Lab Document – Containers

*Note : The examples can be executed with podman as well*

Part A : Basic Commands

This section introduces the basic commands of managing images a containers. The exercises uses standard-readily available images.

1. View Docker information

docker info

1. Pull the latest ubuntu image from docker hub and launch the container

docker pull ubuntu:latest

docker run -itd --name myubuntu <imageid>

1. Check if the container is running

docker ps

1. Try stopping, restarting the container

docker stop <container>

docker start <container>

1. Delete the container

docker rm <container>

1. Directly running Containerized application

docker container run -d -p 80:80 tutum/hello-world

1. Create a container with env variables

docker run --name db -e MYSQL\_ROOT\_PASSWORD=docker -e MYSQL\_DATABASE=docker -e MYSQL\_USER=docker -e MYSQL\_PASSWORD=docker -d mysql:5.6

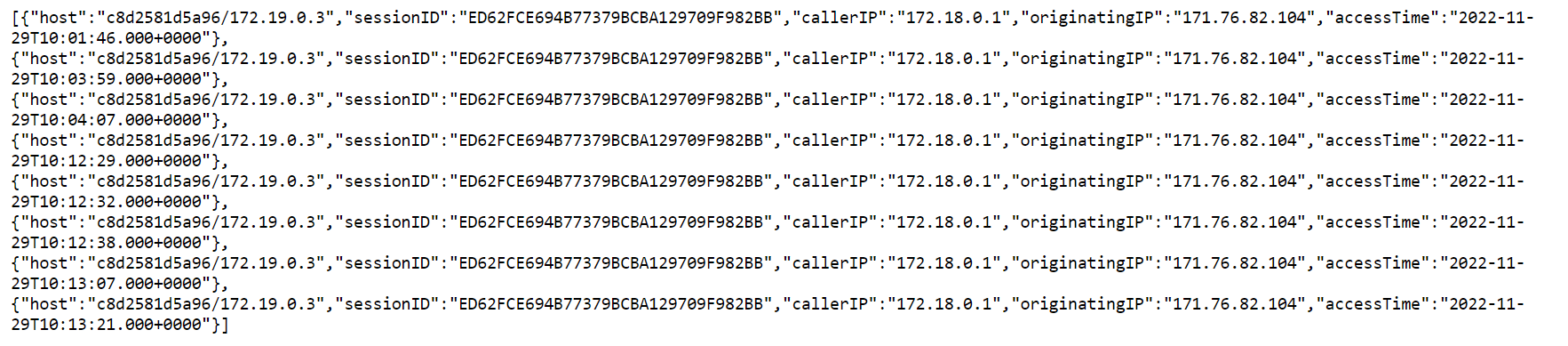
docker exec -it db env

1. Launching two containers – application and database container(s) and linking

docker run --link db:dbserver -e spring.datasource.url=jdbc:mysql://dbserver:3306/docker -p 80:8080 -d brainupgrade/request-logger:microservice

Run the application

<http://localhost:80> and see if the data is getting displayed



1. Check the content of the table in the database

docker exec -it dbserver mysql -u docker -p docker

show tables;

select \* from users;

1. Create another table called log\_table with following structure

Log\_id char(10)

Log\_desc char(50)

);

Check the table using show tables

Exit

1. Save the container into a new image as

Docker commit -t dbserver\_wlog dbserver

1. Run inspect command for both images and containers

docker inspect <container>

docker inspect <image>

Part B : Creating Docker images using Script

In the following section you will create docker images using script, tag it and push to repository

1. Create a file called Dockerfile which will create an ubuntu image with git

FROM ubuntu:latest

MAINTAINER author

RUN apt-get update -y && apt-get install -y git && apt-get install -y curl

Build the image and check if git has been installed or not

1. In this example we will create a simple node application and containerize it

Create a file named index.js with the following contents

var http = require('http');

var server = http.createServer(function(request, response) {

response.statusCode = 200;

response.setHeader('Content-Type', 'text/plain');

response.end('Welcome to the World !');

});

server.listen(3000, function() {

console.log('Server running on port 3000');

});

Create the Dockerfile with the following content

FROM node:carbon

WORKDIR /app

COPY index.js .

EXPOSE 3000

CMD node index.js

Build the file

docker build -t myimage .

1. In this example you will build and containerize a node application

Clone the following repository git clone <https://github.com/slathas24/EKS-Program.git>

Create a directory called NodeApp

copy the following files to from git repo to your NodeApp directory

Main.js

package.json

Create a Dockerfile with the following content

FROM node:alpine

LABEL MAINTAINER "your name "

WORKDIR /app

COPY main.js /app

COPY package.json /app

RUN ["npm","install"]

ENTRYPOINT ["npm"]

CMD ["start"]

EXPOSE 8000

Build the application and test it out

PART C Docker Volumes

1. Create a new Docker volume called "my-data"

docker volume create my-data

2. Inspect the details of the created volume. Check the mount point which will be under

/var/lib/docker/volumes

docker inspect my-data

3. Launch a container that maps a target folder (/app) to the created volume

docker run -itd --name myubuntu1 --mount source=my-data,target=/app <imageid>

4. Create some files inside the /app folder of the container by first doing a docker exec

docker exec -it myubuntu1 /bin/bash

cd /app

touch samplefile1

touch samplefile2

5. Exit the container

6. Run a new container with the same volume binding

docker run -itd --name myubuntu2 --mount source=my-data,target=/app <imageid>

7. Go into the new container and verify that the files are visible

docker exec -it myubuntu2 /bin/bash

cd /app

ls -al

8. Stop and delete both the containers

docker stop myubuntu

docker stop myubuntu2

docker rm myubuntu

docker rm myubuntu2

9. Browse to /var/lib/docker/volumes/my-data/\_data and verify that the files are still present and available for mounting to a new container

10. Delete the volume by using the following command

docker volume rm my-data

PART D : Bind mounts

1. Create a folder on host

mkdir /root/target

Launch a container to create a bind mount to the above folder

docker run -itd --name myubuntu --mount type=bind,source=/root/target,target=/app <image id>

1. Go into the container and create a sample file under /app & exit the container
2. Verify that the created file is visible under /root/target

PART E : CONTAINERIZING MICROSERVICES

In this example we will containerize both Spring Boot application and React application as separate containers.

