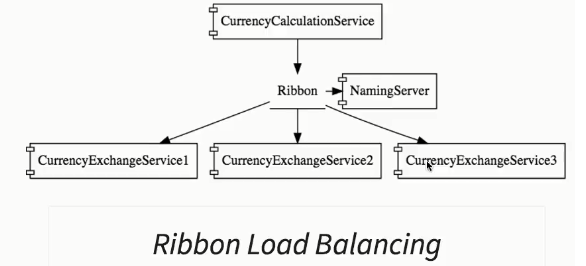
Role of Docker and Kubernetes in Dynamic Scaling :- how to deploy your software applications as portable, self-sufficient containers that can run on almost any server. As a platform for preparing and running distributed software applications, Docker is often compared to configuration management tools like Puppet or Ansible, or virtualization tools like VirtualBox or VMware, but Docker is actually in a category of its own, offering new ways to run software applications.



Here, whenever the CurrencyCalculation service needs to talks to the currencyexchange service, What it does, it would talk to the naming server, find out what are the active instances and ribbon would be used to distribute the load amongst the active currencyExchange service instances.

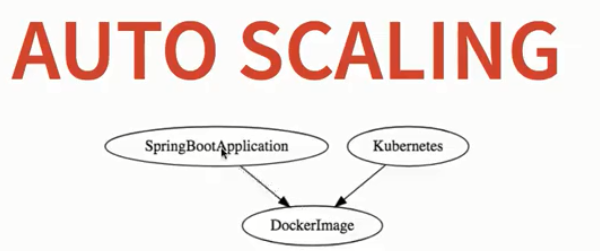
Untill now, we were bringing instances of each of these services manually. We were doing the right click -> changing the port -> and bringing up the new instance of the currency exchange service but when we talk about the real world ( amazon, google etc ), all this happens automatically.

So somebody needs to decide that a new instance of the currency exchange service should be made active, somebody should decide that an existing instance of a currency exchange service needs to be killed. That’s obviously based on the amount of load. So If I am having huge number of requests for the currency exchange service coming in, then I would want more instances active, if I have less amount of load, I would want less instances active. So who makes that decision ? How do you make those choices ?

That’s a very important factor when it comes to implement the microservices in the real world. The most important thing that you need to recognize, is the fact that the implementation of this functionality goes outside the boundary of spring boot or spring cloud.

A typical solution for that, would involve creating containers and having them managed by kubernetes.

Docker is one of the most popular ways to create containers.



Container : - Container is something which you can use to package and have your application in a really runnable state so you have the application jar ( for eg CurrencyConversionService or currencyExchangeService), we can package it and we can create a image for it ( DockerImage/ContainerImage).

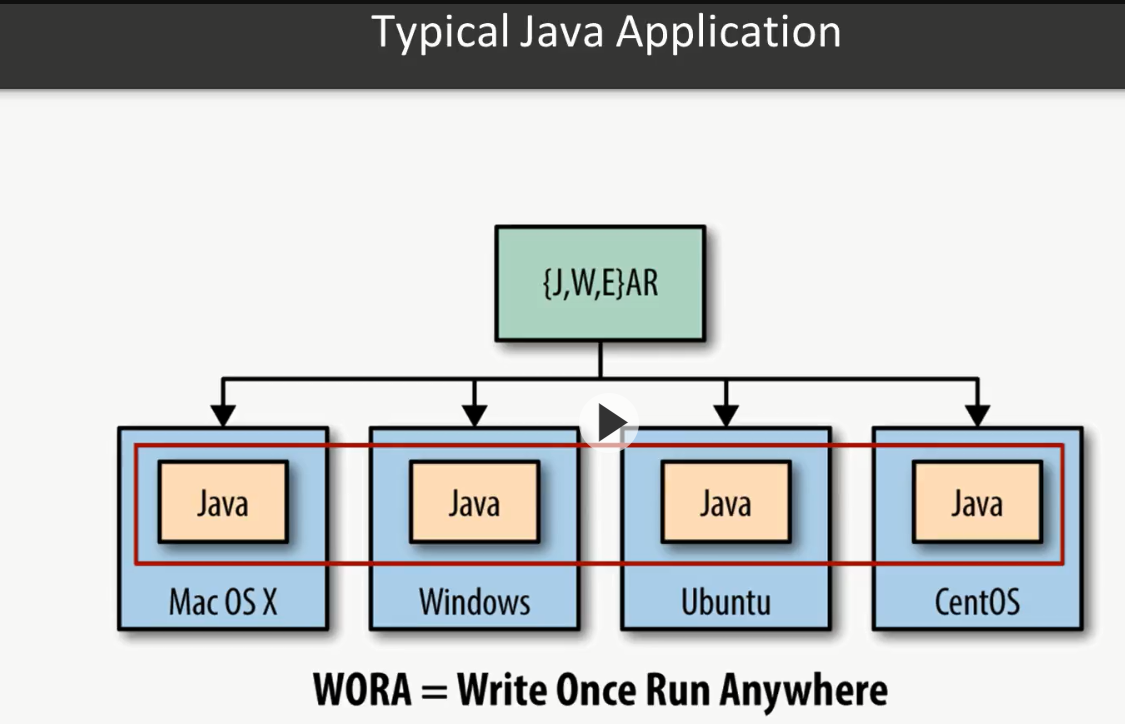
Once you have the container, you can take it and run it anywhere and those kind of containers are managed by Kubernetes.

