A pod in kubernetes can exists in 5 different lifecycle status
1. pending = pod has been accepted and is under creation status
2. running = atleast one of the container within pod has been started and readinessProbe has been passed.
3. successded = if all the containers within the pod exited with zero exit code then pod is said to be successful completed execution.
4. failed = if atleast one container within the pod has exited with non-zero status then it is marked as failed.
5. crashloopbackoff = if a pod is repeatedly failing after restart, the pod will not be scheduled for creation and kept with a status crashloopbackoff

What are labels and annotations in kubernetes?
labels are the arbitary key/value pairs we can attach to the kubernetes objects, those are used for identifying the objects in the kubernetes cluster.
For an kubernetes object we can any number of labels, but the key of the label should appear only once and should be unique.

We can declare labels within the Resource spec at the time of creating the object. or we can attach labels to the object at runtime as well.

healthcare-pod.yml

```
apiVersion: v1
kind: Pod
metadata:
name: healthcarepod
labels:
app: healthcare
environment: development
spec:
containers:
- image: healthcare:1.0
name: healthcare
ports:
- containerPort: 8081
name: http
protocol: TCP
```

we can display all the labels attached to pod using below command.
```
kubectl get pods --show-labels
```

we can query pods based on label key/value using -l switch
```
kubectl get pods -l environment=development
```

we add labels to a object at runtime after creation as well
```
kubectl label pods podname key=value
```

Annotations

Annotations are used for attaching arbitary non-identifier information to an kubernetes object. we can say them as documentation helpers. annotations are used by tools and libraries to retrieve metadata information associated to an object.

Unlike labels we can query the kubernetes objects based on annotations.

```
healthcare-pod.yaml
apiVersion: v1
kind: Pod
metadata:
name: healthcarepod
labels:
app: healthcare
environment: development
annotations:
author: sriman
imageregistry: http://reg.rconnect.org/k8s-virtual-repo
spec:
containers:
- image: healthcare:2.0
name: healthcare
ports:
- containerPort: 8081
name: http
protocol: TCP
```

----

pod
1. pod spec
2. readinessProbe
3. livenessProbe
4. resources
- requests
- limits
5. labels
6. annotations
7. pod lifecycle

----

config maps

What are ConfigMaps in kubernetes why do we need them?
A ConfigMap is an object used to store non-confidential data in key-value pairs. Pods can consume ConfigMaps in four different ways.
1. environment variables
2. command-line arguments
3. configuration files in volumes
4. We can write code dynamically in a container application running in the pod to read config map values using config map api.

ConfigMaps are the way through which we can separate configuration from the application by externalizing it. We can build docker images without configuration being hardcoded with them, rather we can create configmap with file and mount them as volumes in podspec. so that each we run a pod we can pass different file inputs by changing config map.

Now to use a ConfigMap in container application of a pod first we need to create it using spec.

```
healthcare-configmap.yaml
-----------------------
apiVersion: v1
kind: ConfigMap
metadata:
name: healthcaredbconfig
labels:
env: test
data:
driverClassname: com.mysql.cj.jdbc.Driver
url: "jdbc:mysql://localhost:3306/db"
username: root
password: root
```

```
healthcare-pod.yaml
------------------
apiVersion: v1
kind: Pod
metadata:
name: healthcare
labels:
app: healthcare
spec:
containers:
- image: reg.rconnect.org/k8s-virtual-repo/healthcare:2.0
name: healthcare
env:
- name: db.driverClassname
valueForm:
configMapKeyRef:
name: healthcaredbconfig
key: driverClassname
- name: db.url
valueForm:
configMapKeyRef:
name: healthcaredbconfig
key: url
ports:
```

- containerPort: 8081
name: http
protocol: TCP

we can get into running container of the pod using below command
kubectl exec -it podname /bin/bash
kubectl attach podname

What is ConfigMap?
ConfigMap is an kubernetes object type which is used for storing non-confidential arbitary key/value pairs in kubernetes cluster. It is mainly useed for separating the application from its configuration and store it separately.

