**Docker Images and Containers**

Completion requirements

**Done: View**

**5. Exercise: Working with Docker Images and Containers**

**What You'll Do**

* In the previous exercise, we took a Django application, built a Docker image for it, then created a Docker container to run the application.
* This exercise summarizes frequently used CLI commands for working with Docker images and containers.

**Instructions**

**Prerequisites**

* This exercise will use the Docker image and container used in the previous exercise.
* Confirm that you have the Docker image from the previous exercise listed in the Images section of either Docker Desktop or VS Code's Docker Explore panel.
* In this exercise, the directory location from which you perform CLI commands does not matter. All the commands are global (able to be used from anywhere).
* You can choose to use any bash terminal - VS Code's integrated terminal, or Git Bash (Windows), or Terminal (macOS).

**Docker image commands**

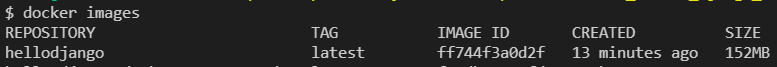
* Open a bash terminal. The directory location does not matter.

**List Docker images**

* To list the available Docker images, run the command:

docker images

* You should see output that includes the **hellodjango**image:



* The columns include the following information about the listed Docker images:

**REPOSITORY**: Names of images

**TAG**: Tags can be used to logically tag Docker images

**IMAGE ID**: Unique identifier for each Docker image

**CREATED**: Time since Docker image was created

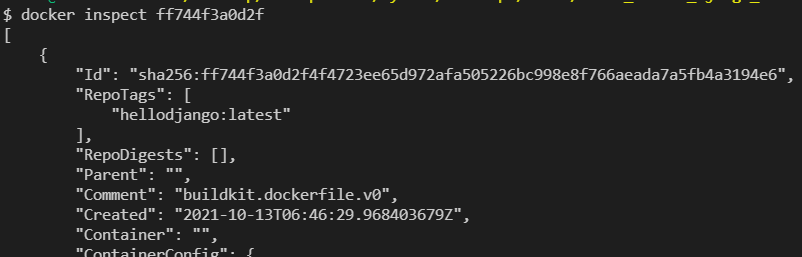
**SIZE**: Size of Docker image

* Note and copy the Image ID of the **hellodjango**image. We will use this in a moment.

**Inspect images or containers**

* The **inspect**command shows the configuration details of an image or container, along with network and other information.
* Run the command below, using the Image ID you copied earlier for your own image:

docker inspect <ImageID>

* The beginning of the output from this command should look like this:  
  
* We will look more closely at the output of the **docker inspect** command in a later lesson.

**Create and start Docker container from image**

* The following command will create a Docker container (an instance of an image) from our image and start it:

docker run --rm -d  -p 8000:8000/tcp hellodjango:latest

* + \**Remember this command for the workshop*
  + NOTE: This is the same command run by VS Code if you use the **VS Code Command Palette -> Docker Images: Run -> hellodjango -> latest** option, as you did in the previous exercise. We are simply running it ourselves from the command line, this time.
  + The **--rm** flag tells Docker to remove this container automatically when it exits with **docker stop** or **docker kill**, which you will learn about later in this exercise. At that time, we will also look at what happens when we don't use this flag.
  + The **-d** flag tells Docker to run the container in detached mode (in the background, without showing any log messages to the terminal) and to print the container ID.
  + The **-p** flag publishes the container's ports. We will discuss ports in more detail next week.
  + You can see view the full documentation on the docker run command here: <https://docs.docker.com/engine/reference/commandline/run/>
* You should see output similar to this:  
  

**Get information**

List Docker containers

* To list the Docker containers that are currently running, run the command:

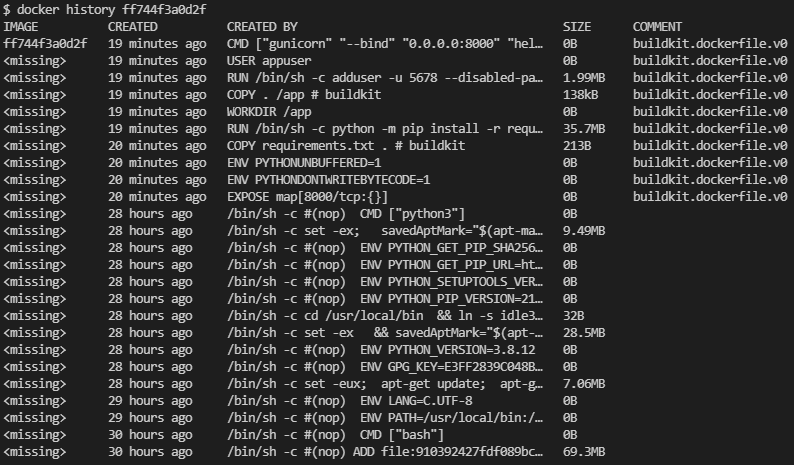
docker ps

* The below output should show a container built from the **hellodjango**Docker image, as in the example below:  
  
* Notice that in the **PORTS** column for this container, the port **8000** is listed. The acronym **tcp** stands for **Transmission Control Protocol** and is a standard data transfer protocol.
* This is the port used to access the Django application running in this Docker container.
* Also take note of the value for **CONTAINER ID**, which can be used in other commands.
* You can also try out the command **docker ps -a**, which lists all Docker containers on the system, whether they are running or not.

