**DOCKER === IP 10.0.75.1 DOCKER HUB ID : 151297me**

An **image** is an executable package that includes everything needed to run an application--the code, a runtime, libraries, environment variables, and configuration files .

A **container** is a runtime instance of an image--what the image becomes in memory when executed (that is, an image with state, or a user process). You can see a list of your running containers with the command, docker ps, just as you would in Linux .

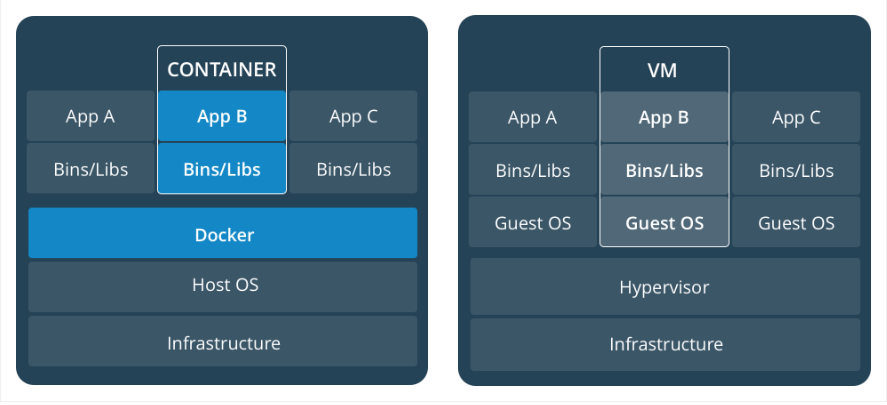
**VIRTUAL MACHINE VS DOCKER (CONTAINER)**

A **container** runs natively on Linux and shares the kernel of the host machine with other containers. It runs a discrete process, taking no more memory than any other executable, making it lightweight.

By contrast, a **virtual machine** (VM) runs a full-blown “guest” operating system with virtual access to host resources through a hypervisor. In general, VMs provide an environment with more resources than most applications need.

**WHAT IS A HYPERVISOR ?**

A **hypervisor**, also known as a virtual machine monitor, is a process that creates and runs virtual machines ([VMs](https://www.vmware.com/topics/glossary/content/virtual-machine)). A hypervisor allows one host computer to support multiple guest VMs by virtually sharing its resources, like memory and processing.



**DOCKER TOOLBOX RELEASES**

<https://github.com/docker/toolbox/releases>

**DOCKER TOOLBOX INSTALLATION STEPS**

<https://docs.docker.com/toolbox/toolbox_install_windows/>

**When Docker Complains of Hyper-V , just remove the line number 69 in start.sh file to below line :**

"${DOCKER*MACHINE}" create -d virtualbox --virtualbox-no-vtx-check $PROXY*ENV "${VM}"

Create a Spring Boot Project

Create a test RestController

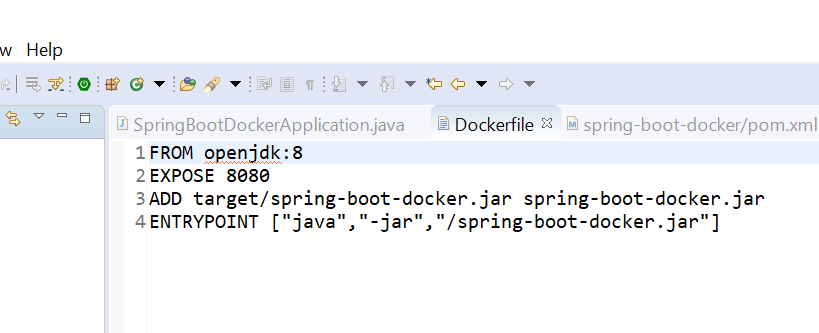
Create a Dockerfile

Give a custom jar name in pom.xml using <finalName> annotation

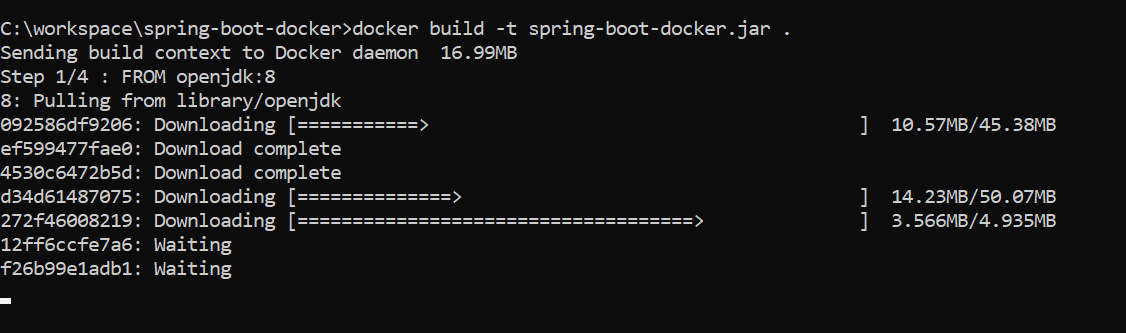
Run maven install phase

That will create jar in target folder

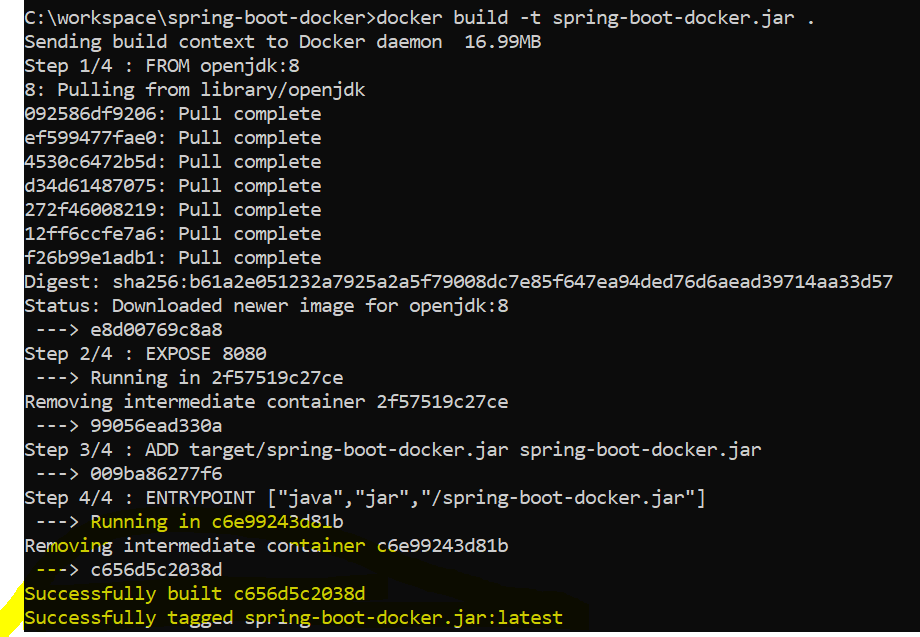
Bare bones Dockerfile contents :



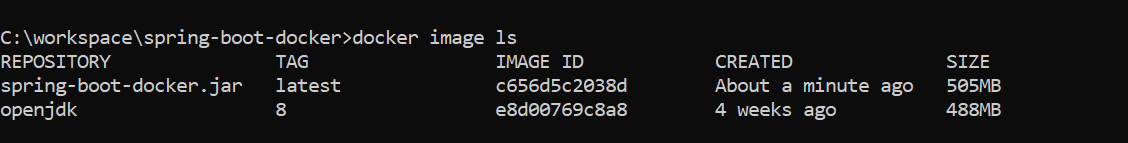
**After giving docker build -t command , docker will start building your image as below :**