There are 4 ways in which we can use a config map in kubernetes.
1. environment variables
2. command line arguments while pulling the image
3. configuration files being mounted as volume mounts
4. through kubernetes configmap api container application can access the config maps.

#1 How to pass configMap as an env variables within kubernetes pod.

```
apiVersion: v1
kind: Pod
metadata:
name: healthcare
labels:
app: healthcare
spec:
containers:
- image: healthcare2.0
name: healthcare
env:
- name: datasource.url
valueFrom:
configMapKeyRef:
name: healthcaredbconfigmap
key: db.url
```

#2 we can mount configMap with configuration files as volume mounts in pod spec

healthcare-configmapfiles.yaml

```
------------------------------
apiVersion: v1
kind: ConfigMap
metadata:
name: healthcareconfig
data:
# file with keys/values
bank-accounts.properties: |
Smith=9338393
Samuel=3939944
Arnold=0393904
```

healthcare-pod.yaml

```
apiVersion: v1
kind: Pod
metadata:
name: healthcarepod
labels:
app: healthcare
spec:
containers:
```

```
- image: healthcare:2.0
name: healthcare
volumeMounts:
- name: healthcareconfig
mountPath: "/config"
readOnly: true

volumes:
- name: healthcareconfig
configMap:
name: healthcareconfig
items:
- key: bank-accounts.properties
path: "bank-accounts.properties"
```

In the above we defined configMap as volume with name healthcareconfig. Now within container we mount volumes to a mountPath of the container using volumeMounts

----

Kubernetes Secrets

Kubernetes secrets let you store and manage sensitive information such as passwords, encryption keys, ssh keys etc. Instead of writing the password, sshkeys directly within pod spec it is always recommended to place them in kubernetes secrets and refer them.

Kubernetes secrets are stored by default in unencrypted base64-encoded format strings. by default they can retrieved as plain-text values by using api.

A secret can be used with a pod in 3 ways:
- As files in a volume mounted on one or more containers
- environment variables
- can be used by kubelet while pulling the images.

When creating a secret we can specify a type: attribute with value as one of the standard kubernetes defined secret types. It is a facilitator in helping the developers to understand what data is available within the secret. It is not mandatory to define secret type while creating the secret and by default it takes the type as :opaque (arbitary data)

built-in secret types:

Opaque:	arbitary user-defined data
kubernetes.io/service-account-token	service account token (kubernetes secret) (system secret)
kubernetes.io/dockercfg	serialized ~/.dockercfg file
kubernetes.io/dockerconfigjson	serialized ~/.docker/config.json file
kubernetes.io/basic-auth	username and password type
kubernetes.io/ssh-auth	credentials for ssh authentication
kubernetes.io/tls	ssl certificate

mysql-dbsecrets.yaml

```
apiVersion: v1
kind: Secret
metadata:
name: mysqldbcred
type: kubernetes.io/basic-auth
```

```
stringData:
username: root
password: root
```

healthcare-pod.yaml

```
apiVersion: v1
kind: Pod
metadata:
name: healthcarepod
labels:
app: healthcare
environment: development
spec:
containers:
- image: healthcare:2.0
name: healthcare
env:
- name: datasource.username
valueFrom:
secretKeyRef:
name: mysqldbcred
key: username
- name: datasource.password
valueFrom:
secretKeyRef:
name: mysqldbcred
key: password
```

Persistence Volumes and Claims
Persistence Volume (pv) is a piece of storage defined in kubernetes cluster that is created by kubernetes administrator or dynamically provisioned using Storage class.

Persistence Volume claim: is a request for a storage by a user. which can be consumed with a pod.

There are few attributes to be know about persistence volumes:
1. storageClassName:
is an identifier we can bind to the persistence volume so that in persistence volume claim we can request persistence volume of that storageClassName only.

