DOu DevOps united

Testing in DevOps

DOu – Certified Tester in DevOps (CTD)
Exercise Solutions

HO-5.1.4(HO-0)
Step 1 - Docker Setup

- Create & Launch AWS Ubuntu 18.04 instance(t2.small, 8GB HD)with all traffic

  (Refer to “Testing in DevOps_Exercise-Solutions_Pre-requisite_V0.1” slide-deck)
- Connect with putty
- Run below commands on terminal
  - sudo apt-get update
  - sudo apt-get install docker.io
  - docker –version

    Output e.g., Docker version 17.12.0-ce, build c97c6d6

HO-5.1.4(HO-0)
Exercise - Demonstrate how Dockers can be applied on a container in a virtualized environment

- Task I
  - Pull Ubuntu image
  - Run the image & enter into container
  - Play with container
- Task II
  - Develop the environment
  - Build the image
  - Sharing the image
- Task III
  - Docker Swarm

HO-5.1.4(HO-0)
Exercise Solution - Demonstrate how Dockers can be applied on a container in a virtualized environment

- Exercise Solution is given in upcoming slides.

HO-5.1.4(HO-0)
Task I: Let us play with docker

Run the below commands on terminal

- sudo docker pull ubuntu
- sudo docker images
- sudo docker run -it -d ubuntu
- sudo docker info
- sudo docker container ls -a
- sudo docker exec -it <<container id>> /bin/bash

  Note: You will enter inside the container.
- exit

  Note: You will exit from the container, but the container status will be Up

HO-5.1.4(HO-0)
Task I: Let us play with docker…

Run the below command on terminal

- sudo docker run -it ubuntu /bin/bash
- Exit

  Note: You will exit from the container and container status will be exited

When you run this command, the following happens (assuming you are using the default registry configuration):

1. If you do not have the `ubuntu` image locally, Docker pulls it from your configured registry, as though you had run `docker pull ubuntu` manually
2. Docker creates a new container, as though you had run a `docker container create` command manually
3. Docker allocates a read-write file system to the container, as its final layer
4. Docker creates a network interface to connect the container to the default network, since you did not specify any networking options
5. Docker starts the container and executes `/bin/bash`. Because the container is running interactively and attached to your terminal (due to the -it flag), you can provide input using your keyboard while the output is logged to your terminal.
6. When you type `exit` to terminate the `/bin/bash` command, the container stops but is not removed. You can start it again or remove it.

Note: For more commands refer to Docker Cheat sheet slide at the end of this deck

HO-5.1.4(HO-0)
Task II: Docker Image build & share

- In next few slides we will create a docker image starting from a base image containing python, get the required packages and run a python program and check its output using a browser from the host machine
- We will then upload the image to a registry

HO-5.1.4(HO-0)
Task II - Developing an Environment(AWS)

- Code is available at https://github.com/umangsaltuniv/DockerSwarm
- Create an empty directory by running the command **mkdir <foldername>** e.g. **mkdir mydocker** and cd into the new directory
- Create a file called Dockerfile by running the command **touch Dockerfile**
- Run command to open Dockerfile

  vi Dockerfile
- Copy and paste the code from github repo into this file and save it(by pressing **esc** then write **:wq** then press **enter**)

```
# Use an official Python runtime as a parent image
FROM python:2.7-slim
# Set the working directory to /app
WORKDIR /app
# Copy the current directory contents into the container at /app
ADD . /app
# Install any needed packages specified in requirements.txt
RUN pip install -r requirements.txt


# Make port 80 available to the world outside this container
EXPOSE 80
# Define environment variable
ENV NAME World
# Run app.py when the container launches
CMD ["python", "app.py"]
```

HO-5.1.4(HO-0)
Task II - Developing an Environment(AWS)…

- Create a file called **requirements.txt**
- **vi requirements.txt**
- Copy and paste the following two lines into this file and save it

```
Flask
Redis
```

- Create a file called **app.py**
- **vi app.py**
- Copy and paste the code from github repo into this file and save it

```
from flask import Flask
from redis import Redis, RedisError
import os
import socket
# Connect to Redis
redis = Redis(host="redis", db=0, socket_connect_timeout=2, socket_timeout=2)
app = Flask(__name__)
@app.route("/")
def hello():
    try:
        visits = redis.incr("counter")
    except RedisError:
        visits = "<i>cannot connect to Redis, counter disabled</i>"
    html = "<h3>Hello {name}!</h3>" \
           "<b>Hostname:</b> {hostname}<br/>" \
           "<b>Visits:</b> {visits}"
    return html.format(name=os.getenv("NAME", "world"), hostname=socket.gethostname(), visits=visits)
            if __name__ == "__main__":
            app.run(host='0.0.0.0', port=80)
```