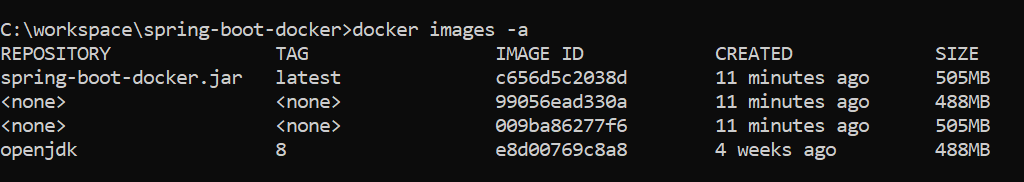
**After the image is build cmd will look like this**



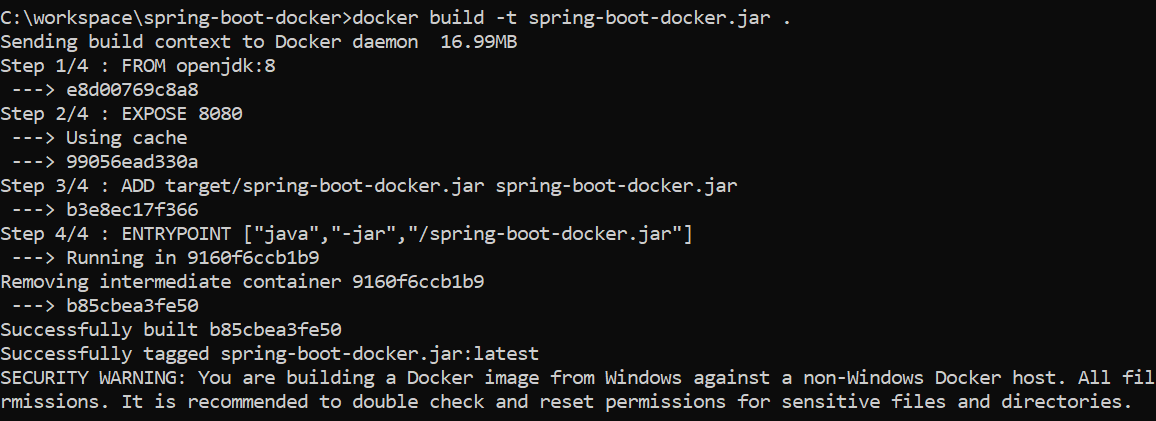
**To see the current images that we successfully built do as below :**



**We can also use below command :**

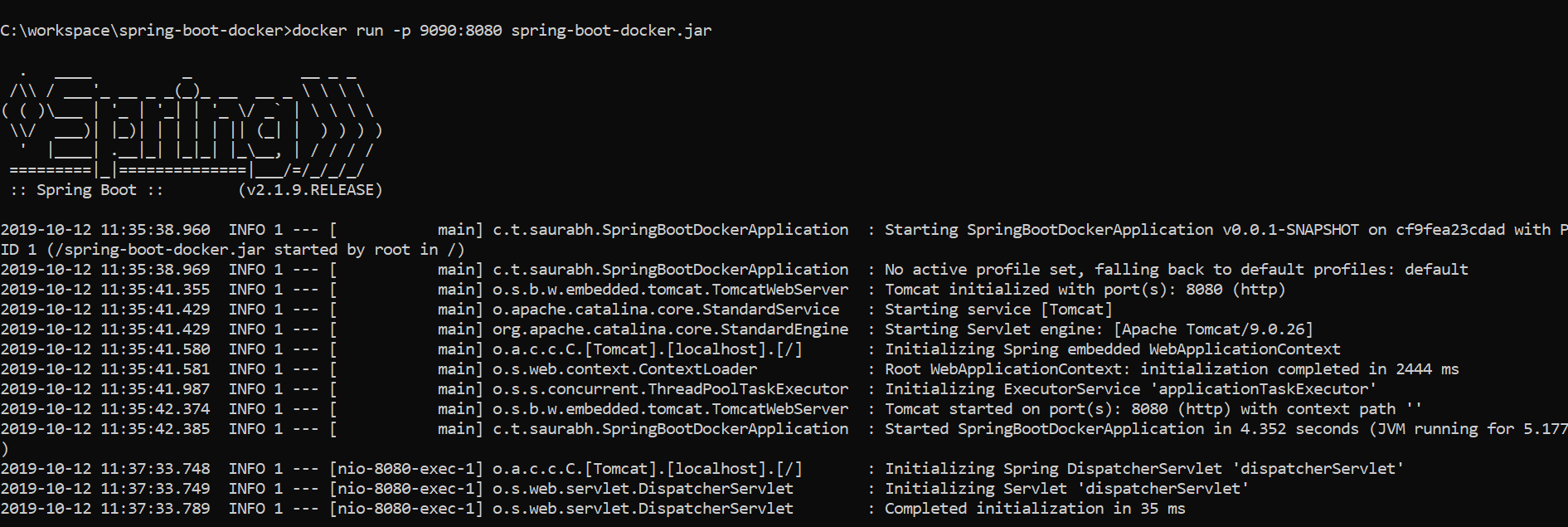


**Docker image build cmd logs**

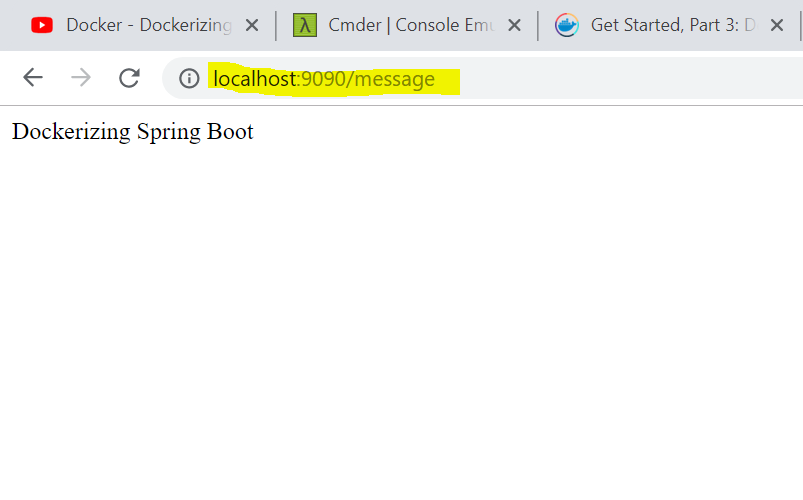


**If image runs successfully after the command**

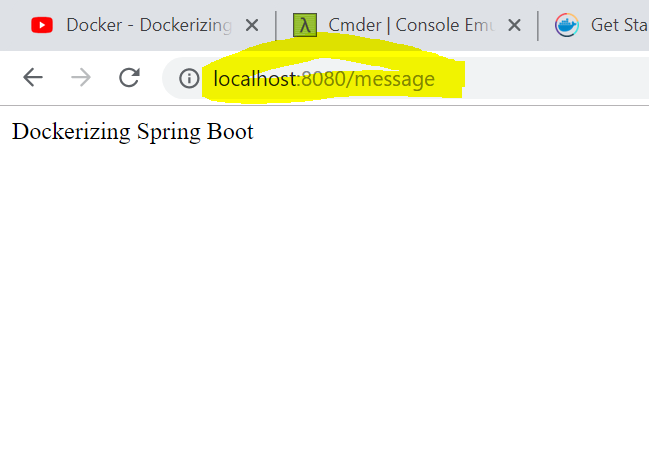
**Docker run -p 9090:8080 spring-boot-docker.jar , you will see below in CMD**