Kubernetes is an open-source system for automating deployment, scaling, and management of containerized applications.( <https://kubernetes.io/>).

Production-Grade Container Orchestration - > Automated container deployment, scaling, and management

So what we can do for our existing applications, we can create docker image just for all of them, Once you have the docker images for all of them, you can have kubernetes managed them and react to the amount of load present in that specific thing. Containers and Dockers and Kubernetes.

**Introduction to Docker and Problem it solves : -**

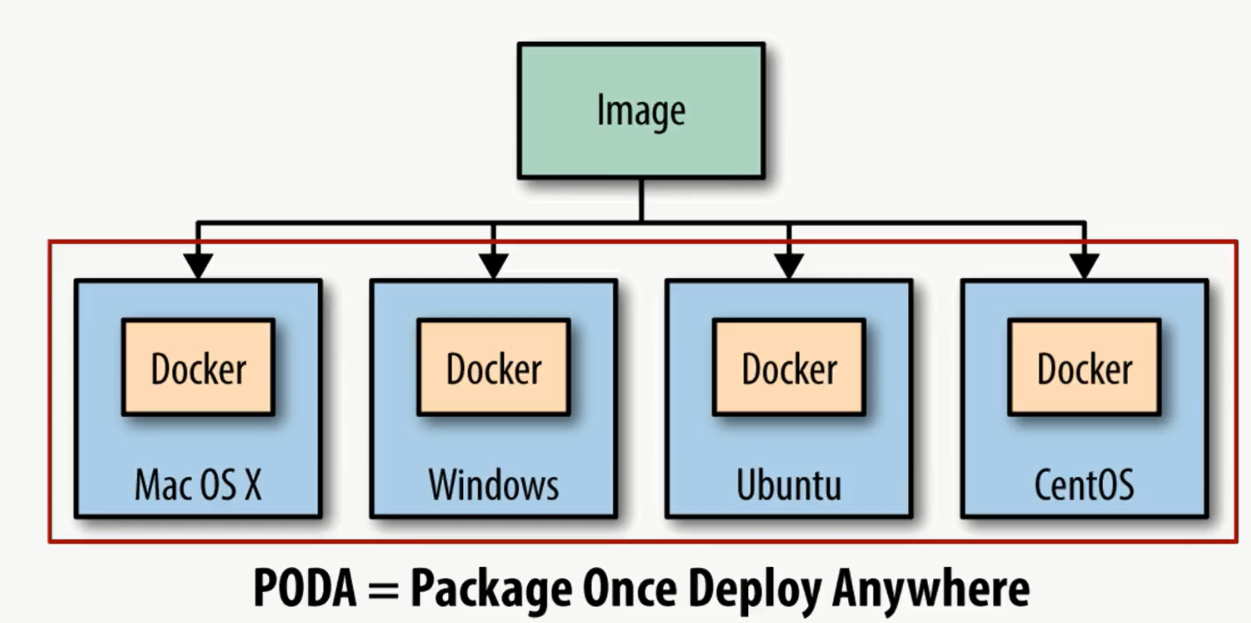


So let's see an example of a typical Java application. So this is right ones run everywhere.

