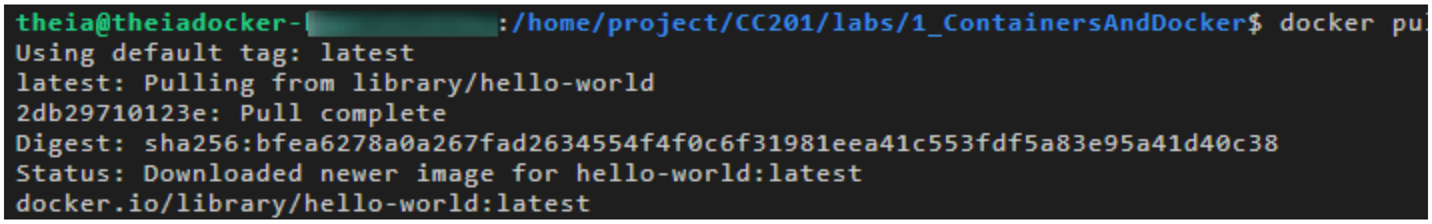
You should see an empty table (with only headings) since you don't have any images yet.



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

2. Pull your first image from Docker Hub.

```
docker pull hello-world
```

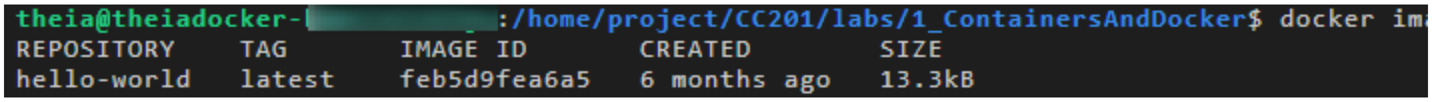


```
theia@theiadocker-[redacted]:/home/project/CC201/labs/1_ContainersAndDocker$ docker pu
Using default tag: latest
latest: Pulling from library/hello-world
2db29710123e: Pull complete
Digest: sha256:bfea6278a0a267fad2634554f4f0c6f31981eea41c553fdf5a83e95a41d40c38
Status: Downloaded newer image for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



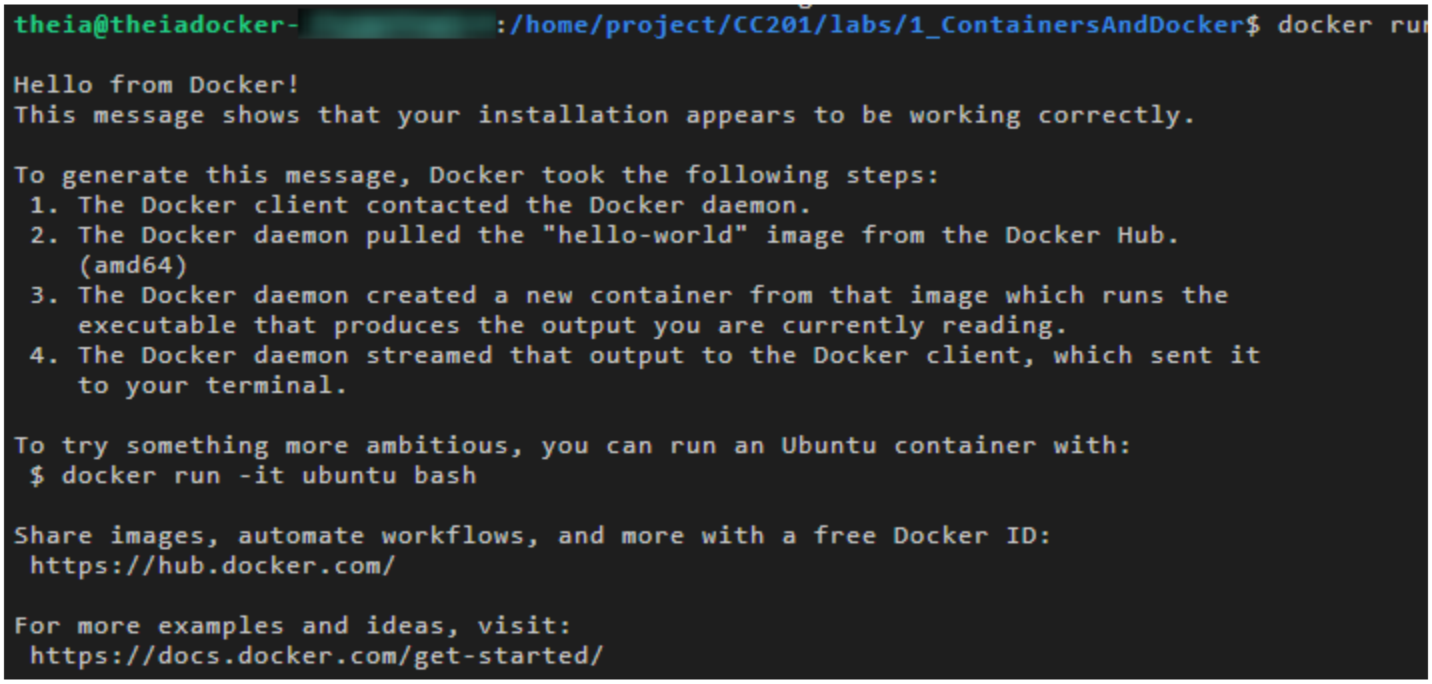
```
theia@theiadocker-[redacted]:/home/project/CC201/labs/1_ContainersAndDocker$ docker ima
REPOSITORY    TAG       IMAGE ID       CREATED         SIZE
hello-world   latest    feb5d9fea6a5   6 months ago    13.3kB
```

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.

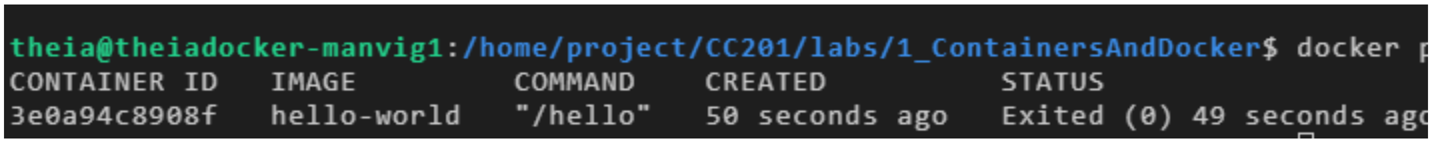


