Day 1
==========
Virtualization
==========
This is the process of running multiple OS's parallelly on
a single pice of h/w.
Here we have h/w(bare metal) on top of which we have host os
and on the host os we install an application called as hypervisor
On the hypervisor we can run any no of OS's as guest OS

The disadvantage of this approach is these application running on the
guest OS have to pass through n number of lavers to access the H/W
resources.

Containarization
==========
Here we have bare metal on top of whcih we install the host Os
and on the hsot OS we install an application called as Docker Engine
On the docker engine we can run any application in the form of containers
Docker is a technology for creating thse containers

Docker achive what is commonly called as "process isolation"
ie all the applications(processes) have some dependency on a specific
OS.This dependency is removed by docker and we can run them on any
OS as containers if we have Docker engine installed

These containers pass through less no of layers to access the h/w
resources
also organizations need not spend money on purchasing licenses of
different
OS's to maintian various applications

Docker can be used at the the stages of S/W development life cycle
Build---->Ship--->Run

==========
==
Docker comes in 2 flavours
Docker CE (Community Edition)
Docker EE (Enterprise Edition)

Setup of Docker on Windows
==========
1 Download docker desktop from
  https://www.docker.com/products/docker-desktop

2 Install it

3 Once docker is installed we can use Power shell
  to run the docker commands

Day 2
==========
======

Create an ubuntu linux machine using vagrant
==========

1 Download oracle virtual box from
  https://www.virtualbox.org/wiki/Downloads

2 Install it

3 Download and install vagrant
  https://www.vagrantup.com/downloads

4 Download the vagrant file and copy it into an empty folder

5 Open cmd prompt

6 Change directory to the folder where the vagrantfile is copied
  cd path_of_folder

7 vagrant up

8 USername and password is:vagrant

==========

Using AWS
==========
1 Login in AWS account

2 Create ane new Ubuntu 20 instance

3 To connect to this ubuntu instance use gitbash
  https://git-scm.com/downloads

==========\

Installing docker on Linux
==========
1 Open get.docker.com

2 Copy and paste the below 2 commands

  curl -fsSL https://get.docker.com -o get-docker.sh

  sh get-docker.sh

==========

Images and Containers
==========
A Docker image is a combination of bin/libs that are necessary
for a s/w application to work.Initially all the s/w's of docker
are available in the form of docker images

A running instance of an image is called as a container

==========
Docker Host: The server where docker is installed is called
docker host

Docker client: This is CLI of docker which accepts the docker commands
from the users and passes to a background process called docker deamon

Docker deamon: This accepts the commands comming from docker client
and routes them to work on docker images or contaienr or the registry

Docker registry: This is the location where docker images are stored
This if of 2 type
1 Public (hub.docker.com)
2 Private: This is set up on one of our internal servers

==========
Day 3
==========
Important Docker commands
==========
Working on docker images
1 To download a docker image
  docker pull image_name

2 To upload an image into docker registry
  docker push image_name/image_id

3 To search for an image
  docker search image_name

4 To get detailed info about an image
  docker image inspect image_name/image_id

5 To delete a docker image
  docker rmi image_name/image_id

6 To create a docker image from a container
  docker commit container_name/container_id image_name

7 To create a docker image from a docker file
  docker build -t image_name .

8 To see the list of images that are present on our docker host
  docker images
  or
  docker image ls

9 To delete all the images
  docker system prune -af

10 To see the complete history for a docker image
   docker image history image_name/image_id

==========
Working on docker containers
==========
11 To see the list of all the running containers
   docker container ls

12 To see the list of all the containers (running and stopped)
   docker ps -a

13 To start a stopped container
   docker start container_name/container_id

14 To stop a running container

```
   docker stop container_name/container_id

15 To restart a container
   docker restart container_name/container_id

16 To delete a stopped container
   docker rm container_name/container_id

17 To delete a running container
   docker rm -f container_name/container_id

18 To stop all running containers
   docker stop $(docker ps -aq)

19 To delete all the stopped containers
   docker rm $(docker ps -aq)

20 To delete all containers (running and stopped)
   docker rm -f $(docker ps -aq)

21 To see the logs generated by a container
   docker logs container_name/container_id

22 To see the ports used by the container
   docker port container_name/container_id

23 To run an application in a container from outside
   docker exec -it container_name/container_id command
   Eg: To start a bash shell in the container
   docker exec -it container_name/container_id bash

24 To go into a container
   docker attach container_name/container_id

25 To create a docker container
   docker run image_name/image_id
   Run command options
   ------------------------
   --name: Used to give a customised name to the container
   -it: USed to open interactive terminal in the container
   -d : Used to run a container in the background in detached mode
   -e: Used to pass environment varibales to a container
   -v: Used to mount an external device or folder as a volume
   --volumes-from: Used to share volumes between multiple containers
   -p: USed for port mapping,it will link the container port with
       a port of the host machine
       eg: -p 8080:80 Here 80 is the container port(internal port) and
       8080 is the host port(external port).
   -P : This will automatically perform port mapping it will link
         the container port with a host port that is greater than 30000
   --link : Used to create a microservices architecture where we can link
            multiple containers
   -rm: Used to remove a container on exit
   --network: USed to run a container on a specific network
   -c: Used to specify an upper limit on the amount of cpu that a
container
       can use
   -m: Used to specify an upper limit on the memory that a container can
use
```

```
================================================================================
==
Day 4
================================================================================
===
Working on docker networks
==================================
26 To see the list of all docker networks
   docker network ls

27 To create a network
   docker network create --driver network_type network_name

28 To get detailed info about a network
   docker network inspect network_name/network_id

29 To delete a docker network
   docker network rm network_name/network_id

30 To Connect a running container to a network
   docker network connect network_name/network_id
container_name/container_id

31 To disconnect a running container to a network
   docker network disconnect network_name/network_id
container_name/container_id

================================================================================
Working on docker volumes
==================================
32 To see the list of volumes
   docker volume ls

33 To create a new docker volume
   docker volume create volume_name

34 To delete a docker volume
   docker volume rm volume_name/volume_id

35 To get detailed inof about a volume
   docker volume inspect volume_id/volume_name

================================================================================
================================================================================
=====
UseCase 1
===============
Create an nginx container in detached mode
docker run --name webserver -p 8888:80 -d nginx

To check if the nginx contianer is running
docker container ls

To access the ngonx from browser
public_ip_dockerhost:8888
================================================================================
===
UseCase 2
=================
Create a tomcat container and name it appserver
```

```
docker run --name appserver -p 9090:8080 -d tomee

To access tomcat from browser
Public_ip_of_dockerhost:9090

========================================================================
====
UseCase 3
====================
Create jenkins container and do automtic port mapping
docker run  --name jenkins -d -P jenkins

To see the ports used by jenkins
docker port jenkins

To access the jenkins from browser
public_ip_of_dockerhost:port_no_from_previous_step



============================================================
UseCase 4
Start centos as a container and launch interactive terminal on it
docker run --name c1 -it centos
exit
============================================================

Usecase 5
Create an ubuntu container and launch interactive terminal in it
docker run --name u1 -it ubuntu
exit

========================================================================
=====
UseCase 6

Create a mysql container and go into its bash shell
Login as mysql root user and create few tables.

1 Create a mysql container
  docker run --name db -d -e MYSQL_ROOT_PASSWORD=intelliqit mysql:5

2 To open interactive bash shell in the container
  docker exec -it db bash

3 To login into the db as root user
  myssql -u root -p
  Enter password "intelliqit"

4 To see the list of availble databases
  show databases;

5 To move into any of the above database
  use db_name;
  Eg: use sys;

6 To create emp and dept tables here
  Open https://justinsomnia.org/2009/04/the-emp-and-dept-tables-for-
mysql/
  Copy the code from emp and dept tables and paste in the myswl container
```

```
7 To check if the emp and dept tables are created
  select * from emp;
  select * from dept;
================================================================================
===
Day 5
================================================================================
===
=====================================================================
Creating a multi container architecture(microservices architecture)
=====================================================================
Various types of development and testing environments can be
created using docker in the following ways

1  --link  option (depricated)
2  docker-compose
3 Docker networking

4 Python Scripts
5 Ansible

--link Option: This is a docker run command option and it is
depricated.

UseCase 1
=============
Create 2 busybox containers and link them

1 Create a busybox container
  docke run  --name c1 -it busybox

2 To come out of the container without exit
  ctrl+p,ctrl+q

3 Create another busybox container and link it with the c1 container
  docker run  --name c1 --link c1:mybusybox -it busybox

4 Check if c2 can ping to c1
  ping c1


UseCase 2
=====================
Create a mysql container and a wordpress container and
link them

1 Create mysql:5 as a container
  docker run  --name db -d -e MYSQL_ROOT_PASSWORD=intelliqit mysql:5

2 Create a wordpress container and link it with a mysql container
  docker run --name mywordpress -d -p 8888:80 --link db:mysql wordpress

3 To check if wordpress is linked with mysql container
  docker inspect mywordpress
  Search for "Links" section

4 To acces the wordpress from browser
  public_ip_of_dockerhost:8888
```

UseCase 3
==========
Create a jenkins container and link with 2 tomcat containers
one for QAserver and another for prodserver

1 Create a jenkins container
  docker run --name jenkins -d -p 5050:8080 jenkins

2 To access jenkins from browser
  public_ip_of_dockerhost:5050

3 Create a tomcat container as qaserver and link with jenkins
  docker run --name qaserver -d -p 6060:8080 --link jenkins:myjenkins
tomcat

