**After Docker Desktop Install and Login**

Now, we can open the terminal to check some Docker commands.

**docker –version**

 command should return the Docker version. In my case, this command returns

*Docker version 19.03.2, build 6a30dfc*

**docker info**

 command should return detailed information of the installed Docker machine of the host machine. Initially, you have zero images and containers.

**docker pull hello-world**

 This command should pull a hello-world image from the Docker registry. Now again, if we run the Docker info command, we can see that the image count is 1. But the container count is still zero.

**docker run hello-world**

 This command should create an instance of the hello-world image. It will return a long message:

***Hello from Docker!***

***This message shows that your installation appears to be working correctly.***

If we run the Docker info command, we can see that the image count is 1 and the container count is also one. So the Docker installation is complete.

**Setup MySQL**

Now we will create and run an image of the MySQL database. From our terminal, we will run the below command. Here, **-d** in this command indicates that the Docker command will run in detached mode.

Docker run -d -p 6033:3306 --name=docker-mysql --env="MYSQL\_ROOT\_PASSWORD=root" --env="MYSQL\_PASSWORD=root" --env="MYSQL\_DATABASE=book\_manager" mysql

Hopefully, the MySQL image is pulled and running as a container. To check this, we can run

**docker image ls** and **docker container ls** commands. In my case, the responses of these commands are:

mysql latest b8fd9553f1f0 3 days ago 445MB

hello-world latest fce289e99eb9 8 months ago 1.84kB

CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES

5db66654ba6a mysql "docker-entrypoint.s…" 13 minutes ago Up 13 minutes 33060/tcp, 0.0.0.0:6033->3306/tcp docker-mysql

Now we can check by logging in to MySQL.

docker exec -it docker-mysql bash;

It will take us inside the docker-mysql container. Then we will log in to MySQL using

**mysql –u root –p**

  using password root. Then, we will run the

**show databases;**

 command to see if the database setup is complete or not.

In my case, it returns the below result:

+--------------------+ | Database | +--------------------+ | book\_manager | | information\_schema | | mysql | | performance\_schema | | sys | +--------------------+ 5 rows in set (0.00 sec)

So we can say that the book\_manager database was created inside the docker container. We can externally use this database from our host machine by using port 6033. Now we have to import the database script to the Docker MySQL database. The SQL script is available [here](https://github.com/sanjoy-sust/book-manager/blob/master/sql/book_manager.sql). Run the following command to import this script to docker-mysql.

docker exec -i docker-mysql mysql –u root –p root book\_manager <book\_manager.sql

Hopefully, the book\_manager script executed successfully. You can confirm by executing the following command.

$ docker exec -it docker-mysql bash;

root@5db66654ba6a:/# mysql -uroot -p

Enter password:

mysql> show databases;

+--------------------+ | Database | +--------------------+ | book\_manager | | information\_schema | | mysql | | performance\_schema | | sys | +--------------------+ 5 rows in set (0.00 sec) mysql> use book\_manager Database changed

mysql> show tables;

+------------------------+ | Tables\_in\_book\_manager | +------------------------+ | author | | book | | book\_author | | book\_publisher | | book\_tag | | bookshelf | | publisher | | tag | +------------------------+ 8 rows in set (0.01 sec) mysql>

**Application Clone and Build Project**

I have already pushed my code to my GitHub repository. Anyone can clone the codebase from [here](https://github.com/sanjoy-sust/book-manager). I think your host machine has Gradle setup. So now we run the **gradle build** commandto build the project. So the executable jar file is created at the **build/jar** directory of your cloned project.

Now open the Dockerfile. We can see that the file contains the following commands:

FROM java:8

VOLUME /tmp

EXPOSE 10222

ADD /build/libs/book-manager-1.0-SNAPSHOT.jar book-manager-1.0-SNAPSHOT.jar

ENTRYPOINT ["java","-jar","book-manager-1.0-SNAPSHOT.jar"]

This file contains sequential commands to execute in docker. It will create an image of java 8. and also it will copy jar file from host machine to docker image and then run command which is given at entrypoint arguments. Now we will build a docker image by using this **Dockerfile**.

docker build -f Dockerfile -t book\_manager\_app .

This command will create a Docker image named **book\_manager\_app** to the Docker machine. Here, the **-f**  command indicates the Docker file name. Now we will run this image as a container.

docker run -t --link docker-mysql:mysql -p 10222:10222 book\_manager\_app

The **--link** command will allow the **book\_manager\_app** container to use the port of MySQL container and **-t**  stands for**--tty**, which will allocate a pseudo-terminal.

After running this command, we will hit <http://localhost:10222/book> from our host machine browser, and it will return a list of books.