**Lab description**

Docker has taken DevOps by storm and ignited interest in containers. Containers offer many of the benefits of virtual machines but in a much more efficient, less resource-intensive system. Containers allow you to package up an application in an isolated environment that can be executed across machines in a reproducible manner. No more "well, it worked on my machine" syndrome when you leverage Docker for your application development and deployment.

In this Lab, you will get up and running with Docker on Linux using an AWS virtual machine. You will work with images from the public Docker registry, run a handful of containers, and create your own image from which to create containers.

### Lab Objectives

Upon completion of this Lab you will be able to:

* Install Docker on Linux using an AWS virtual machine
* Add a user to the Docker group
* Find and use images from the public Docker Registry
* Build your own images using Dockerfiles

### Lab Prerequisites

You should be familiar with:

* What containers are
* Working at the command line in Linux

### Lab Environment

Before completing the Lab instructions the environment will look as follows:

A close-up of a logo

Description automatically generated

After completing the Lab instructions the environment should look similar to:

A diagram of a container

Description automatically generated

# Getting Started with Docker on Linux for AWS

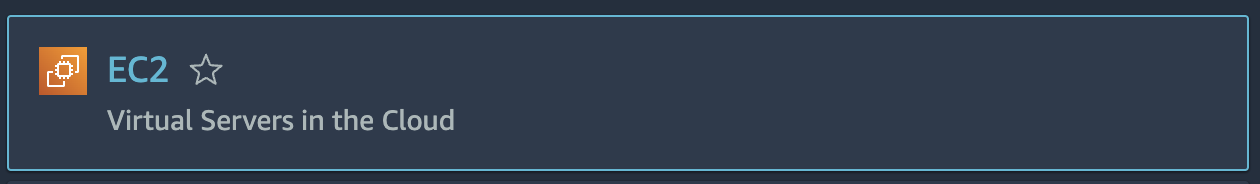
**STEP1:  
  
Introduction**

In next lab step you will use a shell on a Linux host to install Docker.

In this lab step, you will connect to an EC2 instance using EC2 Instance Connect and access a shell.

**Instructions**

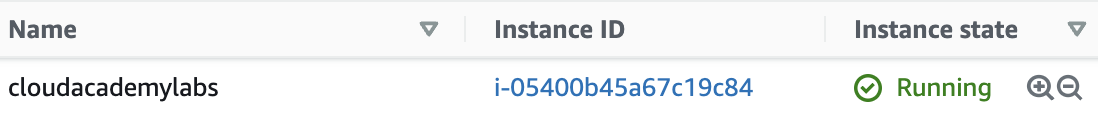
1. In the AWS Management Console search bar, enter *EC2*, and click the **EC2** result under **Services**:



2. To see available instances, click **Instances** in the left-hand menu:

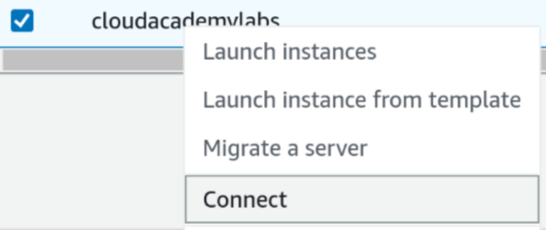
alt

The instances list page will open, and you will see an instance named **cloudacademylabs**:



If you don't see a running instance then the lab environment is still loading. Wait until the **Instance state** is **Running**.

3. Right-click the **cloudacademylabs** instance, and click **Connect**:



The **Connect to instance** form will load.

4. In the form, ensure the **EC2 Instance Connect** tab is selected:

alt

You will see the instance's **Instance ID** and **Public IP address** displayed.

5. In the **User name** textbox, enter *ec2-user*:



*Note*: Ensure there is no space after ec2-user or connect will fail.

6. To open a browser-based shell, click **Connect**:

alt

A browser-based shell will open in a new window ready for you to use.

If you see an error, it is likely, that the environment hasn't finished setting up. When you see the following button show that the lab's cloud environment has fully provisioned, you are ready to use the virtual machine.

Keep this window open, you will use it in later lab steps.

You can also connect to the instance using your preferred SSH client and the PPK (Windows) or PEM (Mac/Linux) key files in the **Credentials** section of this lab.

### Step:2 **Introduction**

You will install Docker using the yum package manager that is available on Amazon Linux. Docker comes in two flavors: Community Edition (CE) and Enterprise Edition (EE). The community edition is open source and available free of charge. It includes the core Docker functionality and was previously called "Docker Engine." The enterprise edition requires a subscription and includes a support package, a certification program to create trusted containers and extensions, and other features. You will be installing the community edition. You will also see how to add your user to the docker group to use the Docker commands without elevating to root permissions.

**Instructions**

 1. Enter the following to install Docker:

[**Copy code**](https://cloudacademy.com/lab/start-with-docker-linux-aws/installing-docker-amazon-linux/?context_id=129&context_resource=lp)

1

sudo yum -y install docker

2. Enter the command below to start Docker as a service:

[**Copy code**](https://cloudacademy.com/lab/start-with-docker-linux-aws/installing-docker-amazon-linux/?context_id=129&context_resource=lp)