But on the each of those operating system, Java must be installed, and Java in the proper version.

So for example if the source application was written in Java dot 1.6, on every computer there will be need for Java 1.6 at least, or Java 1.7, or Java 1.8. But it needs to be installed and this is a prerequisite.

**Docker gives us one step further.**



so we are creating our Java application and we can package it into Docker image.

Then on the every operating system we don't need to install anything besides Docker.

We are starting Docker image using Docker platform, and every dependency is downloaded automatically, we don't need to install anything manually, Docker will keep this up to and everything will work on every operating system. So we are one step ahead.

So in Docker we have couple of phases.

First phase( Build Phase ) is a beautiful, it provides tools you can use to create containerization applications. Developers package the application, its dependencies, and infrastructure as a read-only template. And those templates are called **Docker image.**

Ship phase allows you to share this application in a secure and collaborative manner. Docker images are stored, chart, and managed in a Docker repository, it is called a Docker registry. **Docker Hub** is a public available registry(default), but you can also have your own private registry and use it in your private enterprise network.

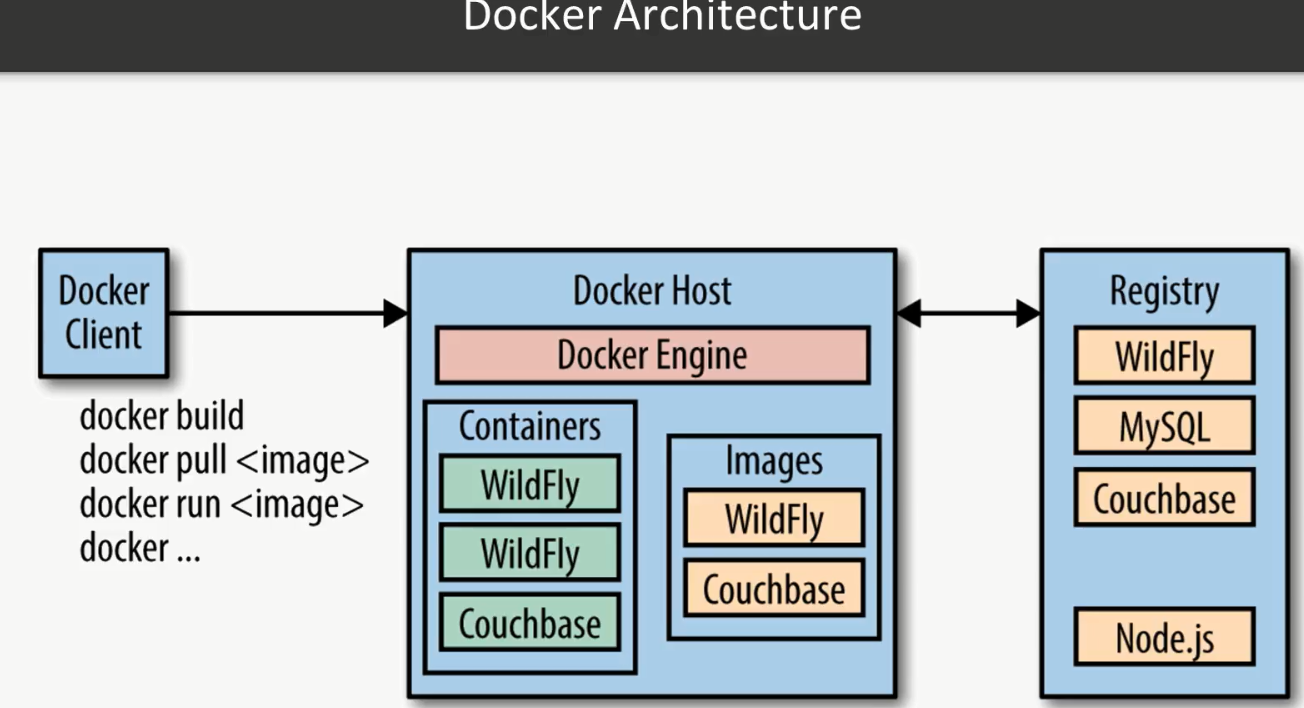
Last phase ( Run Phase ) in which we are interested is called run, and run phase gives us the ability to deploy, manage, and Scout this application. So we can scale those applications independently.**Docker container is a runtime representation of an image.** Every container can be run, started, scaled, stopped, moved, and deleted.

