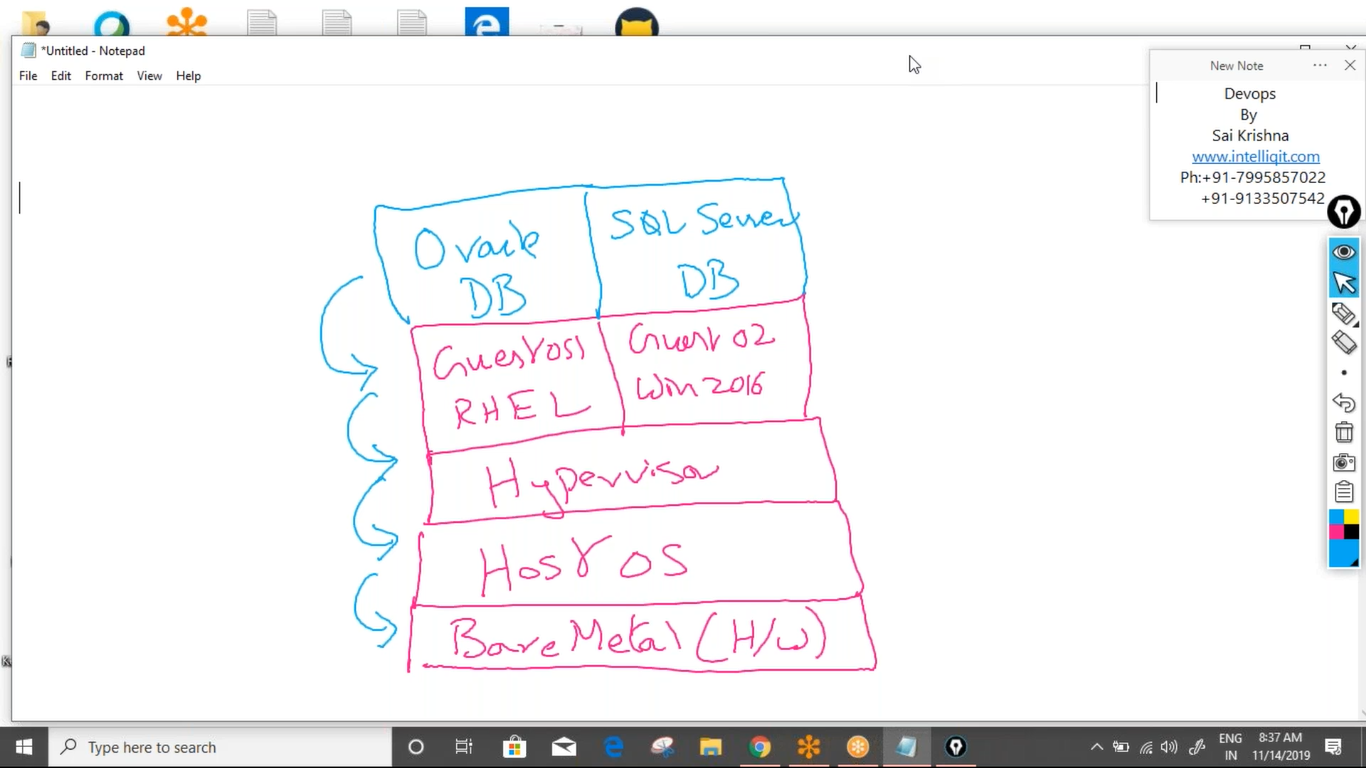
Docker:

Docker is a tool for implementing containers

Virtualization

Here we have a bare metal on top of it we install the host OS. On the host OS we can install the s/w called hypervisor (vm ware, es hex I, cytrix hen, Microsoft hyper v etc). on the hypervisor we can install guest operating systems and on the guest os we can install the application that we want. This feature enables us to run multiple os on one single server. The disadvantage is these applications have to pass through multiple layers in order to access the hardware resources

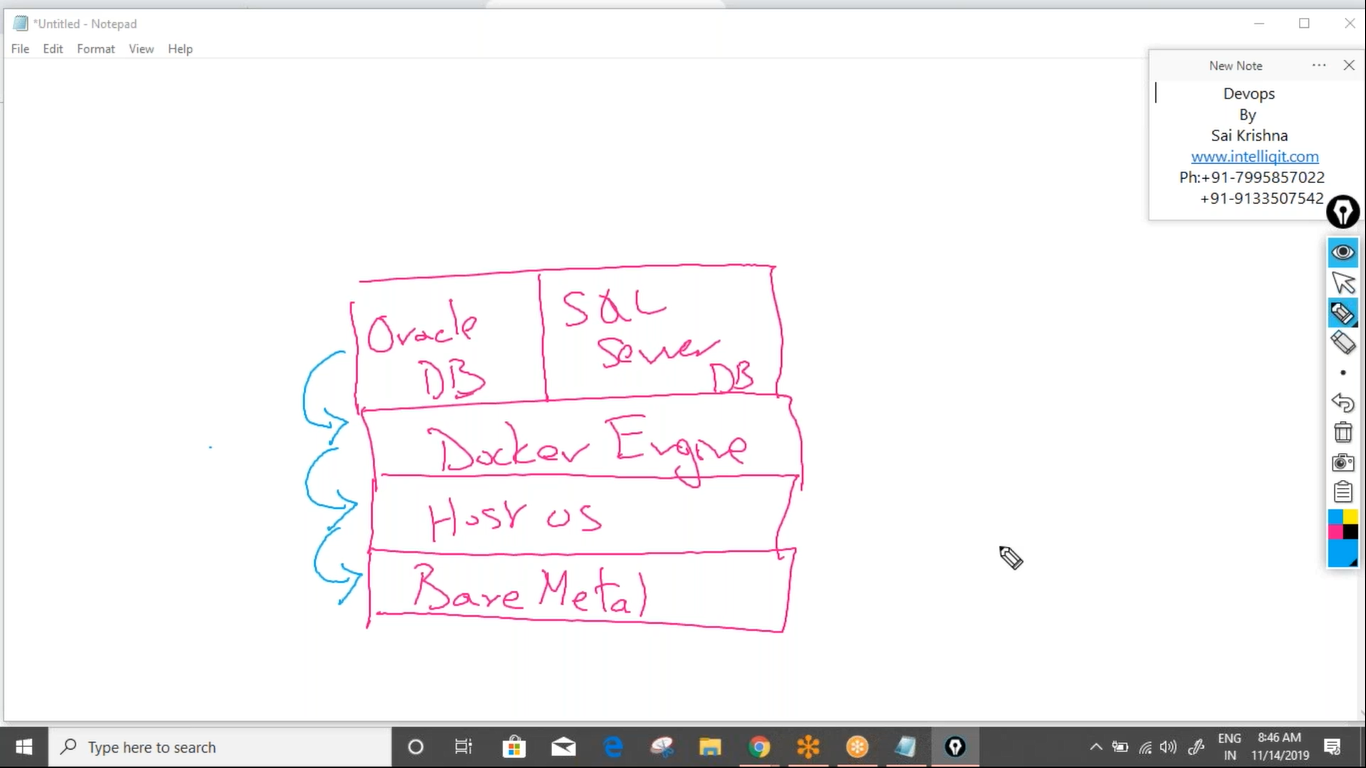


Here we have a bare metal on top which we install the host os on the host os we install the s/w called docker engine and on the docker engine we can install the application that we require these application pass through less number of layers in order to access the hardware resources. Docker performs process isolation that is it removes the dependencies that an application has on the underlining operating system and it allows these application to run directly on top of the docker engine