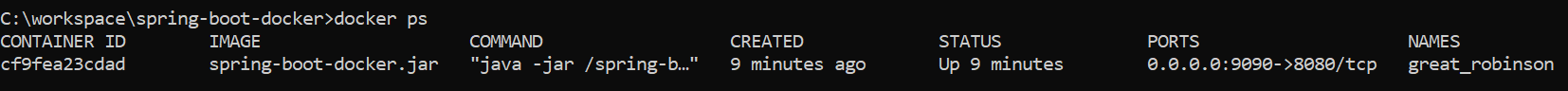
**Rest endpoint /message is successfully running on masked port 9090 against the exposed port 8080**



**Also locally its up too**

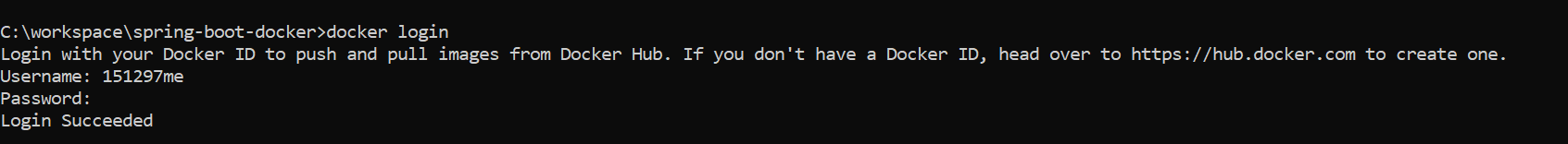


**All running images in docker hub can be checked as below**

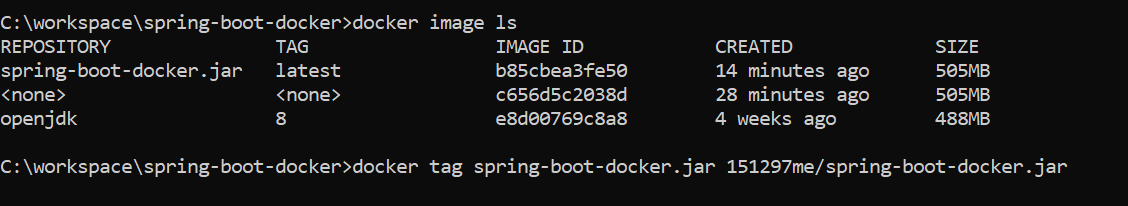


# PUSH AN IMAGE INTO DOCKER HUB

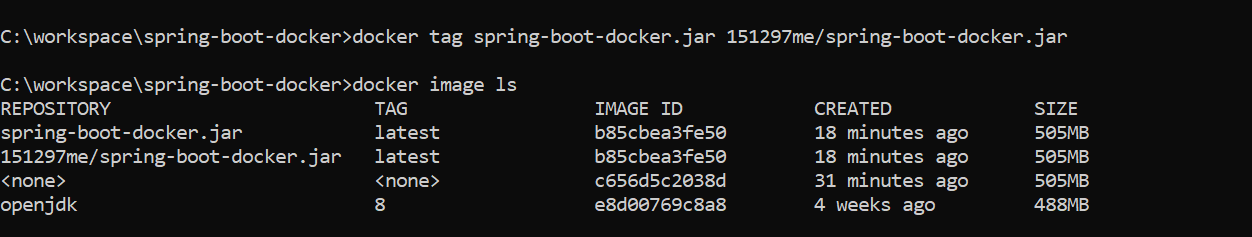
**Login as below in Docker Hub**



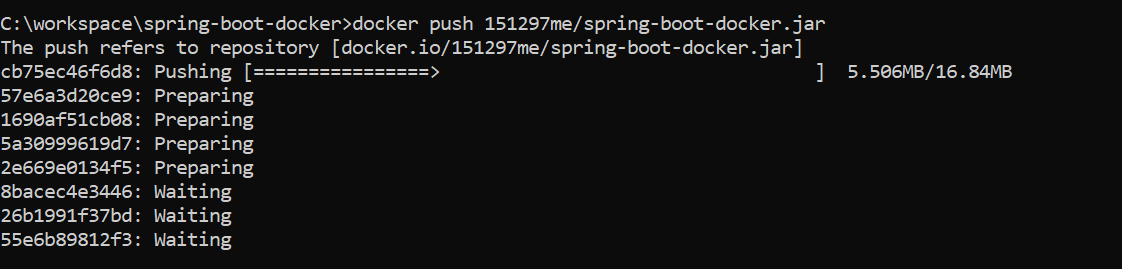
**Before pushing image into Docker Hub we need to provide it a TAG as below**



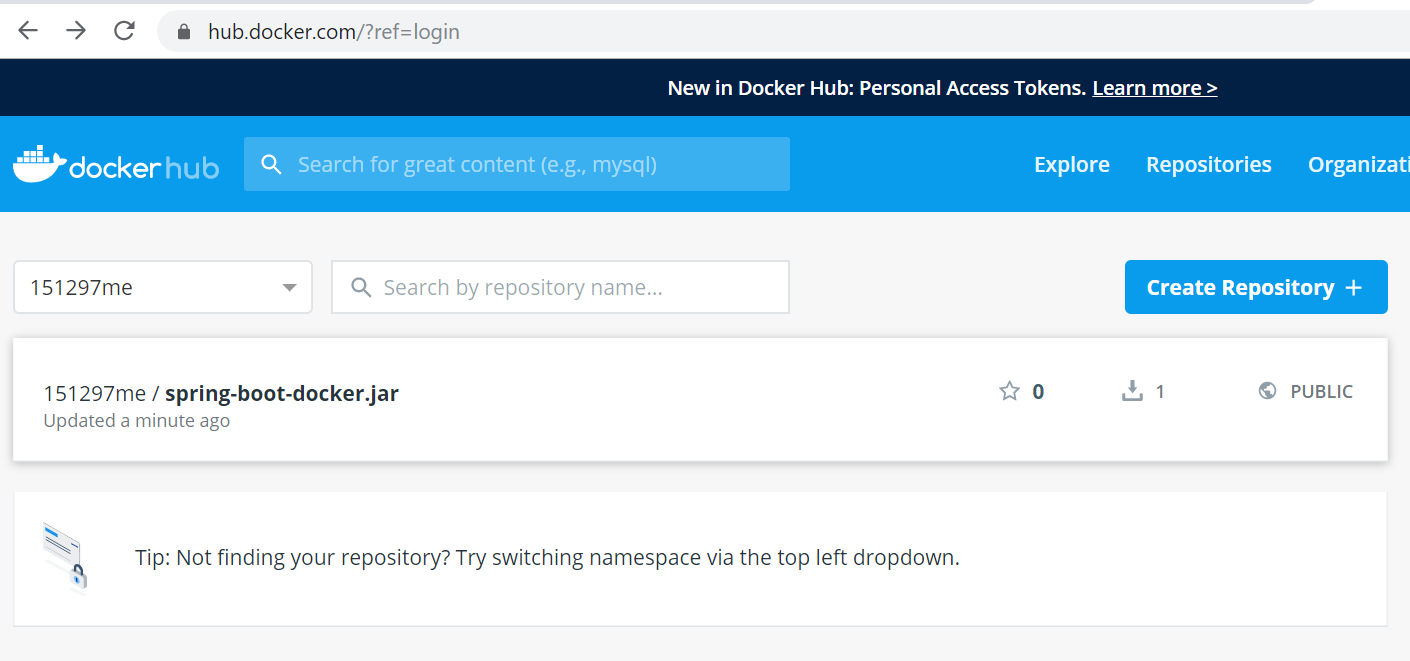
**We verified that the tag we gave is reflected**