Docker Architecture : -

1. Docker client - this is a common light interface that is installed on some machine. We can use that client to connect to Docker host.
2. Docker host - Docker host is a Docker application that will be installed on our local machine.

So there is an execution of images, creating containers, and so on and so on.

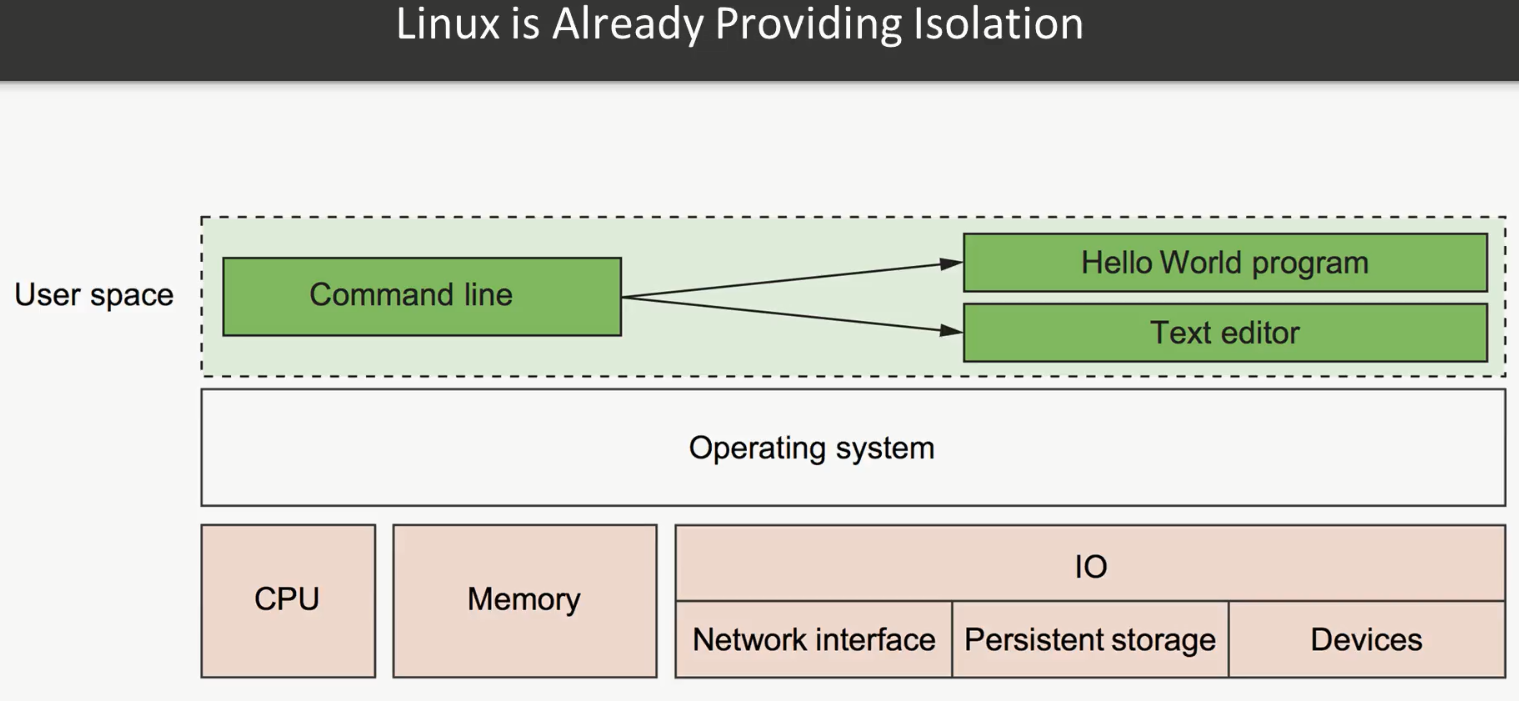
1. Docker registry - we can have a lot of images. for example image for whitefly, for MySQL database, Couchbase database, Node.js, and so on and so on.



