DOCKER

Docker is a tool designed to make it easier to create, deploy, and run applications by using containers. Containers allow a developer to package up an application with all of the parts it needs, such as libraries and other dependencies, and ship it all out as one package. By doing so, thanks to the container, the developer can rest assured that the application will run on any other Linux machine regardless of any customized settings that machine might have that could differ from the machine used for writing and testing the code.

In a way, Docker is a bit like a virtual machine. But unlike a virtual machine, rather than creating a whole virtual operating system, Docker allows applications to use the same Linux kernel as the system that they're running on and only requires applications be shipped with things not already running on the host computer. This gives a significant performance boost and reduces the size of the application.

> docker version → Shows the version of docker

> docker info → Shows docker's information

> docker container run -it --publish (or -p) 80:80 nginx → This will create nginx container and run it, -it means interactive, the first 80 means port used in local machine, the second 80 means the port that will be exposed in the container it is better to use the application's default port which 80 for nginx.

> docker container ls → This show our running container

> docker container ls -a → This will show all the container in our system whether they are running or not.

> docker container rm containerId(e.g fa123f45) → This will delete(remove) the container with the given id)

> docker images → This will shows docker images in our local machine.

> docker image rm imageId(e.g gd1234gr45) → This will delete(remove) the image with the given id.

> docker pull imageName(e.g nginx) → This will pull/download nginx docker image

> docker container run -d -p 8080:80 --name containerName(e.g mynginx) nginx → This will run the nginx image in our local machine in the docker container. The -d means detached.

> docker container ps / docker ps → This will show docker running containers.

> docker container ps -a / docker ps -a → This will show all docker containers.

> docker container run -d -p 3306:3306 --name mysql --env MYSQL_ROOT_PASSWORD=@Mysql! Mysql → This pull mysql create a container and run the image, it also set the root password.

> docker container stop mysql → This will stop mysql running container.

> docker container rm myapache -f → This will force remove apache running container. Without the force it won't be removed because it's running.

ENTERING YOUR CONTAINER

> docker container exec -it mynginx bash → This will give you access into running nginx container files.

> cd usr/share/nginx/html → Thi is the nginx server web root where you can deploy your application.

> exit → Takes you out of the container to your local machine.

> docker rm $(docker ps -aq) -f → This will remove all containers, -f force to remove running containers.

> docker container run -d -p 8080:80 -v $(pwd):/usr/share/nginx/html --name nginx-website nginx → This will create nginx container and map it web root to the current directory you are in, in your machine. After that you can then add all your website in your local directory and it will be published. The pwd wont work on windows so you have to add the complete path. -v mean volume.

CREATING IMAGE FROM YOUR APPLICATION

1. You create a Dockerfile file in the directory of what you want to create the image from.
2. In the file you add the below.

//Start of script for creating docker image of nginx and our application

```
FROM nginx:latest

WORKDIR /usr/share/nginx/html // this is the nginx web root

COPY . .  // this mean copy everything from current directory to the image.
```

//End of script for creating docker image of nginx and our application

> docker image build -t dockerUsername/imageName . → This will create docker image for the given name. The command should be run while you are in the folder containing the Dockerfile, the dot(.) in the command means use in Dockerfile in the current directory.

> docker push dockerUsername/imageName → This will push the docker image to the user account in docker website.

COMPOSING NODE APP AND MONGODB IN DOCKER

We first create a Dockerfile in our application root directory as shown below.

```
FROM node:11

WORKDIR /usr/src/app

COPY package*.json ./

RUN npm install

COPY . .

EXPOSE 3000

CMD ["npm", "start"]
```

Then we also create docker-compose.yml file in our app root directory as shown below.

```
version: '3'
services:
 app:
   container_name: docker-node-mongo
   restart: always
   build: .
   ports:
     - '80:3000'
   links:
     - mongo
 mongo:
   container_name: mongo
   image: mongo
   ports:
     - '27017:27917'
```

We then add .dockerignore in the app root directory as shown below.

```
node_modules
npm-debug.log
```

To use the above compose and run our node app with mongo we follow the following steps.

> docker-compose up → To run the compose we created in yml file. This will download all the necessary images from docker hub and create containers to run them.

> docker-compose up -d → Same as above but it will run in background/detached mode.

> docker-compose down → This will remove all the containers and remove the network, because docker create network for you applications.

The application can now be deployed anywhere and run using docker compose.
THE END

CREATING SPRING BOOT or JAVA IMAGE

> docker build -f Dockerfile -t theNameOfTheImage . → This will create an image from the current working directory.

> docker run -d -p 8080:8080 dockerImageName → this will run the given docker image.