1

sudo systemctl start docker

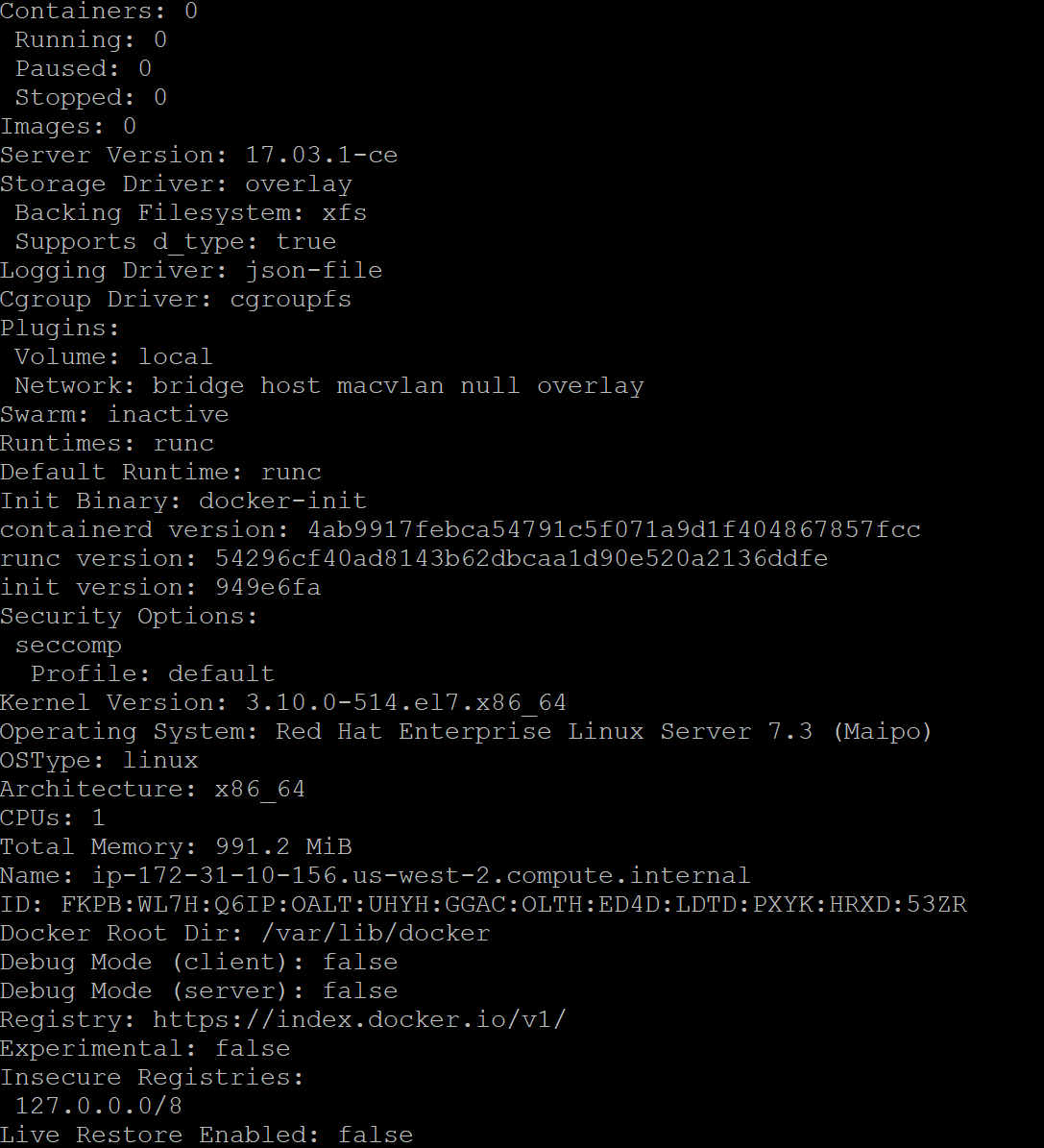
3. Verify Docker is running by entering:

[**Copy code**](https://cloudacademy.com/lab/start-with-docker-linux-aws/installing-docker-amazon-linux/?context_id=129&context_resource=lp)

1

sudo docker info

This will output system-wide information about Docker. The information will resemble the following:



You can see some useful information, such as the number of containers, and the version of the Docker server. Docker adopts a client-server architecture, so the server doesn't have to be running on the same host as the client. In your case, you are using the Docker command line interface (CLI) client to connect to the server, called the Docker daemon.

**Summary**

In this Lab Step, you added the official Docker CE repositories to the yum package manager and installed Docker CE. You also learned your first Docker command to print system-wide information relevant to Docker.

**Step3:**

### Introduction

In the last Lab Step, you used root permissions to run a Docker command. In this Lab Step you will learn how to use Docker commands without elevating to root permissions. This may not always be what you want and should only be performed for trusted users. This is because the actions are equivalent to granting root permissions to a user. It will be convenient in the following Lab Steps to not have to enter sudo for every command.

### Instructions

1. Try entering the same command without using root permission:

[**Copy code**](https://cloudacademy.com/lab/start-with-docker-linux-aws/using-docker-without-root-permission-on-linux/?context_id=129&context_resource=lp)

1

docker info

You will receive a permission denied error message similar to:

alt

By default, the Docker daemon will reject requests from users that aren't part of the **docker** group. If you encounter this message in your travels, you can either use root permission or add your user to the docker group.

2. Verify the docker group exists by searching for it in the groups file:

[**Copy code**](https://cloudacademy.com/lab/start-with-docker-linux-aws/using-docker-without-root-permission-on-linux/?context_id=129&context_resource=lp)

1

grep docker /etc/group

If you don't see a line beginning with "docker:", you will need to add the group yourself by entering:

[**Copy code**](https://cloudacademy.com/lab/start-with-docker-linux-aws/using-docker-without-root-permission-on-linux/?context_id=129&context_resource=lp)

1

sudo groupadd docker

3. Add your user to the docker group:

[**Copy code**](https://cloudacademy.com/lab/start-with-docker-linux-aws/using-docker-without-root-permission-on-linux/?context_id=129&context_resource=lp)

1

sudo gpasswd -a $USER docker

The groups of the currently logged in user is cached, you can verify this by entering groups.

4. You can login again to have your groups updated by entering:

[**Copy code**](https://cloudacademy.com/lab/start-with-docker-linux-aws/using-docker-without-root-permission-on-linux/?context_id=129&context_resource=lp)

1

newgrp docker

Now you will have docker in your list of groups if you enter groups.

Note: It is convenient to not have to terminate your current ssh session by using newgrp, but terminating the ssh session and logging in again will work just as well.

5. Verify that your user can successfully issue Docker commands by entering:

[**Copy code**](https://cloudacademy.com/lab/start-with-docker-linux-aws/using-docker-without-root-permission-on-linux/?context_id=129&context_resource=lp)

1

docker info

Note: if you don't see the system-wide Docker information, you may need to restart the Docker daemon by entering sudo systemctl restart docker.

### Summary

In this Lab Step, you learned about the docker group which has permisison to issue Docker commands. You also learned how to add your user to the group.

**Step 4:**

### Introduction

In this lab step, you will learn about how Docker commands are organized and how to get help for commands on the command line. If you are familiar with using the Linux command line or the git version control system, you will find many of the commands familiar in name and action. Even if you aren't familiar, the commands are organized in an intuitive way that makes learning to use Docker commands as painless as practicable.

The commands are organized into common commands and a more exhaustive list grouped around the management of a specific component of Docker. You use each with a different syntax. For a common command, the usage is:

[**Copy code**](https://cloudacademy.com/lab/start-with-docker-linux-aws/getting-docker-help-from-the-command-line/?context_id=129&context_resource=lp)

1

docker command-name [options]

and the usage for a management group command is:

[**Copy code**](https://cloudacademy.com/lab/start-with-docker-linux-aws/getting-docker-help-from-the-command-line/?context_id=129&context_resource=lp)

1

docker management-group command-name [options]

Most of the common commands can be accessed in the short form or by specifying the management grouping they also fall under. Some commands are only available by specifying the management group. You will get the hang of this and review a couple of the main concepts in the following instructions.

