Docker Training

Table of Contents

Prerequisites to be installed in Windows 10

Install Docker :
https://www.youtube.com/watch?v=_9AWYlt86B8

Download and Install Docker-compose here:
https://github.com/docker/compose/releases/

Install Cygwin
https://www.youtube.com/watch?v=QonIPpKodCw

https://github.com/lakelse/videos/tree/master/01-install-cygwin-on-windows-youtube

In Windows 10 you need to install Cygwin to run docker commands from the terminal

Prerequisites to be installed in Linux

Install Docker
https://linuxconfig.org/how-to-install-docker-on-ubuntu-20-04-lts-focal-fossa

Download and Install Docker-compose here:
https://github.com/docker/compose/releases/

In Linux you use the terminal to run docker commands using sudo.

1 Creating Docker Files to create Docker Images

example 1 : Hello World Without Arguments

Create a folder for your Docker Image and open this folder:

hello0/

create a C++ file : helloworld.cpp with this content:

```
#include <iostream>
using namespace std;

int main()
{
  cout << "Hello world 0!" << endl;
  return 0;
}
```

create a file called: Dockerfile with this content:

```
FROM amytabb/docker_ubuntu16_essentials
COPY HelloWorld /HelloWorld
WORKDIR /HelloWorld/
RUN g++ -o HelloWorld helloworld.cpp
CMD ["./HelloWorld"]
```

Run the folowing commands to create the Docker Image:

docker pull amytabb/docker_ubuntu16_essentials

sudo docker pull amytabb/docker_ubuntu16_essentials // In Ubuntu terminal

docker build -t hello0 .

sudo docker build -t hello0 . // In Ubuntu terminal

Launch the Docker Image using Docker:

docker run -it hello0

sudo docker run -it hello0 // In Ubuntu terminal

Example 2 : Hello World With Arguments

Create a folder for your Docker Image and open this folder:

hello1/

create a C++ file : helloworld.cpp with this content:

```
#include <iostream>
#include <string>
using namespace std;

int main(int argc, char **argv)
{
  cout << "Hello world 1, with arguments!" << endl;

  string val;
  for (int i = 1; i < argc; i++){
          val = argv[i];
       cout << "Argument " << i << " " << val << endl;
  }
  return 0;
}
```

Cleate a shellscript: run_hello1.sh with this content:

```
#!/bin/sh
./HelloWorld/HelloWorld1 $VAR1 $VAR2 $VAR3
```

create a file called: Dockerfile with this content:

```
FROM amytabb/docker_ubuntu16_essentials
ENV NAME VAR1
ENV NAME VAR2
ENV NAME VAR3
COPY run_hello1.sh /run_hello1.sh
COPY HelloWorld /HelloWorld
WORKDIR /HelloWorld/
RUN g++ -o HelloWorld1 helloworld1.cpp
WORKDIR /
CMD ["/bin/sh", "/run_hello1.sh"]
```

Run the folowing commands to create the Docker Image:

```
docker build -t hello1 .
sudo docker build -t hello1 .                    // In Ubuntu terminal

docker run -it -e VAR1='23' hello1
sudo docker run -it -e VAR1='23' hello1          // In Ubuntu terminal
```

Example 3: Simple Java Hellow World

Create a folder for your Docker Image and open this folder:
hellojava/

create a Java file : HelloWorld.java with this content:

```
public class HelloWorld {
   public static void main(String[] args) {
      System.out.println("Hello, World");
   }
}
```

Complie this Class:
javac HelloWorld.java

Pull the Alpine Docker Image from the Docker Repository
docker pull alpine:latest

create a file called: Dockerfile with this content:

```
FROM alpine:latest
ADD HelloWorld.class HelloWorld.class
RUN apk --update add openjdk8-jre
ENTRYPOINT ["java", "-Djava.security.egd=file:/dev/./urandom", "HelloWorld"]
```

Run the folowing commands to create the Docker Image:
docker pull alpine:latest
sudo docker pull alpine:latest // In Ubuntu terminal

docker build --tag "docker-hello-world:latest" .
sudo docker build --tag "docker-hello-world:latest" . // In Ubuntu terminal

Launch the Docker Image using Docker:
docker run docker-hello-world:latest
sudo docker run docker-hello-world:latest // In Ubuntu terminal

Example 4 : Hello World Apache Webserver

Create a folder for your Docker Image and open this folder:

helloweb/

create a HTML file : index.html with this content:

```
<html>
<body>
<h1>Hellow World</h1>
</body>
</html>
```

create a file called: Dockerfile with this content:

```
FROM centos:latest
MAINTAINER NewstarCorporation
RUN yum -y install httpd
COPY index.html /var/www/html/
CMD [“/usr/sbin/httpd”, “-D”, “FOREGROUND”]
EXPOSE 80
```