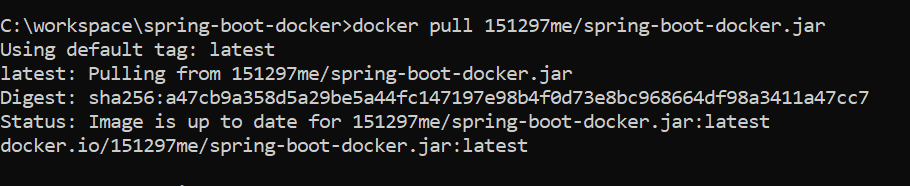
**After tagging , we start pushing image into the Docker Hub using the tag we provided to it as below:**



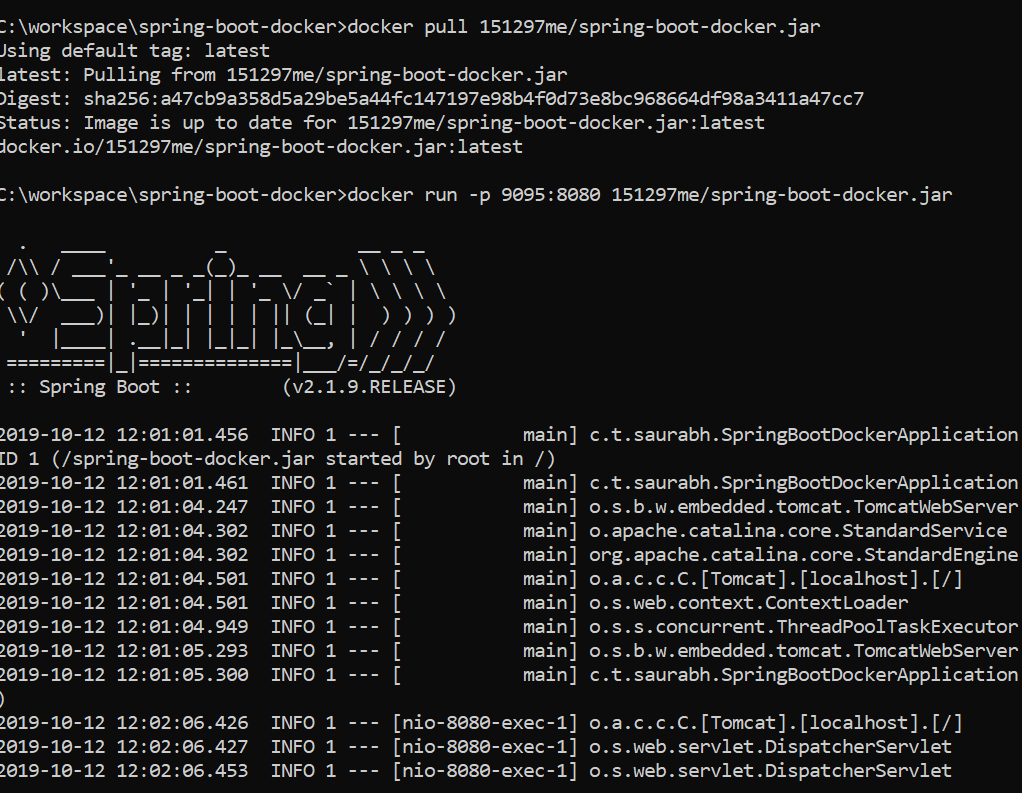
**Image is successfully pushed to Docker Hub 😊**



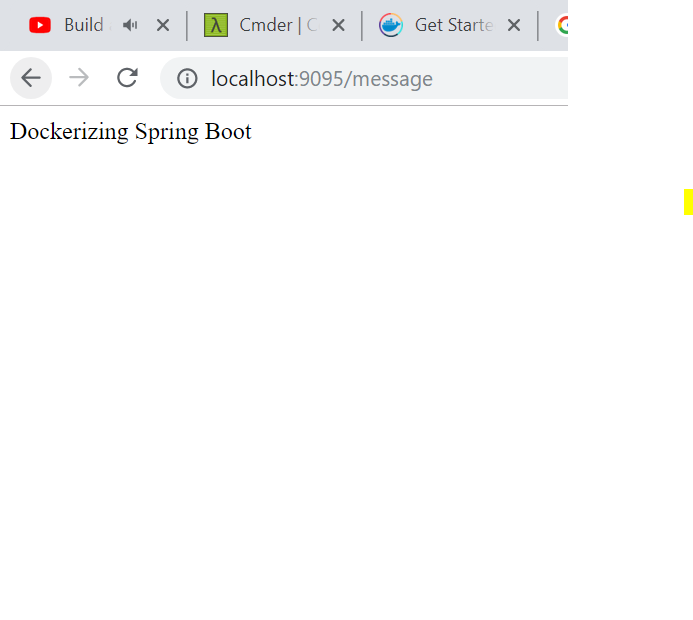
**Now pull image from Docker Hub using Tag name as below , successfully pulled**



**Run pulled up image on port number 9095**

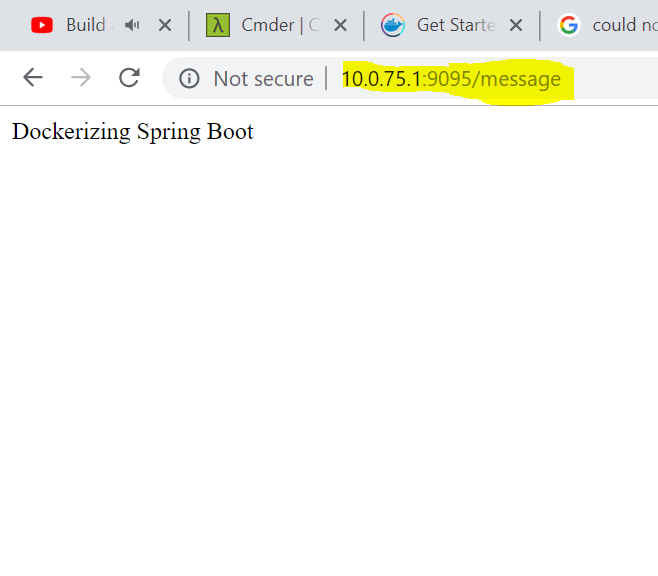


**Service up on port 9095**



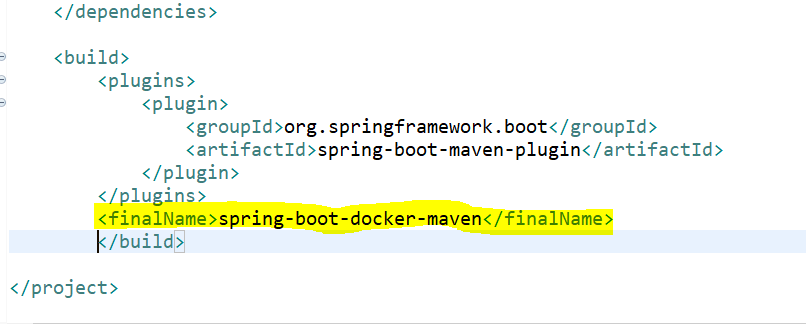
**Verified Running on the actual system IP as well**