### Instructions

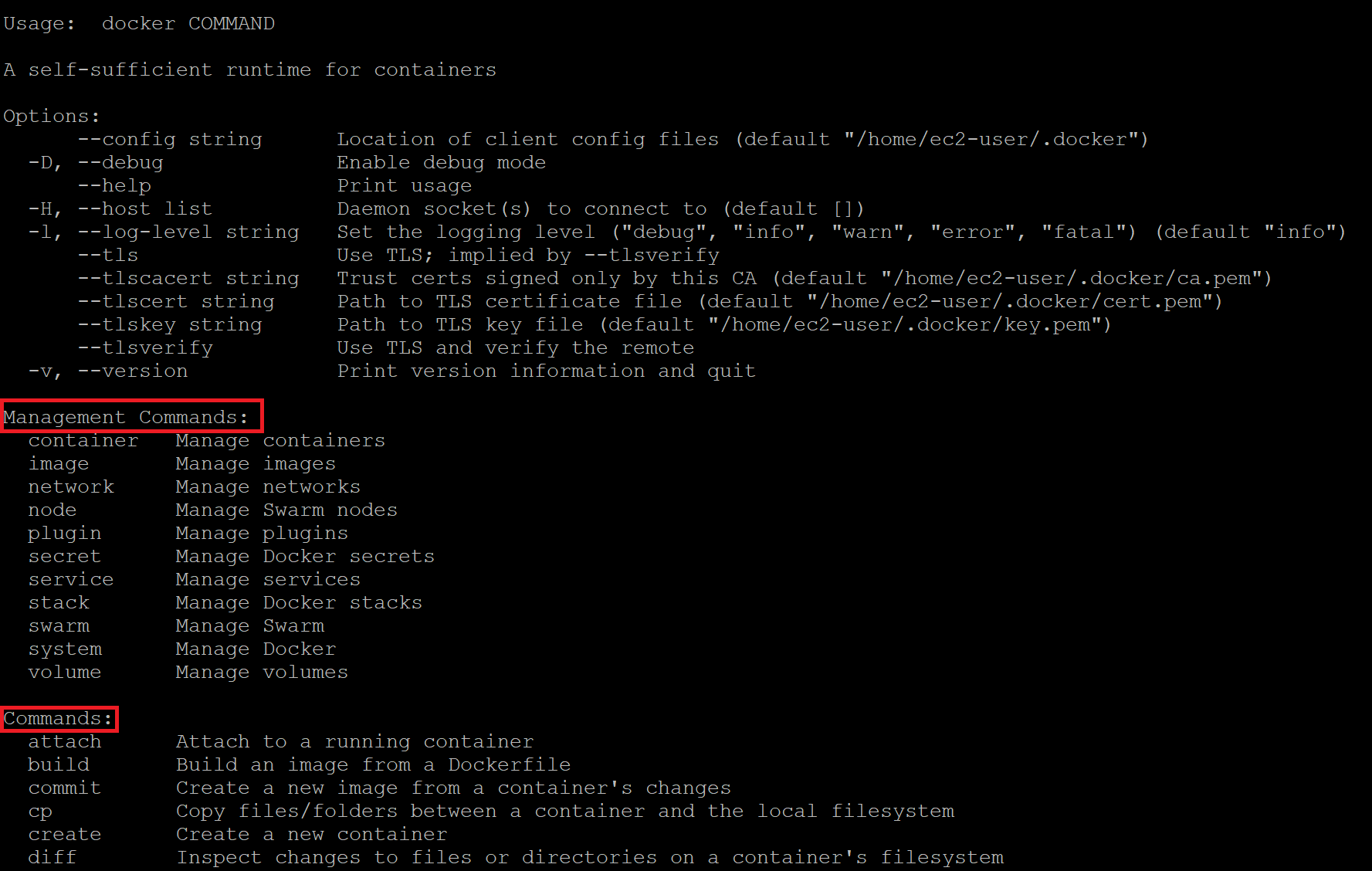
1. To see a list of the commands in Docker, simply enter:

[**Copy code**](https://cloudacademy.com/lab/start-with-docker-linux-aws/getting-docker-help-from-the-command-line/?context_id=129&context_resource=lp)

1

docker --help

The management commands and common commands are highlighted in the image below:



2. Enter the following management command:

[**Copy code**](https://cloudacademy.com/lab/start-with-docker-linux-aws/getting-docker-help-from-the-command-line/?context_id=129&context_resource=lp)

1

docker system info

The output should look familiar. That is because info is a common command that you used in the last Lab Step (docker info) to see the same system-wide information.

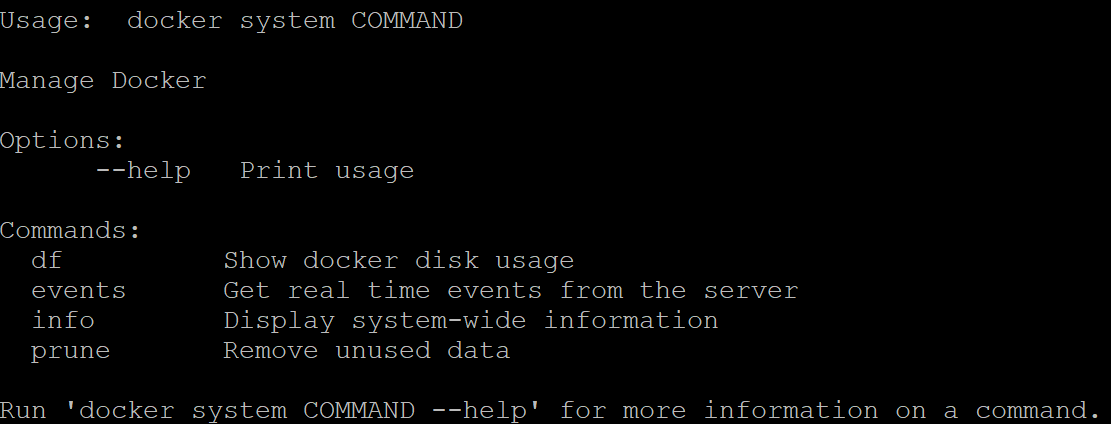
3. To see all of the commands grouped under system, enter:

[**Copy code**](https://cloudacademy.com/lab/start-with-docker-linux-aws/getting-docker-help-from-the-command-line/?context_id=129&context_resource=lp)

1

docker system --help

You will be presented with output similar to:



Notice the info command previously entered.  events is a common command, but df and prune are only available through the system management command.  You can use df to see the disk usage for Docker and prune to clean up unused data.

