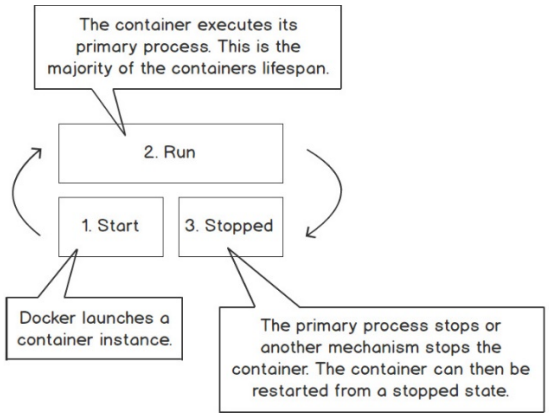
RUN AND MANAGE CONTAINERS

Overview

In this lab, you will learn the fundamentals of containerization, the benefits of running applications in containers, and the basic Docker life cycle commands to manage containerized instances.



Procedure

Running containers

- Enter the docker container run command in a Bash terminal. This instructs Docker to run a container called hello-world:

  $ docker container run hello-world

  In your own words, explain what just happened?

- Use the docker container ls command to see what containers are running on your system. In your Bash terminal, type the following command:

  $ docker container ls

  What output do you get on your screen?

- Use the docker container ls -a command to display all the containers, even the stopped ones:

  $ docker container ls -a

  What output do you get this time on your screen?

- You can query your system to see what container images Docker cached locally. Execute the docker image ls command to view the local cache:

$ docker image ls

What is the image tag associated with your hello-world container?

Managing container life cycles

- In your terminal, execute the docker image pull command to download the Ubuntu 18.04 container image:

  $ docker image pull ubuntu:18.04

  What does the 18.04 represent in the preceding command?

- Use the docker image pull command to download the Ubuntu 19.04 base image:

  $ docker image pull ubuntu:19.04

- Use the docker image ls command to confirm that the container images are downloaded to the local container cache:

  $ docker image ls

  What are the contents of the local container cache?

- Before running the new images, use the docker image inspect command to get verbose output about what makes up the container images and how they differ. In your terminal, run the docker image inspect command and use the image ID of the Ubuntu 18.04 container image as the main argument:

  $ docker image inspect IMAGE ID

  When was this image created?

- Now Inspect the Ubuntu 19.04 image. Run the docker image inspect command in the Ubuntu 19.04 container image ID:

  $ docker image inspect IMAGE ID

  When was this image created?

  What can you conclude after inspecting both container images?

- Use the docker container run command to start an instance of the Ubuntu 18.04 container in daemon mode or in background (hint: use -d flag):

  $ docker container run -d ubuntu:18.04

- Check the status of the container using the docker container ls -a command:

  $ docker container ls -a

  What is the status of your ubuntu container and explain?

- Run the docker container run command again, passing in the -i flag to make the session interactive (expecting user input), and the -t flag to allocate a pseudo-tty handler to the container. This will

essentially link the user's terminal to the interactive Bash shell running inside the container. You can also give the container a human-readable name by passing in the --name flag. Type the following command in your Bash terminal:

$ docker container run -i -t -d --name ubuntu1 ubuntu:18.04

- Execute the docker container ls -a command again to check the status of the container instance:

  $ docker container ls -a

- You now have an Ubuntu container up and running. You can run commands inside this container using the docker container exec command. Run the exec command to access a Bash shell, which will allow us to run commands inside the container. Similar to docker container run, pass in the -i and -t flags to make it an interactive session. Also pass in the name or ID of the container, so that Docker knows which container you are targeting. The final argument of docker exec is always the command you wish to execute. In this case, it will be /bin/bash to start a Bash shell inside the container instance:

  $ docker container exec -it ubuntu1 /bin/bash

  What is the output of your terminal? What is the hostname of your container and where is it taken from?

- From inside the container, you are very limited in terms of what tools you have available. Unlike a VM image, container images are extremely minimal in terms of the packages that come preinstalled. The echo command should be available, however. Use echo to write a simple message to a text file:

  # echo "Hello world from ubuntu1" > hello-world.txt

- Run the exit command to exit from the Bash shell of the ubuntu1 container. You should return to your normal terminal shell:

  # exit

- Now create a second container called ubuntu2 that will also run in your Docker environment using the Ubuntu 19.04 image:

  $ docker container run -i -t -d --name ubuntu2 ubuntu:19.04

- Run docker container exec to access a shell of this second container. Remember to use the name or container ID of the new container you created. Likewise, access a Bash shell inside this container, so the final argument will be /bin/bash:

  $ docker container exec -it ubuntu2 /bin/bash

- Run the echo command inside the ubuntu2 container instance to write a similar hello-world-type greeting:

  # echo "Hello-world from ubuntu2!" > hello-world.txt

- Currently, you have two Ubuntu container instances running in your Docker environment with two separate hello-world greeting messages in the home directory of the root account. Use docker container ls to see the two running container images:

  $ docker container ls

- Instead of using docker container exec to access a shell inside our containers, use it to display the output of the hello-world.txt files you wrote by executing the cat command inside the containers:

  $ docker container exec -it ubuntu1 cat hello-world.txt

  Run the same cat command in the ubuntu2 container instance:

  $ docker container exec -it ubuntu2 cat hello-world.txt

  Explain in detail what just happened? Why were you moved back to the context of your main terminal in both cases?

- In a similar manner to that you used to execute commands inside our running containers, you can also stop, start, and restart them. Stop one of your container instances using the docker container stop command. In your terminal session, execute the docker container stop command, followed by the name or container ID of the ubuntu2 container:

  $ docker container stop ubuntu2

- Use the docker container ls command to view all running container instances:

  $ docker container ls

  Which container is up and running?

- Execute the docker container ls -a command to view all container instances, regardless of whether they are running, to see your container in a stopped state:

  $ docker container ls -a

  What is the output of the preceding command?

- Use the docker container start or docker container restart command to restart the container instance:

  $ docker container start ubuntu2

  This command will return no output, although some versions of Docker may display the container ID or the name of the container.

- Verify that the container is running again by using the docker container ls command:

  $ docker container ls

  What can you conclude about the “Status” of the newly started/restarted container?

- The final stage of the container management life cycle is cleaning up the container instances you created. Use the docker container stop command to stop the ubuntu1 container instance:

  $ docker container stop ubuntu1

- Perform the same docker container stop command to stop the ubuntu2 container instance:

  $ docker container stop ubuntu2

- When container instances are in a stopped state, use the docker container rm command to delete the container instances altogether. Use docker container rm followed by the name or container ID to delete the ubuntu1 container instance:

  $ docker container rm ubuntu1

  This command will return no output, although some versions of Docker may return the container ID.

  Perform this same step on the ubuntu2 container instance:

  $ docker container rm ubuntu2

- Execute docker container ls -a to see all containers.

  $ docker container ls -a

  What is the output of your display?

  What happened to all the other containers you worked with?

- To completely reset the state of our Docker environment, delete the base images you downloaded during this lab as well. Use the docker image ls command to view the cached base images:

  $ docker image ls

- Execute the docker image rmi command followed by the image ID to delete the first image ID:

  $ docker image rmi IMAGE ID

  Similar to docker image pull, the rmi command will delete each image and all associated layers.

  What happened when you executed the preceding command?

Note:

To streamline the process of cleaning up your environment, Docker provides a prune command that will automatically remove old containers and base images:

$ docker system prune -fa

Executing this command will remove any container images that are not tied to an existing running container, along with any other resources in your Docker environment.