Lab 1 – Docker intro

Task – Install and setup Docker

Prerequisite – EC2 (t2.micro) with ports 22 and 80 open to the internet.

1. SSH connect to your EC2 you have generated
2. Run the following commands within the EC2

sudo apt update

sudo apt install -y curl

curl <https://get.docker.com> | sudo bash

sudo usermod -aG docker $(whoami)

1. Reboot your terminal with `sudo reboot`, closing the terminal then opening a new one
2. Run the command `docker run –rm hello-world`

Stretch goal – Using nginx from DockerHub Image run an nginx image

Lab 2 – Docker images

Task – Login to Docker and push and pull images

Prerequisite – EC2 (t2.micro) with ports 22 and 80 open to the internet with docker installed.

1. Create an account on DockerHub <https://hub.docker.com/>
2. Login to DockerHub CLI with `docker login`. Enter username and password when prompted (when entering passwords on Linux no feedback when a key is pressed)
3. Search for the official node image with `docker search <image>`
4. Pull down the official node image with `docker pull <image name>`
5. View all images with `docker images`
6. Tag the Node image with the below

docker tag <old image name> <your username>/<name of new image>

1. Push the image up to DockerHub with the below

docker push <your username>/<name of new image>

1. Remove all local images with `docker rmi $(docker images -a -q)`

Stretch goal – Pull down an image of Python version 3.9.13-buster and push it up to your profile with version number 1.23.45

Lab 3 – Docker containers

Task – Host nginx as a docker container

Prerequisite – 1x EC2 instances, with port 22, 80, 3000 open. The EC2 must not have nginx installed already on it and have Docker installed

1. Run the following commands

docker run -d -p 80:80 --name <custom name> nginx

docker run -d -p 3000:80 –name <other custom name> nginx

1. View all docker containers running with `docker ps`
2. Access the public IP of your EC2 on port 80 AND 3000 to see nginx running

Stretch goals – Experiment with other container flags using this doc info https://docs.docker.com/engine/reference/commandline/run/

Lab 4 – Docker container commands

Task – Interact with docker container

Prerequisites – Same as Lab 3

Most of the following exercises will require researching the Docker docs for specific commands

1. Exec into a running nginx container and modify the index.html (default index is stored at /usr/share/nginx/html/index.html). Command to exec in is:

docker exec -it <container name> bash

1. Print out all logs from all running containers
2. Run a container using the alpine image
3. Rename the container running the alpine image
4. Remove all containers

Lab 5 – Simple DockerFile

Task – Create a custom nginx image using Dockerfile

Prerequisites – Same as Lab 4

1. Make a new directory to work in and add a file called `Dockerfile` to it
2. Add the below to the Dockerfile

FROM nginx:latest

RUN echo "<your *custom* *HTML*>" > /usr/share/nginx/index.html

1. Navigate to the Dockerfile location and run the below command

docker build -t <docker username>/<name *of* *image*> .

1. Run the container using the commands from previous modules

Stretch goal – Modify the Dockerfile to update the package manager and install curl. Test Curl is installed by exec into the container and curling localhost.

Lab 6 – Intermediate Dockerfile

Task – Create a Dockerfile for an existing app

Using the code included and example Dockerfile create a Dockerfile for Task 1 at the following repo <https://gitlab.com/Reece-Elder/dockerfileexercise>.

Lab 7 – Dockerignore

Task – Use Dockerignore to ignore a file

Prerequisites – Same as Lab 3 and using the repo [https://gitlab.com/Reece-Elder/dockerfileexercise Task 1](https://gitlab.com/Reece-Elder/dockerfileexercise%20Task%201).

1. Create a file called secretData.txt that contains “Hello World”. Move this file to the git repo location with the Dockerfile
2. Create a .dockerignore file and add the following `\*.txt` to ignore .txt files
3. Docker build and exec into the container to check secretData.txt isn’t in the container

Lab 8 – Docker Networks

Task – Use Docker Network to connect Task 1 and a reverse proxy nginx

1. From the EC2 with Docker installed create a new network with the command:

docker network create new-network

1. Create a container running task 1 from [https://gitlab.com/Reece-Elder/dockerfileexercise Task 1](https://gitlab.com/Reece-Elder/dockerfileexercise%20Task%201) keeping note of the name of the container
2. Create an nginx.conf with the following (replacing webapp with the name of the task 1 container)

events {}

http {

    server {

        listen 80;

        location / {

            proxy\_pass http://webapp:5000;

        }

    }

}

1. Create a Dockerfile to create an nginx container replacing the default nginx.conf with the new nginx.conf
2. Access the public IP of the nginx reverse proxy to access task 1

Lab 9 – Bind Mounts and Volumes

Task – Use bind mounts and volumes to configure a three-tier app

Prerequisites – EC2 with Docker installed and ports 22, 80, 5000 open

Task 2 from the repo https://gitlab.com/Reece-Elder/dockerfileexercise should be completed. It will require the following:

* flask-app container - (Custom Dockerfile)
* mysql container - (Custom Dockerfile + Volume Mounting)
* nginx container - (Bind Mounting)
* User defined network

Once you have created all containers you should curl localhost or access the public IP on the browser.

Using a volume mount, you should persist the data from your MySQL database.

Commands to use Bind Mounting and Volume Mounting are below:

Create a new volume - `docker volume create <name of volume>`  
Mount a volume to a container:

docker run -d -p 80:80 --name nginxvolume –mount type=volume, source=<name of volume> ,target=</path/on/container> <name of image>

Mounting a Bind mount is the same apart from setting `mount type=bind`.

Lab 10 – Docker Compose Intro

Task – Using Docker Compose for basic container

Prerequisite – EC2 with ports 22, 80 open, docker installed and nginx not installed

1. Install docker compose with install script from <https://gitlab.com/Reece-Elder/devops-installscripts>
2. Make a directory to work in
3. Create a file called `docker-compose.yaml` and add the below:   
   ! IMPORTANT ! – Watch out for tabs and spaces, use sets of 2x (or 4x) spaces as YAML as picky

version: "3.8"

services:

  nginx:

    image: nginx:alpine

    ports:

    - “80:80”

1. Start the docker compose with `docker-compose up -d`
2. Check the container is running with `docker-compose ps`
3. Access the browser <ip>/ to see nginx running

Lab 11 – Docker Compose Multiple services

Task – Use Docker Compose to spin up multiple docker containers

Using Docker Compose you should spin up all containers needed for Task 1 at the repo <https://gitlab.com/Reece-Elder/dockerfileexercise>

Within the Docker-compose.yaml you can have multiple services using local or online hosted images.

By specifying the Dockerfile to use for an image you can get docker-compose to build the image before running containers. `docker-compose up -d` will build the images and updated before running any containers

version: "3.8"

services:

  my-custom-image:

    image: [HOST]/[AUTHOR]/[APPLICATION]:[TAG]

    build: ./path/to/Dockerfile

Stretch goal – Use Docker Compose to create containers for Task 2 at the repo.