**View Docker history**

* With the **history**command, you can view all of the instructions that were executed with an image when running the container, including those that were run automatically by the Dockerfile.
* Viewing the history allows you to view all the instructions that were executed and make sure they are correct. This can be helpful in the event an error occurs.
* To view the history, run the command:

docker history <ImageID>

* Use the IMAGE ID from the **docker images** command.
* The output will look similar to this:  
  
* Notice how it shows all of the commands that were declared in the Dockerfile, which were executed when starting the container.

**List top processes**

* We can list the top processes that are running within a container.
* To do this, run the command:

docker top <ContainerID>

* Use the Container ID from the **docker ps** command.
* The output will show the top processes running in the container.
* You should expect to see two Django **wsgi** processes running. The wsgi process is the entry point to run the Django web application. It is normal to see two of these processes.

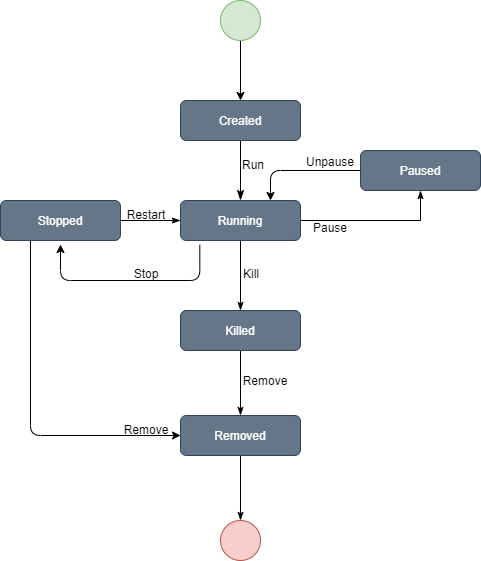
**View statistics on running container**

* We can also get a live stream of statistics on the running Docker container.
* For example, we can get a sense of the CPU usage, memory usage, and other operating system resource usage.
* To do this, run the command:

docker stats <ContainerID>

* You should see output similar to this:  
  
* Since this is a live feed, it will show the current stats, then continuously refresh them on your screen.
* Try leaving the feed open, and go to [http://127.0.0.1:8000](http://127.0.0.1:8000/) in your web browser. You should see the value in the **NET I/O** column increase.
* Use **Ctrl-c**stop the live feed.
* You can also try using the same command with the**--no-stream** flag to only pull the most recent result.

**Docker lifecycle commands**

* The next series of commands allow us to change the status of the Docker container in the Docker container's lifecycle.
* An illustration of the Docker container lifecycle and how the commands fit in the lifecycle is shown below.   
  
* We already have seen the **docker run**command in use when we created and started the container.

**Pausing and unpausing the container**

* The pause command pauses the processes in a running Docker container.
* To pause the Docker container, run the command:

docker pause <ContainerID>

* When used successfully, the command will echo back your Container ID to you.
* Check the status of the container with the **docker ps** command, and it should now show as **(Paused)** in the **STATUS** column, as in this example:  
  
* To unpause a Docker container, run the command:

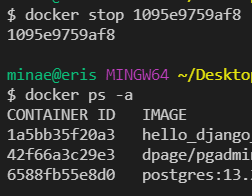
docker unpause <ContainerID>

* The successful execution of this command will also echo your Container ID back to you.
* Check **docker ps** again to confirm that the container is no longer showing **(Paused)** under **STATUS**.

**Stopping the container**

* To stop a running container, run the below command along with the container ID:

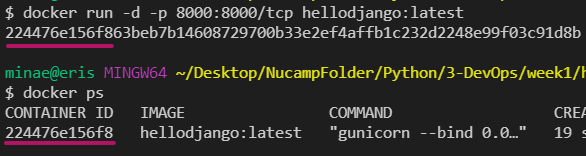
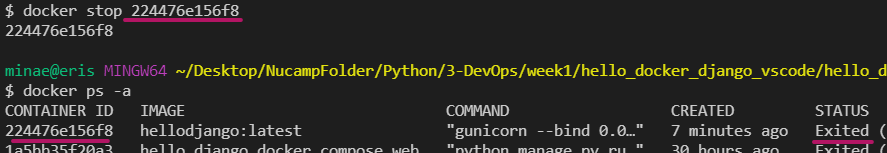
docker stop <ContainerID>

