Virtualization

==========

This is the process of running multiple OS's parallelly on a single piece 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.

Containerization

==========

Here we have bare metal on top of which we install the host OS
and on the host 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 these containers

Docker achieve what is commonly called as "process isolation"
i.e. 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 maintain various applications

Docker can be used at 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

==========

Install docker on Linux

==========

1 Create an Ubuntu instance on AWS
2 Connect to it using git bash
3 Execute 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 the CLI of docker where the user can execute the
docker commands, The docker client accepts these commands and passes them
to a background process called "docker deamon"

Docker deamon: This process accepts the commands coming from the docker client
and routes them to work on docker images or containers or the docker registry

Docker registry: This is the cloud site of docker where docker images are
stored. This is of two types
1 Public Registry(hub.docker.com)
2 Private Registry (Setup on one of our local servers)

==========
Important docker commands
==========
Working on docker images
==========
1 To pull a docker image
 docker pull image_name

2 To search for a docker images
 docker search image_name

3 To upload an image into docker hub
 docker push image_name

4 To see the list of images that are downloaded
 docker images
 or

docker image ls

5 To get detailed info about a docker image
docker image inspect image_name/image_id

6 To delete a docker image that is not linked to any container
docker rmi image_name/image_id

7 To delete a image that is linked to a container
docker rmi -f image_name/image_id

8 To save the docker image as a tar file
docker save image_name

9 To untar this tar file and get image
docker load tarfile_name

10 To delete all image
docker system prune -af

==========

11 To create a docker image from a dockerfile
docker build -t image_name .

12 To create an image from a customised container
docker commit container_id/container_name image_name

Working on docker containers
==========

13 To see the list of running containers
docker container ls

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

15 To start a container
docker start container_id/container_name

16 To stop a container
docker stop container_id/container_name

17 To restart a container
docker restart container_id/container_name
To restart after 10 seconds
docker restart -t 10 container_id/container_name

18 To delete a stopped container
docker rm container_id/container_name

19 To delete a running container

docker rm -f container_id/container_name

20 To stop all running container
docker stop $(docker ps -aq)

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

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

23 To get detailed info about a container
docker inspect container_id/container_name

24 To see the logs genearated by a container
docker logs container_id/container_name

25 To create a docker container
docker run image_name/image_id
run command options

--name: USed to give a name to the container
--restart: Used to keep the container in runnign condition
-d: Used to run the container in detached mode in background
-it: Used to open interactive terminal in the container
-e: Used to pass environment varibales to the container
-v : Used to attach an external device or folder as a volume
--volumes-from: Used to share volume between multiple containers
-p : Used for port mapping.It will link the container port with
host port.Eg: -p 8080:80 Here 8080 is host port(external port)
and 80 is container port(internal port)
-P: Used for automatic port mapping where the container port is
mapped with some host port that is greate than 30000
--link : Used to create a link between multiple containers to create a
microservices architecture.
--network: Used to start a container on a specific network
-rm : Used to delete a container on exit
-m: Used to specify the upper limit on the amount of memeory that
a container can use
-c: Used to specify the upper limit on the amout of cpu a container can use
-ip: Used to asssign an ip to the container

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

27 To run any process in a container from outside the container
docker exec -it container_id/container_name process_name
Eg: To run the bash process in a container
docker exec -it container_id/container_name bash

28 To come out of a container without exit

ctrl+p,ctrl+q

29 To go back into a container from where the interactive terminal is running
docker attach container_id/container_name

30 To see the processes runnign in a container
docker container container_id/container_name top

Working on docker networks
==========
31 To see the list of docker networks
docker network ls

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

33 To get detailed info about a network
docker network insepct network_name/network_id

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

35 To connect a running container to a network
docker netowork connect network_name/network_id container_name/container_id

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

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

38 To create a docker volume
docker volume create volume_name

39 To get detailed info about a volume
docker volume inspect volume_name/volume_id

40 To delete a volume
docker volume rm volume_name/volume_id
==========
Day 3
==========
UseCase 1
==========
Create an nginx contaienr in detached mode and name it webserver
Also perfrom port mapping

docker run --name webserver -p 8888:80 -d nginx

To check if the nginx container is running
docker container ls

To access the nginx container from the leve of browser
public_ip_of_dockerhost:8888