Docker can spin up the necessary environments dev testing prod etc in relatively very less amount of time and fixed hardware allocation is not done for docker due to this reason on the run time the containers can use required amount of hardware resources based on their usage

Docker can be used on at all the three stages of build ship and run, that is dev testing and prod

Docker comes in two flavors community edition and enterprise edition



Docker install:

1. Open get.docker.com-----copy below two command on terminal

Curl -fsSL <https://get.docker.com> -o get-docker.sh

Sh get-docker.sh

Docker images:

An image is a combination of binaries and libraries which are necessary for a specific s/w application

Containers:

An running instance of an image is called as a container, any number of containers can be created from one image

Docker hosts:

The operating system on which docker is installed is called as a docker host

Docker client:

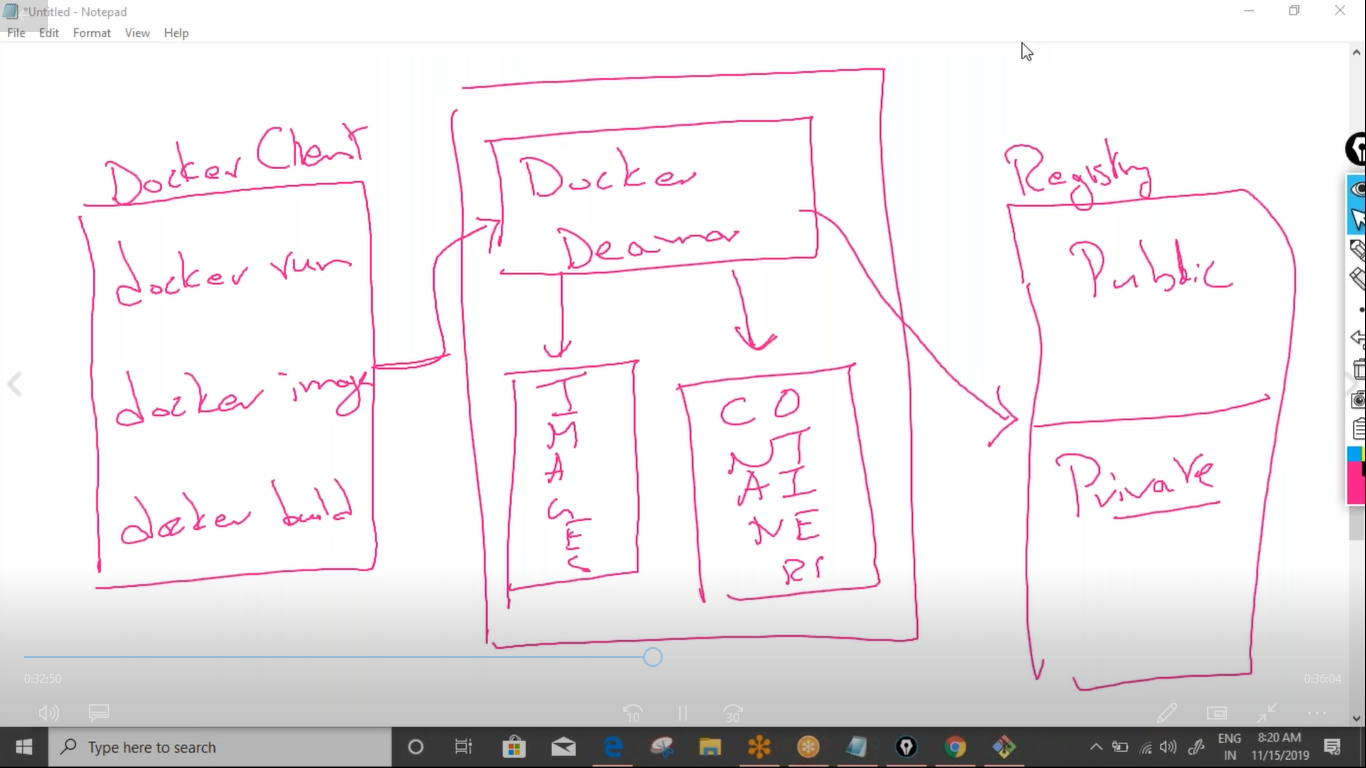
This is a background s/w which accepts the docker commands and passes them to another background process called as a docker daemon

Docker daemon :

This process accepts the commands coming from the docker client and routs them to either docker images or containers or the docker registry

Docker registry:

This is repository where we can store docker images, this is of two types public and private. Public registry is hub.docker.com and this is maintained by a docker corporation. Private registry is created within our own server and only our team members can be able to access it



Important docker commands:

1. To download docker image

Docker pull image\_name

1. To upload docker image

Docker push image\_name

1. To see the list of images that are present in our docker host

Docker images or docker images ls

1. To delete a docker image

Docker rmi image\_id/image\_name

1. To create a docker image from container

Docker commit container\_id/container\_name new\_image\_name

1. To create a docker image from a docker file

Docker build –t new\_image\_name .