4 Create another tomcat container as prodserver and link with jenkins
  docker run --name prodserver -d -p 7070:8080 --link jenkins:myjenkins
tomcat

==========
Day 6
==========
Use Case
==========
Setup a postgres db and link with adminer container to access db from
browser

1 Create a postgres db container
  docker run --name db -d -e POSTGRES_PASSWORD=intelliqit -e
POSTGRES_USER=myuser
                                                -e POSTGRES_DB=mydb
postgres

2 Create an adminer container and link witj postgres
  docker run --name myadminer -d -p 9999:8080 --link db:postgres adminer

3 To access from level of browser
  public_ip_of_dockerhost:9999
==========
=====

UseCase
==========
Setup LAMP architecture where a mysql container can be linked
with an apache and php container

1 Create a mysql container
  docker run --name db -d -e MYSQL_ROOT_PASSWORD=intelliqit mysql

2 Create an apache and link with mysql container
  docker run --name apache -d -p 9090:80 --link db:mysql httpd

3 Create a php container and link with apache and mysql containers
  docker run --name php -d --link db:mysql --link apache:httpd php:7.2-
apache

4 To check if php container is linked with mysql and apache container
  docker inspect php
  Search for "Links" section

```
================================================================
Day 7
================================================================
================================================================================
=====
UseCase
=================
Create a testing environment where a selenium hub container
should be linked with 2 node containers one with chrome
installed and other with firefox installed.The testers should be
able to run the cross browser,cross platform automation
test scripts

1 Create a selenium hub image
  docker run --name hub -d -p 4444:4444 selenium/hub

2 Create a chrome node and link with the hub container
   docker run --name chrome -d -p 5901:5900 --link hub:selenium
                                          selenium/node-chrome-debug

3 Create a firefox node and link with the hub container
  docker run --name firefox -d -p 5902:5900 --link hub:selenium
                                          selenium/node-firefox-debug

4 Check if all 3 containers are running
  docker container ls

5 The above 2 containers are GUI containers and to access the GUI of
  these containers
  a) Install VNC viewer from
https://www.realvnc.com/en/connect/download/viewer/
  b) Open vnc viewer
  c) public_ip_of_dockerhost:5901 and 5902
  d) Click on continue--->Enter password:secret

================================================================================
=====



Docker Compose
=====================
This is used for creating a multi container architecture which is
reusable
Docker compose uses yml files

Install docker-compose
============================
1 Open https://docs.docker.com/compose/install/
2 Click on Linux tab
3 Copy and paste the commands

4 To check the version of docker compose
  docker-compose --version

================================================================================
====
Day 8
```

```
================================================================================
=====
Docker Compose
======================
This is used for creating a multi container architecture which is
reusable
Docker compose uses yml files

Install docker-compose
============================
1 Open https://docs.docker.com/compose/install/
2 Click on Linux tab
3 Copy and paste the commands

4 To check the version of docker compose
  docker-compose --version

================================================================================
=
UseCase
Create a docker compose file to setup a mysql container linked with
a database container

vim docker-compose.yml

---
version: '3.8'

services:
 mydb:
  image: mysql:5
  environment:
   MYSQL_ROOT_PASSWORD: intelliqit

 mywordpress:
  image: wordpress
  image:
  ports:
   - "8888:80"
  links:
   - mydb:mysql
...

To setup the containers from the above file
docker-compose up -d

To stop the containers
docker-compose stop

To stop and delete the containers
docker-compose down


================================================================
UseCase
Create a docker compose file to setup ci-cd environment
where a jenkins is linked with 2 tomcat containers

vim docker-compose.yml
version: '3.8'
```

```
services:
 myjenkins:
  image: jenkins/jenkins
  ports:
  - 5050:8080
  container_name: myjenkins

 qaserver:
  image: tomcat
  ports:
   - 6060:8080
  links:
   - myjenkins:jenkins
  container_name: qaserver

 prodserver:
  image: tomcat
  ports:
   - 7070:8080
  links:
   - myjenkins:jenkins
  container_name: prodserver
...

==========================================================================
==
UseCase
================
Create a docker compsoe file to setup the LAMP architecture
---
version: '3.8'

services:
 mydb:
  image: mysql
  environment:
   MYSQL_ROOT_PASSWORD: intelliqit
  container_name: mydb

 apache:
  image: httpd
  ports:
   - 9090:80
  container_name: apache
  links:
   - mydb:mysql

 php:
  image: php:7.2-apache
  links:
   - mydb:mysql
   - apache:httpd
  container_name: php
...
=================================================================
UseCase
Create a docker compose file to setup start mysql and link with adminer
application
 vim docker-compose.yml
```

```
---
version: '3.8'

services:
 mydb:
  image: mysql
  environment:
   MYSQL_ROOT_PASSWORD: intelliqit
  container_name: mydb


 myadminer:
  image: adminer
  ports:
   - 8888:8080
  container_name: myadminer
...
```

==========

Day 8

==========

==========

UseCase

==========

Create a docker compose file to setup the selenium testing environment where a selenium hub container is linked with 2 node containers one with chrome and other with firefox

```
vim docker-compose.yml
---
version: '3.8'

services:
 hub:
  image: selenium/hub
  ports:
   - 4444:4444
  container_name: hub

 chrome:
  image: selenium/node-chrome-debug
  ports:
   - 5901:5900
  links:
   - hub:selenium
  container_name: chrome

 firefox:
  image: selenium/node-firefox-debug
  ports:
   - 5902:5900
  links:
   - hub:selenium
  container_name: firefox
...
```

To setup the above architecture

```
docker-compose up -d
```

To check the running containers

```
docker container ls

To delete the containers
docker-compose down

=========================================================================
===
Docker Volumes
====================
Containers are ephemeral(temporary) but the data processed by the
containers should be persistent.Once a container is delete all the
data of the container is lost
To preserve the data even if the container is deleted we can use volumes

Volumes are classified into 3 types
====================================
1 Simple docker volume
2 Sharable dokcer volumes
3 Docker volume containers

Simple Docker volumes
============================
These voluems are used only for preserving the data on the host
machine even if the containers is deleted

UsedCase
============
Create a directory /data and mount it as a volume on an ubuntu container
Create some files in the mounted volumes and check if the files
are preserved on the host machine even after the container is deleted

1 Create /data directory
  mkdir /data

2 Create an ubuntu container and mount the above directory as volume
  docker run --name u1 -it -v /data ubuntu
  In the container u1 go into /data directory and create some files
  cd /data
  touch file1 file2 file3
  exit

3 Identify the locatiuon where the mounted data is preserved
  docker inspect u1
  Search for "Mounts" section and copy the "Source" path

4 Delete the container
  docker rm -f u1

5 Check if the data is still present
  cd "source_path_from_step3"
  ls
=========================================================================
Sharable Docker volumes
============================
These volumes are sharabale between multiple containers

Create 3 centos containers c1,c2,c3.
Mount /data as a volume on c1 container ,c2 should use the volume
used by c1 and c3 should use the volume used by c2
```

1 Create a centos container c1 and mount /data
  docker run --name c2 -it -v /data centos

2 Go into the data folder create files in data folder
  cd data
  touch f1 f2

3 Come out of the container without exit
   ctlr+p,ctlr+q

4 Create another centos container c2 and it should used the voluems used by c1
  docker run --name c2 -it --volumes-from c1 centos

5 In the c2 container go into data folder and create some file
  cd data
  touch f3 f4

6 Come out of the container without exit
   ctlr+p,ctlr+q

7 Create another centos container c3 and it should use the volume used by c2
  docker run --name c3 -it --volumes-from c2 centos

8 In the c3 container go into data folder and create some file
  cd data
  touch f5 f6

9 Come out of the container without exit
   ctlr+p,ctlr+q

10 Go into any of the 3 contianers and we will see all the files
   docker attach c1
   cd /data
  ls
  exit

12 Identify the location the where the mounted data is stored
   docker inspect c1
   Search for "Mounts" section and copy the "Source" path

13 Delete all containers
   docker rm -f c1 c2 c3

14 Check if the files are still present
   cd "source_path_from"step12"

==========
Day 9
==========
==========

Docker volume containers

These volumes are bidirectoinal ie the changes done on host
will be reflected into container and changes done by container
will be reflected to host machine

1 Create a volume

```
  docker volume create myvolume

2 To check the location where the mounted the volume works
  docker volume inspect myvolume

3 Copy the path shown in "MountPoint" and cd to that Path
  cd "MountPoint"

4 Create few files here
  touch file1 file2

5 Create a centos container and mount the above volume into the tmp
folder
  docker run --name c1 -it -v myvolume:/tmp centos

6 Change to tmp folder and check for the files
  cd /tmp
  ls
  If we create any files here they will be reflected to host machine
  And these files will be present on the host even after deleting the
  container.

==========================================================================
==
UseCase
============
Create a volume "newvolume" and create tomcat-users.xml file in it
Create a tomcat container and mount the above volume into it
Copy the tomcat-users.xml files to the required location

1 Create a volume
  docker volume create newvolume

2 Identify the mount location
  docker volume inspect newvolume
  Copy the "MountPoint" path

3 Move to this path
  cd "MountPoint path"

4 Create a file called tomcat-users.xml
  cat > tomcat-users.xml
  <tomcat-users>
      <user username="intelliqit" password="intelliqit" roles="manager-
script"/>
  </tomcat-users>

5 Create a tomcat container and mount the above volume
  docker run --name webserver -d -P -v newvolume:/tmp tomcat

6 Go into bash shell of the tomcat container
  docker exec -it webserver bash

7 Move the tomcat-users.xml file into conf folder
  mv /tmp/tomcat-users.xml conf/

==========================================================================

Creating customsied docker images
=========================================
```

This can be done in 2 ways
1 Using docker commit command
2 Using dockerfile

Using the docker commit command
==========
UseCase
==========
Create an ubuntu container and install some s/w's in it
Save this container as an image and later create a new container
from the newly created image.We will find all the s/w's that we
installed.