**10.0.75.1**



# BUILD AND PUSH DOCKER IMAGES USING MAVEN PLUGIN

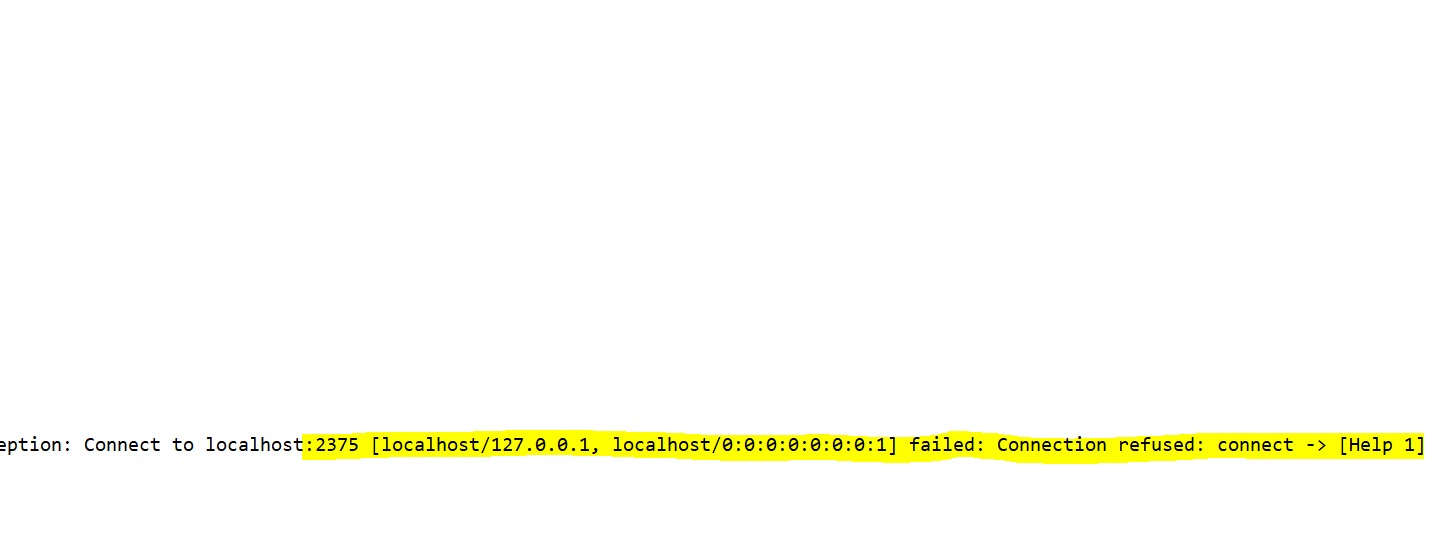
Create a JA R again from a bar bones spring boot project and name it as below :



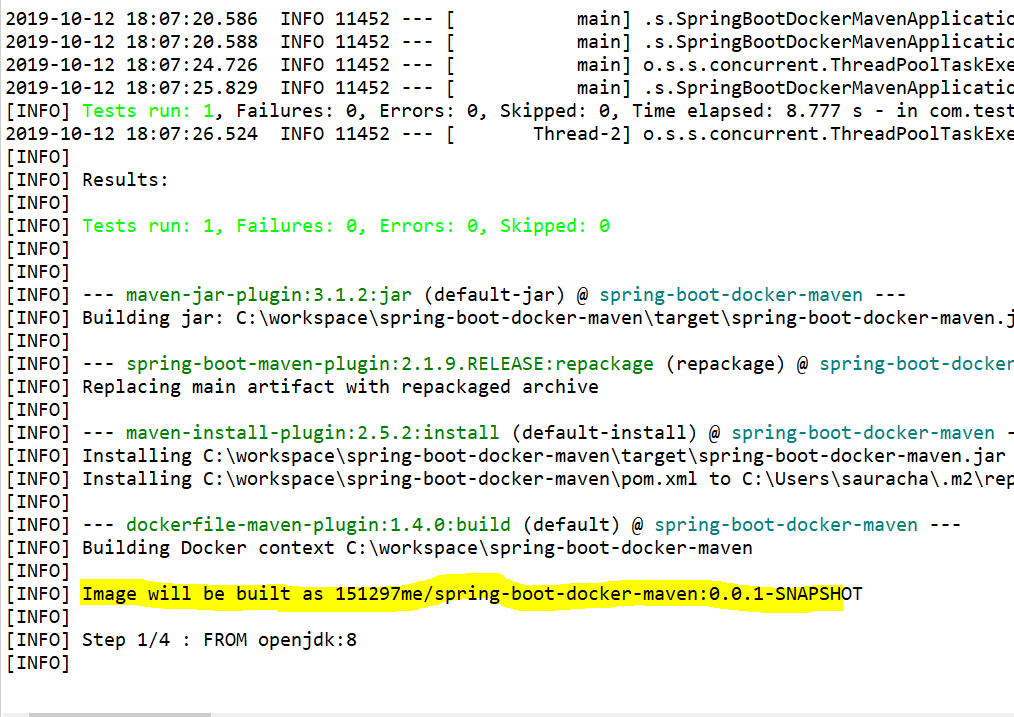
Run maven install to generate the jar in the target directory .

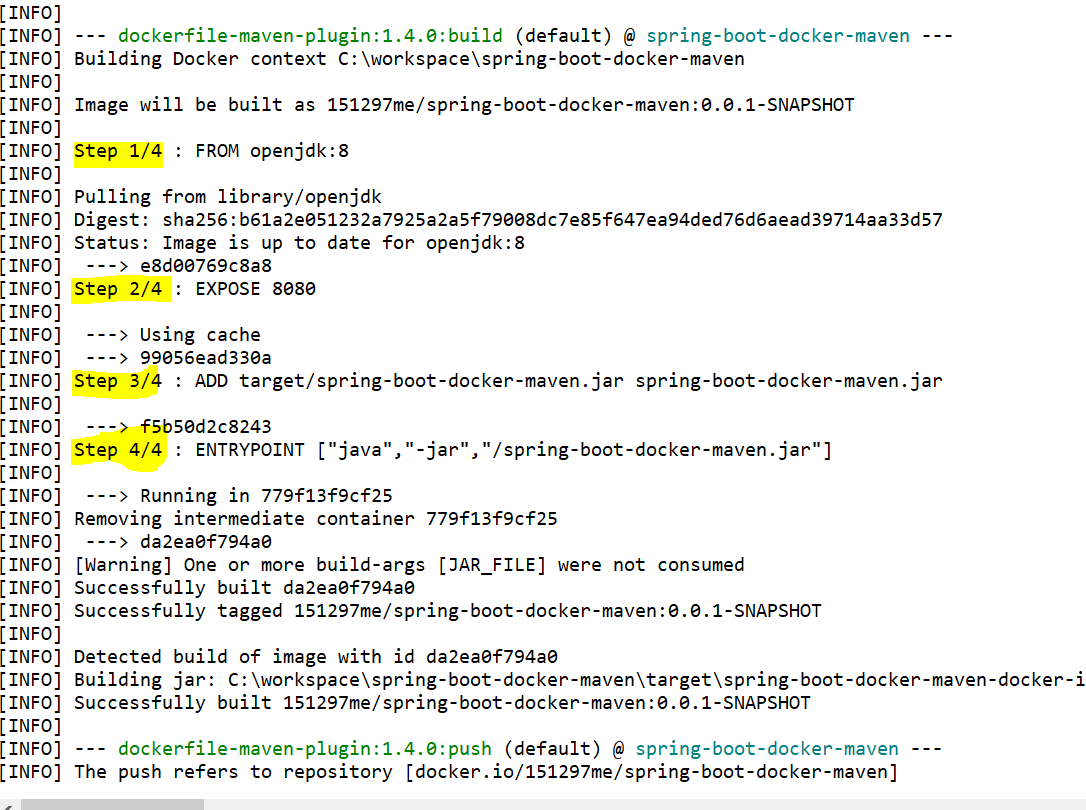
While building using dockerfile-maven-plugin ; got into this issue below