Note: . represent the current working directory

1. To delete all docker unused images

Docker system prune –a

1. To save a docker images as a tar or archived file

Docker save image1 image 2…

1. To get detailed info about the docker image

Docker image inspect image\_name/image\_id

Working in containers:

1. To see the list of running containers

Docker containers ls

1. To see the list of containers (running and stopped)

Docker ps –a

1. To start a stopped containers

Docker start container\_name/container\_id

1. To stop a runnig containers

Docker stop container\_name/container\_id

1. To restart a containers

Docker restart container\_name/container\_id

To restart after 20 seconds

Docker restart –t 20 container\_name/container\_id

1. To delete a stopped container

Docker rm container\_name/container\_id

1. To delete a running container

Docker rm -f container\_na a`me/container\_id

1. To stop all running containers

Docker stop $(docker ps – aq)

1. To delete all stopped containers

Docker rm $(docker ps – aq)

1. To delete all containers (running and stopped)

Docker rm -f $(docker ps – aq)

1. To get detailed logs of a containers

Docker logs container\_id

1. To see complete info about a container

Docker inspect container\_id

1. To see the ports used by the container

Docker port containet\_id

1. To come out of a container without exit

Cntl+p, ctrl+q

1. To go back into the same container

Docker attach container\_id

1. To create a container

Docker run inmage\_name

Run command options:

--name gives a name to the container

-d run the container in detached mode in background( as a daemon)

-it used for opening interactive terminal in the container

--link used to liking multiple containers to create a micro service architecture

--network used for running container on a specific network

-e used for passing environment variables to a container

-v used for attaching volumes to the container

--volumes-from used for sharing volumes b/w containers

-p used for port mapping it will map the container port with the docker host port so that it can used for external communication

Eg: -p 8080:80 here 80 is the container and is called as internal port and 8080 is docker host port and is called as external port

-P used for automatic port mapping. The internal port of the container will be automatically mapped with the some port on a docker host which is grater than 300000

-m used for specifying the upper limit on the memory that the container can use

-c used to allocate specific number of cpu’s to a container

-h used for specifying host name for the container

-rm to delete container on exit

Working on docker networks:

1. To see the list of docker networsk

+Docker network ls

1. To create a new docker network

Docker network create –driver driver\_name network\_name

1. To get detailed info about a network

Docker network inspect netwok\_name/network\_id

1. To delete a network

Docker network rm netwok\_name/network\_id

1. To attach a running container to a network

Docker network connect netwok\_name/network\_id containet\_name/container\_id

1. To disconnect a container from a network

Docker network disconnect netwok\_name/network\_id containet\_name/container\_id

Working on docker volumes:

1. To see the list of docker volumes

Docker volume ls

1. To create a volume

Docker volume create volume\_name

1. To get a detained info about a docker volume

Docker volume inspect volume\_name/volume\_id

1. To delete a volume

Docker volume rm volume \_name/volume\_id

1. Start tomcat as a container in a detach mode. Map the tomcat port 8080 with the host port 5050

Docker run --name webserver -p 5050:8080 -d tomcat

1. Start jenkins as a container name it devserver. Map 8080 port of Jenkins container with 6069 of host

Docker run --name webserver -p 6060:8080 -d Jenkins

1. Start nginx as a container name it appserver. Give it default port

Docker run --name webserver -P -d nginx

To identify port of nginx

Docker port appserver

1. Start Ubuntu as a container and open interactive terminal init

Docker run --name u1 -it Ubuntu

To come out of type exit

1. Start mysql as a container and login into the container create tables

Docker run --name mydb -d -e MYSQL\_ROOT\_PASSWORD=srik mysql:5

1. Open interactive terminal

Docker exec -it mysql bash

1. To login into the database as root

Mysql –u root –p enter password:srik

1. To move into databse: use sys
2. create emp and dept tables

Creating micro services architecture:

Multiple containers can be likened with each other to create multi container architecture. This

Can be done in the fallowing ways

1. using --link option
2. docker compose
3. docker networking
4. using python scripts

Using –link option

This is a run command option which is used to creating link b/w multiple containers and this option is deprecated

1. start a busybox container

docker run --name b1 -it busybox

1. come out of terminal without exit

ctrl+p.cnrl+q

1. start another busybox and link with b1

docker run --name b2 –it --link b1:b1-alias busybox

1. In the b2 container ping b1

ping b1

Create a dev environment where a mysql container is linked with wordpress container

1. Start mysql as a container

Docker run --name mydb -d -e MYSQL\_ROOT\_PASSWORD=srik mysql:5

1. Start wordpress and link with mysql

Docker run --name myword -d -p 8888:80 –link mydb:mysql wordpress

CI/CD

1. Start Jenkins

Docker run --name devserver -d -p 5050:8080 jenkins

1. Start tomcat

Docker run --name QAserver -d -p 6060:8080 --link devserver:jenkins tomcat:8

1. Start prod server tomcat8

Docker run --name prodserver -d -p 7070:8080 --link devserver:jenkins tomcat:8

Create LAMP architecture where the operating system is linux databse is mysql application server is apache and programing language is php

1. Start my sql

Docker run --name mydb -d -e MYSQL\_ROOT\_PASSWORD=srik mysql:5

1. Start apche container and link with mysql

Docker run --name apache -d -p 9988:80 --link mydb:mysql httpd

1. Start php containet and link with apache and mysql containers

Docker run --name php -d --link mydb:mysql –link apache:httpd php:7.2-apche

Create master slave setup of Jenkins suing docker containers

1. Start jenkins

Docker run --name master -d -P jenkins

1. Strt Ubuntu as slave

Docker run --name slave -it --link master:jenkins ununtu

1. In Ubuntu container

Apt-get update

Apt-get install -y wget

Download slave.jar file

Wget master:8080/jnlpJars.slave.jar

Create a testing environment where a selenium hub container is linked with two node containers one with firefox and other with chrome install.net

1. Start selenium hub as a container

Docker run --name hub -d -p 4444:4444 selenium/hub

1. Start chrome container and link with hub

Docker run --name chrome -d -p 5902:5900 --link hub:selenium selenium/node-chrome-debug

1. Start firefox and link with hib

Docker run --name firefox -d -p 5901:5900 --link hub:selenium selenium/node-firefox-debug

1. The above two are GUI containers see by using VNC viewer
2. Download and inatll VNC
3. Open VNC viewer
4. Ipadress:port
5. Click continue enter password------password secret

Docker compose:

The disadvantage of --link option is we have to give multiple commands for creating a multi container architecture. This becomes challenging when we want to link multiple containers frequently and also delete them once the activity is finished. We can handle this in much more efficient way using docker compose. Docker compose used yaml file where we specify info about the different docker containers that we want to start and they should be linked with each other. The main advantage of docker compose is reusability

Installing docker compose:

Sudo curl –L [https://github.com/docker/compose/releases/download/1.24.1/docker-compose-$(uname -s)-$(uname -m)](https://github.com/docker/compose/releases/download/1.24.1/docker-compose-$(uname%20-s)-$(uname%20-m)) –o /user/local/bin/docker-compose

Sudo chmod +x /user/local/bin/docker-compose

<https://docs.docker.com/compose/insatll/>

Create a docker compose file for linking mysql container with a wordpress container