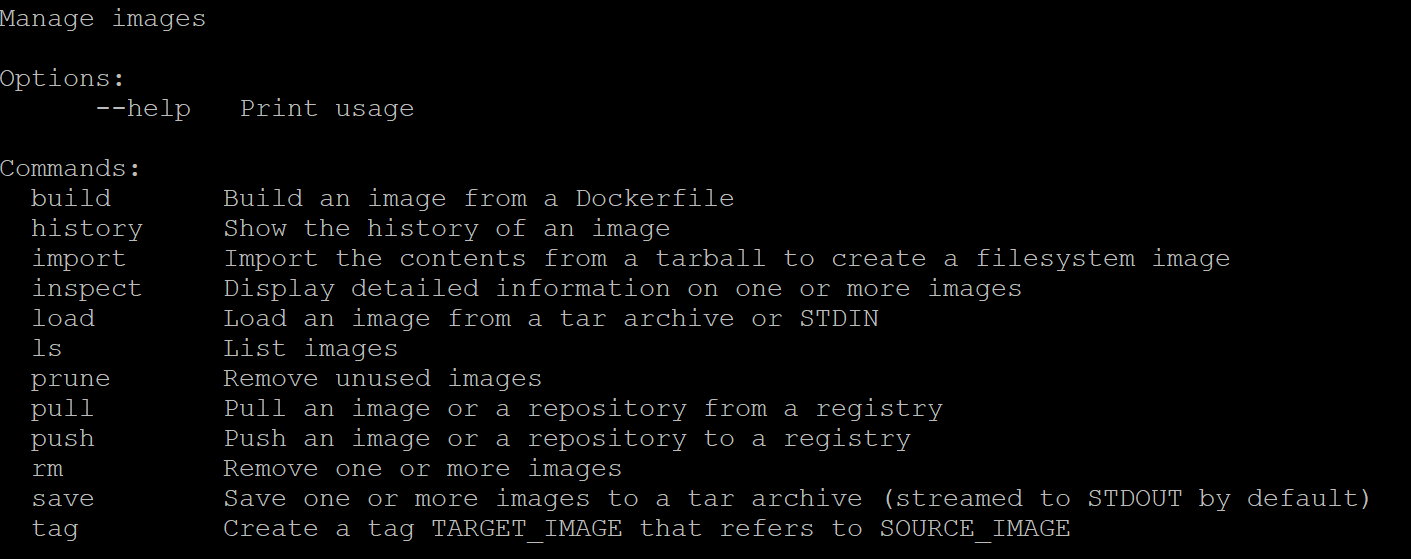
Using the --help you can explore everything there is to know about Docker commands. To prepare for the upcoming Lab Steps, you will now focus in on a few important components.

4. To view the commands grouped with images, enter:

[**Copy code**](https://cloudacademy.com/lab/start-with-docker-linux-aws/getting-docker-help-from-the-command-line/?context_id=129&context_resource=lp)

1

docker image --help



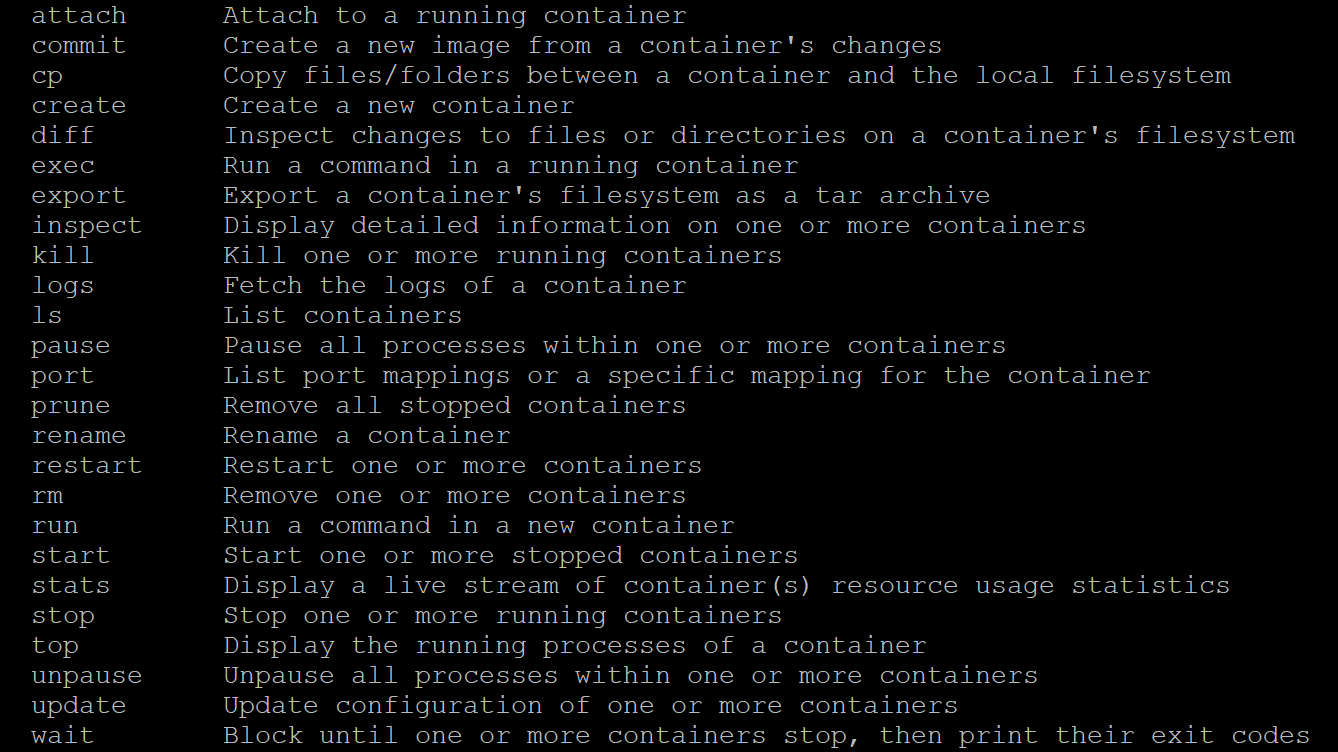
Images are read-only snapshots that containers can be created from. Images are built up in layers, one image built on top of another. Because each layer is read-only, they can be identified with cryptographic hash values computed from the bytes of the data. Layers can be shared between images as another benefit of being read-only. You can, and will later, build your own images. The build command accomplishes that. When you build your own image, you will select a base image to build on top of with your custom application. A pull can be used to pull or download an image to your server from an image registry, while push can upload an image to a registry. Read through the other command descriptions so you are aware of what else is available.

5. To view the commands grouped with containers, enter:

[**Copy code**](https://cloudacademy.com/lab/start-with-docker-linux-aws/getting-docker-help-from-the-command-line/?context_id=129&context_resource=lp)

1

docker container --help



A container is another core concept in Docker. Containers run applications or services, almost always just one per container. Containers run on top of an image. In terms of storage, a container is like another layer on top of the image, but the layer is writable instead of read-only. You can have many containers using the same image. Each will use the same read-only image and have its own writable layer on top. Containers can be used to create images using the commit command, essentially converting the writable layer to a read-only layer in an image.

The relationship between images and containers aside, run is used to run a command on top of an image. A container can be stopped and started again. ls is used to list containers. It is aliased to ps and list as well. Read through the other commands in the list to see what else is available for working with containers.

