**Docker**

**What is Docker?**

Docker is a container management service and 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 the parts it needs, such as libraries and other dependencies and ship it all out as one package.

**Life before Docker?**

Before docker we used Hypervisors which is used to maintain/create the virtual machines which have its own OS, virtual machines use host resources.

Graphical user interface

Description automatically generated with medium confidence

**What is Docker Image?**

An image contains all the things that your application needs to run, it contains the software(Java 1.8), libraries, dependencies. When we run these command, an image is downloaded to our machine. So local image was created from registry.

**What is difference between Image and Container in Docker?**

Image is a static version and container is a running version of your image. For the same image multiple containers can be running.

Image is a class and container is an object, so from the image we can create a multiple containers like objects.

**What is DockerClient and DockerDaemon?**

DockerClient is responsible for sending commands to the docker daemon.

DockerDaemon is responsible for managing our local stuff and also pull something from the Image Registry if something is not available on our local. Docker Daemon also helps us to create an image and push that image to registry.

If you are using Window 10 and are using docker toolbox

=> Use 192.168.99.100 instead of localhost.

hub.docker.com is registry in which we push our images.

**Command** :

docker run -p 5000:5000 in28min/todo-rest-api-h2:1.0.0.RELEASE

***Explanation*** : When we run the above command docker daemon checks whether todo-rest-api-h2:1.0.0.RELEASE is present in our local machine or not. If it is not present then it will pull from in28min docker hub account and download the image in our local. The run keyword runs the image in a new container. By default we cannot access container port, so we have to explicitly host the containerport. For we use -p 5000:5000 => -p {hostport}:{containerport}.

**Command** :

docker run -d -p 5000:5000 in28min/todo-rest-api-h2:1.0.0.RELEASE

This command is used to run the container in detached mode. To see the logs we can use **docker logs [container id]**. To follow the logs we can command **docker logs -f [container id]**.

Command :

docker run -d –restart=always -p 5000:5000 in28min/todo-rest-api-h2:1.0.0.RELEASE

Whenever docker desktop restart, the container will restart by using **--restart=always** in docker command.

**Building a docker image from Dockerfile :**

Createa docker file with name Dockerfile and add below content

FROM openjdk:8-jdk-alpine

EXPOSE 8080

ADD target/hello-world-rest-api.jar hello-world-rest-api.jar

ENTRYPOINT ["sh", "-c", "java -jar /hello-world-rest-api.jar"]

In goals section do clean install so an executable jar will be created. Use this command to build the image from project directory.

docker build -t vishalkumar392/hello-world-rest-api:dockerfile1 .

dot( . ) represents current folder build context. **ENTRYPOINT** wont replace and arguments which passed while running the container while **CMD** will replace. **ADD** is used to add jar file to container.

**MYSQL docker image** :

docker pull mysql

docker run --detach --env MYSQL\_ROOT\_PASSWORD=vishal --env MYSQL\_USER=vishal --env MYSQL\_PASSWORD=vishal --env MYSQL\_DATABASE=student --name mysql --publish 3307:3306 mysql:latest

When we stop and restart the mysql container data will wont be lost.

If now we run our spring boot application in our STS IDE with datasource url to 3307 in properties file we can connect to the container which is running!!.

**Important point** : When we launch a spring boot java container which depends on our mysql container running on port 3307, by default those two containers can’t talk with each other. Because by default this two containers are launched into Bridge network. So the default networking mode of docker is Bridge.

**Commands to connect to containers** :

**docker container run -p 8080:8080 --link=mysql -e RDS\_HOSTNAME=host.docker.internal vishalkumar392/student-service:dockerfile1**

Here --link=mysql represents the name of mysql container and host.docker.internal is used to replace the localhost in our application properties file.

Other way to make containers talk to each other :

**Create a custom network** :

docker network create vishal

docker run --detach --env MYSQL\_ROOT\_PASSWORD=vishal --env MYSQL\_USER=vishal --env MYSQL\_PASSWORD=vishal --env MYSQL\_DATABASE=student --name mysql --publish 3307:3306 --network=vishal mysql:latest

docker container run -p 8080:8080 --network=vishal -e RDS\_HOSTNAME=host.docker.internal vishalkumar392/student-service:dockerfile1

**Volumes in Docker :**

Volumes are the preferred mechanism for persisting data generated by and used by Docker containers.

In our case we are using volumes to persit data of our mysql/java docker containers.

What should it contain? This should contain the data which is present inside the mysql database. So, this volume should map to a folder inside the MYSQL container which contains the MYSQL data.

Which is the folder inside the MYSQL image which contain the MYSQL data?

This is standarad folder inside a mysql. It's /var/lib/mysql. So, all the data created would be inside this folder.

**docker run --detach --env MYSQL\_ROOT\_PASSWORD=vishal --env MYSQL\_USER=vishal --env MYSQL\_PASSWORD=vishal --env MYSQL\_DATABASE=student --name mysql --publish 3307:3306 --network=vishal --volume mysql-database-volume:/var/lib/mysql mysql:latest**

what we are doing is, we are creating a voulme, we are creating a volume on the host machine which maps to this particular directory inside the conatiner.

volumes help us to persist data to host. Docker data volumes help us to share data between file system and the Docker container. so we are creating something on the host file system and mapping it to a folder on the docker conatiner

|  |  |
| --- | --- |
| **commands** | **Use** |
| docker conatiner ls | It shows the containers which are running. |
| docker container ls -a | It shows the containers which are stopped and running. |
| docker container stop [container id] | It stops the container with that id. |
| docker stop $(docker ps -a -q) | It stops all running containers. |
| docker container kill [container id] | It stops immediately. |
| docker container pause [container id] | It pauses the container. |
| docker container unpause [container id] | It unpauses the container. |
| docker container inspect [container id] | It shows the container information like, the host ports etc. |
| docker rm [container id] | It remove a container. |
| docker container prune | It removes all stopped containers. |
| docker rm $(docker ps -a -q) | It removes all stopped containers. |
| docker images | It shows all images. |
| docker image remove [container id] | It removes the image. |
| docker image prune -a | It removes all images. |
| docker image history [container id] | It shows the history of image. |
| docker pull mysql | It pulls the image |
| docker search mysql | It tells about image. |
| docker network create vishal | It creates a custom network vishal. |
| docker network ls | It show all availble networks. |
| docker network inspect [name] | It show the info of network. |