The name of the file should be docker-compose.yaml

---

vesrion: '3'

services:

  mydb:

    image: mysql:5

    environment:

      MYSQL\_ROOT\_PASSWORD: srik

  mywordpress:

    image: wordpress

    ports:

      - 8888:80

    links:

      - mydb:mysql

To create this architecture command id

Docke- compose up –d

And to remove this is docker-compose stop

Create a docker compose file for setting up the ci/cd environment where a jenkins container should be linked with two tomcat containers

---

vesrion: '3'

services:

  devserver:

    image: jenkins

    ports:

      - 6060:8080

  qaserver:

    image: tomcat

    ports:

      - 7070:8080

    links:

      - devserver:jenkins

  prodserver:

    image: tomcat

    ports:

      - 8080:8080

    links:

      - devserver:jenkins

Creating docker compose for LAMP architecture

---

vesrion: '3'

services:

  mydb:

    image: mysql:5

    environment:

      MYSQL\_ROOT\_PASSWORD: srik

  apache:

    image: httpd

    ports:

      - 7070:80

    links:

      - mydb:mysql

  php:

    image: php:7.2-apache

    links:

      - mydb:mysql

      - apache:httpd

Docker-compose –f abc.yaml up –d

Testing environment:  
---

vesrion: '3'

services:

  hub:

    image: selenium/hub

    ports:

      - 5555:4444

  firefox:

    image: selenium/node-firefox-debug

    ports:

      - 5901:5900

    links:

      - hub:selenium

  chrome:

    image: selenium/node-chrome-debug

    ports:

      - 5902:5900

    links:

      - hub:selenium

Docker volumes:

Docker containers are ephemeral but the data it process should be permanent. In such cases we use docker volumes. A docker volume is external device or the directory that is mounted on a container in such a way that even after a container is deleted the volume data will remain on host machine. Docker supports two types of volumes

1. Simple docket volumes
2. Docker volume containers

Simple docker volume:

These volumes are used when we want to preserve the data even after the container is deleted

1. Create directory/data
2. Start Ubuntu as a container And mount /data on it

Docker run --name u1 -it -v /data Ubuntu

1. Delete the container and check if the data is still present on the / machine

Cd data

Touch f1 f2

Exit

1. Identify the location where data is stored

Docker inspect u1

Goto mount section and copy the “source” path

1. Dele container

Docker rm –f u1

1. Check if data is present

Cd “source\_path”

Docker volume containers:

These are sharable docker volumes which can be shared b/w multiple containers

1. Create 3 ubuntu containers c1 c2 c3 on c1 mount volume /data
2. C2 should use the volume used by c1 c3 container should use the volume by c2
3. Create directory/data
4. Start Ubuntu as a container And mount /data on it

Docker run --name c1 -it -v /data Ubuntu

1. Cd data-move into data

Touch f1 f2

Come out without exit cntl+p, cntl+q

1. Create another ubuntu c2

Docker run --name c2 -it --volumes-from c1 ubuntu

1. Cd data-move into data

Touch f1 f2

Come out without exit cntl+p, cntl+q

1. Create another ubuntu c3

Docker run --name c3 -it --volumes-from c2 ubuntu

1. Cd data-move into data

Touch f1 f2

Come out without exit cntl+p, cntl+q

1. Identify the location where data is stored

Docker inspect c1

Goto mount section and copy the “source” path

1. Dele container

Docker rm –f c1 c2 c3

1. Check if data is present

Cd “source\_path”

Creating docker image:  
---------------------------------------------------------------------------------------------------

Customized docker images can be created in two ways

1. Docker commit
2. Dockerfile

Using docker commit command:

Create an Ubuntu container and install git and maven init and save it as snapshot(image) and dele the container

1. Create Ubuntu

Docker run --name c1 -it Ubuntu

1. Apt-get update

Apt get install –y git maven

Exit

1. Save container as image

Docker commit c1 itelliq\_ubuntu

1. Delete the Ubuntu container

Docker rm –f c1

1. Create new container from the above image

Docker run --name c1 -it itelliq\_ubuntu

Creating docker images using Dockerfile:

Docker file is a text based file which used certain predefined key word for creating docker images

Important key word in Dockerfile

FROM----this is used to specify the base image from which customized docker images can be created

MAINTAINER ----this represents the name of the organization or the author who created this dockerfile

CMD-----this is used for executing a command or process in a container even though the control is outside

RUN----this is generally used for package management within a container like updates the repositories installing s/w etc

USER---- this is used to specify the default user who should login into the container

VOLUME----this is used to mount the default volume on a container

EXPOSE----this is used to open a port within a container

COPY----used for copying files from host to container

ADD----this is also used for copying files from host to container it can also be used for downloading files from remote server

WORKDIR----used to specify the default working directory that should be used once the container starts

ENV----used for specifying which varibels should be passed as env variables

ARG----used to specify which version of docker image should be created

ENTRYPOINT-----every container initiates a process called as default process as long as this default process is running the container will be in running status. This is used to specify that default process

LABEL-----this is used to store meta data about the docker image. This meta data is stored in the format of key value pairs

Create a dockerfile from nginx base image and specify the maintainer as intelliq

1. Vim dockerfile

FROM nginx

MAINTAINER intelliq

1. Create image from above docker file

Docker build -t mynginx .

Create a docker file from centos base image and when we start a container from this image it should execute ls –la o/p of the container

1. Vim dockerfile

FROM centos

MAINTAINER intelliq

CMD [“data’]

CMD [“ls”, “-la”]

1. Create a image from above file

Docker build –t mycent

1. Start containers from above image

Docker run –name c1 mycent

Note: only one CMD per docker file is allowed. If we have more than one last one will be executed

Create a dockerfile from Ubuntu base image and install git init, start a container from above image

1. Vim dockerfile

FROM centos

MAINTAINER intelliq

RUN apt-get update

RUN apt-get install –y git

1. Create an image from above dicker file

Docker build –t myubuntu

1. Create container

Note: When we save in dockerfile concept the memory it consumes is very less rather than in image format. So we create a dockerfile and send it to git repo. Whoever wants can download and create image

Cache busting:

Whenever we create an image from a dockerfile docker stores those instruction in the docker cahe. Next time when we edit the same dockerfile it will execute only the new instructions and the older instruction it will read from the cache. This is the tie saving mechanism of docker. The disadvantage of this process is if the docker file is edited after a huge time gap we might end up installing s/w from a repo which is updated long time back

FROM Ubuntu

RUN apt-get update

RUN apt-get install –y git

If we create image from the above docket file and later we add one more statement

RUN aot-get install –y default-jdk

It will execute only the latest statement this might result in installing java from a repo which is updated log time back. To overcome this problem we can use cache busting by using && symbol

FROM Ubuntu

MAINTAINER inte

RUN apt-get update && apt-get install –y git tree

Ansible image:

----------------------------------------------------------------------------------------------------

Create a docker file from Ubuntu base mage and install ansible in it

FROM Ubuntu

MAINTAINER inte

RUN apt-get update

RUN apt-get install –y software-properties-common

RUN apt-add-repository ppa:ansible/ansible

RUN apt-get update

RUN apt-get install –y ansible

Docker build –t ansible .