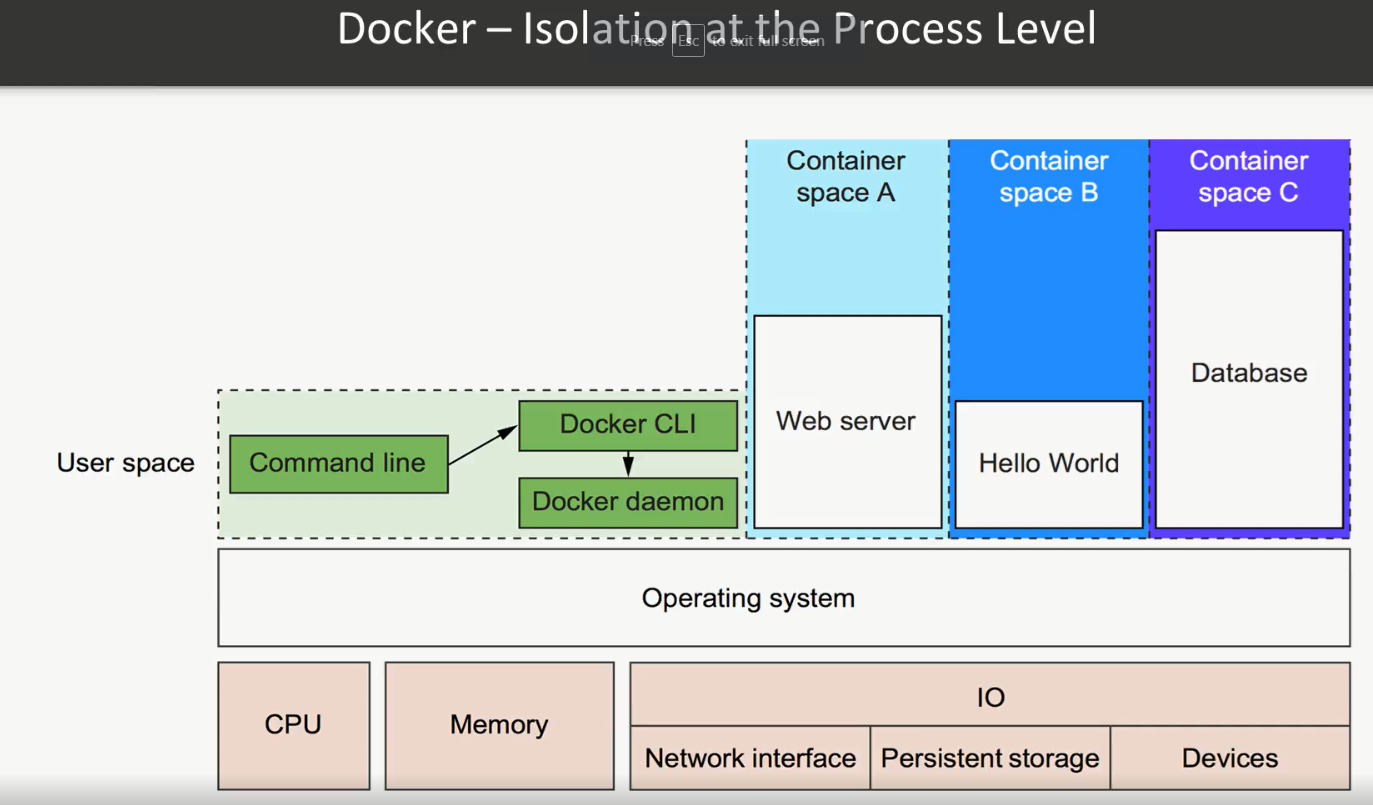
**Docker Engine is just basically fetching the image from a registry, and create container out of it.** We can create multiple containers using the same image, so that image could be shared between a lot of companies, a lot of people, and every one of them can create own container that will be running inside of the own machine.