1 Create an ubuntu container
  docker run --name u1 -it ubuntu

2 In the container update the apt repo and install s/w's
  apt-get update
  apt-get install -y git

3 Check if git is installed or not
  git --version
  exit

4 Save the customised container as an image
  docker commit u1 myubuntu

5 Check if the new image is created or not
  docker images

6 Delete the previousely create ubuntu container
  docker rm -f u1

7 Create an new container from the above created image
  docker run --name u1 -it myubuntu

8 Check for git
  git --version

==========

Dockerfile
==========
Dockerfile uses predefined keyword to create customsied
docker images.

Important keyword in dockerfile
==========
FROM : This is used to specify the base image from where a
customised docker image has to be created

MAINTAINER : This represents the name of the organization or the
author that has created this dockerfile

RUN :Used to run linux commands in the container
     Generally it used to do s/w installtion or
     running scripts

USER : This is used to specify who should be the default user

to login into the container

COPY : Used to copy files from host to the customised image that
we are creating

ADD : This is similar to copy where it can copy files from host
to image but ADD can also downlaod files from some remote server

EXPOSE : USed to specify what port should be used by the container

VOLUME : Used for automatic volume mounting ie we will have a volume
mounted automatically when the container start

WORKDIR : Used to specify the default working directory of the container

ENV : This is used to specify what environment varibles should
be used

CMD : USed to run the default process of the container from outside

ENTRYPOINT : This is also used to run the default process of the
container

LABEL: Used to store data about the docker image in key value pairs

SHELL : Used to specify what shell should be by default used by the image

==========
====

UseCase
==========
Create a dockerfile to use nginx as abse image and specify
the maintainer as intelliqit

1 Create docker file
vim dockerfile

```
FROM nginx
MAINTAINER intelliqit
```

2 To create an image from this file
docker build -t mynginx .

3 Check if the image is created or not
docker images

Day 10
==========
UseCase
==========
Create a dockerfile from ubuntu base image and install
git in it

1 Create dockerfile
vim dockerfile

```
  FROM ubuntu
  MAINTAINER intelliqit
  RUN apt-get update
  RUN apt-get install -y git

2 Create an image from the above file
  docker build -t myubuntu .

3 Check if the new image is created
  docker images

4 Create a container from the new image and it should have git installed
  docker run  --name u1 -it myubuntu
  git --version
```

==========

Cache Busting

==========

When we create an image from a dockerfile docker stores all the executed isntructions in a its cache.Next time if we edit the same docker file and add few new instructions and build an image out of it docker will not execute the previously executed statements Instead it will read them from the cache
This is a time saving mechanism
The disadvantage is if the docker file is edited with a huge time gap then we might end up installing s/w's that are outdated

Eg:

```
FROM ubuntu
RUN apt-get update
RUN apt-get install -y git
RUN apt-get install -y tree
```

If we build an image from the above dockerfile docker saves all these instructions in the dockercache and if we add the below statement

```
RUN DEBIAN_FRONTEND=noninteractive apt-get -yq install maven
```

only this latest statement will be excuted

To avoid this problem and make docker execute all the instructions once more time without reading from cache we use "cache busting"

```
docker build --no-cache -t myubuntu .
```

==========

Day 11

==========

Create a shell script to install multiple s/w's and copy this into the docker image and execute it a the the time os creating the image

```
1 Create the shell script
  vim script.sh
  apt-get update
  for x in tree git
  do
      apt-get install -y $x
  done

2 Give excute permissions on that file
  chmod u+x script.sh
```

```
3 Create the dockerfile
  vim dockerfile
  FROM ubuntu
  MAINTIANER intelliqit
  COPY ./script.sh /
  RUN ./script.sh

4 Create an image from the dockerfile
  docker build -t myubuntu .

5 Create a container from the above image
  docker run  --name u1 -it myubuntu

6 Check if the script.sh is present in / and also see if tree and git are
installed
  ls  /
  git --version
  tree
================================================================================
===
UseCase
================
Create a dockerfile from ubuntu base image and install ansible in it

1 Create a dockerfile
  FROM ubuntu
  MAINTAINER intelliqit
  RUN apt-get update
  RUN apt-get install -y software-properties-common
  RUN apt-get install -y ansible

2 Create an image from the above file
  docker build -t ansible .

3 Create a container from the above image and check if ansible is present
  docker run --name a1 -it ansible
  ansible --version

================================================================================

================================================================

Create a dockerfile from ubuntu base image and make
/data as the default volume

1 Create a dockerfile
  vim dockerfile
  FROM ubuntu
  MAINTAINER intelliqit
  VOLUME /data

2 Create an image from the above dockerfile
  docker build -t myubuntu .

3 Create a container from the above image and check for the volume
  docker run  --name u1 -it myubuntu
  ls (we should see the data folder)

4 Go into the data folder and create some files
```

```
  cd data
  touch file1 file2
  exit

5 Check for the mount section and copy the source path

6 Delete the container
  docker rm -f u1

7 Check if the files are still present
  cd "source path"

========================================================================
Create a dcoker file from nginx base image and expose 80 port

1 vim dockerfile
  FROM nginx
  MAINTAINER intelliqit
  EXPOSE 90

2 Create an image
  docker build -t mynginx .

3 Create a container from above image
  docker run --name n1 -d -P mynginx

4 Check for the ports exposed
  docker port n1

========================================================================
Create a dockerfile from ubunt base image and downlaod jenkins.war
into it

1 Create a dockerfile
  vim dockerfile
  FROM ubuntu
  MAINTIANER intelliqit
  ADD https://get.jenkins.io/war-stable/2.263.4/jenkins.war  /

2 Create an image from the above dockerfile
  docker build -t myubuntu .

4 Create a container from this image
  docker run --name u1 -it myubuntu

5 Check if jenkins.war is present
  ls

========================================================================
======
Day 12
========================================================================
===
Create a dockerfile from jenkins base image and make
root as the deafult user

1 vim dockerfile
FROM jenkins/jenkins
MAINTAINER  intelliqit
USER root
```

2 Create an image from the above dockerfile
  docker build -t myjenkins .

3 Create a container
  docker run --name j1 -d -P myjenkins

4 Go into the bash shell and check if the user is root
  docker exec -it j1 bash
  whoami

==========
Create a dcokerfile from ubuntu base image and install
java in it,docwnload jenkins.war and make
"java -jar jenkins.war" as the default process of the
container

1 vim dockerfile
  FROM ubuntu
  MAINTAINER intelliqit
  RUN apt-get update
  RUN apt-get install -y openjdk-8-jdk
  ADD https://get.jenkins.io/war-stable/2.263.4/jenkins.war /
  ENTRYPOINT ["java","-jar","jenkins.war"]

2 Create an image from the above file
  docker build -t myubuntu .

3 Create a container from the above image and we will see that
  it behaves like a jenkins container
  docker run --name u1 -it myubuntu

4 Check the default process that is running
  docker container ls

==========
Day 13
==========
UseCase
==========
Create a dockerfile from ubuntu base image and make it behave
like nginx

1 Create a dockerfile
  vim dockerfile
  FROM ubuntu
  MAINTAINER intelliqit
  RUN apt-get update
  RUN apt-get install -y nginx
  ENTRYPOINT ["/usr/sbin/nginx","-g","daemon off;"]
  EXPOSE 80