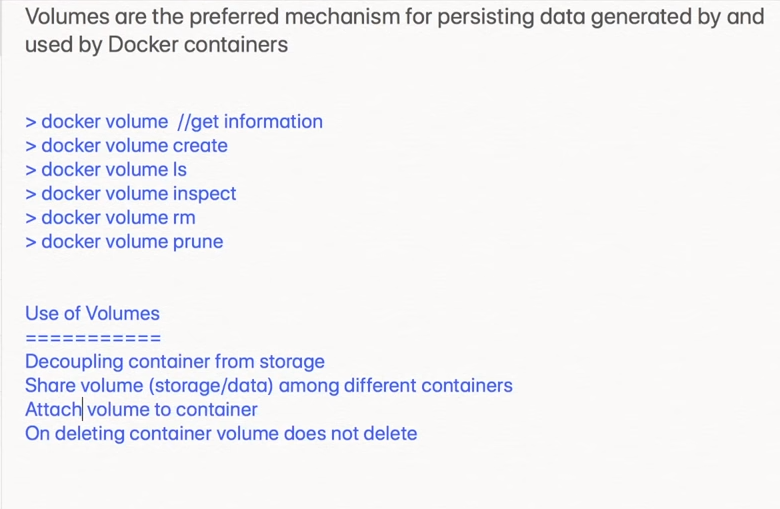
Plugin started to build the image

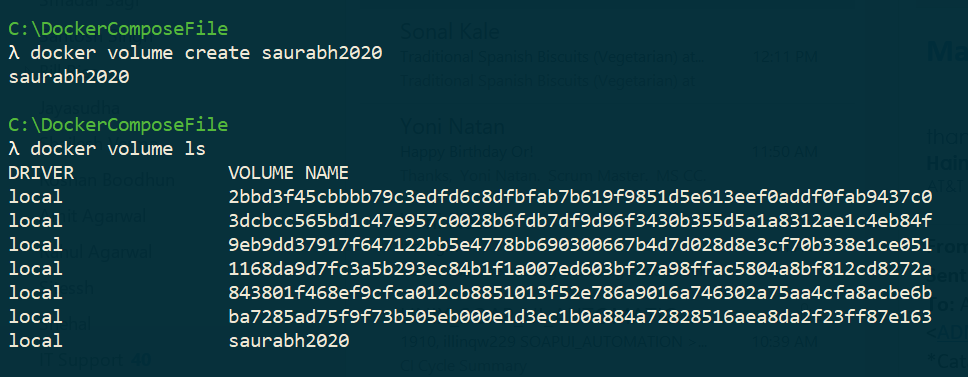


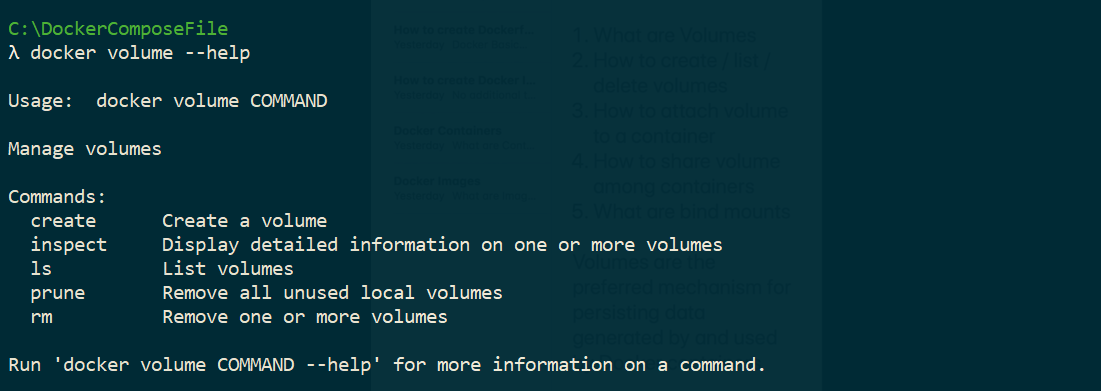


Docker Compose Version Details : <https://docs.docker.com/compose/compose-file/>

**DOCKER VOLUMES**

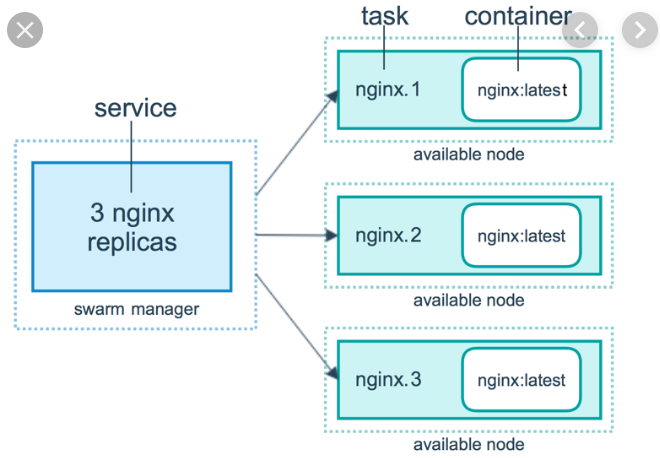


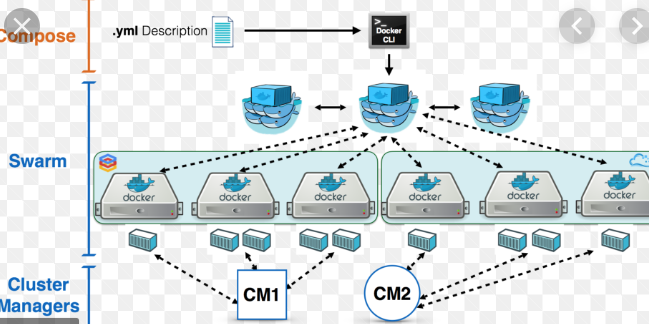


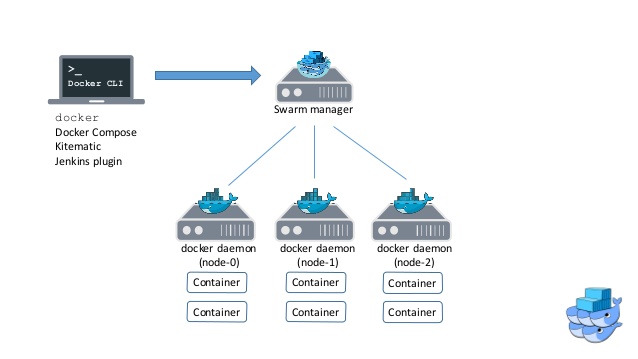


**Bind Mount is a physical volume mount using Docker Volume Options .**

**DOCKER SWARM**

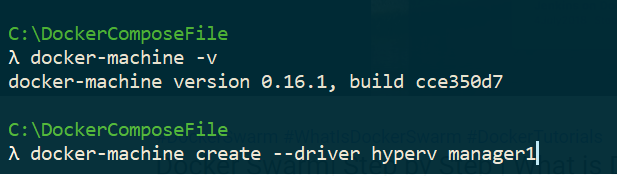






**docker-machine -v**

**creating a manager node :**



**Containerization Options**

**Kubernetes**

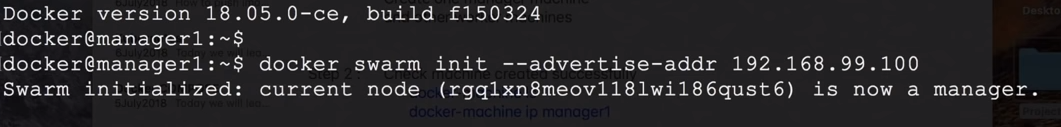
**Docker Swarm**

**Apache Mesos**

**Find IP of a docker-machine**

**docker-machine ip {machineName}**

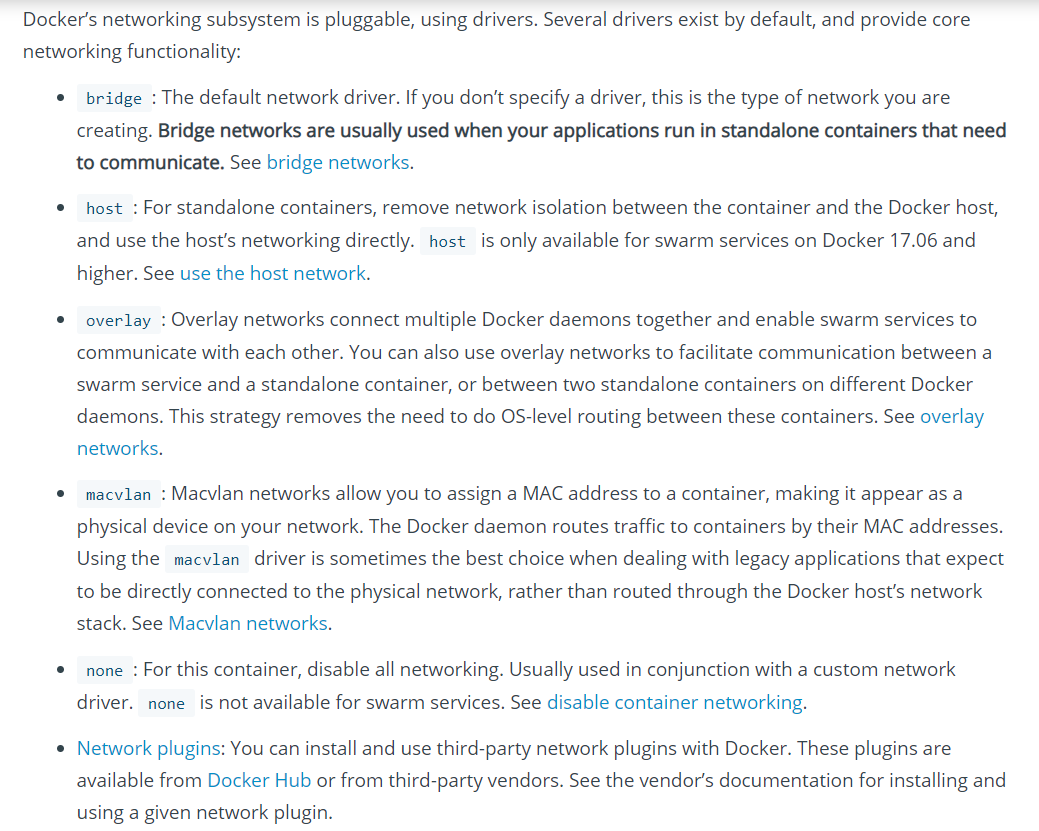
**Command to designate a docker-machine as the manager**



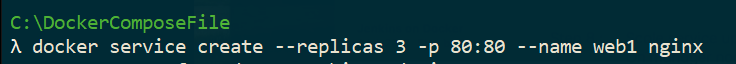
**Docker node ls (only be run from manager)**

**Docker-swarm join token worker (to add a worker node to the swarm)**

**Docker-swarm join token manager (to add a manager node to a swarm)**



**SCALING DOCKER SWARM CONTAINERS BY REPLICAS**



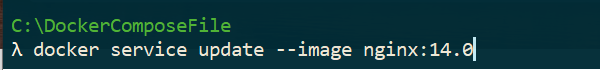
**To check all the replicas if they started running**