### Summary

In this lab step, you observed the command layout used by Docker. You also reviewed the core Docker concepts of images and containers as well as frequently used commands for each. You will see these commands in action in the following lab steps.

**Step 5**

### Introduction

As you learned in the previous Lab Step, containers run a command on top of an image. There are many available images to choose from. Images are collected in repositories. A repository can have many versions of an image. Tags are used to manage versions. You will use images from repositories inside the default Docker Registry, Docker Hub. Docker Hub hosts official images as well as community-contributed images. Docker Hub offers free and paid accounts. You get an unlimited amount of public repositories and one free private repository with the free account.

In this Lab Step, you will run your first container. You will learn how to search Docker Hub, pull images, and work with containers running on top of the images. You will get practice with many common commands when working with Docker.

### Instructions

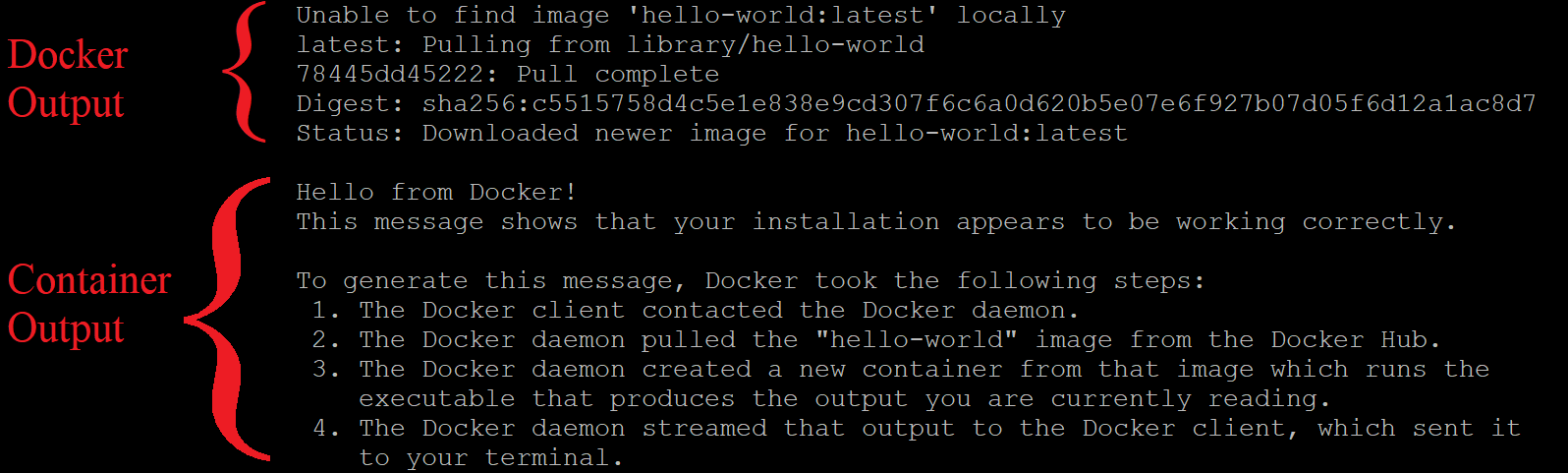
1. Enter the following to see how easy it is to get a container running:

[**Copy code**](https://cloudacademy.com/lab/start-with-docker-linux-aws/running-your-first-docker-container/?context_id=129&context_resource=lp)

1

docker run hello-world

It can be just that easy to run a container! You will see output similar to:



There are two sections to the output: output from Docker as it prepares to run the container and the output from the command running in the container.

Take a look at the Docker output first:

1. In the first line, Docker is telling you that it couldn't find the image you specified, hello-world, on the Docker Daemon's local host. The latest portion after the colon (:) is a tag. The tag identifies which version of the image to use. By default, it looks for the latest version.
2. In the next line, it notifies you that it automatically pulled the image. You could manually perform that task using the command docker pull hello-world. The library/hello-world is the repository it's pulling from inside the Docker Hub registry. library is the account name for official Docker images. In general, images will come from repositories identified using the pattern account/repository.
3. The last three lines confirm the pull completed and the image has been downloaded.

The container output is the output that gets written by the command that runs in the container. Read the output to learn more about how it was produced. You ask, but what was the command? You only specified the image name, hello-world. The image provides a default command in this case which prints the output you see.

2. Re-run the same command:

[**Copy code**](https://cloudacademy.com/lab/start-with-docker-linux-aws/running-your-first-docker-container/?context_id=129&context_resource=lp)

1

docker run hello-world

Notice this time the Docker output is not included. That is because the specific version of the image was found locally and there is no need to pull the image again.

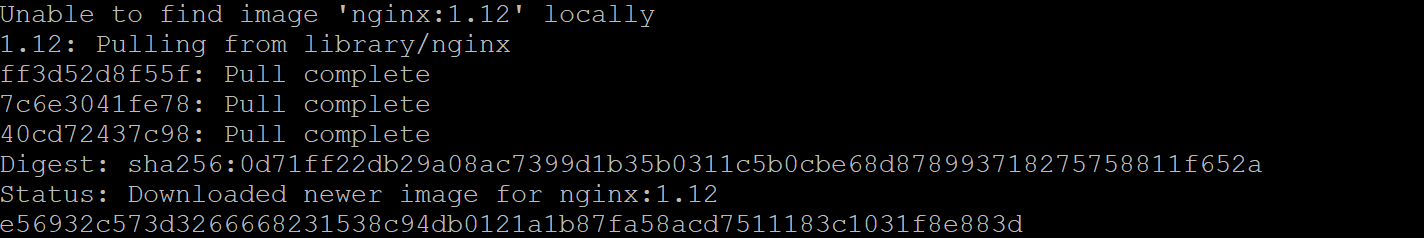
3. Try running a more complex container with some options specified:

[**Copy code**](https://cloudacademy.com/lab/start-with-docker-linux-aws/running-your-first-docker-container/?context_id=129&context_resource=lp)

1

docker run --name web-server -d -p 8080:80 nginx:1.12

This runs the nginx web server in a container using the official nginx image. You will see output similar to:



This time you specified the tag 1.12 indicating you want version 1.12 of nginx instead of the default latest version. There are three **Pull complete** messages this time, indicating the image has three layers. The last line is the id of the running container. The meanings of the command options are:

* --name container\_name: Label the container container\_name. In the command above, the container is labeled web-server. This is much more manageable than the id, 31f2b6715... in the output above.
* -d: Detach the container by running it in the background and print its container id. Without this, the shell would be attached to the running container command and you wouldn't have the shell returned to you to enter more commands.
* -p host\_port:container\_port: Publish the container's port number container\_port to the host's port number host\_port. This connects the host's port 8080 to the container port 80 (http) in the nginx command.

You again used the default command in the image, which runs the web server in this case.