Run the folowing commands to create the Docker Image:

```
docker pull centos:latest
sudo docker pull centos:latest                    // In Ubuntu terminal

docker build -t webserver:v1 .
sudo docker build -t webserver:v1 .               // In Ubuntu terminal
```

Launch the Docker Image using Docker:

```
docker run –dit –p 1234:80 webserver:v1
sudo docker run –dit –p 1234:80 webserver:v1      // In Ubuntu terminal
```

Open your browser to :

http://localhost:1234

Example 5: Java Spring Boot Microservice

Create a Spring Boot project that will generate JAR file: hello-docker-0.0.1-SNAPSHOT.jar

Java Code:

```
package com.example.howtodoinjava.hellodocker;

import java.util.Date;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

@SpringBootApplication
public class HelloDockerApplication {

   public static void main(String[] args) {
      SpringApplication.run(HelloDockerApplication.class, args);
   }
}

@RestController
class HelloDockerRestController {
   @RequestMapping("/hello/{name}")
   public String helloDocker(@PathVariable(value = "name") String name) {
      String response = "Hello " + name + " Response received on : " + new Date();
      System.out.println(response);
      return response;

   }
}
```

Add Maven Docker Plugins to POM file

```
<plugin>
   <groupId>com.spotify</groupId>
   <artifactId>dockerfile-maven-plugin</artifactId>
   <version>1.3.4</version>
   <configuration>
      <repository>${docker.image.prefix}/${project.artifactId}</repository>
   </configuration>
</plugin>
<plugin>
   <groupId>org.apache.maven.plugins</groupId>
   <artifactId>maven-dependency-plugin</artifactId>
   <executions>
      <execution>
         <id>unpack</id>
         <phase>package</phase>
         <goals>
            <goal>unpack</goal>
         </goals>
         <configuration>
            <artifactItems>
               <artifactItem>
                  <groupId>${project.groupId}</groupId>
                  <artifactId>${project.artifactId}</artifactId>
                  <version>${project.version}</version>
               </artifactItem>
            </artifactItems>
         </configuration>
      </execution>
   </executions>
</plugin>
```

create a file called: Dockerfile with this content:

```
FROM openjdk:8-jdk-alpine
VOLUME /tmp
ADD target/hello-docker-0.0.1-SNAPSHOT.jar hello-docker-app.jar
ENV JAVA_OPTS=""
ENTRYPOINT [ "sh", "-c", "java $JAVA_OPTS -Djava.security.egd=file:/dev/./urandom -jar
/hello-docker-app.jar" ]
```

Now use maven command mvn clean install or use maven command mvn clean build

mvn clean install

2 Portainer

Now we will look at docker container management commands and manage then manage them in Portainer

Install Portainer

docker volume create portainer_data

sudo docker volume create portainer_data // In Ubuntu terminal

docker run -d -p 8000:8000 -p 9000:9000 --name=portainer --restart=always -v /var/run/docker.sock:/var/run/docker.sock -v portainer_data:/data portainer/portainer-ce

sudo docker run -d -p 8000:8000 -p 9000:9000 --name=portainer --restart=always -v /var/run/docker.sock:/var/run/docker.sock -v portainer_data:/data portainer/portainer-ce

#open browser : http://localhost:9000

user:admin
password:adminadmin

3 Getting Familiar with docker commands

Running an Ubuntu Container and accessing it from the terminal

docker run -it ubuntu bash
sudo docker run -it ubuntu bash

Now run these commands from the terminal
ls -l
pwd
whoami
more /etc/os-release
lscpu
top
uname -a

Listing docker Images

docker images
sudo docker images // ubuntu Linux

Listing All Docker Containers

docker container ls -a
sudo docker container ls -a // ubuntu Linux

Starting a listed container

(assuming a container was listed as: 0a0631eeea21)

docker start -i 0a0631eeea21
sudo docker start -i 0a0631eeea21

Run and Delete a container afterwards

docker run --rm -t ubuntu bash
sudo docker run --rm -t ubuntu bash // ubuntu Linux

Deleting all Docker Containers

(its dangerous but it give you back all the disk space used)
docker container prune
sudo docker container prune // ubuntu Linux

Pulling (down) Docker containers

(Windows users remove sudo from each line)
sudo docker pull ubuntu
sudo docker pull sonarqube
sudo docker pull postgres
sudo docker pull nginx
sudo docker pull ubuntu
sudo docker pull mariadb
sudo docker pull httpd
sudo docker pull centos

Show Details of Docker containers

sudo docker container ls

List all Containers

sudo docker container ls -l
sudo docker container ls -a
sudo docker container ls --all

List all Images

sudo docker images

Stopping a container

sudo docker stop ubuntu

Deleting a container

sudo docker stop hello-world
sudo docker rm hello-world
sudo docker stop 724a4081c966
sudo docker rm 724a4081c966

Deleting an Image (This is permanent)

sudo docker images
sudo docker rmi hello-world

Running containers :

sudo docker run hello-world
sudo docker run -it ubuntu bash

Launch a container in the interactive mode

sudo docker run -i -t ubuntu:latest /bin/bash

Attached Mode

In this mode, the console you are using to execute docker run will be attached to standard input, output and error.
That means your console is attached to the container's process.