* It will echo the Container ID back to you. Check **docker ps** to verify that the container is no longer listed.
* Also check **docker ps -a**. This should show both started and stopped containers.
* However, pay attention here to the fact that this does *not* show the container you just stopped, as in the example below:  
  
* You can see that the container with the ID of 1095e9759af8 was stopped in the above example, but is not listed in **docker ps -a**, which should list all containers, including stopped ones.
* Why is this?
* Recall that when you started the container using **docker run** earlier in this exercise, we used the flag **--rm**. This flag **removes**a container when it is stopped (or killed). Let's try using docker run again, but *without*the **--rm** flag.

**Docker run without --rm flag**

* Run this command:

docker run -d -p 8000:8000/tcp hellodjango:latest

* Recall that the **-d** flag runs the container in detached mode and prints the Container ID.
* Run **docker ps** and notice that the Container ID is the first 12 characters of the resulting output from **docker run**:  
  
* Once again, note your Container ID (the one with the **IMAGE**of **hellodjango:latest**) and use it to stop the container with **docker stop <ContainerID>:**  
  ****
* If you check **docker ps** afterward, the container will not be listed.
* However, if you check **docker ps -a**, you should see the stopped container there, listed with a **STATUS**of **Exited**:  
  
* This is because when we started the container originally with the **docker run** command, we did not include the **-rm** flag.

**Docker start & restart**

* If a container is stopped but not removed, we can restart it using the **docker start**command:
* To restart the Docker container, run the same command you did the first time you started the container from the image:

docker start <ContainerID>



* If we need to stop then start a container in one command, the command **docker restart <ContainerID>** will accomplish this:

docker restart <ContainerID>

**Kill the container**

* The kill command is used to send something called a KILL signal to a container to stop it immediately.
* This is a command that you would use if **docker stop** did not work. **docker stop** would be preferred to this. By analogy, think about turning your computer off via the operating system's Shut Down option (docker stop), versus pushing the power button (docker kill).
  + This [StackOverflow discussion](https://stackoverflow.com/questions/28339469/which-one-should-i-use-docker-kill-or-docker-stop) sheds more light on this.
* Be sure to use the current Container ID. This should be the Container ID listed in **docker ps,**for the container using the **hellodjango:latest** **i**mage.
* Run the command:

docker kill <ContainerID>

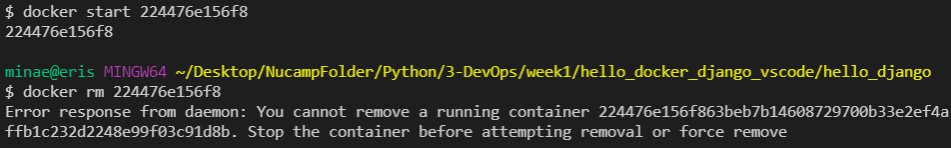
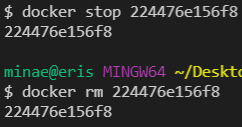


* Like **docker stop**, when the container was started without the**--rm** flag, **docker kill** will not remove the container.
* You should be able to enter the **docker ps -a**command and see the killed container still listed there:  
  

**Remove Docker container**

* To remove a Docker container means that the Docker container will be deleted entirely.
* You will not be able to view it in **docker ps -a**, and you will not be able to **start** or **restart** it.
* For practice, first use what you have learned in the previous sections to start the Docker container that you killed in the previous section.
* Then run the following command to remove the container entirely:

docker rm <ContainerID>

* You will see an error like this:  
  
* Go ahead and **stop**the container, then run the **rm** command again:  
  
* You can confirm with **docker ps -a** that the container no longer exists.
* Even though the Docker container was removed, the Docker image still exists in the image repository.
* You can confirm this as well by running the command **docker images** once more:  
  
* Another instance (container) of the Docker image can be made at any time using the **docker run** command.

**Remove image from Docker repository**

* We can also remove a Docker image from the Docker image repository on our system.
* Below is the command to do this.
  + **NOTE**: We use this image in future exercises, so if you remove it, you will need to re-build it later, using the instructions for building an image in the previous exercise. **Do not actually remove this image unless you wish to rebuild it afterward for practice**.
* To remove the image, run this command, using the Image ID shown in **docker images**:

docker rmi <ImageID>

* The below output shows the Docker image being removed:   
  
* After removing the Docker image, you can run**docker images** once more to confirm that it no longer appears in the repository.
* You can also confirm this in the Docker panel of VS Code, or in Docker Desktop.
* If you remove it, make sure to practice rebuilding it again so that it is available for next week.