* Clone the repository : <https://github.com/slathas24/EKS-Program.git>
* Navigate to the folder EKS-Program
* To build the API
  + Navigate to the folder EKS-Program/**smart-bank-app-api**/
  + Create a Dockerfile

FROM adoptopenjdk/maven-openjdk11 as build

WORKDIR /build

COPY . .

RUN ls -l

RUN mvn clean package

FROM prasadlvi/openjdk-11-jre:latest

COPY --from=build /build/target/\*.jar app.jar

EXPOSE 8080

ENTRYPOINT ["java"]

CMD ["-jar","app.jar"]

* + Build the image and test it out using curl or Browse

http://<host>:8080/catalogue/

* To build the front end
  + Navigate to folder EKS-Program/**smartbankfrontend**/
  + Create a Dockerfile

FROM node

WORKDIR /app

COPY . .

RUN npm install

EXPOSE 3000

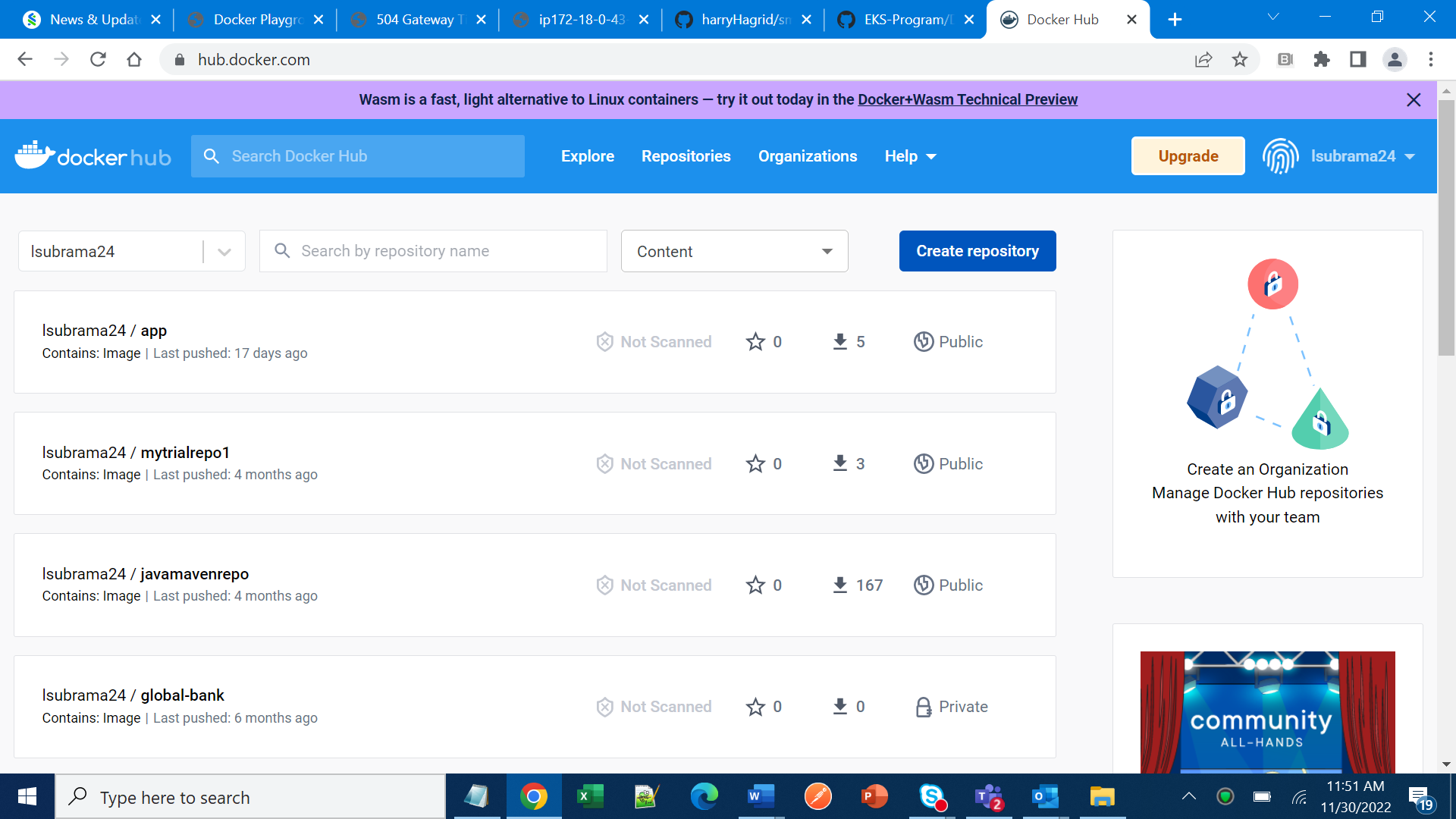
ENTRYPOINT [ "npm"]

CMD [“start”]

* + Try running the program using curl or Browser

PART E : Uploading to docker hub

1. Register to dockerhub if not done - hub.docker.com
2. Create a repository



1. From the prompt give the command docker login
2. Specify the docker login and password
3. Create a tag <imageid> <tagname> username/reponame:tagname (for example : docker push lsubrama24/app:tagname)
4. Push it into repository using docker push