2 Create an image from the above dockerfile
  docker build -t myubuntu .

3 Create a container from the above image and it will work like nginx
  docker run --name n1 -d -P myubuntu

```
4 Check the ports used by nginx
  docker container ls

5 To access nignx from browser
  public_ip_of_dockerhost:port_no_captured_from_step4

================================================================================
====
UseCase
Create a dockerfile fromj centos base image and install
httpd in it make httpd as the default process
1 Create the index.html
  <html>
        <body>
                <h1>Welcome to IntelliQIT</h1>
        </body>
 </html>


2 Create the dockerfile
  vim dockerfile
  FROM centos
  MAINTAINER intelliqit
  RUN yum -y update
  RUN yum -y install httpd
  COPY index.html /var/www/html
  ENTRYPOINT ["/usr/sbin/httpd","-D","FOREGROUND"]
  EXPOSE 80

3 Create an image from the above dockerfile
  docker build -t mycentos .

4 Create a container from the above image
  docker run --name c1 -d -P mycentos

5 Check the ports used by container
  docker container ls

6 To access the from browser
  public_ip_of_dockerhost:port_from_step5


================================================================================
CMD and ENTRYPOIT
------------------------
Bothe of them are used to specify the default process that should be
triggered when the container starts but the CMD instruction can be
overridden with some other process passed at the docker run command

Eg:
FROM ubuntu
RUN apt-get update
RUN apt-get install -y nginx
CMD ["/usr/sbin/nginx","-g","daemon off;"]
EXPOSE 80

Though the default process is to trigger nginx we can bypass that
and make it work on some other process
```

```
docker build -t myubuntu .
Create a container
docker run --name u1 -it -d myubuntu
Here if we inspect the default process we will see that
nginx as the default process
docker container ls

on the otherhand we can modify that default process to something else
docker run --name u1 -d -P myubuntu ls -la
Now if we do "docker container ls"  we will see the dafault process
to be "ls -la"

================================================================================
Docker Networking
====================
Docker uses 4 types os networks
1 Bridge: This is the deafult network of docker when contianers are
          running on a single docker host

2 Host: This is used when we want to run a single container on a
dockerhost
         and this contianer communicates only with the host machine

3 Null: This is used for creating isolated containers ie these containers
        cannot communicate with th host machine or with other containers

4 Overlay: This is used when containers are running in a distributed
environment
           on multiple linux servers



UseCase
===============
Create 2 bridge networks intelliq1 and intelliq2
Create 2 busybox containers c1,c2 and c3
c1 and c2 should run on intelliq1 network and shoul ping each other
c3 should run on intelliq2 network and it should not be able to ping c1
or c2
Now put c2 on intelliq2 network,since c2 is on both intelliq1 and
intelliq2
networks it should be able to ping to both c1 and c3
but c1 and c3 should not ping each other directly

1 Create 2 bridge networks
  docker network create --driver bridge intelliq1
  docker network create --driver bridge intelliq2

2 Check the list of available networks
  docker network ls

3 Create a busybox container c1 on intelliqi1 network
  docker run --name c1 -it --network intelliq1 busybox
  Come out of the c1 container without exit ctrl+p,ctrl+q

4 Identify the ipaddress of c1
  docker inspect c1
```

```
5 Create another busybox container c2 on intelliq1 network
  docker run --name c2 -it --network intelliq1 busybox
  ping ipaddress_of_c1    (It will ping)
  Come out of the c2 container without exit ctrl+p,ctrl+q

6 Identify the ipaddress of c2
  docker inspect c2

7 Create another busybox container c3 on intelliq2 network
  docker run --name c3 -it --network intelliq2 busybox
  ping ipaddress_of_c1  (It should not ping)
  ping ipaddress_of_c2  (It should not ping)
  Come out of the c3 container without exit ctrl+p,ctrl+q

8 Identify the ipaddress of c3
  docker inspect c3

9 Now attach intelliq2 network to c2 container
  docker network connect intelliq2 c2

10 Since c2 is now on both intelliq1 and intelliq2 networks it should
ping
   to both c1 and c3 containers
   docker attach c2
   ping ipaddress_of_c1  (It should  ping)
   ping ipaddress_of_c3  (It should  ping)
   Come out of the c2 container without exit ctrl+p,ctrl+q

11 But c1 and c3 should not ping each other
   docker attach c3
   ping ipaddress_of_c1  (It should not ping)

==========================================================================
==
Day 14
==========================================================================
==
==========================================================================
====
UseCase
=============
Create a custom bridge network and create a docker compose file
to start postgres and adminer container on the above created
network

1 Create a custom bridge network
  docker network create --driver bridge --subnet 10.0.0.0/24 intelliqit

2 Create a docker compose file
  vim docker-compose.yml
---
version: '3.8'

services:
 db:
  image: postgres
  environment:
   POSTGRES_PASSWORD: intelliqit
   POSTGRES_USER: myuser
   POSTGRES_DB: mydb
```

```
 adminer:
  image: adminer
  ports:
   - 8888:8080

networks:
 default:
  external:
   name: intelliqit
...

3 To create the containers
  docker-compose up -d

4 To see if adminer and postgres contianers are created
  docker container ls

5 To check if they are running on intelliqit network
  docker inspect container_id_from_Step4

========================================================================
UseCase
Create a docker compose file which starts jenkins and 2 tomee as
containers also this compose file should create 2 networks
abc,xyz.On abc network it should run the jenkins container
and xyz network it should run the 2 tomee containers

vim docker-compose.yml
---
version: '3.8'

services:
 myjenkins:
  image: jenkins/jenkins
  ports:
   - 5050:8080
  networks:
   - abc


 qaserver:
  image: tomee
  ports:
   - 6060:8080
  networks:
   - xyz


 prodserver:
  image: tomee
  ports:
   - 7070:8080
  networks:
   - xyz

networks:
 abc: {}
 xyz: {}
```

```
--------------------------------------------------------------------------
-----
UseCase
================
Create a docker compose file to start mysql and wordpress as container
it shoudl also create 2 volumes one for wordpress and other for mysql

vim docker-compose.yml
version: '3.8'

services:
 db:
  image: mysql:5
  environment:
   MYSQL_ROOT_PASSWORD: intelliqit
  volumes:
   - db:/var/lib/mysql

 wordpress:
  image: wordpress
  ports:
   - 9090:80
  volumes:
   - wordpress:/var/www/html

volumes:
 db:
 wordpress:

To create containers from the above file
docker-compose up -d

Check if 2 new volumes are created
docker volume ls



==========================================================================
==

UseCase
=================
Create a dockerfile to create a customsied jenkins image
and this dockerfile should be built as image from the docker composefile

vim dockerfile
FROM  jenkins/jenkins
MAINTAINER intelliqit
USER root
RUN apt-get update
RUN apt-get install -y git

vim docker-compose.yml
---
version: '3.8'

services:
 jenkins:
  build: .
```

```
 mytomcat:
  image: tomee
  ports:
   - 8080:8080
...

To create containers from the above file
docker-compose up -d

Check if a new jenkins image has be created
docker images

================================================================================
Day 15
================================================================================
=
================================================================================
====
Container Orchestration
==============================
This is the process of handling docker containers running
on multiple linux servers in a distributed environment


Advantages
=================
1 Load Balancing
2 Scalling
3 Rolling update
4 High Availability and Disaster recovery(DR)

LoadBalancing
==================
Each container is capable of sustaining a specific user load
We can increase this capacity by running the same application
on multiple containers(replicas)

Scalling
==============
We should be able to increase or decrease the number of containers
on which our applications are running without the end user
exepriencing any downtime.

Rolling update
======================
Application running in a live environment should be upgraded or
downgraded to a different version without the end user having any
downtime

Disaster Recovery
=====================
In case of network failuers or server crashes still the container
orchestration tools maintain the desired count of containers
and thereby provide the same service to the end user

Popular container orchestration tools
=========================================
1 Docker Swarm
2 Kubernetes
3 OpenShift
```

4 Mesos

==========
Setup of Docker Swarm
==========
1 Create 3 AWS ubuntu instances
2 Name them as Manager,Worker1,Worker2
3 Install docker on all of them
4 Change the hostname
  vim /etc/hostname
  Delete the content and replace it with Manager or Worker1 or Worker2
5 Restart
  init 6
6 To initilise the docker swarm
  Connect to Manager AWS instance
  docker swarm init
  This command will create a docker swarm and it will also generate
  a tokenid
7 Copy and paste the token id in Worker1 and Worker2

==========
Ports used by docker swarm
==========
TCP port 2376 for secure Docker client communication. This port is
required for Docker Machine to work. Docker Machine is used to
orchestrate Docker hosts.

TCP port 2377. This port is used for communication between the nodes of a
Docker Swarm or cluster. It only needs to be opened on manager nodes.

TCP and UDP port 7946 for communication among nodes (container network
discovery).
UDP port 4789 for overlay network traffic (container ingress networking).

==========
Load Balancing:
Each docker containers has a capability to sustain a specific
user load.To increase this capability we can increase the
number of replicas(containers) on which a service can run

UseCase

Create nginx with 5 replicas and check where these replicas are
running

1 Create nginx with 5 replicas
  docker service create --name webserver -p 8888:80 --replicas 5 nginx

2 To check the services running in swarm
  docker service ls

3 To check where these replicas are running
  docker service ps webserver

4 To access the ngonx from browser

```
  public_ip_of_manager/worker1/worker2:8888

5 To delete the service with all replicas
  docker service rm webserver
================================================================================
UseCase
===========
Create mysql with 3 replicas and also pass the necessary environment
variables

1 docker service create --name db --replicas 3
                    -e MYSQL_ROOT_PASSWORD=intelliqit  mysql:5

2 To check if 3 replicas of mysql are running
  docker service ps db

================================================================================
=
Day 16
================================================================================
=

Scalling
============
This is the process of increasing the number of replicas or decreasing
the replicas count based on requirement without the end user experiencing
any down time.

UseCase
============
Create tomcat with 4 replicas and scale it to 8 and scale it
down to 2

1 Create tomcat with 4 replicas
  docker service create --name appserver -p 9090:8080 --replicas 4 tomcat

2 Check if 4 replicas are running
  docker service ps appserver

3 Increase the replicas count to 8
  docker service scale appserver=8

4 Check if 8 replicas are running
  docker service ps appserver

5 Decrese the replicas count to 2
  docker service scale appserver=2

6 Check if 2 replicas are running
  docker service ps appserver

================================================================================
Rolling updates
======================
Services running in docker swarm should be updated from once
version to other without the end user downtime

UseCase
===========
Create redis:3 with 5 replicas and later update it to redis:4
```

```
also rollback to redis:3

1 Create redis:3 with 5 replicas
  docker service create --name myredis --replicas 5 redis:3

2 Check if all 5 replicas of redis:3 are running
  docker service ps myredis

3 Perfrom a rolling update from redis:3 to redis:4
  docker service update --image redis:4 myredis

4 Check redis:3 replcias are shut down and in tis palce redis:4 replicas
are running
  docker service ps myredis

5 Roll back from redis:4 to redis:3
  docker service update --rollback myredis

6 Check if redis:4 replicas are shut down and in its place redis:3 is
running
  docker service ps myredis


================================================================================
=======
To remove a worker from swarm cluster
docker node update --availability drain Worker1

To make this worker rejoin the swarm
docker node update --availability active Worker1

To make worker2 leave the swarm
Connect to worker2 usig git bash
docker swarm leave

To make manager leave the swarm
docker swarm leave --force

To generate the tokenid for a machine to join swarm as worker
docker swarm join-token worker

To generate the tokenid for a machine to join swarm as manager
docker swarm join-token manager

To promote Worker1 as a manager
docker node promote Worker1

To demote "Worker1" back to a worker status
docker node demote Worker1



====================================================================
FailOver Scenarios of Workers
================================
Create httpd with 6 replicas and delete one replica running on the
manager
Check if all 6 replicas are still running
```