2. accessModes:
- ReadOnly
- ReadWriteOnce = only one node on the cluster will be allowed to write
- ReadWriteMultiple = multiple nodes on the cluster will write on the persistence volume

3. capacity:
defines storage size to be allocated

healthcare-pv.yaml

```
apiVersion: v1
```

```
kind: PersistenceVolume
metadata:
name: hcmysqlvol
spec:
storageClassName: healthcareClass
capacity:
storage: 2Gi
accessModes:
- ReadWriteOnce
hostPath:
path: "/u01/data"
```

healthcare-pvc.yaml

```
apiVersion: v1
kind: PersistenceVolumeClaim
metadata:
name: healthmysqlvolclaim
spec:
storageClassName: healthcareClass
accessModes:
- ReadWriteOnce
resources:
requests:
storage: 2Gi
```

healthcare-pod.yaml

```
apiVersion: v1
kind: Pod
metadata:
name: healthcarepod
spec:
containers:
- image: healthcare:2.0
name: healthcare
volumeMounts:
- name: hcmysqlvol
mountPath: /var/lib/mysql
volumes:
- name: hcmysqlvol
persistenceVolumeClaim:
claimName: healthmysqlvolclaim
```

Kubernetes Controlplane has 3 components
1. Api Manager = Front-end to the kubernetes masterplane through which we interact with kubernetes
2. Scheduler = Schedules the pod on the workernode of the cluster

3. Controller Manager
Daemon process that run on the kubernetes cluster which works at the background continously to bring the cluster to the desired state.
There are 5 different types of Controllers are there
1. ReplicaSet
2. Deployment
3. DaemonSet
4. Service
5. Job

ReplicaSet
We can create a pod within kubernetes cluster through pod manifest/spec. There are few characteristics of the pod
1. One pod spec/manifest creates only One instance of pod under Running
2. Pods will not surviev through crash

If we want to run multiple pods with same containerized application, then we need to submit multiple pod specs, managing the pods through multiple pods spec and monitoring them to ensure always the desired number of pods are running is difficult.

30 = 30 pods specs = difficult job
we need to apply 30 pod specs
tracking, managing the pods as a group is not possible.

How to solve this problem?
Kubernetes has provided ReplicaSet, through which we can achieve desired number of Replicas of a pod are always running within the kubernetes cluster.

We can imagine a ReplicaSet as a Reconciliation loop, where a ReplicaSet controller loops through kubernetes cluster to observer whether desired number of replicas are reached or not and ensures it brings the cluster state to the replicas we specified.
There after it monitors the kubernetes cluster to ensure always the desired replicas are running in case of pod failure or crash as well.

there is no use of writing a replicaSet spec without a pod running, so it is best practise we write pod spec within the replicaSet spec only together and will applied. Here a pod spec is defined as a template indicate ReplicaSet run this pod template of 3 replicas.

From the above we can understand we dont create pod, rather we create ReplicaSet, which ensures it runs the desired number of pod replicas based on the template we defined within spec

```
apiVersion: apps/v1
Kind: ReplicaSet
metadata:
name: healthcareReplicaSet
labels:
app: healthcare
```

```
#replicaSet specification
spec:
replicas: 3
selector:
matchLabels:
app: healthcarepod
environment: production
#pod information [type of pod]
template:
metadata:
labels:
app: healthcarepod
environment: production
spec:
containers:
- image: reg.rconnect.org/k8s-virtual-repo/healthcare:2.0
name: healthcare
ports:
- containerPort: 8081
name: http
protocol: TCP
```

How to scale the replicas?

```
kubectl scale replicasets replicasetname --replicas=4
kubectl edit relicaset replicassetname
```

will open the runtime spec you can modify and apply.

In general we dont use directly the ReplicaSet for running the pods within kubernetes cluster rather we use Deployment around the ReplicaSet, since deployment has more features than replicaSet.