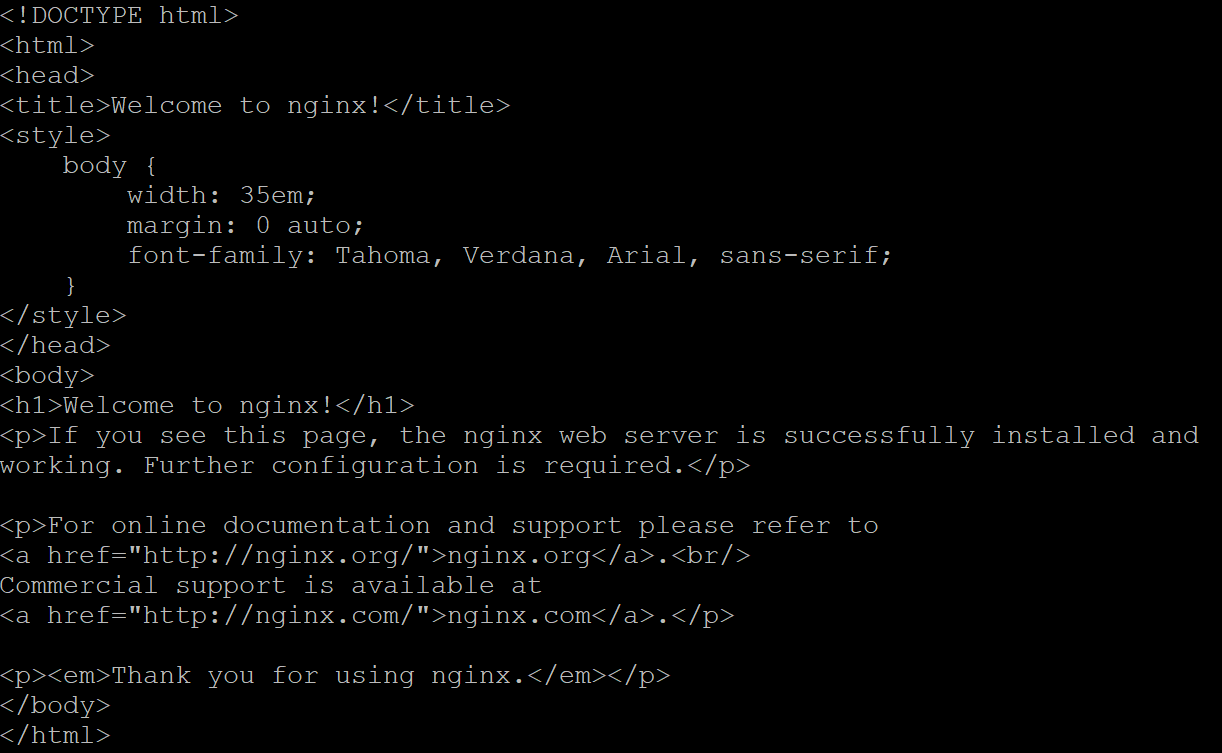
4. Verify the web server is running and accessible on the host port of 8080:

[**Copy code**](https://cloudacademy.com/lab/start-with-docker-linux-aws/running-your-first-docker-container/?context_id=129&context_resource=lp)

1

curl localhost:8080

This command sends an HTTP GET request (a standard web browser request) to localhost port 8080. You will be returned an HTML document, which is the default nginx web page, verifying the nginx server is running in the container:



5. To list all running containers, enter:

[**Copy code**](https://cloudacademy.com/lab/start-with-docker-linux-aws/running-your-first-docker-container/?context_id=129&context_resource=lp)

1

docker ps

alt

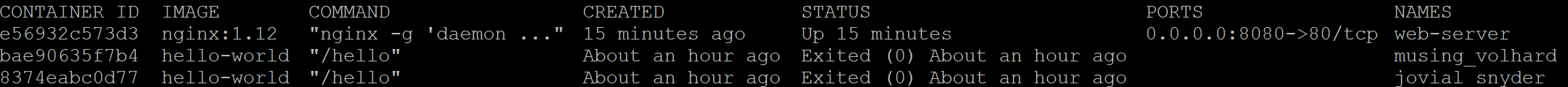
You will see a table with one row below the table column headings, indicating one running process. The (truncated) container id is the same as the last line of the docker run output. One new thing is that you can see the command that is running in the container in the third column. The friendly name you specified is in the last column.

6. Enter the following to see a list of all running and stopped containers:

[**Copy code**](https://cloudacademy.com/lab/start-with-docker-linux-aws/running-your-first-docker-container/?context_id=129&context_resource=lp)

1

docker ps -a



This time you can see the two hello-world containers from the start of the Lab Step. They simply wrote a message and then stopped when the command finished, whereas the nginx server is always listening for requests until you stop it. Notice that Docker automatically assigned random friendly names for the hello-world containers, musing\_volard, and jovial\_snyder in the image above. These names are useful if you need to reference a container that you didn't assign a name to by yourself.

7. To stop the nginx server, enter:

[**Copy code**](https://cloudacademy.com/lab/start-with-docker-linux-aws/running-your-first-docker-container/?context_id=129&context_resource=lp)

1

docker stop web-server

8. Verify the server is no longer running by running:

[**Copy code**](https://cloudacademy.com/lab/start-with-docker-linux-aws/running-your-first-docker-container/?context_id=129&context_resource=lp)

1

docker ps

This will not return any rows in the table. You can also verify that the default nginx web page is no longer being served by running curl localhost:8080 again.

9. To start running the command in the web-server container again, enter:

[**Copy code**](https://cloudacademy.com/lab/start-with-docker-linux-aws/running-your-first-docker-container/?context_id=129&context_resource=lp)

1

docker start web-server

This is different from re-running the original docker run command, which would make a second container running the same command instead of using the stopped container.

10. To see the container's output messages, enter:

[**Copy code**](https://cloudacademy.com/lab/start-with-docker-linux-aws/running-your-first-docker-container/?context_id=129&context_resource=lp)

1

docker logs web-server

alt

Docker logs messages written to standard output and standard error. In the case of nginx, it writes a line for each request that it receives.

11. You can run other commands in a running container. For example, to get a bash shell in the container enter:

[**Copy code**](https://cloudacademy.com/lab/start-with-docker-linux-aws/running-your-first-docker-container/?context_id=129&context_resource=lp)

1

docker exec -it web-server /bin/bash

Which will cause your shell prompt to change to something similar to:

alt

This indicates you are at a shell prompt in the container using the root container user. The -it options tell Docker to handle your keyboard events in the container. Enter some commands to inspect the container environment, such as ls and cat /etc/nginx/nginx.conf. When finished, enter exit to return to the VM ssh shell. Your shell prompt should change to confirm you are no longer in the container bash shell.

