Lab 2. Getting Started with Docker

By the end of this lab exercise, you should be able to:

- Launch containers with Docker.
- List common options used with the **`docker container run`** command.
- Run web applications and access those with port mapping.
- Manage container lifecycle and learn how to debug them.
- Cleanup

Validate the Setup

Begin by checking the Docker version you have installed using the following command:

```
docker version
```

Validate further by running a smoke test:

```
docker run hello-world
```

This should launch a container successfully and show you a **`hello world`** message.
This smoke test validates the following:

- You have Docker client installed.
- Docker daemon is up and running.
- Docker client is correctly configured and authorized to talk to Docker daemon.
- You have non-blocking internet connectivity.
- You are able to pull an image from Docker registry and run a container with it.

Launching Your First Container

Before you begin launching your first container, open a new terminal and run the following command to analyze the events of the Docker daemon:

```
docker system events
```

When you launch the above command, you may not see any output. Keep this window open, and it will start streaming events as you proceed to use the Docker CLI in another window.

Now, launch your first container with the following command:

```
docker container run centos ps
```

where,

- **docker** is the command line client.
- The **container run** command is used to launch a container. You can alternatively just use the **run** command here.
- CentOS is the image.
- **ps** is the actual command which is run inside this container.

Create a few more containers with the same image but with different commands as follows:

```
docker container run centos uptime
```

```
docker container run centos uname -a
```

```
docker container run centos free
```

You can check the events in the window where you have started running the **docker system events** command earlier. It shows what's happening on the Docker daemon side.

Listing Containers

Check your last run container using the following commands:

```
docker ps -l
```

```
docker ps -n 2
```

```
docker ps -a
```

where,

- **-l**: last run container
- **-n xx**: last xx number of containers
- **-a**: all containers (even if they are in ‘stopped’ state)

Learning About Images

Containers are launched using images which are pulled from the registry. Observe the following

command:

```
docker container run centos ps
```

Here the image name is **centos** which can actually be expanded to:

```
registry.docker.io/docker/centos:latest
```

where the following are the fields:

- registry: **registry.docker.io**
- namespace: **docker**
- repo: **centos**
- tag: **latest**

Use the following commands to list your images from your local machine and pull the **nginx** image from DockerHub. Examine the layers of an image with the **history** command:

```
docker image ls
```

```
docker image pull nginx
```

```
docker image history nginx
```

Default Run Options

You have learned how to launch a container. However, this container is ephemeral and exits immediately after running the command. To make the container persistent, and in order to be able to interact with it, lets try adding a couple of new options:

```
docker run -it centos bash
```

where,

- **-i**: (interactive) provides standard in to the process running inside the container.
- **-t**: provides a pseudo-terminal in order to interact.

You could further add the **--rm** options to ensure the container is deleted when stopped:

```
docker run --rm -it alpine sh
```

Note: Use ^d to come out of the shell opened within the container.

Another useful and important option is **-d**, which will launch the process in a contained environment and detach from it. This allows the container to keep running in the background:

```
docker container run -idt --name redis redis:alpine
```

Launching and Connecting to Web Applications

So far, you have launched a few simple containers, with one-off commands. It's time to now learn how to launch a web application, and learn how to connect to it.

You could launch a container with the **nginx** web server as follows. Observe the new **-p** option:

```
docker container run -idt -p 8080:80 nginx
```

where,

- **-p** allows you to define a port mapping.
- **8080** is the host-side port.
- **80** is the container-side port, the one on which the application is listening.

Once you configure the port mapping, access that application on your browser by visiting **http://IPADDRESS:8080**.

***NOTE**: Replace **IPADDRESS** with the *actual IP* in case your Docker host is remote, or with *localhost* if using Docker Desktop.*

With the command above, using the **-p** option, you explicitly define the host-side port. You can also have Docker pick the port automatically with the **-P** option:

```
docker container run -idt -P nginx
```

where,

- The host port is automatically chosen and incremented starting with 32768.
- The container port is automatically read from the image.