Deployments

In general deployment is the process and packaging and place the application within the server to bring it running.
In kubernetes we are deploying the pods on to the kubernetes cluster to make them running.
There are manys was we can deploy a pod and run in kubernetes cluster
1. pod spec
2. replicaSet spec

The third-way through which we can have a pod application running in kubernetes is through Deployment Spec. We use deployment provide declarative updates for pods and replicas.

We describe desired state in a Deployment, and the Deployment Controller changes the actual state to the desired state at a controlled rate.

Usecase:-
1. Create a deployment to Rollout a ReplicaSet = through deployment spec we can tell number of replicas to be running to ReplicaSet
2. We can update an existing pod template, so that Deployment creates a new RelicaSet with changed pod template and rolls the update slowly.
3. if the rolling update is not good, we can rollback to the previous state of the pod as well.
4. Scaleup the deployment to handle more load
5. we can pause a deployment and make multiple changes to pod template and apply at one

shot.

Deployment Spec:-

```
apiVersion: apps/v1
kind: Deployment
metadata:
name: healthcare-deployment
labels:
application: healthcare
spec:
replicas: 2
selector:
matchLabels:
templatename: healthcaretemplate
template :
metadata:
labels:
templatename: healthcaretemplate
spec:
containers:
- image: reg.rconnect.org/k8s-virtual-repo/healthcare:2.0
name: healthcare
ports:
- containerPort: 8081
name: http
protocol: TCP
```

```
kubectl set image deployment/deploymentname label=image --record
kubectl edit deployment deploymentName
kubectl rollout history deployment deploymentName
kubectl scale deployment deploymentName --replicas=2
```

kubectl rollout undo deployment deploymentName --to-revision=2 = if we want to rollout to previous revisions
----
Service
Service is about networking the pods that are running on the kubernetes cluster. There are many reasons we would use service in kubernetes
1. To expose multiple replicas of a backend pod applications to the front-end over fixed ip address we use service.
2. To expose a pod/relicaset/deployment to be accessed to the external world.

There 5 types are supported by kubernetes

1. ClusterIP Service = we can expose replicas of a backend application pods using a fixed/static cluster ip address, so that it can consumed as part of FrontEnd application pods that are running within the kubernetes cluster.
It is not meant for exposing the pod over the external network. it acts as a load balancer to route request to multiple replicas of pod running with the cluster.
By default when we create a service, it is created as ClusterIP service only and is assigned with

ip address within the Cluster Range.

2. Headless Service = In headless service we specify ClusterIP as none, so that the service can be accessible with servicename through dns resolution rather than using ip address.

3. NodePort = Is used for routing external traffic to worknode port of the cluster. Each worker node will be assigned and opended with a fixed port no using which client applications can directly access the pod running on workernode using workernodeip:nodeport

NodePort:= 30000 - 32727
It is not recommended to exposes pods through Nodeport, since the worker nodes are directly exposed to the external world.

4. LoadBalancer
5. Ingress

Amazon EKS Cluster
While working with onpremise environment,
1. the high availability of K8S Master/Controlplane has to be taken care by the ops team, by setting a slave control plane for the master.
2. scaling up the worknode pool has to be handled manually.

Managed Cloud Kubernetes Service like
AWS EKS = Elastic Kubernetes Service
The guarantee availability of Kubernetes Master Node/Controlplane will be taken care by AWS EKS.
In AWS EKS We can setup an Worker Node pool with initial capacity and max capacity, so that autoscaling of these worknodes depends on demand will be handled by eks.

There are 3 ways to setup the EKS Cluster
1. eksctl is an commandline interface through which we can quickly setup an eks cluster
2. amazon cloud console
3. terraform.

#1 To setup an eks cluster we need to create iam role
eks cluster
#AmazonEksClusterPolicy

ODWA106062021 = HmS4:m

Jenkins
What is jenkins what is the purpose of jenkins?
Jenkins is continuous integration tool through which we can achieve ci/cd automation. Jenkins helps us in archestrating various different activities like build, deploy and release of an software application, so that we can achieve continous success of delivery of an application.