```
theia@theiadocker-[redacted]:/home/project/CC201/labs/1_ContainersAndDocker$ docker ru

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
   ```

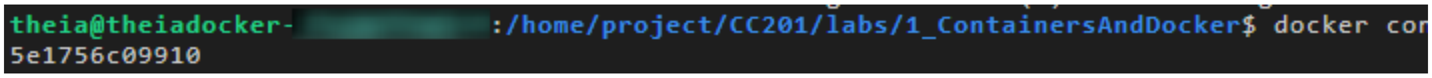
Among other things, for this container you should see a container ID, the image name (`hello-world`), and a status that indicates that the container exited successfully.



```
theia@theiadocker-manvig1:/home/project/CC201/labs/1_ContainersAndDocker$ docker p
CONTAINER ID   IMAGE         COMMAND    CREATED          STATUS
3e0a94c8908f   hello-world   "/hello"   50 seconds ago   Exited (0) 49 seconds ag
```

6. Note the CONTAINER ID from the previous output and replace the **<container_id>** tag in the command below with this value. This command removes your container.

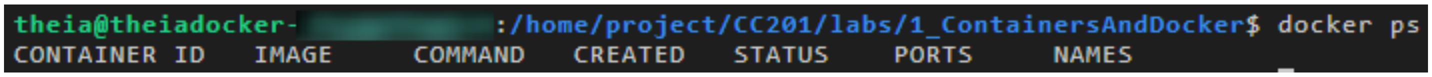
   ```
   docker container rm <container_id>
   ```



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

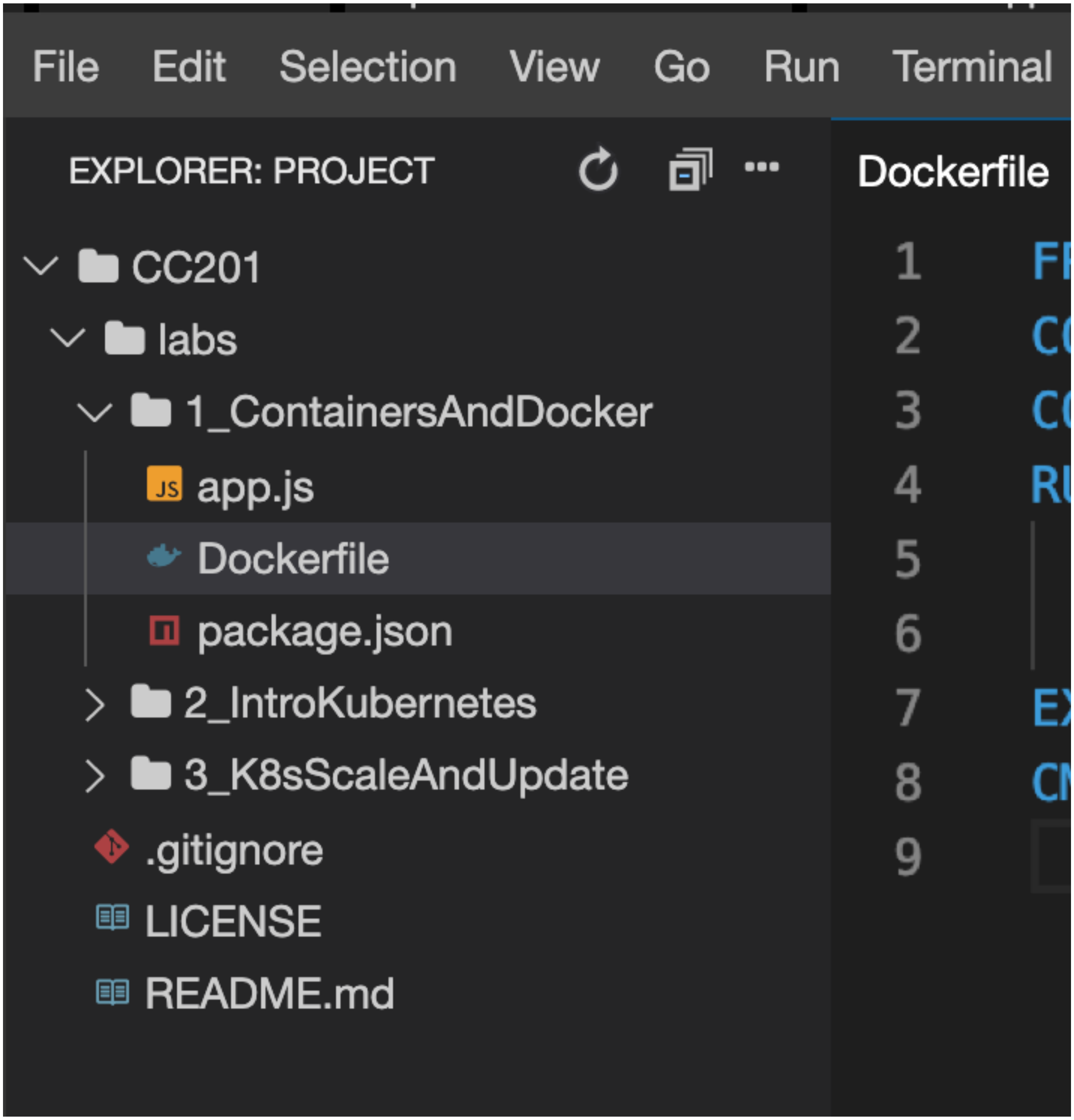
Congratulations on pulling an image from Docker Hub and running your first container! Now let's try and build our own image.

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.

2. 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.



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.

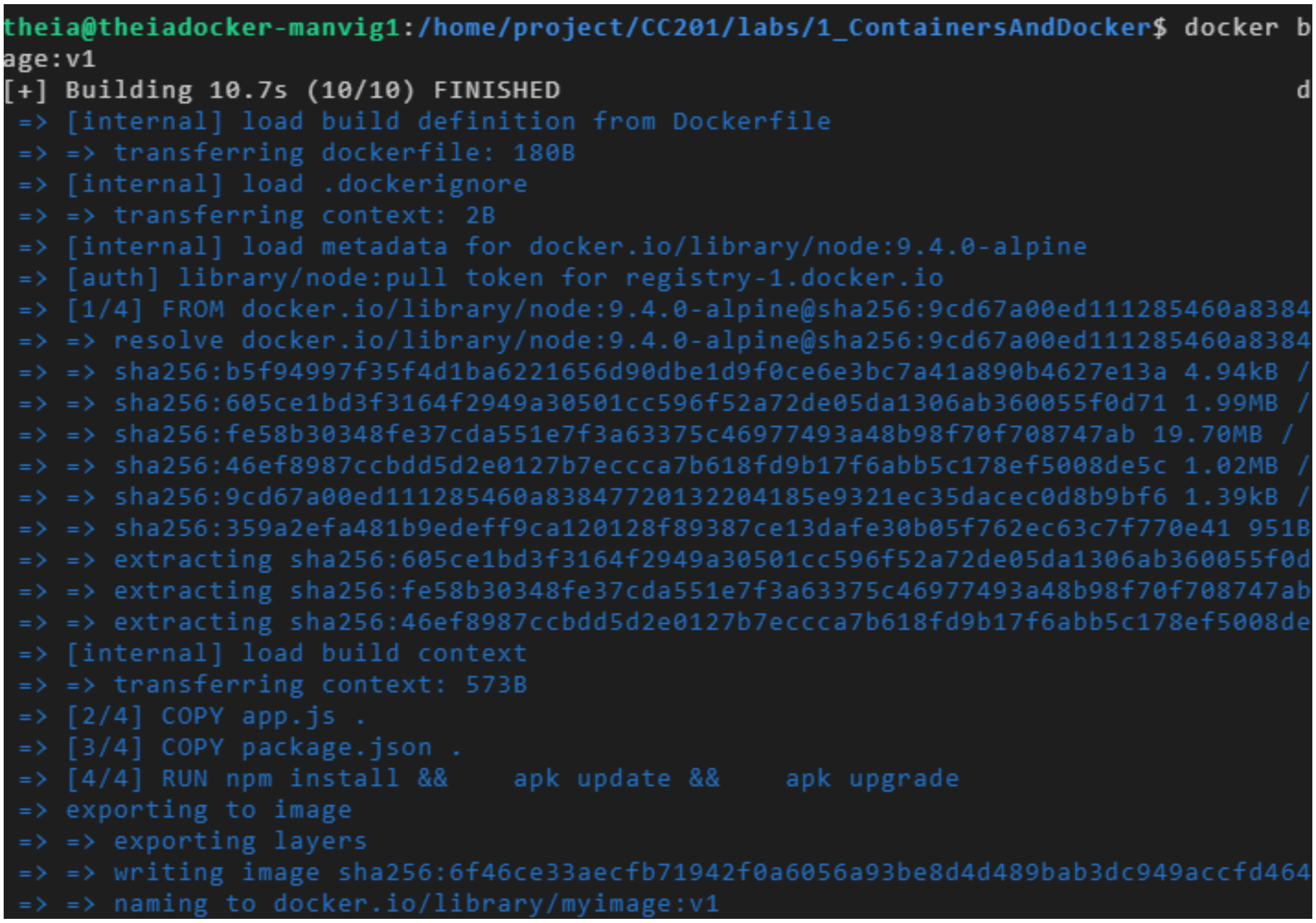
The EXPOSE instruction exposes a particular port with a specified protocol inside a Docker Container.

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
```