==========
UseCase 2
==========
Start a jenkins container in detached mode and also perfrom port mapping
docker run --name myjenkins -d -p 9999:8080 jenkins/jenkins

To see the ports used by the above container
docker port appserver

To access jenkins from browser
public_ip_of_docker_host:poert_no_from_above_command
==========
UseCase 3
==========
Start tomcat as a container and perfrom automatic port mapping
docker run --name appserver -d -P tomee

To see the ports used by the above container
docker port appserver

To access the httpd from borwser
piblic_ip_of_dockerhost:9090

==========
Day 4
==========
UseCase
==========
Start centos as a container and launch interactive terminal in it
docker run --name mycentos -it centos

To come out of the centos container
exit

==========
UseCase
==========
Create a mysql container and login as root user and create some sql tables

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

2 To check if the mysql container is running
docker container ls

3 To go into the bash shell of the container
docker exec -it db bash

4 To login into the database
mysql -u root -p
Password: intelliqit

5 To see the list of databases
show databases;

6 To move into any of the above database
use databasename;
Eg: use sys;

7 To create emp and dept tables here
Open
https://justinsomnia.org/2009/04/the-emp-and-dept-tables-for-mysql/
Copy script from emp and dept tables creation
Paste in the mysql container

8 To see the data of the tables
select * from emp;
select * from dept;

==========
To setup a multi container architecture
==========
1 Using --link run command option (depricated)
2 Docker compose
3 Docker Networkings

4 Python Scripting
5 Ansible Playbooks

==========

UseCase
Create 2 busybooks containers c1 and c2 and link them

1 Create a busybox contaienr and name it c1
docker run --name c1 -it busybox

2 To come out of the c1 contaienr without exit
ctrl+p,ctrl+q

3 Create another busybox container c2 and link it with c1 container
docker run --name c2 -it --link c1:mybusybox busybox

4 Check if c2 is pinging to c1

ping c1

UseCase
==========
Setup wordpress and link it with mysql container

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

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

3 To check if wordpress and mysql containers are running
docker container ls

4 To access wordpress from a browser
public_ip_dockerhost:8080

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

==========
UseCase
==========
Setup CI-CD environment where a Jenkins container is linked with
2 tomcat containers for QAserver and PRodserver

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

2 To access jenkins from browser
public_ip_dockerhost:5050

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

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

5 Check if all 3 containers are running
docker container ls

==========
Day 5
==========
UseCase
==========
Create a postgress container and link with an admienr container to

Note: adminer is a client application of databases.

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

2 Create an adminer application and link with the postgres db
```
docker run --name myadminer -d -p 8888:8080 --link mydb:postgres adminer
```

3 To access postgres from browser
```
public_ip_of_dockerhost:8888
```

==========

Setup LAMP architecture

1 Create mysql container
```
docker run --name mydb -d -e MYSQL_ROOT_PASSWORD=intelliqit mysql
```

2 Create an apache container and link with mysql container
```
docker run --name apache -d -p 9999:80 --link mydb:mysql httpd
```

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

4 To check if php container is linked with apache and mysql
```
docker inspect php
```

==========

UseCase

==========

Create a testing environment where a selenium hub container is linked
with 2 node containers one with chrome and other with firefox installed

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

2 Create a container with chrome installed on it
```
docker run --name chrome -d -p 5901:5900 --link hub:selenium
                            selenium/node-chrome-debug
```

3 Create another container with firefox installed on it
```
docker run --name firefox -d -p 5902:5900 --link hub:selenium
                          selenium/node-firefox-debug
```

4 The above 2 containers are GUI ubuntu containers and we can
access their GUI using VNC viewer

a) Install VNC viewer from
https://www.realvnc.com/en/connect/download/viewer/

b) Open vnc viewer--->Public ip of docker host:5901 or 5902

Click on Continue--->Enter password: secret

==========
Day 6
==========
Docker compose
==========
The disadvantage of "link" option is it is depricated and
the same individual command have to be given multiple times
to setup similar architectures.
To avaoid this we can use docker compose
Docker compose uses yml files to setup the multu container
architecture and these files can be resused any number of time
==========
USeCase
==========
Create a docker compose file to setup a mysql and wordpress
container and link them