You were able to connect to a bash shell because the nginx image has a Debian Linux layer which includes bash. Not all images will include bash, but exec can be used to run any supported command in the container.

12. To list the files in the container's /etc/nginx directory, enter:

[**Copy code**](https://cloudacademy.com/lab/start-with-docker-linux-aws/running-your-first-docker-container/?context_id=129&context_resource=lp)

1

docker exec web-server ls /etc/nginx

This runs the ls command and returns to the ssh shell prompt without using a container shell to execute the command. What commands are supported depends on the layers in the container's image. Up until now you have used two images but don't know how to find more.

13. Stop the nginx container:

[**Copy code**](https://cloudacademy.com/lab/start-with-docker-linux-aws/running-your-first-docker-container/?context_id=129&context_resource=lp)

1

docker stop web-server

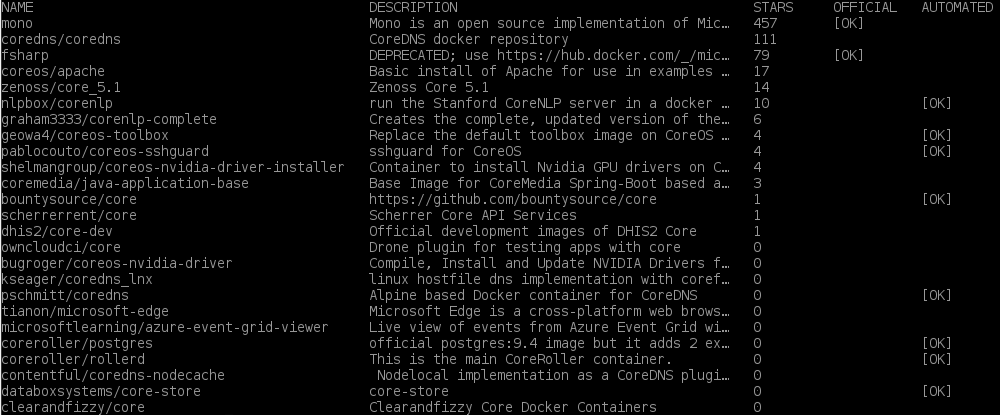
14. Search for an image that you don't know the exact name of, say an image for Microsoft .NET Core, by entering:

[**Copy code**](https://cloudacademy.com/lab/start-with-docker-linux-aws/running-your-first-docker-container/?context_id=129&context_resource=lp)

1

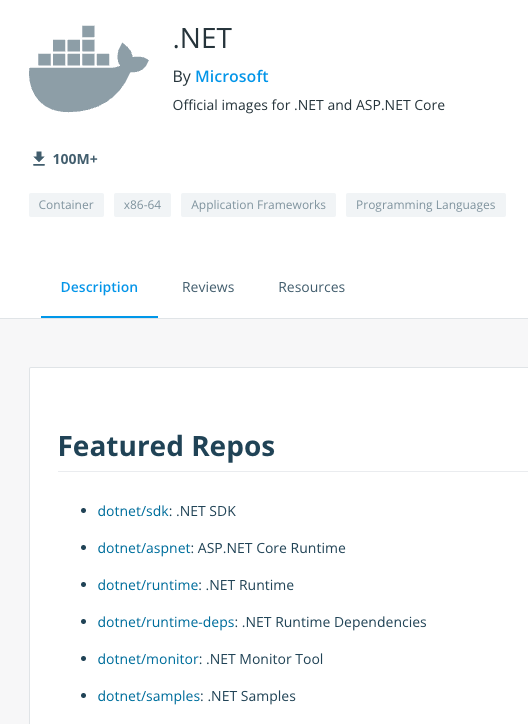
docker search "Microsoft .NET Core"

This searches Docker Hub for images related to the string provided. In this case, the top results related to .NET Core are returned:



This can be useful for recalling the name of an image but not very useful for images that have multiple configuration options. For that you should use the web version of Docker Hub.

15. Navigate to [the following link](https://hub.docker.com/_/microsoft-dotnet). You will be viewing the web page of the .NET repository:



You will find all the supported tags along with helpful documentation for using the images. You will usually find useful documentation and examples for images on Docker Hub. You will notice a Dockerfile beside each group of tags on the dotnet page. A Dockerfile is a set of instructions for creating an image and it is what you will dive into in the next Lab Step.

### Summary

In this Lab Step, you explored the commands related to working with containers. You also ran a few containers. You learned the steps Docker takes to run a container and how to find Docker Hub images to create containers from.

### **Step 6:** Introduction

Docker containers run on top of images. You have seen how to use images on Docker's public registry, the Docker Hub. There are many different images available. It is worth trying to find existing images when you can. Inevitably, you will need to create your own images when making your own applications. In that case, you will still want to invest time into finding the right base layer to add your own layer(s) on top of.

There are a couple of ways to make images. You can use docker commit to create an image from a container's changes. The changes may come from using exec to open a shell in the container like in the previous Lab Step. The other method is using a Dockerfile. A Dockerfile is easier to maintain, easier to repeatedly create images from, and distributions easier. You will create a Dockerfile in this Lab Step. Just know that it is possible to create equivalent images using commits.

Dockerfiles specify a sequence of instructions. Instructions can install software, expose network ports, set the default command for running a container using the image, and other tasks. Instructions can really handle anything required to configure the application. Many of the instructions add layers. It is usually a good idea to keep the number of layers to a reasonable number. There is overhead with each layer, and the total number of layers in an image is limited. When the Dockerfile is ready, you can create the image using the docker build command.

You will see how all of this comes together by creating an image of a Python Flask web app that randomly chooses what type of Cloud Academy content you should look at next. The choice of Python as the example app is arbitrary. You will not focus on the specifics of the programming language. You should be able to repeat the process for any other programming language by following a similar process. Whatever programming language or framework you are working with, you should consult the Docker Hub documentation for the image, as it will usually include advice on how to structure your Dockerfile.

### 

### Instructions

1. Install Git:

[**Copy code**](https://cloudacademy.com/lab/start-with-docker-linux-aws/creating-your-first-docker-image/?context_id=129&context_resource=lp)

1

sudo yum -y install git

You will clone a code repository with the Flask app using Git.

2. Clone the code repository to your virtual machine:

[**Copy code**](https://cloudacademy.com/lab/start-with-docker-linux-aws/creating-your-first-docker-image/?context_id=129&context_resource=lp)

1

git clone https://github.com/cloudacademy/flask-content-advisor.git

3. Change to the apps directory:

[**Copy code**](https://cloudacademy.com/lab/start-with-docker-linux-aws/creating-your-first-docker-image/?context_id=129&context_resource=lp)

1

cd flask-content-advisor

4. Create and start editing a Dockerfile using the nano text editor:

[**Copy code**](https://cloudacademy.com/lab/start-with-docker-linux-aws/creating-your-first-docker-image/?context_id=129&context_resource=lp)

1

nano Dockerfile

Note: The name of the Dockerfile must be Dockerfile with an uppercase "D" and all other letters lowercase

5. Enter the following in the file:

[**Copy code**](https://cloudacademy.com/lab/start-with-docker-linux-aws/creating-your-first-docker-image/?context_id=129&context_resource=lp)

1

2

3

4

5

6

7

8

9

10

11

12

*# Python v3 base layer*

FROM python:3

*# Set the working directory in the image's file system*

WORKDIR /usr/src/app

*# Copy everything in the host working directory to the container's directory*

COPY . .