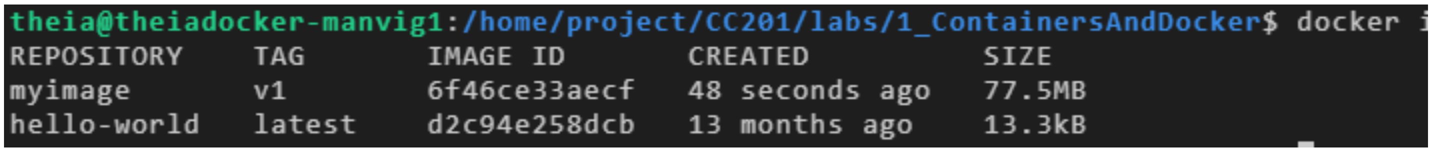
As seen in the module videos, the output creates a new layer for each instruction in the Dockerfile.



```
theia@theiadocker-manvig1:/home/project/CC201/labs/1_ContainersAndDocker$ docker b
age:v1
[+] Building 10.7s (10/10) FINISHED                                                  d
 => [internal] load build definition from Dockerfile
 => => transferring dockerfile: 180B
 => [internal] load .dockerignore
 => => transferring context: 2B
 => [internal] load metadata for docker.io/library/node:9.4.0-alpine
 => [auth] library/node:pull token for registry-1.docker.io
 => [1/4] FROM docker.io/library/node:9.4.0-alpine@sha256:9cd67a00ed111285460a8384
 => => resolve docker.io/library/node:9.4.0-alpine@sha256:9cd67a00ed111285460a8384
 => => sha256:b5f94997f35f4d1ba6221656d90dbe1d9f0ce6e3bc7a41a890b4627e13a 4.94kB /
 => => sha256:605ce1bd3f3164f2949a30501cc596f52a72de05da1306ab360055f0d71 1.99MB /
 => => sha256:fe58b30348fe37cda551e7f3a63375c46977493a48b98f70f708747ab 19.70MB /
 => => sha256:46ef8987ccbdd5d2e0127b7eccca7b618fd9b17f6abb5c178ef5008de5c 1.02MB /
 => => sha256:9cd67a00ed111285460a838477201322041 85e9321ec35dacec0d8b9bf6 1.39kB /
 => => sha256:359a2efa481b9edeff9ca120128f89387ce13dafe30b05f762ec63c7f770e41 951B
 => => extracting sha256:605ce1bd3f3164f2949a30501cc596f52a72de05da1306ab360055f0d
 => => extracting sha256:fe58b30348fe37cda551e7f3a63375c46977493a48b98f70f708747ab
 => => extracting sha256:46ef8987ccbdd5d2e0127b7eccca7b618fd9b17f6abb5c178ef5008de
 => [internal] load build context
 => => transferring context: 573B
 => [2/4] COPY app.js .
 => [3/4] COPY package.json .
 => [4/4] RUN npm install &&     apk update &&     apk upgrade
 => exporting to image
 => => exporting layers
 => => writing image sha256:6f46ce33aecfb71942f0a6056a93be8d4d489bab3dc949accfd464
 => => naming to docker.io/library/myimage:v1
```