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

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

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

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

To stop all the container of the docker compose file
docker-compose stop

To start the container
docker-compose start

To stop and delete
docker-compose down
==========
UseCase
==========

Create a docker compsoe file to setup the CI-CD environment
where a jenkins container is linked with 2 tomee containers
one for qaserver and other for prodserver

vim docker-compose.yml

```
---
version: '3.8'

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

 qaserver:
  image: tomee
  ports:
   - 6060:8080
  links:
   - myjenkins:jenkins

 prodserver:
  image: tomee
  ports:
   - 7070:8080
  links:
   - myjenkins:jenkins
...
```

==========

Day 7

==========

UseCase

==========

Create a docker compose file to setup an adminer app and postgres db

vim docker-compose.yml

```
---
version: '3.8'
services:
  mydb:
    image: postgres
    environment:
      POSTGRES_PASSWORD: intelliqit

  myadminer:
    image: adminer
    ports:
     - 8888:80
    links:
      - mydb:postgres
...
```

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

To stop and delete
docker compose down

UseCase
==========
Create a docker compose file to setup the LAMP architecture
vim lamp.yml
---
version: '3.8'

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

 apache:
  image: httpd
  ports:
   - 8989:80
  links:
   - mydb:mysql

 php:
  image: php:7.2-apache
  links:
   - mydb:mysql
   - apache:httpd
...

To create containers from the above file
docker-compose -f lamp.yml up -d

To delete the containers
docker-compose -f lamp.yml down

==========
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 8
==========
==========
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

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

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

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 apt-get install -y tree
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 .

==========

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 wget
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
```

==========
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 10

==========

Create a dockerfile from jenkins base image and make the default user as root

1 vim dockerfile
FROM jenkins/jenkins
MAINTAINER intelliqit
USER root

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

3 Create a container from the above image
docker run --name j1 -d -P myjenkins

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

==========

Create a docekerfile from nginx base image and expose 90 port

1 vim dockerfile
FROM nginx
MAINTAIENR intelliqit
EXPOSE 90

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

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

4 To check the port
docker port n1

==========

Day 11

==========

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
```

==========

==========

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"

==========
Day 12
==========
Docker Networking
==========
Docker supports 4 types of networks
1 Bridge
2 Host
3 Null
4 Overlay
==========
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)

==========
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 centos image and upload into the public registry

1 Signup into hub.docker.com

2 Create a customsied centos image
a) Create a centos container and install git init
docker run --name c1 -it centos
yum -y update
yum -y install git
exit

b) Save this container as an image
docker commit c1 intelliqit/mycentos

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

4 Push the customised image
docker push intelliqit/mycentos

==========
Day 12
==========
Note: To create network with a specific subnet range
docker network create --driver bridge --subnet=192.168.2.0/24 intelliqit

Docker compose by deafult creates its own customised bridge network and creates containers on the netowork

vim docker-compose.yml

```
---
version: '3.8'

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

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

To setup the containers
docker compose up -d

To see the list of containers
docker container ls

To see the list of networks
docker network ls

To above 2 containers will be running on a new bridge network that is created by docker compose

To delete the containers
docker compose down
This will not only delete the containers it will also delete the networks
that got created.
==========
Day 12
==========

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
```

==========

Create a dockerfile and use it directly in docker-compsoe

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

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

```
services:
 jenkins:
  build: .
  ports:
   - 7070:8080


 mytomcat:
  image: tomee
  ports:
   - 6060:8080

...
```

To start the services
docker-compose up

==========

Docker compose file to create 2 networks and run containers on different network
vim docker-compose.yml

```
---
version: '3.8'

services:
 mydb:
  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: {}
...
```

==========
Day 13
==========
Docker compose file to create 2 containers and also create 2 volumes for both the containers

```
---
version: '3.8'

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

 wordpress:
  image: wordpress
  ports:
   - 9999:80
  volumes:
   wordpress:/var/www/html


volumes:
  mydb:
  wordpress
```

To start the service
docker-compose up -d

To see the list of volumes
docker volume ls

==========
Day 14
==========
Docker Swarm
==========
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

==========

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 nginx 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 15
==========
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

==========
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 - Fault Tolerance
1	-	0
2	-	0
3	-	1
4	-	1
5	-	2
6	-	2
7	-	3
8	-	3

==========

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

==========
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

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

==========

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

...