Docker run –it ansible

Docker file from centos base image and mount data as default volume on it

1. Vim dockerfile

FROM centos

MAINATINER inte

VOLUME /data

1. Create an image from above

Docker build –t centos .

1. Create container from above image

Same process in volume fallow that

Create docker file from ubuntu base image and download jenkins.war into it

1. Vim dockerfile

FROM Ubuntu

MAINATINER inte

ADD [http://mirrors.jenkins.io/war-stable/latest/jenkins.war /](http://mirrors.jenkins.io/war-stable/latest/jenkins.war%20/)

1. Create an image from above

Docker build –t myubuntu .

Every docker container triggers a default process and this we can see in th o/p of docker container ls in the command section as long as this process runs the container will be in running status . if the default process stops the containers goes into exited status

Create docker file from Ubuntu base image and make java –jar jenkins.war as the default process of the container

1. Vim docker file

FROM unbuntu

MAINTAINER intelliq

ADD <http://mirrors.jenkins.io/war-stable/latest/jenkins.war> /

RUN apt-get update

RUN apt-get install –y openjdk-8-jdk

ENTRYPOINT [“java”, “-jar”, “jenkins.war”]

Create a docker file from jenkins base image and make the default user as root. Install git and maven in it

1. Vim docker file

FROM jenkins

MAINTAINER intelliq

USER root

RUN apt-get update

RUN apt-get install –y openjdk-8-jdk

1. Create image and start containers

Docker run –name j1 –d –P myjenkins

1. Start interatctive terminal

Docker exec –it j1 bash

Git –version

Docker networks:

Docker used 4 types of networks

1. Bridge: Default network docker uses the containers running on a single host
2. Host only: This is used when we want to create containers which can communicate only with the host machine and not with the other containers. This is generally used in scenarios where we want to run a single docker container on a host
3. Null network: This is used for creating isolated containers which cannot communicate with the host or with other containers. This is generally used in docker security
4. Overlay network: Also called as ingress network. This is used when docker containers are running in distubuted environment on multiple servers and they want to communicate with each other
5. Create two bridge networks in1 and in2

Docker network create --driver bridge in1

Docker network create --driver bridge in2

1. Create three busybox containers c1 c2 c3

Docker run --name c1 -it --network in1 busybox

Come out without exit that is cntl+p, cntl+q

Identify ipaddress docker inspect c1

Docker run --name c2 -it --network in1 busybox

Ping ipaddress of c1

Come out without exit

1. C1 and c2 should run on in1 network and they should ping each other

C3 should run on in2 network later attach c2 to in2 network also because c2 present on both the network it should ping to both c1 and c3

Docker run --name c2 -it --network in1 busybox

Ping c1 and c2 and come out without exit

1. Now attach in2 to c2

Docker network connect in2 c2

1. Docker attach c2

Now ping to c1 and c3

Container orchestration:

This is the process of running docker containers in a distributed environment on multiple servers

Advantages:

1. Load balancing: We should deploy multiple containers all of them running on one single service. These containers are also called as replicas and they behave like clones
2. Scaling: Depending on business requirement we should be able increase or decrease the replica count without experiencing any downtime
3. Rolling updates: The version of the docker image on which the containers are running should be upgraded or downgraded without any downtime
4. Handling failover scenarios: Incase a container crashes or the server on which these containers are running crashes still the required count of containers should always maintained using this we can handle disaster recovery and also promise high availability

Popular container orchestration tolls:

1. Docker swarm
2. Kubernetes
3. Apache mesos
4. Redhat openshift

Docker swarm setup:  
----------------------------------------------------

1. Launch 3 AWS and install docker on all of them
2. Change the host name

Vim /etc/hostname make them manager, worker1 and worker2

1. Restart the aws instances

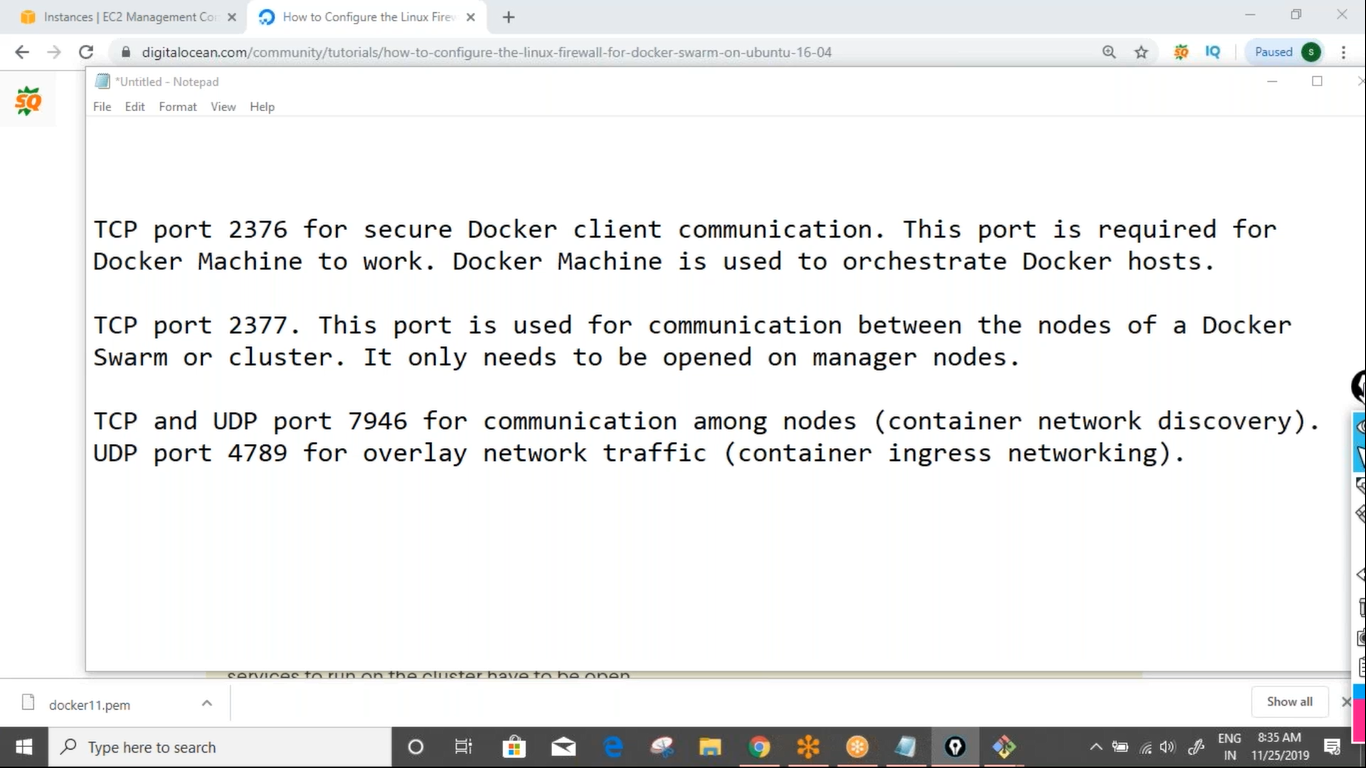
Init 6

1. Connect to manager
2. Create swarm

Docker swarm init --advertise-addr private\_ip\_of\_manager

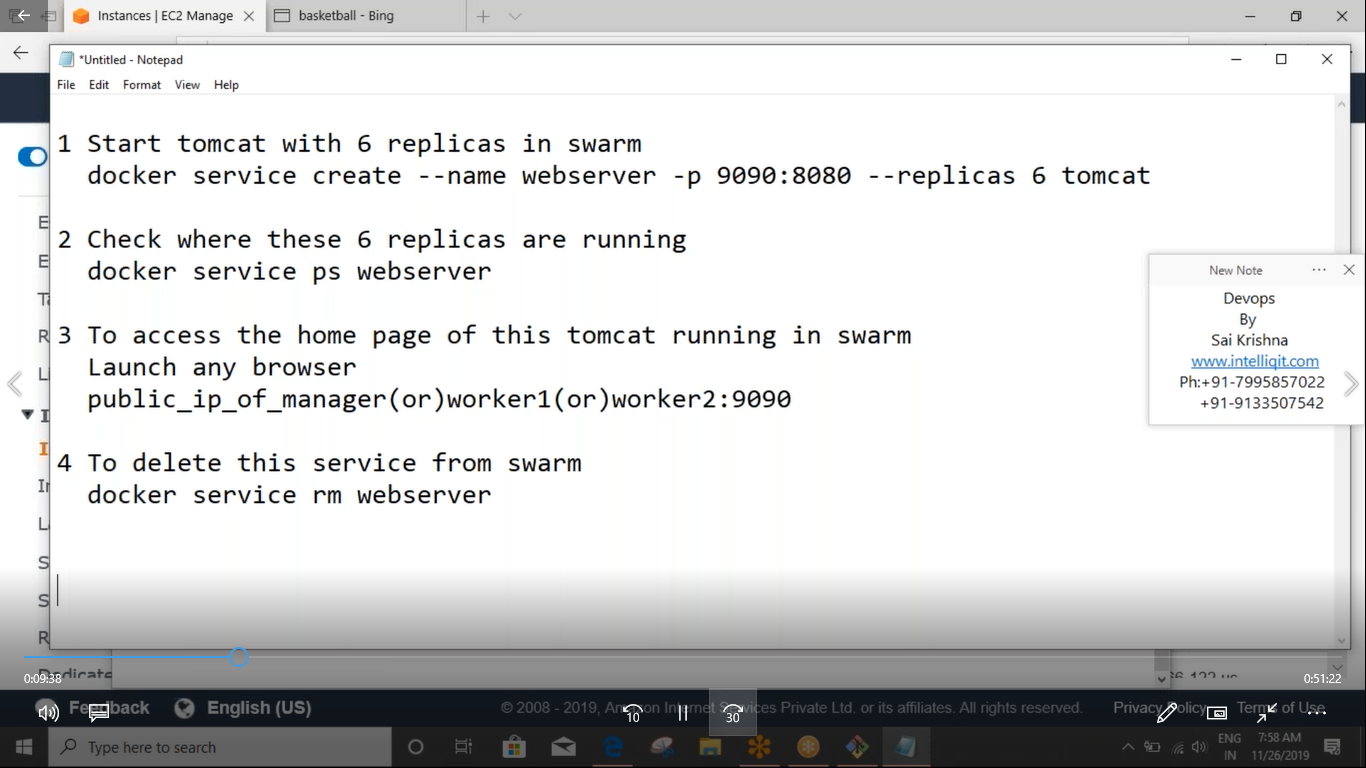
The command will initialize the current node as manager and it will generate command that we need to paste in the worker nodes

Ports for docker swarm



Load balancing:  
----------------------------------------------------------------------------------------------------------------------------------------

Start tomcat with 6 repliacs and check if these replicas are running on manager and the workers