Drain Worker1 from the docker swarm and check if all 6 replicas are running
on Manager and Worker2,make Worker1 rejoin the swarm

Make Worker2 leave the swarm and check if all the 6 replicas are running on Manager and Worker1

1 Create httpd with 6 replicas
  docker service create --name webserver -p 9090:80 --replicas 6 httpd

2 Check the replicas running on Manager
  docker service ps webserver | grep Manager

3 Check the container id
  docker container ls

4 Delete a replica
  docker rm -f container_id_from_step3

5 Check if all 6 replicas are running
  docker service ps webserver

6 Drain Worker1 from the swarm
  docker node update --availability drain Worker1

7 Check if all 6 replicas are still running on Manager and Worker2
  docker service ps webserver

8 Make Worker1 rejoin the swarm
  docker node update --availability active Worker1

9 Make Worker2 leave the swarm
  Connect to Worker2 using git bash
  docker swarm leave
  Connect to Manager

10 Check if all 6 replicas are still running
   docker service ps webserver

==========
=
Day 17
==========
 ==========
FailOver Scenarios of Managers
==========
If a worker instance crashses all the replicas running on that
worker will be moved to the Manager or the other workers.
If the Manager itself crashes the swarm becomes headless
ie we cannot perfrom container orchestration activites in this
swamr cluster

To avoid this we should maintain multiple managers
Manager nodes have the status as Leader or Reachable

If one manager node goes down other manager becomes the Leader
Quorum is resonsible for doing this activity and if uses a RAFT
algorithm for handling the failovers of managers.Quorum also
is responsible for mainting the min number of manager

Min count of manager required for docker swarm should be always
more than half of the total count of Managers

Total Manager Count	-	Min Manager Required
1	-	1
2	-	2
3	-	2
4	-	3
5	-	3
6	-	4
7	-	4

==========
==========
Overlay Networking
==========
This is the deafult network used by swarm
and this network perfroms network load balancin
ie even if a service is running on a specicfic worker we can
access if from orther slave

==========
====
UseCase
==========
Create 2 overlay networks intelliqit1 and intelliqit2
Create httpd with 5 replacs on intelliqit1 network
Create tomcat with 5 replicas on default overlay "ingres" network
and later perform rolling network update to intelliqit2 network

1 Create 2 overlay networks
  docker network create --driver overlay intelliqit1
  docker network create --driver overlay intelliqit2

2 Check if 2 overlay networks are created
  docker network ls

3 Create httpd with 5 replcias on inteliiqit1 network
  docker service create --name webserver -p 8888:80 --replicas 5
                                          --network intelliqit1 httpd

4 To check if httpd is running on intelliqit1 network
  docker service inspect webserver
  This command will generate the output in JSON format
  To see the above output in normal text fromat
  docker service inspect webserver --pretty

5 Create tomcat with 5 replicas on the deafult ingres network
  docker service create --name appserver -p 9999:8080 --replicas 5 tomcat

6 Perform a rolling network update from ingres to intelliqit2 network
  docker service update --network-add intelliqit2 appserver

7 Check if tomcat is now running on intelliqit2 network
  docker service inspect appserver --pretty

Note: To remove from intelliqit2 network
      docker service update --network-rm intelliqit2 appserver

==========
Day 18
==========

==========
Overlay Networking
==========
This is the deafult network used by swarm
and this network perfrom network load balancin
ie even if a service is running on a specicfic worker we can
access if from orther slave

UseCase
==========
Start nginx with 2 repliacs and check if we can acces it from
browser from manager and all workers

1 Create nginx
  docker service create --name webserver -p 8888:80 --replicas 2 nginx

2 Check where these 2 replcas are running
  docker service ps webserver
  These repliacs will be running on only 2 nodes and we will have a third
  node where it it not running

3 Check if we can access nginx from the third node where it is not
present
  public_ip_of_thirdnode:8888

==========
Docker Stack
==========
docker compose + docker swarm = docker stack
docker compose + kubernetes = kompose

Docker compose when implemented at the level of docker swarm
it is called docker stack.Using docker stack we can create an orchestreta
a micro services architecture at the level of production servers

1 To create a stack from a compose file
  docker stack deploy -c compose_filename stack_name

2 To see the list of stacks created
  docker stack ls

3 To see on which nodes the stack services are running
  docker stack ps stack_name

4 To delete a stack
  docker stack rm stack_name

```
==========================================================================
UseCase
================
Create a docker stack file to start 3 replicas of wordpress
and one replica of mysql

vim stack1.yml
---
version: '3.8'

services:
 db:
  image: "mysql:5"
  environment:
   MYSQL_ROOT_PASSWORD: intelliqit

 wordpress:
  image: wordpress
  ports:
   - "8989:80"
  deploy:
   replicas: 3

To start the stack file
docker stack deploy -c stack1.yml mywordpress

To see the services running
docker service ls

To check where the serives are running
docker stack ps mywordpress

To delete the stack
docker stack rm mywordpress


==========================================================================
=======
UseCase
==============
Create a stack file to setup CI-cd architecture where a jenkins
container is linked with tomcats for qa and prod environments
The jenkins contianers should run only on Manager
the qaserver tomcat should run only on Worker1 and prodserver
tomcat should run only on worker2

vim stack2.yml
---
version: '3.8'

services:
 myjenkins:
  image: jenkins/jenkins
  ports:
   - 5050:8080
  deploy:
   replicas: 2
   placement:
    constraints:
     - node.hostname == Manager
```

```
 qaserver:
  image: tomcat
  ports:
   - 6060:8080
  deploy:
   replicas: 3
   placement:
    constraints:
     - node.hostname == Worker1

 prodserver:
  image: tomcat
  ports:
   - 7070:8080
  deploy:
   replicas: 4
   placement:
    constraints:
     - node.hostname == Worker2
...

To start the services
docker deploy -c stack2.yml ci-cd

To check the replicas
docker stack ps ci-cd
=====


================================================================
Day 19
================================================================
UseCase
Create a stack file to setup the selenium hub and nodes architecture
but also specify a upper limit on the h/w

vim stack3.yml
---
version: '3.8'

services:
 hub:
  image: selenium/hub
  ports:
   - 4444:4444
  deploy:
   replicas: 2
   resources:
    limits:
     cpus: "0.1"
     memory: "300M"

 chrome:
  image: selenium/node-chrome-debug
  ports:
   - 5901:5900
  deploy:
   replicas: 3
   resources:
```

```
    limits:
     cpus: "0.01"
     memory: "100M"

 firefox:
  image: selenium/node-firefox-debug
  ports:
   - 5902:5900
  deploy:
   replicas: 3
   resources:
    limits:
     cpus: "0.01"
     memory: "100M"
```

```
=====================================================================
Docker secrets
==========================
This is a feature of docker swarm using which we can pass secret data
to the services running in swarm cluster
These secrets are created on the host machine and they will be
availbale from all the replicas in the swarm cluster

1 Create a dcoker secret
  echo " Hello Intelliqit" | docker secret create mysecret -

2 Create a redis db with 5 replace and mount the secret
  docker service create --name myredis --replicas 5 --secret mysecret
redis

3 Capture one of the replica contianer id
  docker container ls

4 Check if the secret data is available
  docker exec -it container_id cat /run/secrets/mysecret


=====================================================================
=====
Create 3 secrets for postgres user,password and db
and pass them to the stack file

1 Create secrets
  echo "intelliqit" | docker secret create pg_password -
  echo "myuser" | docker secret create pg_user -
  echo "mydb" | docker secret create pg_db -

2 Check if the secrets are created
  docker secret ls

3 Create the docker stack file to work on these secrets
  vim stack6.yml
---
version: '3.1'
services:
  db:
    image: postgres
    environment:
      POSTGRES_PASSWORD_FILE: /run/secrets/pg_password
      POSTGRES_USER_FILE: /run/secrets/pg_user
      POSTGRES_DB_FILE: /run/secrets/pg_db
```

```
    secrets:
     - pg_password
     - pg_user
     - pg_db

  adminer:
    image: adminer
    restart: always
    ports:
      - 8080:8080
    deploy:
     replicas: 2

secrets:
    pg_password:
     external: true
    pg_user:
     external: true
    pg_db:
     external: true

...
================================================================================
Working on docker registry
==========================
This is the location where the docker images are saved
This is of 2 types
1 Public registry
2 Private regsitry

UseCase
Create a customised ubuntu image and upload into the public registry

1 Signup into hub.docker.com

2 Create a customsied ubuntu image
  a) Create a centos container and install git init
     docker run --name u1 -it ubuntu
    apt-get update
    apt-get  install -y  git
    exit

  b) Save this container as an image
     docker commit u1 intelliqit/ubuntu_may_25

3 Login into dockerhub
  docker login
  Enter username and password of dockerhub

4  Push the customised image
   docker push intelliqit/ubuntu_may_25


================================================================================
Private Registry
====================
This can be created using a docker image called as "registry"
We can start this as a container and it will allow us to
push images into the registry
```

```
1 Create registry as a container
  docker run --name lr -d -p 5000:5000 registry

2 Download an alpine image
  docker pull alipne

3 Tag the alpine with the local registry
  docker tag alpine localhost:5000/alpine

4 Push the image to local registry
  docker push localhost:5000/alpine
```

```
================================================================================
===
Day 20
================================================================================
====
Kubernetes
==================

Menions: This is an individual node used in kubernetes
Combination of these minions is called as Kubernetes cluster

Master is the main machine which triggers the container orchestraion
It distributes the work load to the Slaves

Slaves are the nodes that accept the work load from the master
and handle activites load balancing,autoscalling,high availability etc


==============================================
Kubernetes unmanaged setup on Centos
==========================================
Install, start and enable docker service

yum install -y -q yum-utils device-mapper-persistent-data lvm2 >
/dev/null 2>&1
yum-config-manager --add-repo
https://download.docker.com/linux/centos/docker-ce.repo > /dev/null 2>&1
yum install -y -q docker-ce >/dev/null 2>&1


systemctl start docker
systemctl enable docker

================================================================================
============
Disable SELINUX

setenforce 0
sed -i --follow-symlinks 's/^SELINUX=enforcing/SELINUX=disabled/'
/etc/sysconfig/selinux

================================================================================
==================
Disable SWAP

sed -i '/swap/d' /etc/fstab
swapoff -a
```

```
================================================================================
=================
Update sysctl settings for Kubernetes networking

cat >>/etc/sysctl.d/kubernetes.conf<<EOF
net.bridge.bridge-nf-call-ip6tables = 1
net.bridge.bridge-nf-call-iptables = 1
EOF
sysctl --system

================================================================================
==================
Add Kubernetes to yum repository

cat >>/etc/yum.repos.d/kubernetes.repo<<EOF
[kubernetes]
name=Kubernetes
baseurl=https://packages.cloud.google.com/yum/repos/kubernetes-el7-x86_64
enabled=1
gpgcheck=1
repo_gpgcheck=1
gpgkey=https://packages.cloud.google.com/yum/doc/yum-key.gpg
        https://packages.cloud.google.com/yum/doc/rpm-package-key.gpg
EOF

================================================================================
=============
Install Kubernetes
yum install -y kubeadm kubelet kubectl

================================================================================
=========
Enable and start Kubernetes service

systemctl start kubelet
systemctl enable kubelet
================================================================================
============
Repeat the above steps on Master and slaves
================================================================================
==============

On Master=============
===========
Initilise the Kubernetes cluster
----------------------------------------

kubeadm init --apiserver-advertise-address=ip_of_master --pod-network-
cidr=192.168.0.0/16

================================================================================
================

To be able to use kubectl command to connect and interact with the
cluster,
the user needs kube config file.

mkdir /home/centos/.kube
cp /etc/kubernetes/admin.conf /home/centos/.kube/config
chown -R centos:centos /home/centos/.kube
```