==========
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 PersistantVolume: Used to specify the section of storage that should be used for volumes

9 PersistantVolumeClaims: Used to reserver a certain amout of storage for a pod from the
persistant volume.

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

11 HorrizontalPodAutScaller: Used for auto scalling of pods depending on the load

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

==========

Kubernetes on AWS using Kops

1. Launch Linux EC2 instance in AWS (Kubernetes Client)
2. Create and attach IAM role to EC2 Instance.

Kops need permissions to access

- S3
- EC2
- VPC
- Route53
- Autoscaling
- etc..

3. Install Kops on EC2

```
curl -LO https://github.com/kubernetes/kops/releases/download/$(curl -s https://api.github.com/repos/kubernetes/kops/releases/latest | grep tag_name | cut -d '"' -f 4)/kops-linux-amd64
chmod +x kops-linux-amd64
sudo mv kops-linux-amd64 /usr/local/bin/kops
```

4. Install kubectl

```
curl -LO https://storage.googleapis.com/kubernetes-release/release/$(curl -s https://storage.googleapis.com/kubernetes-release/release/stable.txt)/bin/linux/amd64/kubectl
chmod +x ./kubectl
sudo mv ./kubectl /usr/local/bin/kubectl
```

5. Create S3 bucket in AWS

S3 bucket is used by kubernetes to persist cluster state, lets create s3 bucket using aws cli Note: Make sure you choose bucket name that is uniqe accross all aws accounts

```
aws s3 mb s3://project.in.k8s --region us-west-2
```

6. Create private hosted zone in AWS Route53

Head over to aws Route53 and create hostedzone
Choose name for example (sai.in)
Choose type as privated hosted zone for VPC
Select default vpc in the region you are setting up your cluster
Hit create

7 Configure environment variables.

Open .bashrc file

```
vi ~/.bashrc
```

Add following content into .bashrc, you can choose any arbitary name for cluster and make sure buck name matches the one you created in previous step.

```
export KOPS_CLUSTER_NAME=project.in
export KOPS_STATE_STORE=s3://project.in.k8s
```

Then running command to reflect variables added to .bashrc

```
source ~/.bashrc
```

8. Create ssh key pair

This keypair is used for ssh into kubernetes cluster

```
ssh-keygen
```

9. Create a Kubernetes cluster definition.

```
kops create cluster \
--state=${KOPS_STATE_STORE} \
--node-count=2 \
--master-size=t3.medium \
--node-size=t3.medium \
--zones=us-west-2a \
--name=${KOPS_CLUSTER_NAME} \
--dns private \
--master-count 1
```

10. Create kubernetes cluster

```
kops update cluster --yes --admin
```

Above command may take some time to create the required infrastructure resources on AWS. Execute the validate command to check its status and wait until the cluster becomes ready

```
kops validate cluster
```

For the above above command, you might see validation failed error initially when you create cluster and it is expected behaviour, you have to wait for some more time and check again.

11. To connect to the master

```
ssh admin@api.javahome.in
```

Destroy the kubernetes cluster

```
kops delete cluster  --yes
```

Update Nodes and Master in the cluster

We can change numner of nodes and number of masters using following commands

- kops edit ig nodes change minSize and maxSize to 0
- kops get ig- to get master node name
- kops edit ig - change min and max size to 0
- kops update cluster --yes