As usual, use **docker ps** commands to observe the port mapping.

Troubleshooting Containers

To debug issues with the containers, which are nothing but processes running in an isolated environment, the following could be two important tasks:

- Checking process logs.
- Being able to establish a connection inside the container (similar to ssh).

You will learn about both in this subsection.

To start examining logs, find your container id or container name using:

```
docker ps
```

You can use the following command to find logs for a container whose name is **redis**:

```
docker logs redis
```

You have an option of replacing the **redis** name with the actual container id (the first column in the output of the **docker ps** command).

You can follow the logs using the **-f** option and **docker exec** allows you to run a command inside a container:

```
docker logs -f redis
```

Note: Use ^c to exit

The **exec** command allows you to connect to the container and launch a shell inside it, similar to an ssh connection. It also allows running one-off commands:

```
docker exec redis ps
```

```
docker exec -it redis sh
```

With the **logs** and **exec** commands you should be able to get started with essential debugging.

Exercise: Stop, Remove and Clean Up

With the knowledge gained thus far, you have been tasked with setting up the following two apps with Docker:

- Nextcloud
- Portainer

Try launching it on your own before you proceed with the solutions which are below.

Here you will learn how to stop, remove and clean up the containers which you have created.

Find your container id and stop the container before you remove it. You can stop containers using the name or the ID of the container. You can also stop multiple containers at once, as follows:

```
docker stop redis
```

```
docker stop b584 84e6 e4da
```

where,

- **ac69 b584 84e6 e4da** are container ids (as per example)

These could vary on your system. Use the ones you see when you run **docker ps**.

You can start a container that is in a stopped state with:

```
docker start redis
```

To remove a container (destructive process) that is stopped use:

```
docker stop redis
```

```
docker rm redis
```

If the container is in the running state, it cannot be stopped unless a `force` option is used:

```
docker container run -idtP --name web nginx
```

```
docker rm web
```

```
docker rm -f web
```

Images are independent of containers. You can remove images using the following commands:

```
docker image rm 4206
```

```
docker image rm 4f1 4d9 4c1 9f3
```

```
docker image prune
```

where,

`4206` and `4f1 4d9 4c1 9f3` are the beginning characters of the image ids on our host. You can get all your Docker system information by using `docker system info`.

Setting Up Nextcloud

You should now have some insight about how to run applications using containers. Here you will set up the Nextcloud application using the official image; you will create a volume and mount it inside the container. We know the container directory we must mount our volume to is `/var/www/html` because of the documentation. You may look at it for your own information but it is not necessary.

Follow these steps to set up Nextcloud:

```
docker container run -idt -P -v nextcloud:/var/www/html nextcloud
```

You will see Docker download the image and finally print the container ID.