*# Install code dependencies in requirements.txt*

RUN pip install --no-cache-dir -r requirements.txt

*# Indicate that the server will be listening on port 5000*

EXPOSE 5000

*# Set the default command to run the app*

CMD [ "python", "src/app.py" ]

The lines beginning with # are comments and explain what each instruction is doing. Make sure you read the comments. Some highlights are:

* FROM sets the base layer image
* COPY . . copies all of the files in the code repository into the container's /usr/src/app directory
* RUN executes a command in a new layer at the top of the image
* EXPOSE only indicates what port the container will be listening on, it doesn't automatically open the port on the container
* CMD sets the default command to run when a container is made from the image

There are more instruction types, but this lab will only focus on those mentioned. After completing the lab, you can review all of the instructions available in Dockerfiles at the [Dockerfile reference web page](https://docs.docker.com/engine/reference/builder/" \t "_blank).

6. Once you have all of the instructions in the Dockerfile, press Ctrl+x, enter Y when prompted to save, and press enter.

You are now back at the shell prompt.

7. Build the image from the Dockerfile:

[**Copy code**](https://cloudacademy.com/lab/start-with-docker-linux-aws/creating-your-first-docker-image/?context_id=129&context_resource=lp)

1

docker build -t flask-content-advisor:latest .

The -t tells Docker to tag the image with the name flask-content-advisor and tag latest. The . at the end tells Docker to look for a Dockerfile in the current directory. Docker will report what it's doing to build the image. Each instruction has its own step. Steps one and four take longer than the others. Step one needs to pull several layers for the Python 3 base layer image and Step four downloads code dependencies for the Flask web application framework. Notice that each Step ends with a notice that an intermediate container was removed. Because layers are read-only, Docker needs to create a container for each instruction. When the instruction is complete, Docker commits it to a layer in the image and discards the container.

8. Record your VM's public IP address:

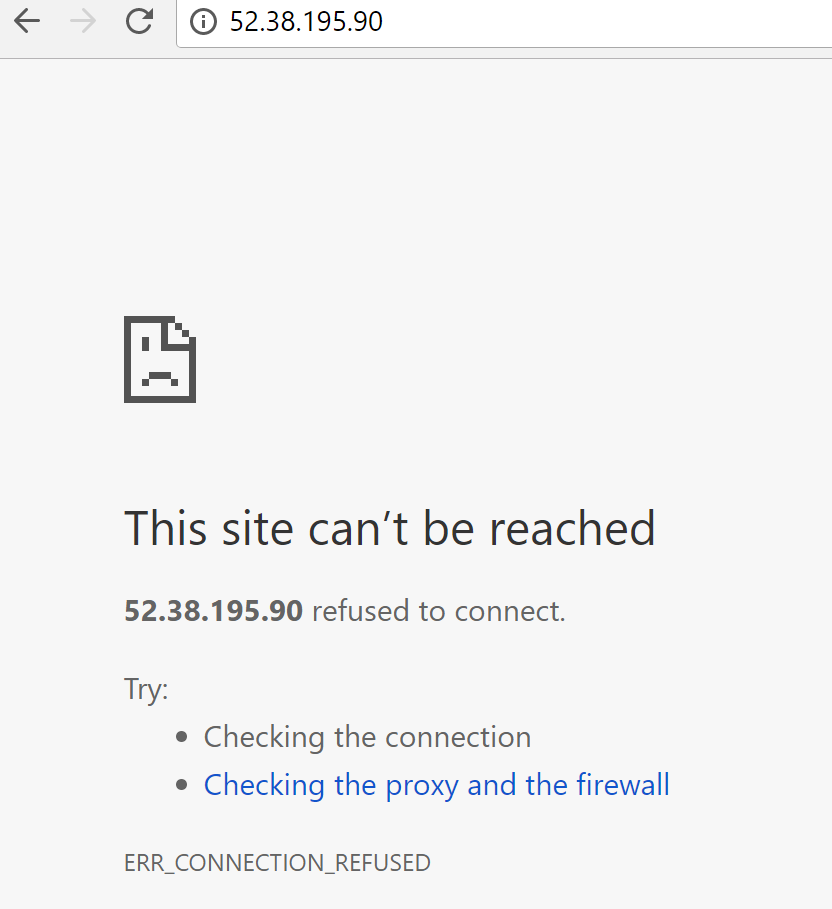
[**Copy code**](https://cloudacademy.com/lab/start-with-docker-linux-aws/creating-your-first-docker-image/?context_id=129&context_resource=lp)

1

curl ipecho.net/plain; echo

You will need your IP to test that the web app is available with your browser.  The echo is only to put the shell prompt on a new line.

9. Open a new browser tab and navigate to the public IP address you just recorded. The browser will fail to load anything since no server is running yet. Keep the tab open for later.



Note: Your public IP will probably be different from the one in the image.

10. Now you can run a container using the image you just built:

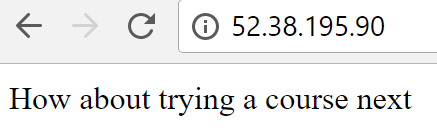
[**Copy code**](https://cloudacademy.com/lab/start-with-docker-linux-aws/creating-your-first-docker-image/?context_id=129&context_resource=lp)

1

docker run --name advisor -p 80:5000 flask-content-advisor

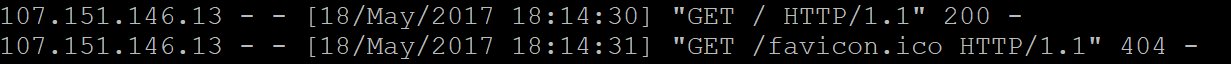
This runs a container named advisor and maps the container's port 5000 to the host's port 80 (http). This time you didn't include -d to run in detached mode.  That is why you see output and you don't have the shell prompt returned to you. If you did run with -d, you could get the same information from docker logs.

11. Return to your browser tab with the public IP and refresh the page:



Now the advisor will be in fine form recommending what content to tackle next after you complete this lab.

12. Return to the shell and notice that some web requests will have been logged corresponding to your browser requests:



There are two requests because the browser automatically requests a favicon in addition to the page content.

### Summary

In this Lab Step, you created your own image using a Dockerfile. You also ran a container using the image that was built.