==========

Kubernetes setup using Kubeadmm

==========

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-1.19.1 kubelet-1.19.1 kubectl-1.19.1
```

==========
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/ec2-user/.kube
cp /etc/kubernetes/admin.conf /home/ec2-user/.kube/config
chown -R ec2-user:ec2-user /home/ec2-user/.kube

==========
Deploy calico network
kubectl apply -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

==========
==========
To see the list of nodes in the Kubernetes cluster
  kubectl get nodes

2 To get info about the nodes along with ipaddress and docker version etc
  kubectl get nodes -o wide

3 To get detailed info about the nodes
  kubectl describe nodes node_name

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

To see the list of pods
kubectl get pods

To get info about the pods along with ipaddress
kubectl get pods -o wide

To get detailed info about the pods
kubeclt describe pods webserver

==========
Create a mysql pod and also pass the necessary environment variables
kubectl run --image mysql:5 db --env MYSQL_ROOT_PASSWORD=intelliqit

Check if the pod is running
kubectl get pods

To delete the mysql pod
kubectl delete pods db

==========
Kubernetes Definition file
==========
Kubernetes performs container orchestration uisng certain definition
file.These files are created using yml and they have 4 top level
fields

apiVersion:
kind:
metadata:
spec:

apiVersion: Every kubernetes object uses a specific Kubernetes code
library that is called apiVersion.Only once this code library is imported
we can start working on specific objects

kind: This represents the type of Kubernetes object that we want to us
    eg: Pod,Replicaset,Service etc

metadata: Here we give a name to the Kubernetes object and also some
        labels.These labels can be used later for performing group
        activites

spec: This is where we store info about the exact docker image,container name
    environment varibales,port mapping etc

Kind   apiVersion
==========
Pod   v1
Service   v1
NameSpace   v1
Secrets   v1
ReplicationController v1
PersistantVolume v1

PersistantVolumeClaim v1
HorrizontalPodAutoScaller v1
ReplicaSet apps/v1
Deployment apps/v1
DaemoSet apps/v1

==========

UseCase-1
Create a pod definition file to start an nginx in a pod
name the pod as nginx-pod,name the container as webserver

vim pod-defintion1.yml

```
---
apiVersion: v1
kind: Pod
metadata:
 name: nginx-pod
 labels:
  author: intellqit
  type: reverse-proxy
spec:
 containers:
  - name: appserver
    image: nginx
...
```

To create a pod from the above file
kubectl create -f pod-defintion1.yml

To see the list of pods
kubectl get pods

To see the pods along with the ipaddress and name of the slave where it is running
kubectl get pods -o wide

To delete the pods created from the above file
kubectl delete -f pod-definition1.yml

==========
Create a pod defintion file to start a postgres container
Name of the container should be mydb,pass the necssary environment
variables,this container should run in a pod called postgres-pod

and give the labels as author=intelliqit and type=database

vim pod-definition2.yml

```
---
apiVersion: v1
kind: Pod
metadata:
 name: postgres-pod
 labels:
  author: intelliqit
  type: database
spec:
 containers:
  - name: mydb
    image: postgres
    env:
     - name: POSTGRES_PASSWORD
       value: myintelliqit
     - name: POSTGRES_USER
       value: myuser
     - name: POSTGRES_DB
       value: mydb
```

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

To delete the pods
kubectl delete -f pod-definition2.yml

==========

UseCase 3
Create a pod defintion file to start a jenkins container in a pod
called jenkins-pod,also perform port mapping to access the jenkins
from a browser

vim pod-definition3.yml

```
---
apiVersion: v1
kind: Pod
metadata:
 name: jenkins-pod
 labels:
  author: intelliqit
  type: ci-cd
spec:
 containers:
  - name: myjenkins
    image: jenkins
```

```
    ports:
     - containerPort: 8080
       hostPort: 8080
...
```

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

To see the list of pods along with nodes where they are running
kubectl get nodes -o wide

To get the external ip of the node
kubectl get node -o wide

To access then jenkins from browser
external_ip_of_slavenode:8080

==========
Create a pod definition file to setup tomcat

```
---
apiVersion: v1
kind: Pod
metadata:
  name: tomcat-pod
  labels:
    author: intelliqit
    type: appserver
spec:
  containers:
    - name: mytomcat
      image: tomee
      ports:
        - containerPort: 8080
          hostPort: 9090
```

==========
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: tomee
      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 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
      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

==========

Create a mysql deployment
vim deployment2.yml

```
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: mysql-deployment
  labels:
    type: db
    author: intelliqit
spec:
  replicas: 3
  selector:
```

```
  matchLabels:
    type: db
 template:
  metadata:
    name: mysql-pod
    labels:
      type: db
  spec:
    containers:
      - name: mydb
        image: mysql
        ports:
          - containerPort: 3306
            hostPort: 8080
        env:
          - name: MYSQL_ROOT_PASSWORD
            value: intelliqit
```

```
================================================================================
Kubernetes setup using Kubeadm
========================================
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-1.19.1 kubelet-1.19.1 kubectl-1.19.1