Detached Mode

In detached mode, you can follow the standard output of your docker container with docker logs -f <container_ID>

Running Containers in Attached mode:

docker run -rm mongo ..... Mongo listens to port 27017
Ctrl C to stop it

Running Containers in Detached mode (as a service):

sudo docker run -d -p 80:80 gvelrajan/hello-world:v1.0
sudo docker run -d -p 8080:80 gvelrajan/hello-world:v1.0
sudo docker ps

Run and then delete the container:

sudo docker run --rm -t ubuntu bash

Running a webserver with internal port 80 and external port 4000 and 4001

sudo docker run --rm -it -p 4000:80 simple-webserver
sudo docker run --rm -it -p 4001:80 simple-webserver

You access the following webserver instances from the browser at:
http://localhost:4000
http://localhost:4001

Pruning Docker Containers Images and volumes

Pruning unused Docker Containers

The following removes all containers
sudo docker container prune

The following removes containers created more than 5 minutes ago:
sudo docker container prune --force --filter "until=5m"

Prune unused Docker Images

sudo docker image prune

Limiting images pruned to only images created more than 24 hours ago:

sudo docker image prune -a --filter "until=24h"

Prune Volumes (this is non recoverable)

sudo docker volume prune

Removing volumes which are not labelled with the keep label:

sudo docker volume prune --filter "label!=keep"

4 The Dockerfile Syntax

FROM centos // you are specify "Super - Container" usually the O/S web server or Database
MAINTAINER <author's detail>
COPY <src> ... <dst>
ADD <src> ... <dst> // similar to add but handles tar commands
ENV <key> <value>
ENV DEBUG_LVL 3
ENV APACHE_LOG_DIR /var/log/apache
USER <UID>|<UName> // <UID> numerical user ID <UName> :valid user Name
WORKDIR <dirpath>
VOLUME <mountpoint>
EXPOSE <port>[/<proto>] [<port>[/<proto>]...]
EXPOSE 7373/udp 8080
RUN <command>
CMD <command> // shell command executed during launch of the container
ENTRYPOINT <command> // shell command executed during the launch of the container.
ONBUILD <INSTRUCTION>
ONBUILD ADD config /etc/appconfig

And Example of a Dockerfile :

```
##############################################
# Dockerfile to build an Apache2 image
##############################################
# Base image is Ubuntu
FROM ubuntu:14.04
# Author: James Bond
MAINTAINER James Bond <james.bond007@gmail.com>
# Install apache2 package
RUN apt-get update && \
apt-get install -y apache2 && \
apt-get clean
```

Running Containers with a Hard coded port

_____

```
sudo docker container run --name mongo mongo -p 8081:271017 mongo
sudo docker container rm -f mongo     .....manually deletes the container
```

Running Containers with Dynamically allocated port

_____

```
sudo docker container run -d --name mongo -p 271017 mongo
```

Pausing/Unpausing a Docker Container

_____

```
sudo docker run -i -t ubuntu:latest /bin/bash
```

5 Docker Swarms

Difference between Docker and Docker Swarm

As a platform, Docker has revolutionized the manner software was packaged.
Docker Swarm or simply Swarm is an open-source container orchestration platform and is the native clustering engine for and by Docker.
Swarm turns a pool of Docker hosts into a virtual, single host.

Worker Nodes

Worker nodes are also instances of Docker Engine whose sole purpose is to execute containers.

Create Swarm

docker swarm init ....do this on the first node to manage the swarm

Joining a swarm

docker swarm join --token SWMTKN-1-4zfe8c7vezompmxgnvr59z9fzj4dop9lggde6qk4qs71nytr97-03cjgmeh8yiun807y7adz6k79 10.154.2.105:2377

Creating workers

docker swarm join-token worker
.. it will display :
To add a worker to this swarm, run the following command:
sudo docker swarm join --token SWMTKN-1-4zfe8c7vezompmxgnvr59z9fzj4dop9lggde6qk4qs71nytr97-2srtr4qxcnvc9bqmr9nn6s76z 10.154.2.105:2377

Joining another node as a worker to this swarm

sudo docker swarm join --token SWMTKN-1-4zfe8c7vezompmxgnvr59z9fzj4dop9lggde6qk4qs71nytr97-2srtr4qxcnvc9bqmr9nn6s76z 10.154.2.105:2377