==========
==========
Deploy calico network
kubectl create -f
https://docs.projectcalico.org/v3.9/manifests/calico.yaml

==========
==========
For slaves to join the cluster
kubeadm token create --print-join-command

==========
==========
Check the pods of kube-system are running

kubectl get pods -n kube-system

==========
===
Day 21
==========
Kubernetes
==========

Menions: This is an individual node used in kubernetes
Combination of these minions is called as Kubernetes cluster

Master is the main machine which triggers the container orchestraion
It distributes the work load to the Slaves

Slaves are the nodes that accept the work load from the master
and handle activites load balancing,autoscalling,high availability etc

Kubernetes uses various of types of Object

1 Pod: This is a layer of abstraction on top of a container.This is the samallest
  object that kubernetes can work on.In the Pod we have a container.
  The advantage of using a Pod is that kubectl commands will work on the Pod and the
  Pod communicates these instructions to the container.In this way we can use the
  same kubectl irresepective of which technology containers are in the Pod.

2 Service: This is used for port mapping and network load balancing

3 NameSpace: This is used for creating partitions in the cluster.Pods running
 in a namespace cannot communicate with other pods running in other namespace

4 Secrets: This is used for passing encrypted data to the Pods

5 ReplicationController: This is used for managing multiple replicas of PODs
and also perfroming saclling

6 ReplicaSet: This is similar to replicationcontroller but it is more advanced
where features like selector can be implemented

7 Deployment: This used for perfroming all activites that a Replicaset can do
  it can also handle rolling update

8 Volume: Used to preserve the data even when the pods are deleted

9 Statefulsets: These are used to handle stateful application like data bases
  where consistency in read write operations has to be maintained.

10 Ingress: This object is used for mapping ip with domain name

Kubernetes Architecture
==========
Master Componentes
==========
Container runtime: This can be docker or anyother container technology

apiServer: Users interact with the apiServer using some clinet like ui,command line tool like kubelet.It is the apiServer which is the gateway to the cluster
It works as a gatekeeper for authentication and it validates if a specific
user is having permissions to execute a specific command.Example if we want to
deploy a pod or a deployment first apiServers validates if the user is authorised to perform that action and if so it passes to the next process ie the "Scheduler"

Scheduler: This process accepts the instructions from apiServer after validation
and starts an application on a sepcific node or set of nodes.It estimates how much amount of h/w is required for an application and then checks which
slave have the necessary h/w resources and instructs the kubelet to deploy
the application

kubelet: This is the actual process that takes the orders from scheduler and
deploy an application on a slave.This kubelet is present on both master and slave

controller manager: This check if the desired state of the cluster is always
maintained.If a pod dies it recreates that pod to maintain the desired state

etcd: Here the cluster state is maintained in key value pairs.
It maintains info about the slaves and the h/w resources available on the slaves and also the pods running on the slaves
The scheduler and the control manager read the info from this etcd

and schedule the pods and maintain the desired state

==========
==
Worker components
==========
containerrun time: Docker or some other container technology

kubelet: This process interacts with container run time and the node
and it start a pod with a container in it

kubeproxy: This will take the request from services to pod
It has the intellegence to forward a request to
a near by pod.Eg If an application pod wants to communicate with a db pod
then kubeproxy will take that request to the nearby pod

==========
========
==========
====
Setup of ManagedKubernetes
==========
Free
==========
1 http://katakoda.com
(or)
2 http://playwithk8s.com

Paid
==========
1 Signup for a Google cloud account
2 Click on Menu icon on top right corner--->Click on Kubernetes Engine--
>Clusters
3 Click on Create cluster--->Click on Create

==========
=======
Day 22
==========
=======
==========
UseCase
==========
Create nginx as a pod and name it webserver
kubectl run --image nginx webserver

To see the list of pods running
kubectl get pods

To see more info about the pods like their ip and slave where they are
running
kubectl get pods -o wide

To delete the pod
kubectl delete pods webserver

==========
===
UseCase
==========

Create mysql pod and name it mydb and go into its interactive terminal
and create few tables

```
kubectl run --image mysql:5 mydb --env MYSQL_ROOT_PASSWORD=intelliqit
```

To check the pods
```
kubectl get pods
```

To go into the interactive terminal
```
kubectl exec -it mydb -- bash
```

To login into the db
```
mysql -u root -p
Password: intellqiit
```

Create tables here

==========

Kuberentes Defintion files

==========

Objects in Kubernetes cluster are deployed using these
defintion files
They are created using yml and they generally these 4 top level
fields.

```
apiVersion:
kind:
metadata:
spec:
```

apiVersion : This specifies the code library that has to be imported
to create a particualr kind of Kubernetes object

kind: Here we specify the type kubernetes object that we want to
create(Pod,ReplicaSet,Deployment,Service etc)

metadata: Here we can give additional info about the Pod like
the name of the Pod,some labels etc

spec: This is where exact info about the object that is created is
specified like containers info port mapping,no of replicas etc

==========

kind	apiVersions
Pod	v1
Service	v1
Secret	v1
Namespace	v1
ReplicationController	v1
Volume	v1
ReplicaSet	apps/v1
Deployment	apps/v1
StatefuleSet	apps/v1