================================================================================
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/ec2-user/.kube
cp /etc/kubernetes/admin.conf /home/ec2-user/.kube/config
chown -R ec2-user:ec2-user /home/ec2-user/.kube

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

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

==========--

==========
==========
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: reverse-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: reverse-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

==========

==========

Create a service object of the type LoadBalancer for a tomcat pods
vim servcie2.yml

```
---
apiVersion: v1
kind: Service
metadata:
 name: tomcat-service
spec:
 type: LoadBalancer
 ports:
  - targetPort: 80
    port: 80

 selector:
```

```
  author: intellqit
  type: appserver


vim pod0defintion5.yml
vim pod-definition2.yml
---
apiVersion: v1
kind: Pod
metadata:
 name: tomcat-pod
 labels:
  type: appserver
  author: intelliqit
spec:
 containers:
  - name: tomcat
    image: tomee

...
```

==========

Create a service object of the type load balancer for postgres pod

```
vim service3.yml

---
apiVersion: v1
kind: Service
metadata:
 name: postgres-service
spec:
 type: ClusterIp
 ports:
  - targetPort: 5432
    port: 5432

 selector:
  author: intellqit
  type: db

vim pod-defintion6.yml
apiVersion: v1
kind: Pod
metadata:
 name: mysql-pod
 labels:
  type: db
  author: intelliqit
spec:
 containers:
  - name: mydb
```

```
    image: mysql
    env:
     name: MYSQL_ROOT_PASSWORD
     value: intelliqit
```

==========

Node affinity:This is a feature of Kubernetes whcih attracts pods to a specific slave

==========

To see the list of a labels

```
kubectl get nodes --show-labels
```

To label a slave

```
kubectl label nodes <your-node-name> key=value
```

```
kubectl label nodes gke-cluster-1-default-pool-3cde7c4a-hl74 slave1=intelliqit1
```

==========

Pod Defintion file to implement node affinity

```
apiVersion: v1
kind: Pod
metadata:
  name: nginx
spec:
  affinity:
    nodeAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
        nodeSelectorTerms:
        - matchExpressions:
          - key: slave1
            operator: In
            values:
            - intelliqit1
  containers:
  - name: nginx
    image: nginx
```

==========

Deployment file to implement node affintiy

```
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  labels:
    type: proxy
spec:
  replicas: 2
  selector:
    matchLabels:
      type: proxy
```

```
 template:
   metadata:
     name: nginx-pod
     labels:
       type: proxy
   spec:
     containers:
       - name: mynginx
         image: nginx
     affinity:
       nodeAffinity:
         requiredDuringSchedulingIgnoredDuringExecution:
           nodeSelectorTerms:
             - matchExpressions:
                 - key: slave1
                   operator: In
                   values:
                     - intelliqit1
```

==========

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-
```

Deployment defintion file to use the above taint

```
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: httpd-deployment
  labels:
    type: webserver
spec:
  replicas: 3
  selector:
    matchLabels:
```

```
    type: webserver
  template:
    metadata:
      name: httpd-pod
      labels:
        type: webserver
    spec:
      containers:
        - name: myhtppd
          image: httpd
      tolerations:
        - key: slave3
          operator: Equal
          value: intelliqit3
          effect: NoSchedule
...
```

==========

DaemonSets:These are used to run a single pod on each and every slave,The no salve count will become the desired count of the Daemonsets

==========

```
---
apiVersion: apps/v1
kind: DaemonSet
metadata:
  name: ghost-daemon
  labels:
    type: cms
spec:
  selector:
    matchLabels:
      type: cms
  template:
    metadata:
      name: ghost-pod
      labels:
        type: cms
    spec:
      containers:
        - name: ghost
          image: ghost
...
```

==========

Secrets

==========

This is used to send encrypted data to the definiton files

Generally passwords for Databases can be encrypted using this

Create a secret file to store the mysql password

vim secret.yml

```
---
apiVersion: v1
```

```
kind: Secret
metadata:
 name: mysql-pass
type: Opaque
stringData:
 password: intelliqit
 username: sai
...
```

To deploy the secret
kubectl create -f secret.yml

Create a pod defintion file to start a mysql pod and pass the environment varible using the above secret
vim pod-defitintion5.yml