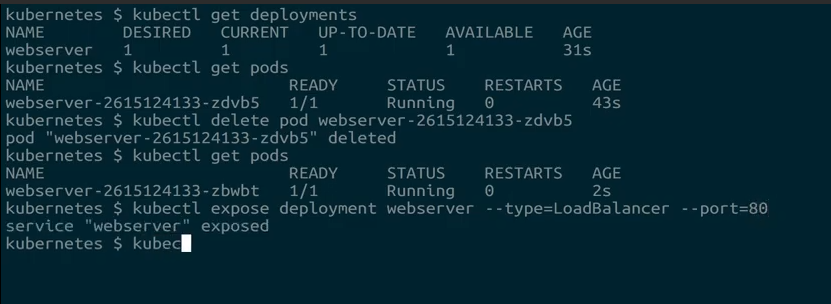
**docker service ps web1**

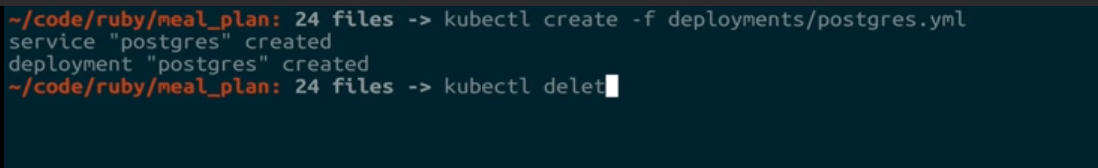
**docker service scale web1 = 2**

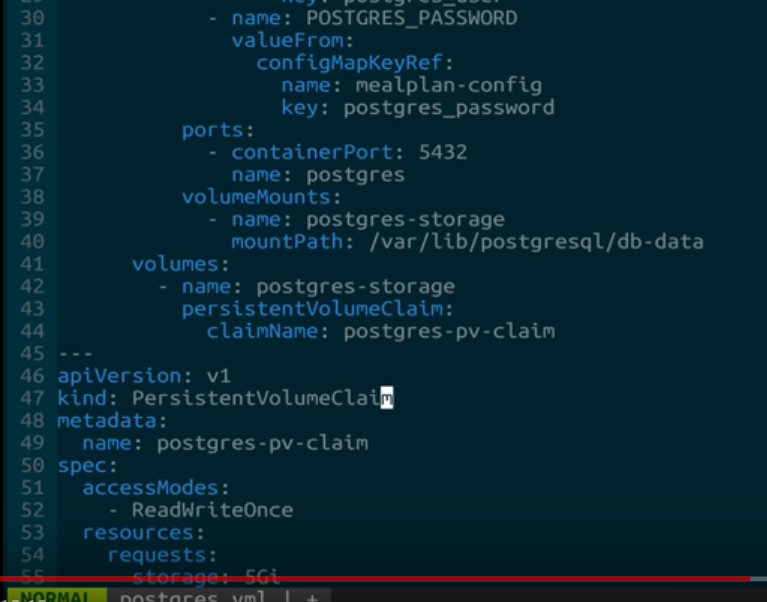
**docker node inspect web1**

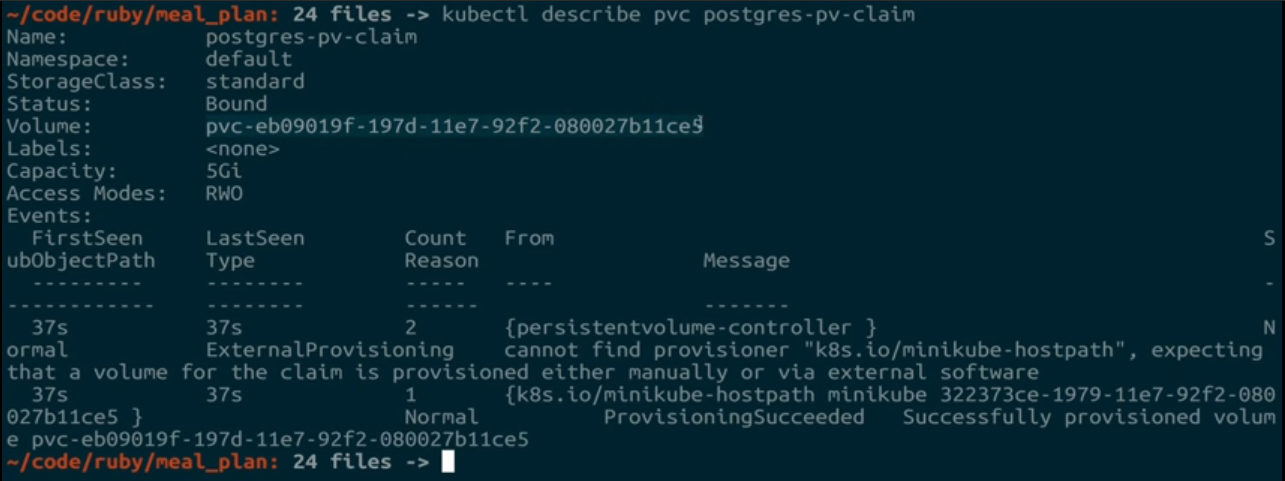
**Docker updating a Service**

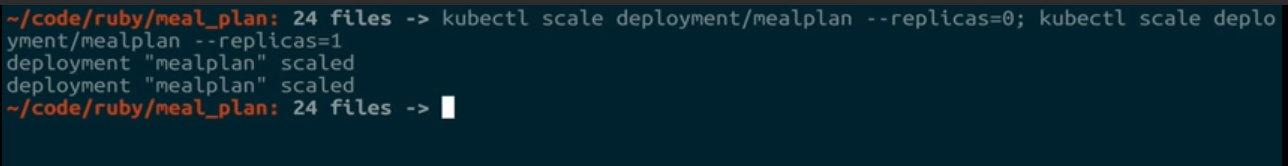












<https://www.youtube.com/watch?v=3dOWkiBEjXY&list=PLH1ul2iNXl7uhEOpPBYyUVktV7nW_15sh>

**Docker Networking**

**CMD vs ENTRYPOINT**

**CMDs can be overridden**

**ENTRYPOINT cannot be overridden but subsequent commands get appended .**

**CMD & ENTRYPOINT can be used together , where CMD can be used as argument for ENTRYPOINT**

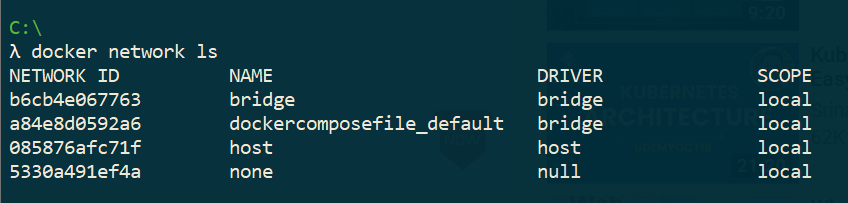
**Both CMD & ENTRYPOINT commands can be overridden from command line .**

**DOCKER BRIDGE NETWORKING DRIVER NOTES**

<https://www.youtube.com/watch?v=Tx12haz-4VA>

**Bridge network is used for containers to communicate within a host on a single machine .**

1. **Default**
2. **Custom**

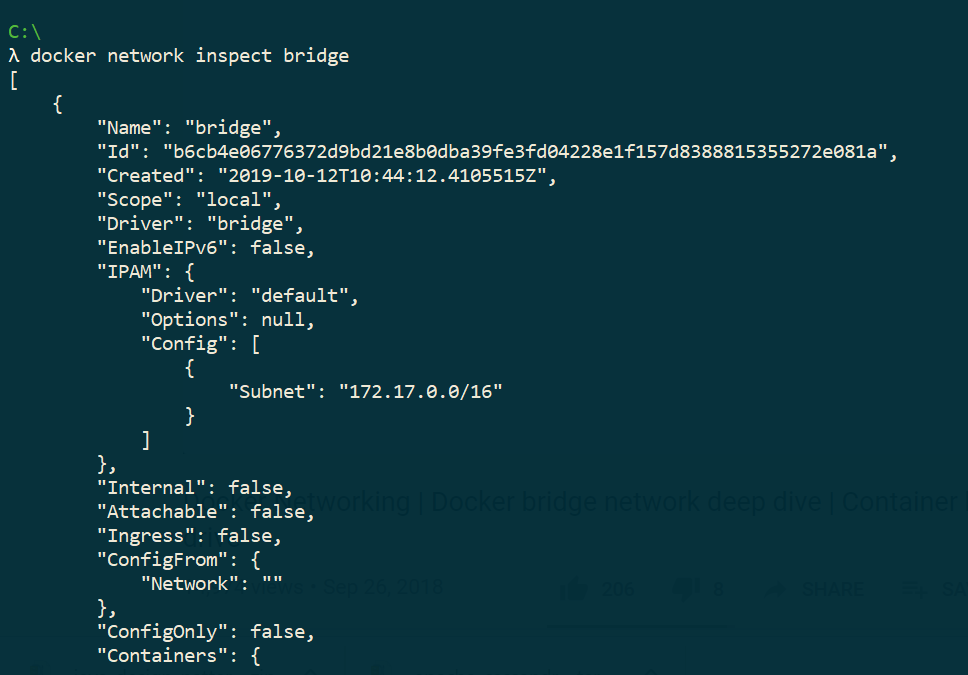


**In Bridge network , containers can only use IP Addresses to communicate , and not the container names .**

**Create a custom bridge network :**

**Docker network create --driver = bridge abc**

**Docker run itd –name =”containerName” --network =” abc alpine”**



**Signing into a container :**

**Docker attach alpine1**

**Custom bridge network allows you to communicate using names**

<https://www.youtube.com/watch?v=3dOWkiBEjXY&list=PLH1ul2iNXl7uhEOpPBYyUVktV7nW_15sh&index=1>

