Cloud Technology Experts

Exercise 1: Docker Practice

For an explanation of docker commands, reference this link:
https://docs.docker.com/get-started/docker_cheatsheet.pdf

1. Run `docker --version` to confirm the installation.
2. Use `docker run hello-world` to pull and run the 'hello-world' image from Docker Hub.
3. Use the command `docker pull ubuntu` to download the Ubuntu image from Docker Hub. Then, display a list of downloaded Docker images with the command `docker images`.
4. Run an Ubuntu container with an interactive shell using `docker run -it ubuntu /bin/bash`. Exit without stopping the container by typing exit.
5. Download the Apache Docker image using the command `docker pull httpd`.
6. Launch the Apache image as a web server with the command `docker run -d -p 80:80 --name my-apache-server httpd`.
7. Visit http://localhost/ and verify web server is running.
8. List all running containers using `docker ps`.
9. Stop the web server by executing the command `docker stop my-apache-server`.
10. List all stopped containers using `docker ps -a` command.
11. Delete the `my-apache-server` container by using the command

`docker rm [container-id]`, where `[container-id]` is the specific ID of the container.

12. Create a new Nginx container with the name `my-web-server` by executing the Docker command: `docker run --name my-web-server -d -p 80:80 nginx`.
13. To verify that the `my-web-server` container is running, you can use the Docker command `docker ps`.

Docker Network

14. Create a new Docker network using `docker network create my-network`.
15. To see all the networks in your Docker environment, execute the command `docker network ls`.
16. Connect your `my-web-server` container to the `my-network` network by using the Docker command: `docker network connect my-network my-web-server`.
17. To verify if the `my-web-server` container is using the `my-network` network, use the `docker inspect my-web-server` command. This command provides detailed information about the container configuration, including its network settings.
18. Alternatively, you can execute the command `docker network inspect my-network` to view a list of containers that are connected to the `my-network`.

Docker Volumes

19. Create a new Docker volume called `my-volume` by executing the command: `docker volume create my-volume`. This command will create a new volume in Docker with the name `my-volume` which you can then use for persistent data storage with Docker containers.
20. To attach the `my-volume` Docker volume to your `my-web-server` container, you need to specify the volume when you run or create the container. If `my-web-server` is already running, you'll need to stop and remove it first, then recreate it with the volume attached.
    - Stop the Existing Container (if it's running): `docker stop my-web-server`
    - Remove the Existing Container: `docker rm my-web-server`
21. Run the Container with the Volume Attached: Use the `docker run` command to recreate `my-web-server` and attach `my-volume`. For example, you might want to attach the volume to a specific directory inside the container, like `/usr/share/nginx/html` for website content. The command looks like this: `docker run --name my-web-server -d -p 80:80 -v my-volume:/usr/share/nginx/html nginx`. In this command, `-v my-volume:/usr/share/nginx/html` attaches `my-volume` to the `/usr/share/nginx/html` directory inside the container. You can adjust the container path (`/usr/share/nginx/html`) based on where you want to use the volume in the container.

22. To update the "Welcome to nginx!" message to "Hello world!" in your Dockerized Nginx server, you should overwrite the `index.html` file in the container with the new message.
    - Access the shell of your running Nginx container named "my-web-server" by executing the command: `docker exec -it my-web-server /bin/bash`.
    - Next, execute the command `echo "Hello world!" > /usr/share/nginx/html/index.html` within the Nginx container to update the content, and then type `exit` to leave the container.
    - To confirm the changes, open your web browser and go to http://localhost/.
23. Use the `docker stop` command to gracefully stop the container.
24. Then remove `my-web-server` using `docker rm my-web-server` command.
25. Finally, delete the volume `docker volume rm my-volume`.

Congratulations!!! You have learned how to use Docker.

Exercise 2: Docker Build and Docker Run

For an explanation of docker commands, reference this link:
https://docs.docker.com/get-started/docker_cheatsheet.pdf

In this exercise, we learn how to:

1. Create an image from Dockerfile and
2. Create a container from the image.

Prerequisite:

Install git from this URL

https://www.git-scm.com/downloads

Do the following commands from your shell/terminal:

Step 1: Clone the application repository

$git clone https://github.com/damianigbe/docker-exercises.git

Step 2: Create a docker image and call the image ‘myimage’

$cd docker-exercises

$cd exercise-1

$docker build -t myimage .

Step 3: run the container

$docker run -d --name mycontainer -p 80:80 myimage

Step 4: access your application from the container

In your browser, enter:

http://127.0.0.1/docs

Or
http://localhost/docs

Congratulations!!! You have completed the process of **Docker build, and run** process.

Exercise 3: Docker Push ‘myimage’ Image to Docker hub

Step 1: create an account on docker hub https://hub.docker.com/ (if you don't have an account)

Step 2: login to docker hub and enter your username and password

$docker login

Step 3: Tag your image for the registry. Note that your image name is ‘myimage’

$docker tag myimage username/myimage

Step 4: Push the tagged image to docker hub

$docker push username/myimage

Step 5: Login to docker hub and verify your image

Congratulations!!!, You have completed the entire **Docker build, push, and run** process.

Exercise 4: CREATING A MICROSERVICE WITH RESTFUL API in FLASK

In this lab, we will create a restful api microservice offering books. We will create the API, Create a Docker Image, tag the image and upload to DockerHub.

Step 1: You already clone the repo in exercise 2 so just go into the directory

$ cd rest-api-microservice-docker/

Step 2: Create the Docker Image using Dockerfile
The Dockerfile is already provided, so now you need to build the Docker image

$ docker build -t flaskbookapi:1.0 .

Step 3: Test the microservice by running the docker container
Now, finally! Fire up a container with the image we just built!

```
$ docker run -p 5000:5000 --name FlaskBookAPI flaskbookapi:1.0
* Serving Flask app "api" (lazy loading)
* Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
  * Debug mode: off
  * Running on http://0.0.0.0:5000/ (Press CTRL+C to quit)
```

Step 4: Test that the API is running as expected

In your browser, enter:

localhost:5000/books

OR (using curl but ensure curl is installed)

Open another terminal on your laptop and curl the API just deployed on a docker container. You can query for all the books, and you can query individual books as follows.

```
controlplane $ curl localhost:5000/books
{"books": [{"id": 1, "title": "Zero to One", "author": "Peter Thiel", "length": 195, "rating": 4.17},
{"id": 2, "title": "Atomic Habits ", "author": "James Clear", "length": 319, "rating": 4.35}]}

controlplane $ curl localhost:5000/books/1
{"book": {"id": 1, "title": "Zero to One", "author": "Peter Thiel", "length": 195, "rating": 4.17}}
controlplane $ curl localhost:5000/books/2
{"book": {"id": 2, "title": "Atomic Habits ", "author": "James Clear", "length": 319, "rating": 4.35}}
```

If you try to query a book that does not exists it will give an error

```
controlplane $ curl localhost:5000/books/3
```

Home Work: Should be ready by the next class

1. Tag the image created in exercise 4 and push the image to the docker registry
2. There is a folder called ‘static-site’ that you cloned in exercise 2. Perform the following inside the directory:
    a. Create the image
    b. Tag the image,
    c. Push the image to Docker registry
    d. run the container from the image
    e. and access the application running inside the container