```
---
apiVersion: v1
kind: Pod
metadata:
 name: mysql-pod
 labels:
  author: intelliqit
  type: db
spec:
 containers:
  - name: mydb
    image: mysql:5
    env:
     - name: MYSQL_ROOT_PASSWORD
       valueFrom:
        secretKeyRef:
         name: mysql-pass
         key: password
...
```

To create pods from above file
kubect create -f pod-defintion5.yml

==========

Create a secret definition file for postgres secret

```
---
apiVersion: v1
kind: Secret
metadata:
  name: postgres-secret
type: Opaque
stringData:
  password: intelliqit
  username: myuser
  dbname: mydb
```

Create postgres deployment and use the above secret

```
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: postgres-deployment
  labels:
    app: db
spec:
  replicas: 2
  selector:
    matchLabels:
      app: db
  template:
    metadata:
      name: postgres-pod
      labels:
        app: db
    spec:
      containers:
        - name: mydb
          image: postgres
          env:
            - name: POSTGRES_PASSWORD
              valueFrom:
                secretKeyRef:
                  name: postgres-secret
                  key: password
            - name: POSTGRES_USER
              valueFrom:
                secretKeyRef:
                  name: postgres-secret
                  key: username
            - name: POSTGRES_DB
              valueFrom:
                secretKeyRef:
                  name: postgres-secret
                  key: dbname
```

==========

Volumes

==========

```
---
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
==========

Persistent volume is the storage that is used by Kubernetes for Volumes
Persistent volume claim is the amount of storage from the persistent volume that will be allocatted to a Pod.It is alos a PVC that is attached to a Pod

vim pv.yml
--
```
apiVersion: v1
kind: PersistentVolume
metadata:
```

```
  name: my-pv
  labels:
    type: local
spec:
  storageClassName: manual
  capacity:
    storage: 4Gi
  accessModes:
    - ReadWriteOnce
  hostPath:
    path: /mnt/data
...
```

To create the persistant volume
kubect apply -f pv.yml

To see the list of pv
kubectl get pv

Create a persistantvolumeclaim definition file

vim pvc.yml

```
---
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: my-pvc
  labels:
    author: intelliqit
spec:
  storageClassName: manual
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi
```

To create the persistant volume claim
kubectl apply -f pvc.yml

To see the list of pvc
kubectl get pvc

Create a pod defintion file to use the above pvc
vim pod-volumes.yml

```
---
apiVersion: v1
kind: Pod
metadata:
  name: nginx-pod
  labels:
```

```
    author: intelliqit
    type: proxy
spec:
  containers:
    - name: mynginx
      image: nginx
      volumeMounts:
        - mountPath: /usr/share/nginx/html
          name: my-volume
  volumes:
    - name: my-volume
      persistentVolumeClaim:
        claimName: my-pvc
```

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

==========
Statefulsets
==========

```
apiVersion: v1
kind: Service
metadata:
  name: mysql
  labels:
    app: mysql
spec:
  clusterIP: None
  selector:
    app: mysql
  ports:
    - name: tcp
      protocol: TCP
      port: 3306
---
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: mysql
spec:
  replicas: 1
  serviceName: mysql
  selector:
    matchLabels:
      app: mysql
  template:
    metadata:
      labels:
        app: mysql
    spec:
```

```
    volumes:
      - name: task-pv-storage
        persistentVolumeClaim:
          claimName: task-pv-claim
    containers:
      - name: mysql
        image: mysql:5.6
        ports:
          - name: tpc
            protocol: TCP
            containerPort: 3306
        env:
          - name: MYSQL_ROOT_PASSWORD
            value: intelliqit

        volumeMounts:
          - name: task-pv-storage
            mountPath: /var/lib/mysql
```

==========

Helm Chart is a very feature-rich framework when you are working with complex Kubernetes cluster and deployment. Helm chart provides a very convenient way to pass values.yaml and use it inside your Helm Chart