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

```
docker images
```



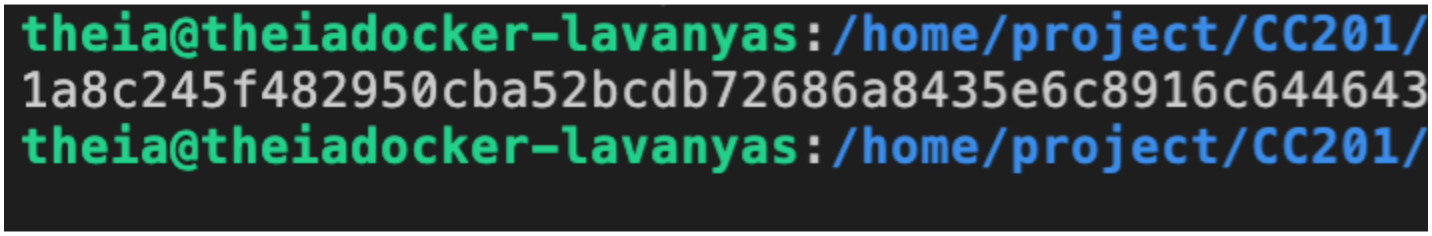
```
theia@theiadocker-manvig1:/home/project/CC201/labs/1_ContainersAndDocker$ docker i
REPOSITORY    TAG       IMAGE ID       CREATED          SIZE
myimage       v1        6f46ce33aecf   48 seconds ago   77.5MB
hello-world   latest    d2c94e258dcb   13 months ago    13.3kB
```

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.

Run the image as a container

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

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

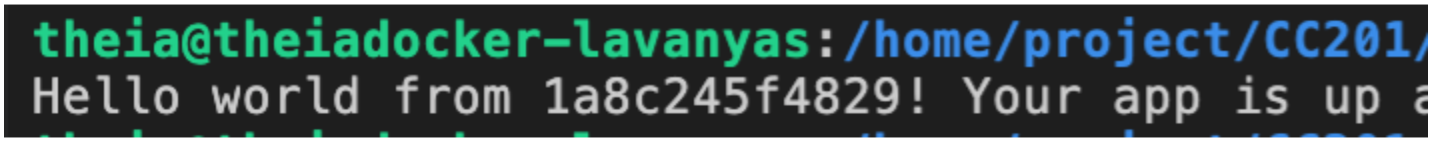


```
theia@theiadocker-lavanyas:/home/project/CC201/
1a8c245f482950cba52bcdb72686a8435e6c8916c644643
theia@theiadocker-lavanyas:/home/project/CC201/
```

The output is a unique code allocated by docker for the application you are running.

2. Run the `curl` command to ping the application as given below.

```
curl localhost:8080
```

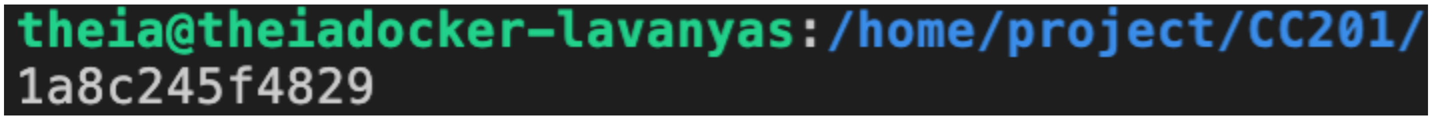


```
theia@theiadocker-lavanyas:/home/project/CC201/
Hello world from 1a8c245f4829! Your app is up a
```

If you see the output as above, it indicates that **'Your app is up and running!'.**

3. Now to stop the container we use `docker stop` followed by the container id. The following command uses `docker ps -q` to pass in the list of all running containers:

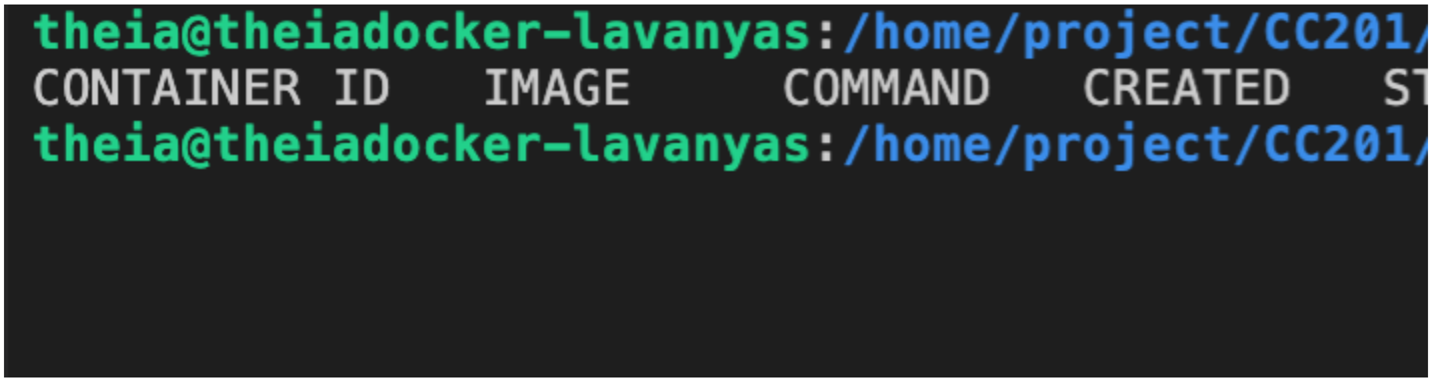
   ```
   docker stop $(docker ps -q)
   ```



```
theia@theiadocker-lavanyas:/home/project/CC201/
1a8c245f4829
```

4. Check if the container has stopped by running the following command.

   ```
   docker ps
   ```

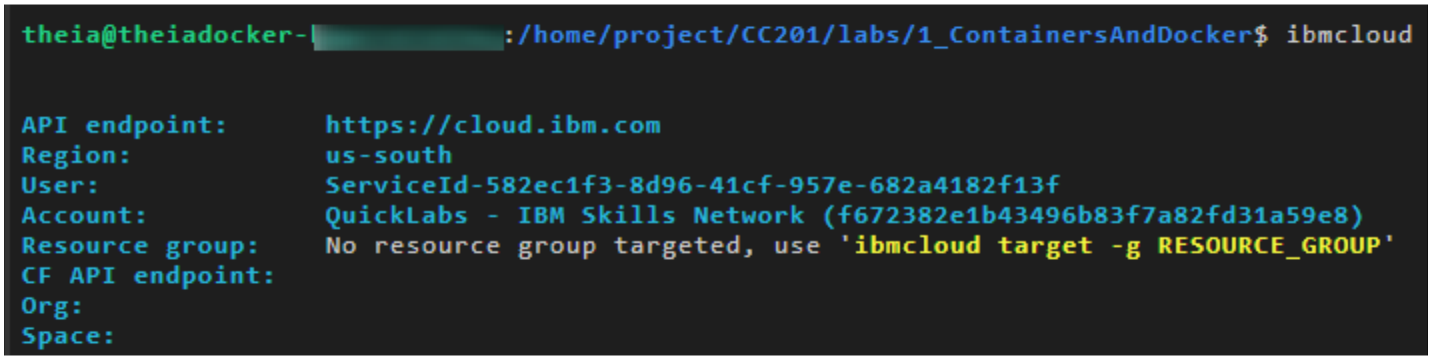


```
theia@theiadocker-lavanyas:/home/project/CC201/
CONTAINER ID   IMAGE      COMMAND    CREATED    ST
theia@theiadocker-lavanyas:/home/project/CC201/
```

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:

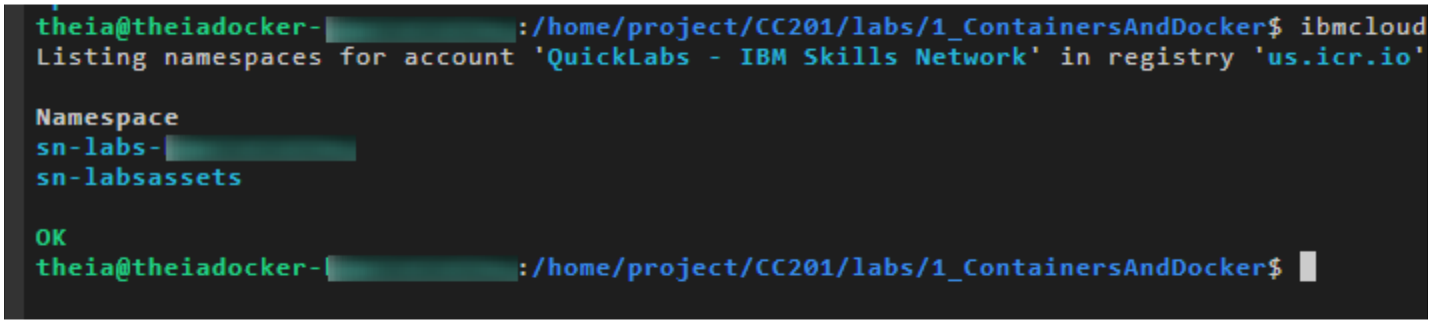
```
ibmcloud target
```



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


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
```



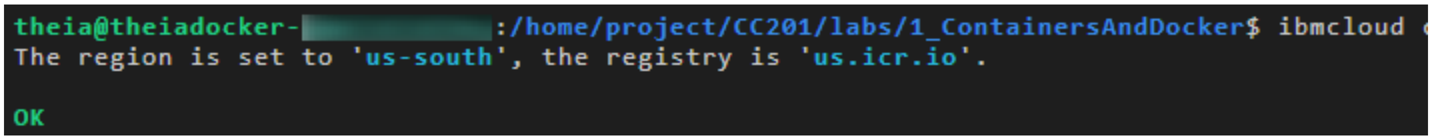
```
theia@theiadocker-[redacted]:/home/project/CC201/labs/1_ContainersAndDocker$ ibmcloud
Listing namespaces for account 'QuickLabs - IBM Skills Network' in registry 'us.icr.io'

Namespace
sn-labs-[redacted]
sn-labsassets

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

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

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.

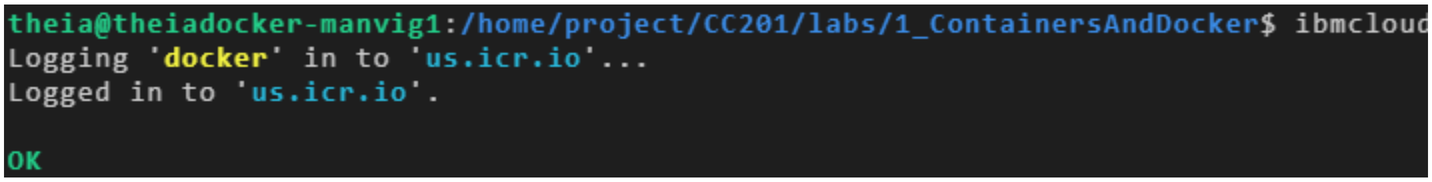
   ```
   ibmcloud cr region-set us-south
   ```



```
theia@theiadocker-[redacted]:/home/project/CC201/labs/1_ContainersAndDocker$ ibmcloud
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-manvig1:/home/project/CC201/labs/1_ContainersAndDocker$ ibmcloud
Logging 'docker' 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
```

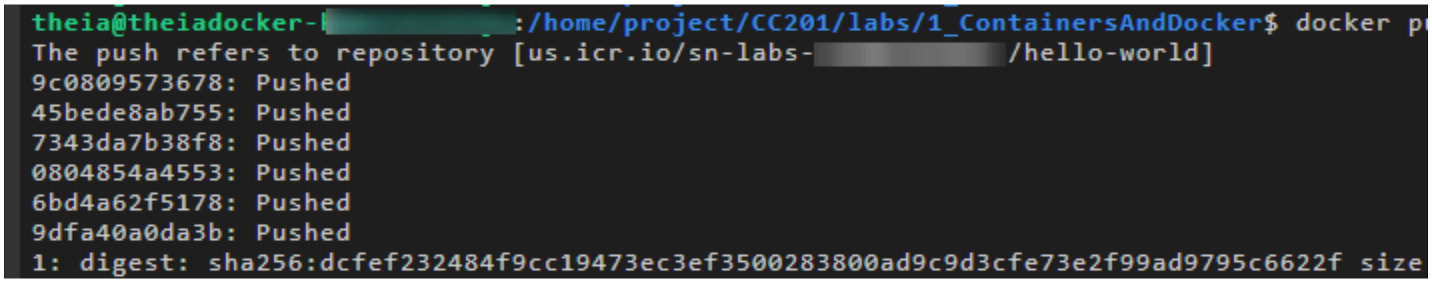
6. Tag your image so that it can be pushed to IBM Cloud Container Registry.