**One of the most important features of Docker is isolation.**

In Linux there's already providing his isolation layer, because we have CPU memory IO and so on and so on, and then there is operating system. And operating system introduced some kind of isolation. Then when we are starting programs in the user space, we are not able to fetch data or interact with a kernel space, and this is a very important security feature. So let's say that we created a simple command-line application that starts HelloWorld program or text editor, then one HelloWorld program or text editor starts it is executed inside of the user space. That HelloWorld program cannot access kernel space with some system information like cryptographic keys or something like this, because if there will be a possibility to access that data then it could be stolen by some malicious program.



**Docker goes one level higher, because it provides an isolation at the process level :**



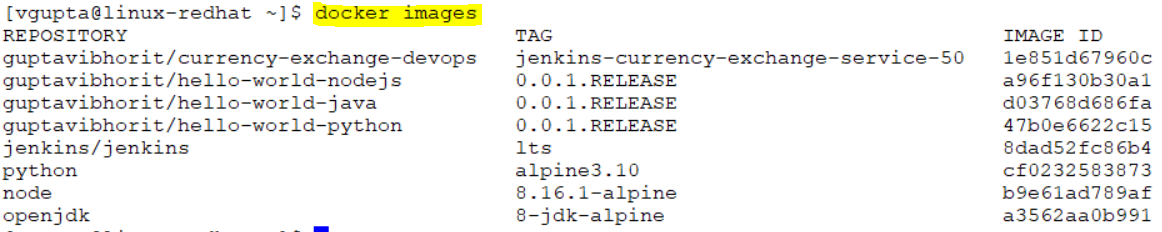
So we can see we have a running Docker means running to programs in userspace. For example two programs or more.

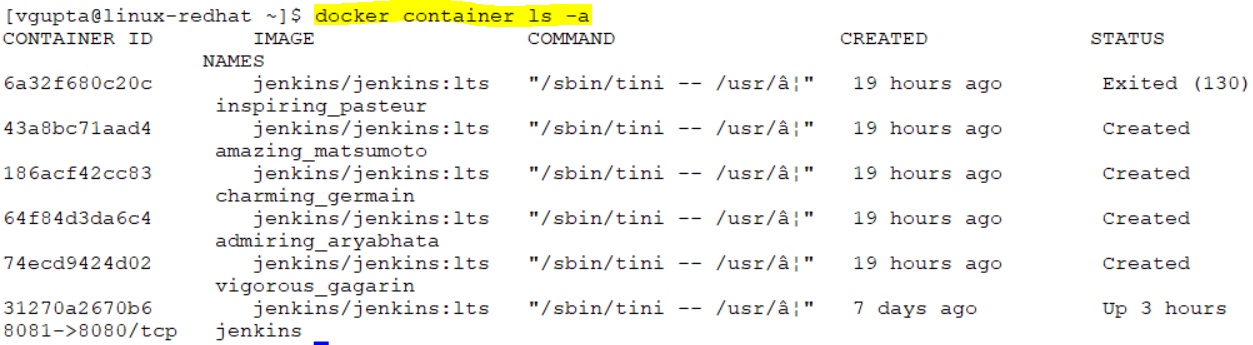
The first is a Docker daemon, as we can see. If it is installed properly, this process should always be running when you are using Docker.

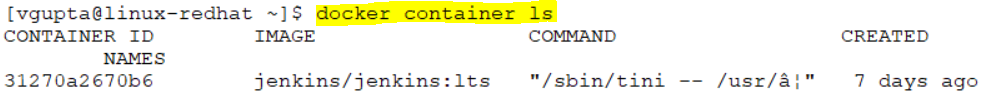
The second is the Docker CLI, this is a Docker program that user interacts with. If you want to start, stop, or install software, you use a command line the Docker program. This picture also shows free running containers. Each is running as a child process of the Docker daemon, wrapped with a container and the delegate process is running in its own memory subspace if the user space. Programs running inside a container can ask us only the own memory and resources, as code by the container. The containers that Docker builds are isolated we respect to eight aspects. Those aspects are PID namespace, UTS namespace that is providing host and domain name, M&T namespace is a file system and structure, IPC namespace is process communication, net namespace is network access, user name space is user names and identifiers, and also route that controls the Sol system route.