Jenkins is written java language its an web application in which it provides console based access. For each project we would want to perform a build, we need to create project within the jenkins and define the steps we want to perform for that project within the jenkins pipeline project.

Jenkins works on master/slave architecture, where on the master we are going to create jobs for each project we want to build, when we run job, it is scheduled on the agent to execute and produce the desired output.

Jenkins supports various types of projects
1. Freestyle project (shell commands)
We write directly the shell script to perform the build activities within it, which would be executed and finaly outcome will be shown
-> check git
-> mvn clean verify
-> artifactory

2. Pipeline project (jenkins2)
-> The entire build and deployment activities of the projects are broken down into stages/steps
stage('git') {}
stage('build'){}
stage('push') {}
pipeline = means sequence of stages that has to be executed in achieving the final state of the system.

-> jenkins works on plugins
-> plugins the components through which we can perform build activities.
- git plugin
- maven
- pipeline stage view
- docker plugin

Steps to install jenkins on ubuntu

#1 wget -q -O - https://pkg.jenkins.io/debian-stable/jenkins.io.key | sudo apt-key add -
#2 sudo sh -c 'echo deb http://pkg.jenkins.io/debian-stable binary/ > /etc/apt/sources.list.d/jenkins.list'
#3 sudo apt update

#4 pre-install openjdk-11-jdk on all the nodes including master and agents.

#5 sudo apt install jenkins
#6 systemctl restart jenkins

What is Jenkins?
For building, deploying and releasing an application there are several sequence of activities are being involved like
1. pull the code from source versioning system
2. build the application
3. publish the artifacts of a build to the artifactory repository
4. setup a testaware environment
- tools, softwares and libraries
5. deploy the application
6. make it accessible
the above steps could vary depends on type of the application and nature of the application we are working on. To ensure faster delivery of our application we have to implement automation across different activities/aspects of the system.
For eg..
1. building an application we need to use maven/gradle build tool
2. artifactory repository for distributing artifacts internal we need to artifactory repositories
3. for setting up infrastructure, we need to use iac tools like
vagrant
terraform
4. for provisioning the softwares required on the machine we need to use
shell scripting
ansible
5. baking and deploying an application
# we can go for manually copy of the application
# or can deliver it as a containerized application

but who will carry out all the above activities running the above tools in setting up the delivery of the application that is where jenkins comes into picture.
Jenkins is a Continous integration tool that archestrate the different aspects/activities involved in delivering an application.

Jenkins has provided bunch of plugins through which we can automate the execution of various different activities defined above, so that opsengineer can script the entire process of build and delivery of the application through the help of plugins

Jenkins Architecture

Jenkins follows master/slave architecture. Master is the central server on which the jenkins is being installed and configured, through which the opsengineer can script the automation flow for his project and can trigger the master to execute.

To ensure scalability of the builds that are carried across the projects, master node distributes the execution of these scripted activities across the slaves/nodes which are responsible for performing the build execution.
Features:-
1. Scalability
2. Scripted Pipeline = the entire build activities of the project can be scripted and can versioned as part of the source code of our application, so that we can keep track of how the build activities of our project has been evolved. we can distribute the activities to the entire team.
3. plugin based architecture = we can add new features and enhance the jenkins system and support new build activities through plugins
4. we can establish and manage parent-child dependencies in executing the build
5. we can run parallel builds which are independent of a project.

How to work with jenkins?
The developer/ops engineer has to write the instructions/script in executing the build activities as described above for a project, this can be done through creating an project in jenkins. There are 2 types of projects are there in jenkins
1. Freestyle project (Jenkins1)
Jenkins will provide you an user interface allowing you to define the build activities directly. opsengineer has to write bash/shell scripting within the userinterface to carry out the activities required for building.
Now all the activities that has to be performed out of a build/release of a project are written together without any differentiation, so that there are few problems with freestyle project.
- we cannot version the project activity script within the source code repository, so we cannot identify or version the build of the project.
- a build/release of a project involves several activities to be conducted, which cannot be represented in freestyle project, so we cannot identify where the failure has been encountered while carrying which activity unless we go through the logs
- we cannot monitor realtime progress of the build activity

2. Pipeline project
Pipeline is defined as sequence of steps/flow of execution that has to be carried in the order definition to achieve the delivery/release of the application is called pipeline.