Create your first Helm Chart
We are going to create our first helloworld Helm Chart using the following command

helm create mynginx

tree mynginx

Update the service.type from ClusterIP to NodePort inside the values.yml

To install the chart
---
helm install <FIRST_ARGUMENT_RELEASE_NAME> <SECOND_ARGUMENT_CHART_NAME>

helm install nginx mynginx

Verify the helm install command
---
helm list -a

Get kubernetes Service details and port
---
kubectl get service

==========

How to ADD upstream Helm chart repository

---
helm repo add <REPOSITORY_NAME> <REPOSITORY_URL>

To add any chart repository you should know the name and repository url.
---
helm repo add bitnami https://charts.bitnami.com/bitnami

Verify the repository
---
helm search repo bitnami

To see the list of repositories added
---
helm repo list

Updating the helm repo

Lets see how you can update your helm repositories. (The update command is necessary if haven't updated your Helm chart repository in a while, so might miss some recent changes)

Here is the command to update Helm repository

helm repo update

Removong a repository
---
helm repo remove bitnami

==========
Demo
==========
In this tutorial, we are going to install WordPress with MariaDB using the Helm Chart on Kubernetes cluster. With this installation, we are going to see - How we can upgrade as well as rollback the Helm Chart release of WordPress. This complete setup inherited the benefits of the Kubernetes .i.e. scalability and availability.

Since we are installing WordPress, so we need to have a database running behind the WordPress application. From the database standpoint, we are going to use MariaDB. Helm chart ships all these components in a single package, so that we need not worry about installing each component separately.

To search for all wordpress relates repositories
helm search hub wordpress

If the output of the above command is too large we can use
helm search hub wordpress --max-col-width=0

Ensure that the binami is installed
---
helm repo add bitnami https://charts.bitnami.com/bitnami
heml repo list

Readme.md
==========
This Readme.md contains the installation instructions and it can be viewed using the following command

helm show readme bitnami/wordpress --version 10.0.3

To update the username and password
vim wordpress-values.yml

wordpressUsername: admin
wordpressPassword: admin
wordpressEmail: selenium.saikrishna@gmail.com
wordpressFirstName: Sai
wordpressLastName: Krishna
wordpressBlogName: mywordpress.com
service:
  type: LoadBalancer

Create a new namespace
kubectl create namespace nswordpress

Versify the namesapce
kubectl get namespace

Run the below command to install wordpess in the namepsace
helm install wordpress bitnami/wordpress --values=wordpress-values.yaml --namespace nswordpress --version 10.0.3

To see the resources running in a specific namespace
watch -x kubectl get all --namespace nswordpress

To remove
kubect uninstall wordpress

Converting k8 defintion files to helm
==========
Objective 1 : - At first we are going to create simple Kubernetes deployment(k8s-deployment.yaml)` and in that deployment we are going to deploy a microservice application.

Objective 2 : - Secondly we are going to `create service(k8s-service.yaml) for exposing the deployment as a service on NodePort.

Objective 3 : - Here we are going to convert Kubernetes deployment(k8s-deployment.yaml) and create service(k8s -service.yaml) into a Helm Chart YAMls.

Step 1
==========
Create deployment.yml file and also a service file of NodePort type

Step 2
==========
helm create demochart
tree demochart

Step 3
==========
Go into the demochart folder
cd demochart

The first YAML which we are converting is chart.yaml but it is optional and does not require any change but it would be nice to update some value with regards to your project name.

vim chart.yml
We can just edit the name (not mandatory)

In templates folder edit deployment.yml
vim deployment.yml
Edit the container port

cd ..

In values.yml
Edit the type: from clusterip to NodePort
port: 8080
image:
 tag: "latest"

==========
Come out of the demochart folder

To install the above chart
helm install mytomcat demochart

To see the components
kubectl get all

To delete
helm uninstall mytomcat
==========
Install prometheus and grafana
==========

Use older version of kubernetes (1.19)

helm repo add prometheus-community https://prometheus-community.github.io/helm-charts
helm repo add stable https://charts.helm.sh/stable
helm repo update

helm install prometheus prometheus-community/kube-prometheus-stack

grafana by default runs on clusterip to make to accessable externally change to nodeport
kubectl patch svc prometheus-grafana -p '{"spec": {"type": "NodePort"}}'

Identify the port used by nodeport and opne firewallrules on gcp
gcloud compute firewall-rules create firewall5 --allow tcp:31764

Username is admin
password: prom-operator

==========

##
##

All the Kubernetes definition files are available in this git repo

https://github.com/krishnain/KubernetesComplete.git

##
##