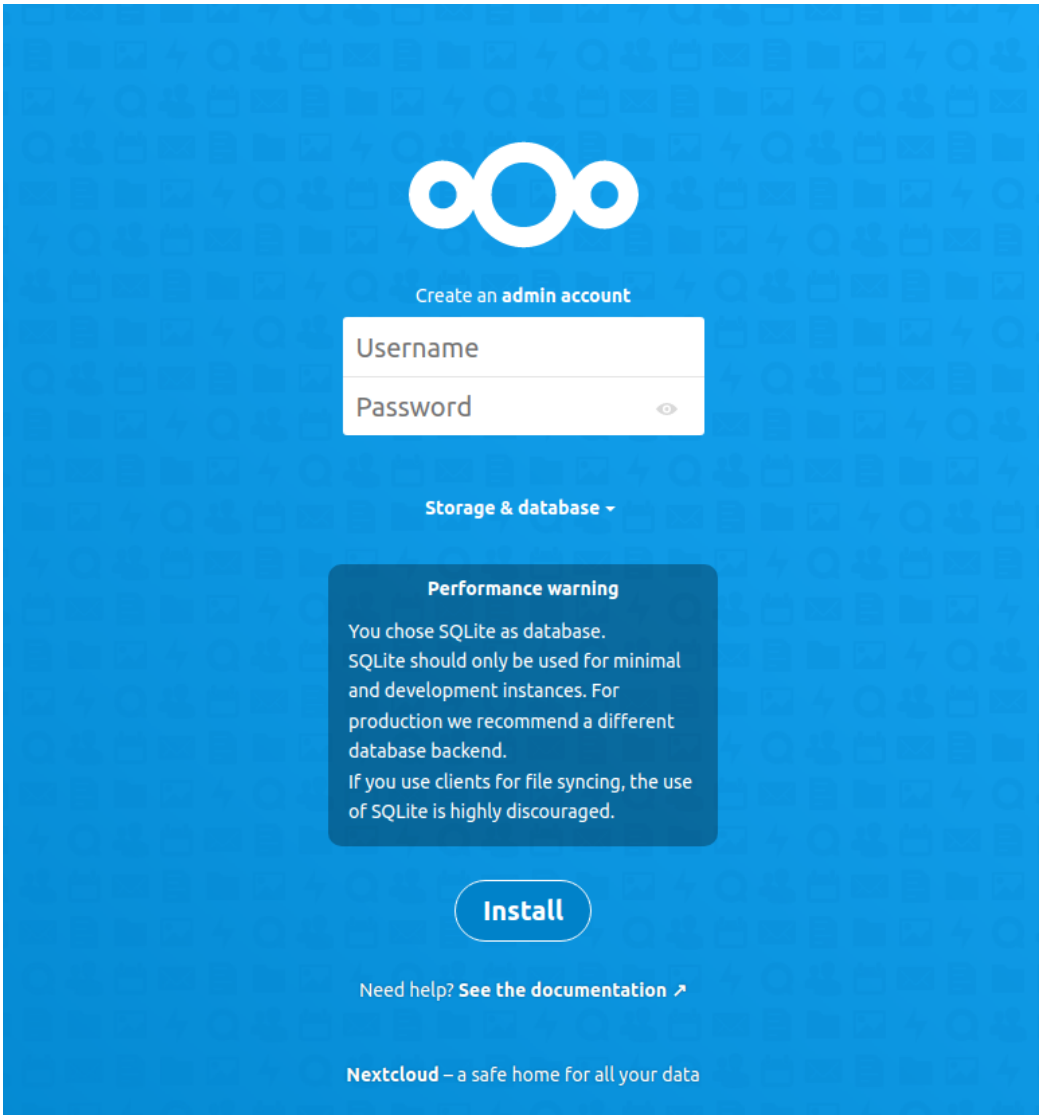
Use the following commands to get the running container port and logs:

```
docker ps
```

```
docker logs ac69
```

```
docker logs -f ac69
```

Example: You can visit the application by using `localhost:32770` on your browser.



Setting up Portainer with Advanced `run` Options

Here you will launch and practice more advanced `run` options by using the Portainer application.