HO-5.1.4(HO-0)
Task II - Build The Image

- Run the following commands from that folder(e.g., mydocker) where all three files exist(Dockerfile, requirements, app)
  - **sudo docker build -t friendlyhello .**
  - **sudo docker images**
  - **sudo docker run -p 4000:80 friendlyhello**
- For AWS, See the output in a browser at

  <http://your aws public dns:4000>

  e.g., http://ec2-3-81-43-72.compute-1.amazonaws.com:4000
- “Hello World” UI will be displayed on browser
- Container id will also appear on Hello World UI on browser

HO-5.1.4(HO-0)
Task II - Sharing The Image

- Make sure you have already registered on docker hub(Refer to setup guide)
- Launch another terminal instance(Right click on top white window header and click Duplicate session)
- Run the following commands
  - **sudo docker login**

    (Enter your docker hub id & password)
  - **sudo docker tag friendlyhello <your docker hub id>/tutorial:firstversion**
  - **sudo docker image ls**

    <You should be able to see your tagged image there>
  - **sudo docker push <Your docker hub id>/tutorial:firstversion**

Please note Tag command syntax is <docker tag image username/repository:tag>

Note: Now you can see tutorial image on your docker hub UI account under Repositories section

HO-5.1.4(HO-0)
Task II - Stopping the Container

- sudo docker container ls -a
- sudo docker container stop <Container NAME or ID>

  Note: Container status will be in Exited
- See example below
  - *sudo docker container ls*

    ```
    CONTAINER ID       IMAGE             COMMAND            CREATED
    1fa4ab2cf395      friendlyhello      "python app.py"     28 seconds ago
    ```
  - *sudo docker container stop 1fa4ab2cf395*
- *Refresh the Hello World UI on browser, now Hello World app will not be displayed there*

HO-5.1.4(HO-0)
Task III: Docker Swarm

- In this exercise, we will try to scale the app by running multiple containers

HO-5.1.4(HO-0)
Task III: Docker Swarm - Running the Service

- We want to run containers and their multiple instances to provide the required horsepower to the service.
- Read the docker-compose.yml which does the following
  a. Pull the image that we uploaded to registry
  b. Run 5 instances of that image as a service called web, limiting each one to use, at most, 10% of the CPU and 50MB of RAM
  c. Immediately restart containers if one fails
  d. Map port 4000 on the host to web's port 80
  e. Instruct web's containers to share port 80 via a load-balanced network called webnet
  f. Define the webnet network with the default settings (which is a load-balanced overlay network)

HO-5.1.4(HO-0)
Task III: Docker Swarm - Starting/Stopping The Service

- Create an empty directory by running the command **mkdir <foldername>** e.g. **mkdir mydockerswarm** and cd into the new directory
- Create a file called by running the command **touch docker-compose.yml**
- Run command to open **docker-compose.yml**

  vi docker-compose.yml
- Copy and paste the code from github repo into this file and save it(by pressing esc then write :wq then press enter)

HO-5.1.4(HO-0)
Task III: Docker Swarm - Starting/Stopping The Service…

- Run the following commands from that folder(e.g. mydockerswarm) where docker-compose.yml file exists
- Run the following commands
  - sudo docker swarm init
  - sudo docker stack deploy -c docker-compose.yml getstartedlab
  - sudo docker service ls (Look for output for the web service, prepended with your app name)
  - sudo docker service ps getstartedlab_web
  - sudo docker container ls -q
  - For AWS, Launch <http://your aws public dns:4000> on browser
    (every time when you refresh the page, you should see a different ID showing load balancing)
- Shutdown the app and the swarm
  - sudo docker stack rm getstartedlab
  - sudo docker swarm leave –force
- Verify all containers will be removed
  - sudo docker container ls -a

HO-5.1.4(HO-0)
Docker Cheat Sheet

- docker create [image]: Create a new container from a particular image.
- docker login: Log into the Docker Hub repository.
- docker pull [image]: Pull an image from the Docker Hub repository.
- docker push [username/image]: Push an image to the Docker Hub repository

Running Docker Containers

- docker start [container]: Start a particular container.
- docker stop [container]: Stop a particular container.
- docker run -ti — rm — image [image] [container] [command]: Create and start a container at the same time, run a command inside it, and then remove the container after executing the command.
- docker pause [container]: Pause all processes running within a particular container.

Using Docker Utilities

- docker version: Display the version of Docker that is currently installed on the system.
- docker images: List all of the images that are currently stored on the system.
- docker container ls –a: List all of the containers.
- docker ps: List all of the containers that are currently running.
- docker exec -it <<container id>> /bin/bash : Enter inside the running container

Cleaning Up Your Docker Environment

- docker kill [container]: Kill a particular container.
- docker kill $(docker ps -q): Kill all containers that are currently running.
- docker rm [container]: Delete a particular container that is not currently running.
- docker rm $(docker ps -a -q): Delete all containers that are not currently running