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

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

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

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

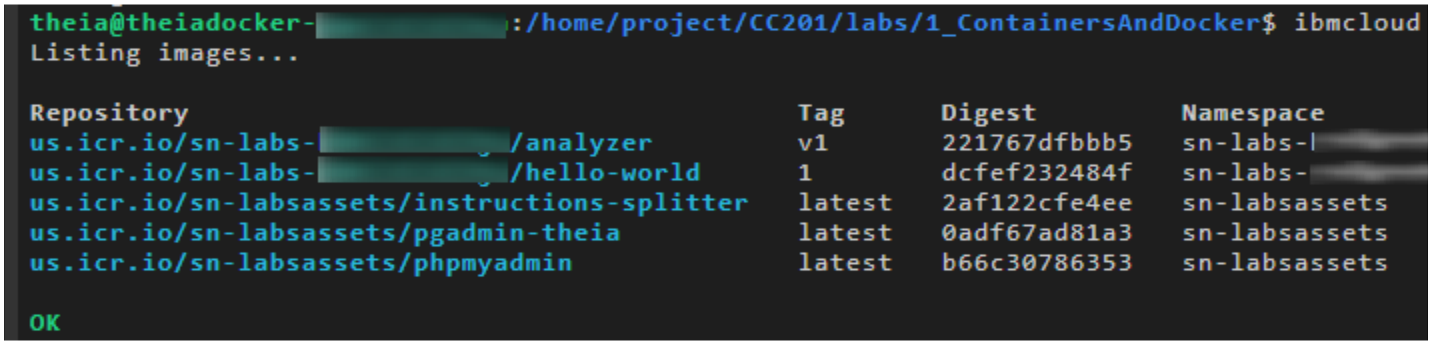


```
theia@theiadocker-[redacted]:/home/project/CC201/labs/1_ContainersAndDocker$ docker pu
The push refers to repository [us.icr.io/sn-labs-[redacted]/hello-world]
9c0809573678: Pushed
45bede8ab755: Pushed
7343da7b38f8: Pushed
0804854a4553: Pushed
6bd4a62f5178: Pushed
9dfa40a0da3b: Pushed
1: digest: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

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.

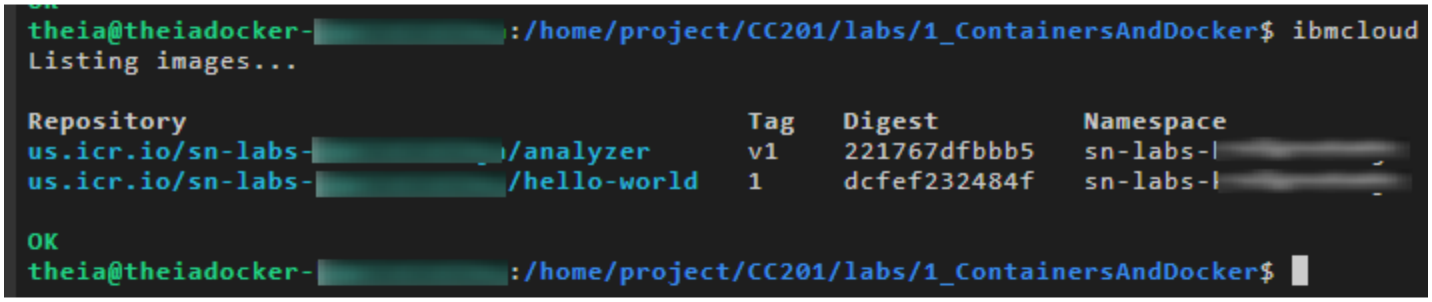
```
ibmcloud cr images
```



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

Repository                                        Tag      Digest         Namespace
us.icr.io/sn-labs-[redacted]/analyzer             v1       221767dfbbb5   sn-labs-[redacted]
us.icr.io/sn-labs-[redacted]/hello-world          1        dcfef232484f   sn-labs-[redacted]
us.icr.io/sn-labsassets/instructions-splitter    latest   2af122cfe4ee   sn-labsassets
us.icr.io/sn-labsassets/pgadmin-theia            latest   0adf67ad81a3   sn-labsassets
us.icr.io/sn-labsassets/phpmyadmin               latest   b66c30786353   sn-labsassets

OK
```

Optionally, to only view images within a specific namespace.

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



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

Repository                                   Tag   Digest         Namespace
us.icr.io/sn-labs-[redacted]/analyzer        v1    221767dfbbb5   sn-labs-[redacted]
us.icr.io/sn-labs-[redacted]/hello-world     1     dcfef232484f   sn-labs-[redacted]

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

You should see your image name in the output.

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

© IBM Corporation. All rights reserved.