==========

Create a pod defintion file to start nginx pod with a name webserver

1 vim pod-defintion1.yml

```
---
apiVersion: v1
kind: Pod
metadata:
 name: nginx-pod
 labels:
  type: proxy
  author: intelliqit
spec:
 containers:
  - name: webserver
    image: nginx

...

2 Create pod from the above file
  kubectl apply -f pod-defintion1.yml

3 To check the list of pods
  kubectl get pods

4 To delete the pods
  kubectl delete -f pod-defintion1.yml

=========================================================================
UseCase
================
Create a postgres-pod and give the labels as author=intelliqit
and type=db,also pass the necessay environment variables

1 vim pod-definition2.yml
apiVersion: v1
kind: Pod
metadata:
 name: postgres-pod
 labels:
  author: intelliqit
  type: db
spec:
 containers:
  - name: mydb
    image: postgres
    env:
     - name: POSTGRES_PASSWORD
       value: intelliqit
     - name: POSTGRES_USER
       value: myuser
     - name: POSTGRES_DB
       value: mydb
...

To create pods from the above file
kubectl apply -f pod-defintion2.yml


=================================================================
UseCase
============
Create a jenkins-pod and also perfrom necessary port mapping
```

```
vim pod-definition2.yml
---
apiVersion: v1
kind: Pod
metadata:
 name: jenkins-pod
 labels:
  type: ci-cd
  author: intelliqit
spec:
 containers:
  - name: jenkins
    image: jenkins/jenkins
    ports:
     - containerPort: 8080
       hostPort: 8080
...

To create the pods from the above file
kubectl apply -f pod-defintion3.yml

To check if the jnekins pod is running
kubectl get pods -o wide

To accesss jenkins from browser
kubectl get nodes -o wide
Capture the external ip of the node where jenkins pod is running
in browser
externalip:8080

================================================================================
Day 23
================================================================================
================================================================================
ReplicationController
======================
This is a high level Kubernets object that can be used for handling
multiple replicas of a Pod.Here we can perfrom Load Balancing
and Scalling

ReplicationController uses keys like "replicas,template" etc in the
"spec" section
In the template section we can give metadata related to the pod and also
use
another spec section where we can give containers information

Create a replication controller for creating 3 replicas of httpd
vim repilication-controller.yml
---
apiVersion: v1
kind: ReplicationController
metadata:
 name: httpd-rc
 labels:
  author: intelliqit
spec:
 replicas: 3
 template:
```

```
  metadata:
   name: httpd-pod
   labels:
    author: intelliqit
  spec:
   containers:
    - name: myhttpd
      image: httpd
      ports:
       - containerPort: 80
         hostPort: 8080
...

To create the httpd replicas from the above file
kubectl create -f replication-controller.yml

To check if 3 pods are running an on whcih slaves they are running
kubectl get pods -o wide

To delete the replicas
kubectl delete -f replication-controller.yml



ReplicaSet
==================
This is also similar to ReplicationController but it is more
advanced and it can also handle load balancing and scalling
It has an additional field in spec section called as "selector"
This selector uses a child element "matchLabels" where the
it will search for Pod based on a specific label name and try to add
them to the cluster

Create a replicaset file to start 4 tomcat replicas  and then perform
scalling
vim replica-set.yml
---
apiVersion: apps/v1
kind: ReplicaSet
metadata:
 name: tomcat-rs
 labels:
  type: webserver
  author: intelliqit
spec:
 replicas: 4
 selector:
  matchLabels:
   type: webserver

 template:
  metadata:
   name: tomcat-pod
   labels:
    type: webserver
  spec:
   containers:
    - name: mywebserver
      image: tomcat
      ports:
```

```
        - containerPort: 8080
          hostPort: 9090
```

To create the pods from the above file
kubectl create -f replica-set.yml

Scalling can be done in 2 ways
a) Update the file and later scale it

b) Scale from the coomand prompt withbout updating the defintion file

a) Update the file and later scale it
  Open the replicas-set.yml file and increase the replicas count from 4 to 6
  kubectl replace -f replicas-set.yml
  Check if 6 pods of tomcat are running
  kubectl get pods

b) Scale from the coomand prompt withbout updating the defintion file
   kubectl scale --replicas=2 -f replica-set.yml

==========
Deployment
==========

This is also a high level Kubernetes object which can be used for scalling and load balancing and it can also perfrom rolling update

Create a deployment file to run nginx:1.7.9 with 3 replicas

vim deployment1.yml

```
---
apiVersion: apps/v1
kind: Deployment
metadata:
 name: nginx-deployment
 labels:
  author: intelliqit
  type: proxyserver
spec:
 replicas: 3
 selector:
  matchLabels:
   type: proxyserver
 template:
  metadata:
   name: nginx-pod
   labels:
    type: proxyserver
  spec:
   containers:
    - name: nginx
      image: nginx:1.7.9
      ports:
       - containerPort: 80
         hostPort: 8888
```

To create the deployment from the above file

```
kubectl create -f deployment.yml

To check if the deployment is running
kubectl get deployment

To see if all 3 pod of nginx are running
kubectl get pod

Check the version of nginx
kubectl describe pods nginx-deployment | less

================================================================================
=
Namespace in kubernetes
==========================
Namespaces are used to create partitions in the Kubernetes cluster
Pods runnign in different namespaces cannot communicate with
each other

To create Namespaces
===========
vim namespace.yml
---
apiVersion: v1
  kind: Namespace
  metadata:
    name: test-ns
...

kubectl apply -f namespace.yaml

To see the list of namespace
================================
kubectl get namespace

Create a pod on that namespace
==================================
vim pod-definition4.yml

---
apiVersion: v1
kind: Pod
metadata:
 name: jdk-pod
 namespace: test-ns
 labels:
  author: intelliqit
spec:
 containers:
  - name: java
    image: openjdk:12
...

To see list of pods in a namespace
====================================
kubectl get pods -n test-ns

To delete a namespace
==========================
kubectl delete namespace test-ns
```

```
================================================================================
=
Day 24
================================================================================
=
Kompose
================
This is used to implement docker compose to create a multi
container architecture in Kubernetes

Implementing docker compose can be done using Kompose
docker compose + docker swarm = docker stack
docker compose + Kubernetes = Kompose

Setup
===========
1 Download Kompose
  curl -L
https://github.com/kubernetes/kompose/releases/download/v1.18.0/kompose-
linux-amd64 -o kompose

2 Give execute permissions
  chmod +x kompose

3 Move it to PATH
  sudo mv ./kompose /usr/local/bin/kompose

4 To check if the installion is successfull
  kompose version

Digital Ocean URL
========================
https://www.digitalocean.com/community/tutorials/how-to-migrate-a-docker-
compose-workflow-to-kubernetes

Create a docker compose file
vim docker-compose.yml
---
version: '3'

services:
 mydb:
  image: mysql:5
  environment:
   MYSQL_ROOT_PASSWORD: intelliq

 wordpress:
  image: wordpress
  ports:
   - 8080:80
  deploy:
   replicas: 3
...

To setup the above architecture in Kubernetes
kompose up

To create kubernetes definition files
kompose convert
```

To delete the above create architecture
kompose down

Note: Practice Kompsoe on katokoda.com

```
==========================================================================
---
apiVersion: v1
kind: Pod
metadata:
 name: redis-pod
 labels:
  author: intelliqit
spec:
 containers:
  - name: redis
    image: redis
    volumeMounts:
     - name: redis-volume
       mountPath: /data/redis
 volumes:
  - name: redis-volume
    emptyDir: {}
```

Create a pod from the above file
kubectl create -f volumes.yml

To check if the volume is mounted
kubectl exec -it redis-pod -- bash

Go to the redis folder and create some files
cd redis
cat > file
Store some data in this file

To kill the redis pod install procps
apt-get update
apt-get install -y procps

Identify the process id of redis
ps aux
kill 1

Check if the redis-pod is recreated
kubectl get pods
We will see the restart count changes for this pod

If we go into this pods interactive terminal
kubectl exec -it redis-pod -- bash

We will see the data but not the s/w's (procps) we installed
cd redis
ls

ps This will not work

```
==========================================================================
===
```