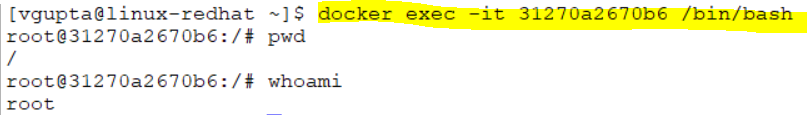
[vgupta@linux-redhat ~]$ docker --version

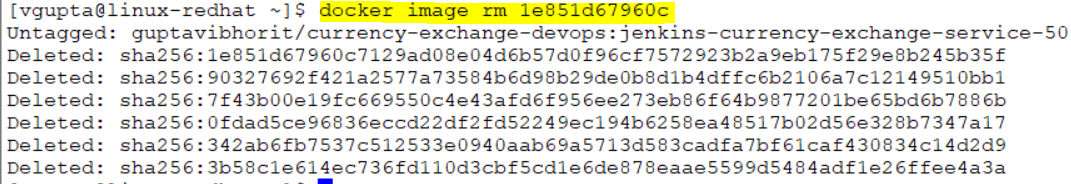
Docker version 19.03.12, build 48a66213fe











[vgupta@linux-redhat ~]$ docker container rm 3127

Error response from daemon: You cannot remove a running container 31270a2670b6ced6413d88afaed03d94616270e818c7025a7d607e863abb3ef4. Stop the container before attempting removal or force remove

[vgupta@linux-redhat ~]$ docker container stop 3127

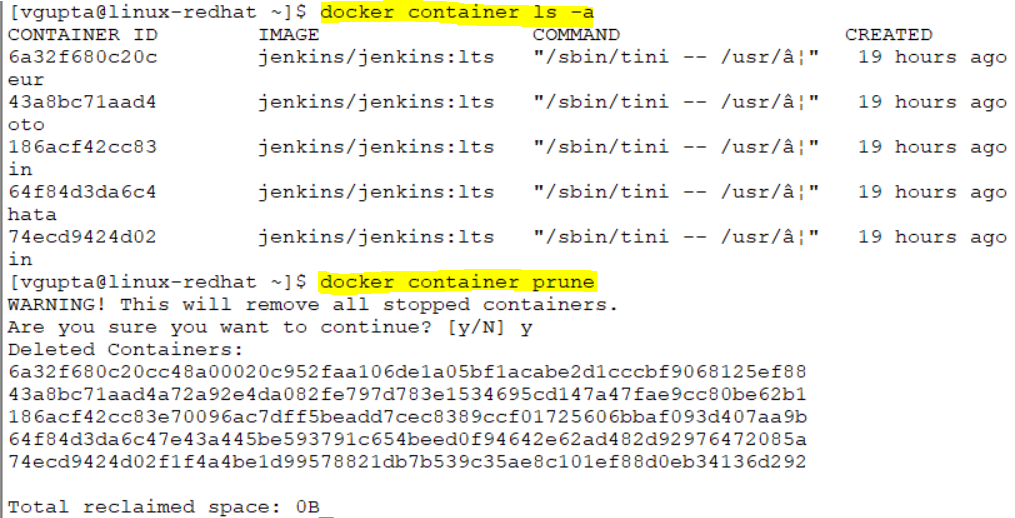
3127

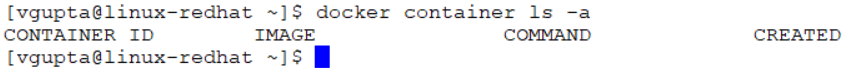
[vgupta@linux-redhat ~]$ docker container rm 3127