**Docker Tutorial for beginners| Writing Dockerfile | Dockerfile Instructions Tutorial**

**Docker images**

**Docker pull imageName : version**

**Docker rmi {imageId || image name}**

**Docker rmi image1 image2 image3**

**We can delete an image locally , if any container is using it**

**Docker image -q**

**This command returns ids of all images . So we can use this as a subquery to delete all the images**

**docker rmi ${docker image -q}**

**docker inspect {imageId || image name}**

**docker run -d -p 8085:8080 image /path to the image**

**docker ps ===== Lists all running containers**

**docker ps -a ====== running containers + stopped containers**

**docker restart { id || name}**

**docker rm {id || name}**

**docker rm {id 1 id 2 …|| name1,name2 …}**

**docker rm -f (Removes docker containers forcefully)**

**docker ps -aq {container id}**

**docker rm ${docker ps -aq} removes all contaners using ids as sub-quey**

**GET INTO CONTAINER to debug issues ……………**

**Docker exec it myweb /bin/bash**

**Docker logs {id || name}**

**Docker inspect {id || name}**

**BUILD A DOCKER IMAGE**

**docker build -t**

**RUN A DOCKER IMAGE**

**Docker run -d -p {imageName} / {location of the image }**

**Detached mode**, shown by the option --**detach** or -d , means that a **Docker** container runs in the background of your terminal. It does not receive input or display output. **docker** run -d IMAGE.

# **What is the difference between the COPY and ADD commands in a Dockerfile ?**

Using Copy we can only copy the local files+folders …but using ADD we can copy local + remote files +folders…

Docker image list