Start my sql with 3 replicas and check where these repliacs are running

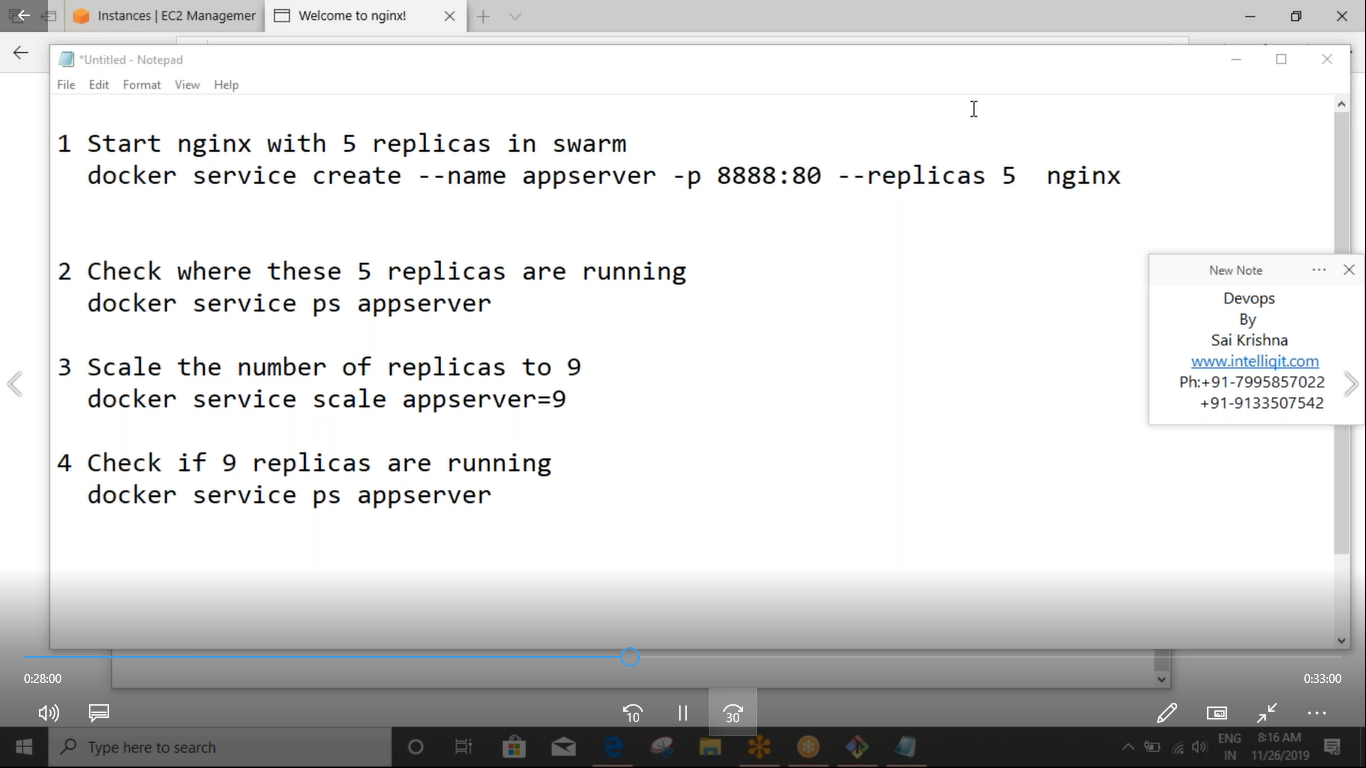
Docker service create --name mydb -e MYSQL\_ROOT\_PASSWORD=srik --replicas 3 mysql:5

Docker service ps mydb

Scaling:  
----------------------------------------------------------------------------------------------------------------------------------

Based on business requirements we should be able to either increase the number of repliacs or decrease the number of replicas without experiencing any downtime

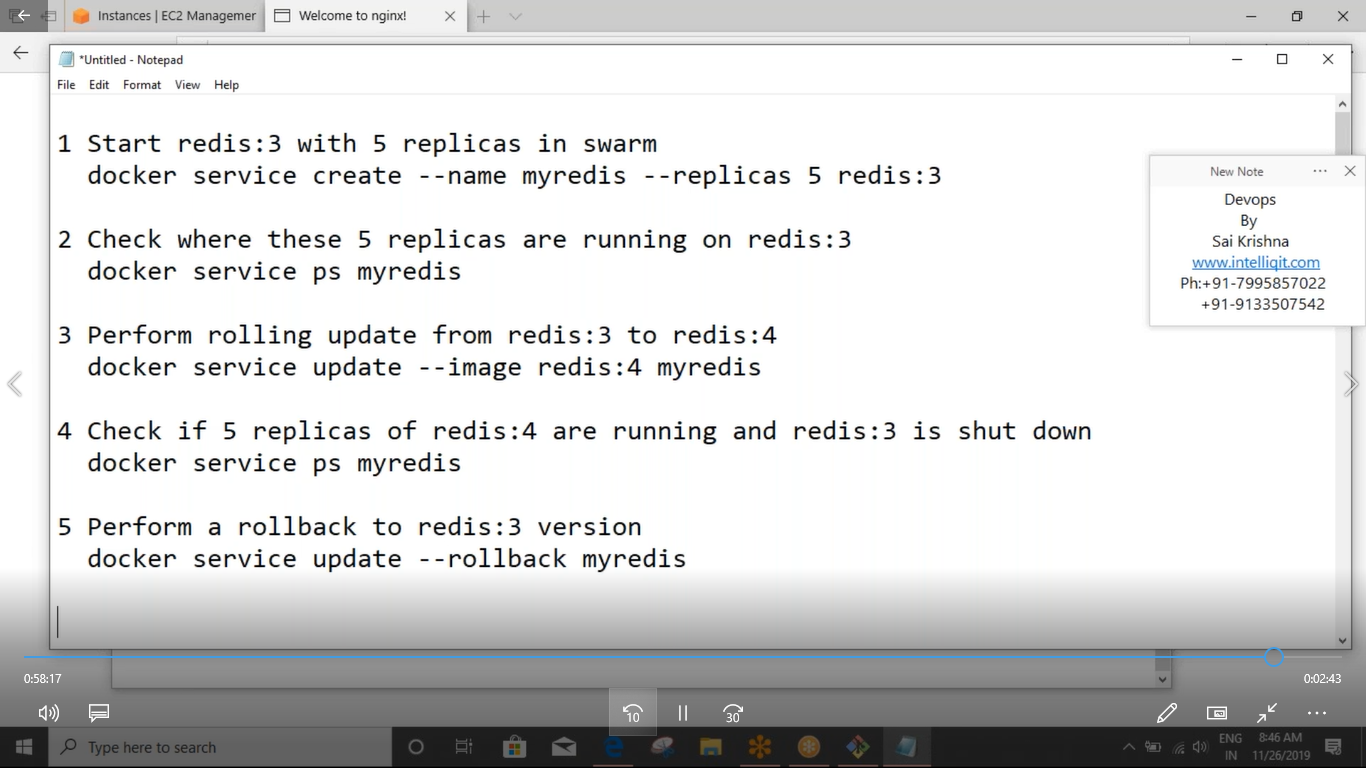
1. Start nginx with 5 replicas and scale it to 9



Rolling update:

We should be able to update the services running in docker swarm from one version to another version without experiencing downtime this is done by docker by effecting one replica after another. This is called as rolling update

Start redis 3 with 5 replicas in docker swarm



Overlay network

Docker swarm by default uses overlay network this is also called as ingress network

1. Create two overlay networks in1 and in2

Docker network create --driver overlay in1

Docker network create --driver overlay in2

1. To see the list of networks

Docker network ls

1. Start tomcat with 3 replicas

Docker service create --name webserver -p 8888:8080 --replicas 3 --network in1 tomcat

1. Check tomcat running on in1

Docker service inspect webserver --pretty

1. Start httpd with 5 repliacs in default ingress network

Docker service create --name appserver -p 9999:80 --replicas 5 httpd

1. Perform rolling update on httpd change to in2 netwok

Docker service update --network-add in2 appserver

1. Check where httpd is running

Docker service inspect appserver --pretty

Docker stack:

Docker compose when implemented in docker swarm is called as docker stack. A docker stack is capable of deploying a complete architecture where multiple containers are linked with eachother in such a way that all these containers can be scaled and orchestrated to better

1. To deploy a stack from a compose file

Docker stack deploy -c docker-compose-filename stack\_name

1. To see list of stacks

Docker stack ls

1. To see the nodes where these stack services are running

Docker stack ps stack\_name

1. To del a stack

Docker stack rm stack\_name

Create a docker stack file for linking three repliacs of a word press with one replica of mysql

---

version: '3'

services:

  mydb:

    image: mysql:5

    environment:

      MYSQL\_ROOT\_PASSWORD

  mywordpress:

    image: wordpress

    ports:

      - 8989:80

    deploy:

      replicas: 3

1. To deploy these sevices in swarm

docker stack deploy -c stack1.yaml wordpress

1. To check were these services are running

Docker statckps wordpress

1. To del this stack from swram

Dcker stack rm wordpress

Note: Docker-compose + docker-swarm = docker stack

Docker-compose+ kubernetes = kompose

To setup the ci-cd env where 1 replica of jenkins should be linked with tomcat as qaserver and prodserver. Jenkins should run on manager qa on worker 1 and prod run on worker2

---

version: '3'

services:

  devserver:

    image: jenkins

    ports:

      - 5050:8080

    deploy:

    placement:

      constraints:

        - node.hostname == Manager

  qaserver:

    image: tomcat

    ports:

      - 6060:8080

    deploy:

      replicas: 2

      placement:

      constraints:

        - node.hostname == worker1

  prodserver:

    image: tomcat

    ports:

      - 7070:8080

    deploy:

    replicas: 3

    placement:

      constraints:

        - node.hostname == worker2

Create a docker stack file for setting up testing env

---

version: '3'

services:

  hub:

    image: selenium/hub

    ports:

      - 4444:4444

    deploy:

      resources:

        limits:

          cpus: "0.1"

          memory: 250M

  chrome:

    image: selenium/node-chrome-debug

    ports:

      - 5901:5900

    deploy:

      resources:

        limits:

          cpus: "0.01"

          memory: 100M

  firefox:

    image: selenium/node-firefox-debug

    ports:

      - 5902:5900

    deploy:

      resources:

        limits:

          cpu: "0.01"

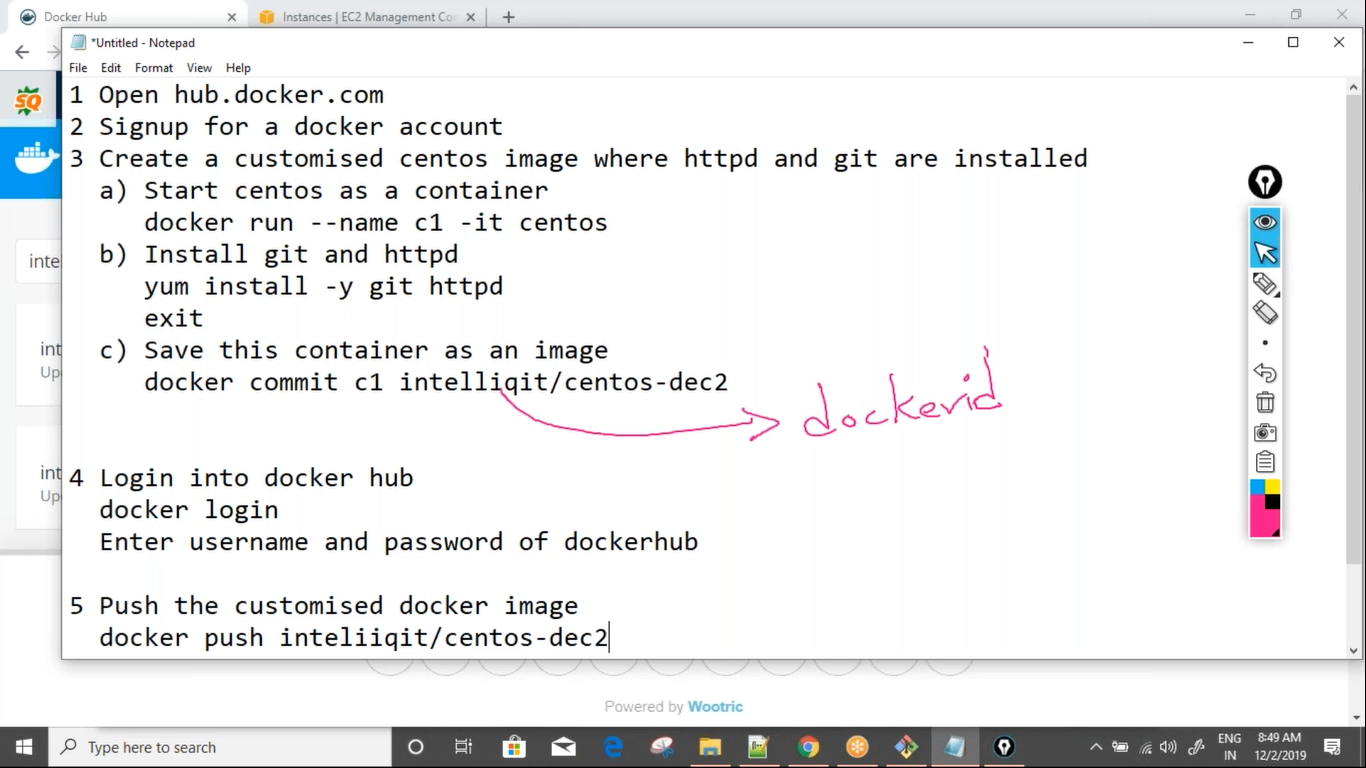
          memory: 100M

Working on docker registry

Registry is a image where docker images are saved. These registries are two types

1. Private registry
2. Public registry

Public registry:



Private registry:

Local registry can be created by using docker image called registry. If we start this registry as a container it starts behaving like docker local registry into which we can push images

Create a local registry and push alphine into it

1. Docker run --name lr –p 5000:5000 –d registry
2. Docker pull alpine
3. Tag the alpine image to local regitry

Docker tag alpine localhost:5000/alphine

1. Push the alpine into local registry

Docker push loaclhost:5000/alpine

To handle failover scenarios effectively we should maintain multiple managers. Once a specific count of managers is created swarm always expects the (total number of managers)/ 2 +1 as the minimum manager count. This sis maintained using raft algorithm. Managers have the status as leader or reachable. If the leader machine shuts down another machine which has reachable stated will be elected as a leader. This is internally done by quorum.