Day 25

```
================================================================================
===
================================================================
Service Object
=====================

This is used for network load balancing and port mapping
It uses 3 ports
1 target port:  Pod or container port
2 port:   Service port
3 hostPort:  Host machines port to make it accessable from external
network

Service objects are classified into 3 types
1 clusterIP: This is the default type of service object used in
  Kubernetes and it is used when we want the Pods in the cluster to
  communicate with each other and not with extrnal networks

2 nodePort: This is used if we want to access the pods from an extrnal
  network and it also performs network load balancing ie even if a pod
  is running on a specific salve we can access it from other slave in
  the cluster

3 LoadBalancer: This is similar to Nodeport and it is used for external
  connectivity of a Pod and also network load balancing and it also
assigns
  a public ip for all the slave combined together


================================================================================
====
Use Case
=================
Create a service defintion file for port mapping an nginx pod

vim pod-defintion1.yml
---
apiVersion: v1
kind: Pod
metadata:
 name: nginx-pod
 labels:
  author: intellqit
  type: proxy
spec:
 containers:
  - name: appserver
    image: nginx
============================================================
vim service1.yml
---
apiVersion: v1
kind: Service
metadata:
 name: nginx-service
spec:
 type: NodePort
 ports:
  - targetPort: 80
    port: 80
```

```
    nodePort: 30008
 selector:
  author: intellqit
  type: proxy

Create pods from the above pod definition file
kubectl create -f pod-definition1.yml
Create the service from the above service definition file
kubectl create -f service.yml
Now nginx can be accesed from any of the slave
kubectl get nodes -o wide
Take the external ip of any of the nodes:30008


UseCase
==================
Create a pod defintion file to start a ghost pod and also create a
service object of the type LoadBalancer

vim pod-defintion7.yml
---
apiVersion: v1
kind: Pod
metadata:
 name: ghost-pod
 labels:
  author: intelliqit
  type: CMS
spec:
 containers:
  - name: ghost
    image: ghost
...

vim service2.yml
---
apiVersion: v1
kind: Service
metadata:
 name: ghsot-service
 labels:
  author: intelliqit
spec:
 type: LoadBalancer
 ports:
  - targetPort: 2368
    port: 2368
 selector:
  type: CMS
  author: intelliqit

================================================================================
UseCase
================
Create a pod-definiton file for httpd pod and create a service object
of type cluster ip for it

vim pod-definition8.yml
---
apiVersion: v1
```

```
kind: Service
metadata:
 name: ghsot-service
 labels:
  author: intelliqit
spec:
 type: LoadBalancer
 ports:
  - targetPort: 2368
    port: 2368
 selector:
  type: CMS
  author: intelliqit
...

vim service3.yml
---
apiVersion: v1
kind: Service
metadata:
 name: httod-service
 labels:
  author: intelliqit
spec:
 ports:
  - targetPort: 80
    port: 80
 selector:
  author: intelliqit
  type: webserver
...

================================================================================
UseCase
==============
Create a deployment file of for tomcat and also create a servcie file
of type node port

vim deployment3.yml
---
apiVersion: apps/v1
kind: Deployment
metadata:
 name: tomcat-deployment
 labels:
  type: appserver
spec:
 replicas: 2
 selector:
  matchLabels:
   type: appserver
 template:
  metadata:
   name: tomcat-pod
   labels:
    type: appserver
  spec:
   containers:
    - name: tomcat
      image: tomee
```

```
...

vim service4.yml
---
apiVersion: v1
kind: Service
metadata:
 name: tomcat-service
 labels:
  author: intelliqit
spec:
 type: NodePort
 ports:
  - targetPort: 8080
    port: 8080
 selector:
  type: appserver
...

================================================================================
=


================================================================================
==
Day 26
================================================================================
==
================================================================================
Kubernetes Project
========================
This is a python based application which is used for accepting a vote
(voting app).This application accepts the vote and passes it to a
temporary db created using redis.From here the data is passed to a
worker application created using .net which anlysises the data and
stores them permananatly in a database created using postgres
From here the results can be seen on an application that is created
using nodejs and this is called as resulta-app

To do this we will create 5 pod definition files
and 4 service files,2 services of type cluster ip for redis and postgres
databases 2 services of type loadbalancer for python voting app and
nodejs result app

Pod Definition Files
================================
vim voting-app-pod.yml

---
apiVersion: v1
kind: Pod
metadata:
  name: voting-app-pod
  labels:
    name: voting-app-pod
    author: intelliqit
spec:
  containers:
    - name: voting-app
      image: dockersamples/examplevotingapp_vote
```

```
      ports:
        - containerPort: 80
...


vim result-app-pod.yml
---
apiVersion: v1
kind: Pod
metadata:
  name: result-app-pod
  labels:
    name: result-app-pod
    author: intelliqit
spec:
  containers:
    - name: result-app
      image: dockersamples/examplevotingapp_result
      ports:
        - containerPort: 80
...


vim worker-app-pod.yml

---
apiVersion: v1
kind: Pod
metadata:
  name: worker-app-pod
  labels:
    name: worker-app-pod
    author: intelliqit
spec:
  containers:
    - name: worker-app
      image: dockersamples/examplevotingapp_worker
...


vim redis-pod.yml
---
apiVersion: v1
kind: Pod
metadata:
  name: redis-pod
  labels:
    name: redis-pod
    author: intelliqit
spec:
  containers:
   - name: redis
     image: redis
     ports:
       - containerPort: 6379
...

vim postgres-pod.yml

---
```

```
apiVersion: v1
kind: Pod
metadata:
  name: postgres-pod
  labels:
    name: postgres-pod
    author: intelliqit
spec:
  containers:
    - name: postgres
      image: postgres
      ports:
        - containerPort: 5432
...


================================================================================
===
Service Defintion file
==============================
vim redis-service.yml
---
apiVersion: v1
kind: Service
metadata:
  name: redis-service
  labels:
    name: redis-service
    author: intelliqit
spec:
  ports:
    - port: 6379
      targetPort: 6379
  selector:
    name: redis-pod
    app: demo-voting-app
...

vim pod-service.yml
---
apiVersion: v1
kind: Service
metadata:
  name: postgres-service
  labels:
    name: postgres-service
    author: intelliqit
spec:
  ports:
    - port: 5432
      targetPort: 5432
  selector:
    name: postgres-pod
    app: demo-voting-app
...

Note: Since "type" is not specified in the "spec" section they  will
be created as clusterIP
```

```
vim voting-app-service.yml
---
apiVersion: v1
kind: Service
metadata:
  name: voting-app-service
  labels:
    name: voting-app-service
    author: intelliqit
spec:
  type: LoadBalancer
  ports:
    - port: 80
      targetPort: 80
  selector:
    name: voting-app-pod
    app: demo-voting-app
...

vim result-app-service.yml
---
apiVersion: v1
kind: Service
metadata:
  name: result-app-service
  labels:
    name: result-app-service
    author: intelliqit
spec:
  type: LoadBalancer
  ports:
    - port: 80
      targetPort: 80
  selector:
    name: result-app-pod
    app: demo-voting-app
...


The above 2 service objects are created as LoadBalancer type ie
they can perfrom network load balancing where we can access the pod
from any slave and also a single public ip will be assigned for all
the salves

================================================================================
Day 30
================================================================================
To deploy the above project using docker stack
vim voting-app.yml
---
version: '3'

services:
 voting-app:
  image: dockersamples/examplevotingapp_vote
  ports:
   - 6060:80
```

```
 redis:
  image: redis
  ports:
   - 6379:6379

 worker-app:
  image: dockersamples/examplevotingapp_worker

 postgres:
  image: postgres
  environment:
   POSTGRES_PASSWORD: intelliqit
  ports:
   - 5432:5432

 result-app:
  image: dockersamples/examplevotingapp_result
  ports:
   - 7070:80


To deploy the above services
docker stack deploy -c voting-app.yml my-voting-app


To see the list of nodes where the stack services are running
docker stack ps my-voting-app

================================================================================
===
To deploy the above file in kubernetes using kompose
================================================================================
=====
Kompose
================
This is used to implement docker compose to create a multi
container architecture in Kubernetes

Implementing docker compose can be done using Kompose
docker compose + docker swarm = docker stack
docker compose + Kubernetes = Kompose

Setup
===========
1 Download Kompose
  curl -L
https://github.com/kubernetes/kompose/releases/download/v1.18.0/kompose-
linux-amd64 -o kompose

2 Give execute permissions
  chmod +x kompose

3 Move it to PATH
  sudo mv ./kompose /usr/local/bin/kompose

4 To check if the installion is successfull
  kompose version

Digital Ocean URL
========================
```

https://www.digitalocean.com/community/tutorials/how-to-migrate-a-docker-compose-workflow-to-kubernetes

==
Day 27
==
Stateful set
==========

```
---
apiVersion: v1
kind: Service
metadata:
 name: nginx
 labels:
  app: nginx
spec:
 ports:
  - port: 80
    name: web
 clusterIP: None
 selector:
  app: nginx
---
apiVersion: apps/v1
kind: StatefulSet
metadata:
 name: web
spec:
 serviceName: "nginx"
 replicas: 3
 selector:
  matchLabels:
   app: nginx
 template:
  metadata:
   labels:
    app: nginx
  spec:
   containers:
    - name: nginx
      image: nginx
      ports:
       - containerPort: 80
         name: web
      volumeMounts:
       - name: www
         mountPath: /usr/share/nginx/html
 volumeClaimTemplates:
  - metadata:
     name: www
    spec:
     accessModes: [ "ReadWriteOnce" ]
     resources:
      requests:
       storage: 1Gi
```

```
================================================================================
=====
Node affinity
================================
kubectl get nodes --show-labels

kubectl label nodes <your-node-name> disktype=ssd

kubectl label nodes gke-cluster-1-default-pool-3cde7c4a-hl74 disktype=ssd
================================================================
apiVersion: v1
kind: Pod
metadata:
  name: nginx
spec:
  affinity:
    nodeAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
        nodeSelectorTerms:
        - matchExpressions:
          - key: disktype
            operator: In
            values:
            - ssd
  containers:
  - name: nginx
    image: nginx

================================================================
Taints and toleration
=======================
Taints and Tolerations
Node affinity, is a property of Pods that attracts them to a set of nodes
(either as a preference or a hard requirement). Taints are the opposite -
- they allow a node to repel a set of pods.

Tolerations are applied to pods, and allow (but do not require) the pods
to schedule onto nodes with matching taints.

Taints and tolerations work together to ensure that pods are not
scheduled onto inappropriate nodes. One or more taints are applied to a
node; this marks that the node should not accept any pods that do not
tolerate the taints.

To create a taint for a node
kubectl taint nodes node1 node=intelliqit:NoSchedule

To delete the tain
kubectl taint nodes node1 node=intelliqit:NoSchedule-

Pod defintion file to use the above taint
apiVersion: v1
kind: Pod
metadata:
  name: nginx-pod
  labels:
    author: intelliqit
spec:
  containers:
  - name: mygninx
```

```
    image: nginx
  tolerations:
  - key: "node"
    operator: "Equal"
    value: "intelliqit"
    effect: "NoSchedule"
```

==========

Helm and Kubernetes

Helm is a package manager. Package managers automate the process of installing, configuring, upgrading, and removing computer programs

Helm has two elements, a client (Helm) and a server (Tiller). The server element runs inside a Kubernetes cluster and manages the installation of charts.