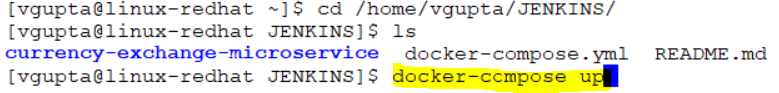
3127

[vgupta@linux-redhat ~]$ docker container ls

CONTAINER ID IMAGE COMMAND CREATED STATUS







docker run guptavibhorit/hello-world-java:0.0.1.RELEASE

docker run -p 5001:5000 guptavibhorit/hello-world-java:0.0.1.RELEASE

docker container start $CONTAINER\_ID

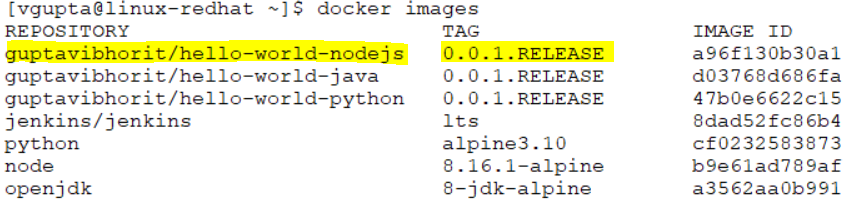
docker container stop $CONTAINER\_ID

docker container logs $CONTAINER\_ID

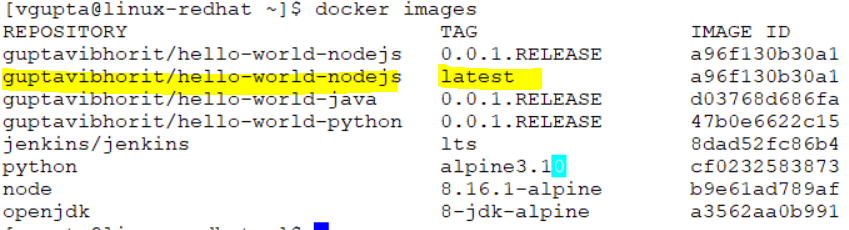
docker image history $IMAGE\_ID



Docker logs -f a5d26b80a445b4aad4777cf74e91b9cba3a59d72834711863e67ef419c0ef178



docker tag guptavibhorit/hello-world-nodejs:0.0.1.RELEASE guptavibhorit/hello-world-nodejs:latest



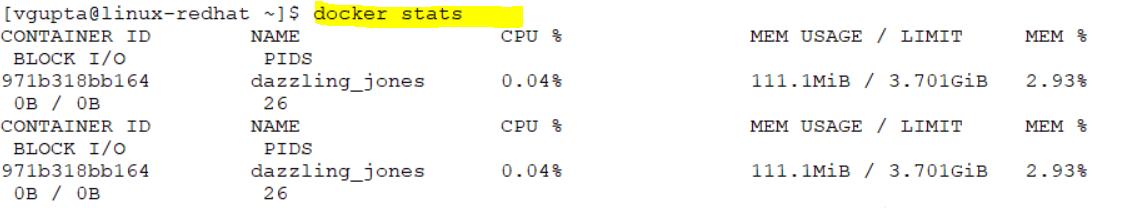
docker pull mysql

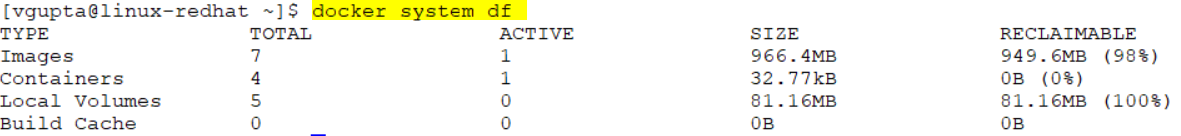
docker container pause $CONTAINER\_ID

docker container unpause $CONTAINER\_ID

docker container inspect $CONTAINER\_ID

docker stats





docker events

docker top c710

docker stats

docker run -m 512m –cpu-quota 50000

docker system df

docker run -dit openjdk:8-jdk-alpine

d- detached mode, it – interactive mode

docker container cp target/docker-in-5steps-todo-rest-api-h2-1.0.0.RELEASE.jar $CONTAINER\_ID:/tmp