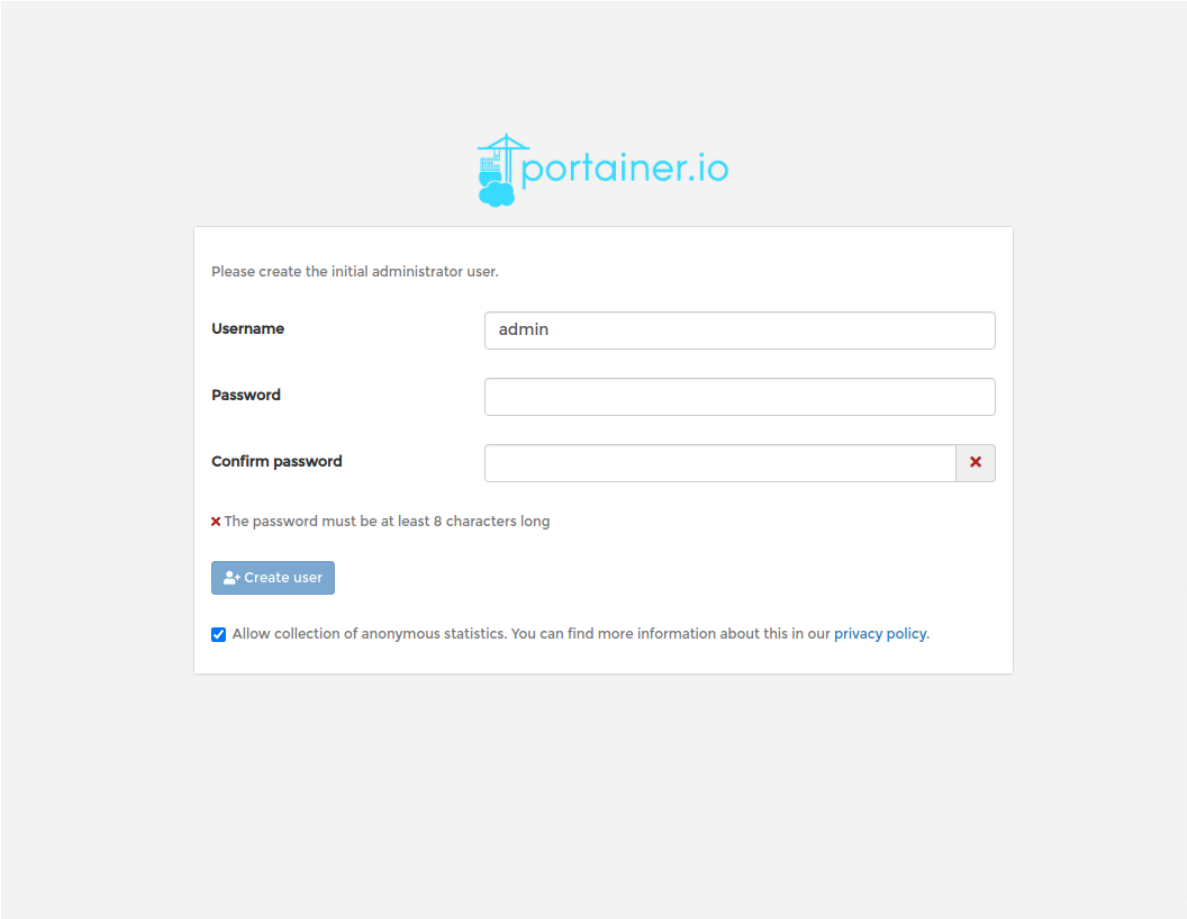
In this case, you need to create a volume before running the container and validate the volume by using the following commands:

```
docker volume create portainer_data

docker volume ls

docker volume inspect portainer_data
```

After creating the volume, run the `docker container run` command along with the volume which we have created:

```
docker run -d -p 9000:9000 -v
/var/run/docker.sock:/var/run/docker.sock -v
portainer_data:/data portainer/portainer
```

Access the portainer application using port 9000 in your browser. To do this, enter `localhost:9000` or `IPADDRESS:9000` in your browser's address bar.



Cleanup

First, we need to stop any running containers. To get all running containers and their IDs run:

```
$ sudo docker ps
```

Keep the first set of unique characters of each container ID handy. These are all you need to enter after the command. Run:

```
$ sudo docker stop containerID1 containerID2 containerID3…
```

Don't type the '...'. That is simply to indicate that you may have more containers.

Now that all the containers have been stopped, we can run:

```
$ sudo docker container prune
```

Type 'y' to continue.

Finally, run:

```
$ sudo docker system prune
```

Type 'y' to continue.

Get your images with:

```
$ sudo docker image ls
```

Take note of the image IDs. Then run:

```
$ sudo docker image rm imageID1 imageID2 imageID3…
```

Run:

```
$ sudo docker system info
```

This should verify that you have 0 containers running, paused, and stopped.

Summary

In this lab, you spun up multiple Docker containers with a diverse set of options. These options allowed you to interact with these containers through their shells and via networking. Finally, you cleaned up your Docker environment so that we can move on to other concepts with a clean slate.