There are 2 types of pipeline syntax supported by jenkins
2.1 declarative pipeline
2.2 scripted pipeline

How to install jenkins on Ubuntu?

#1 wget -q -O - https://pkg.jenkins.io/debian-stable/jenkins.io.key | sudo apt-key add -
#2 sudo sh -c 'echo deb http://pkg.jenkins.io/debian-stable binary/ /etc/apt/sources.list.d/jenkins.list'
#3 sudo apt update
#4 sudo apt install openjdk-11-jdk
#5 sudo apt install jenkins

Agent Node installation

#1 on all the agent nodes install open-jdk11
sudo apt install -y openjdk-11-jdk
#2 install maven
sudo apt install -y maven
#3 install docker if you which to build and run docker containers.

#1 declarative pipline
here the project deployment/delivery activities are written in jenkins domain specific language where we define each activitiy as stage with steps inside it. A pipeline developer dont need to have any programming language background to write declarative pipeline. within each step we need to invoke a plugin to carry out the activities.

syntax:-
```
pipeline {
agent {            or  agent any
label 'agent1'
}
stages {
stage('git checkout') {
steps {
echo "checkout code"
}
}
stage('build') {
steps {
echo "mvn clean verify"
}
}
}
}
```

#2 scripted pipeline
scripted pipeline is written in groovy language and has more control over performing the defining the activities. pipeline developer has to know groovy language scripting to write scripted pipelines, seems to be complex to achieve.

syntax:-
```
node('node1') {
stage('git checkout') {
echo "checout project"
}
stage('build') {
echo "mvn clean verify"
}
}
```

jfrog = The provide software tools for release management, end-end application delivery. using these tools an organization can implement ci/cd workflow easily. There are many tools provided by jfrog

jfrog platform (full suite)
- dependency management of various remote repositories
- container registry for distributed container images
- release management pipelines
- security scanner
- distribution

jfrog artifactory (commercial) | jfrog artifactory oss (only jcentral, maven remote repository proxying)
- dependency management of various remote repository
maven, jcentral, bitnary

jfrog container registry (open source)
- container images of the organization can be distributed locall across
(docker, helm)

jfrog artifactory / container registry supports three types of repositories
- local repository = publishing the produced artifacts so that it can be distributed across various stages of our project delivery
-local-dev (alpha/beta)
-qa (beta)
-stage
-release

- remote repository = proxy the public repository and cache locally, so that we can improve build performance

- virtual repository = hybrid of both,

----

to setup artifactory-oss or artifactory-jcr we need a hosted domain only. if we dont have the machine attached domain name, we can install apache2 server with virtual host configured locally on the same machine where we are running the above softwares.

setting up artifactory-oss.
#1 download jfrog artifactory-oss software
https://jfrog.com/open-source/

#2 guznip and tar extract into /u01/app directory
before proceeding further in running the artifactory setup the apache2 server with virtualhost configuration. repo.rconnect.org

#3 install apache2 server
sudo apt update -y
sudo apt install -y apache2

now we need to write virtualhost configuration on the ServerName: www.repo.rconnect.org with proxyPass configuration pointing to the ip and port on which artifactory oss server is running.

here apache2 is not playing the role of webhosting server rather he acts as a proxy for a back-end server.

We configure apache2 server on port: *.80 with ServerName: www.repo.rconnect.org up receiving the request proxy this request (pass) the request to backend server (artifactory oss) running on http:

```
sudo a2enmod proxy
sudo a2enmod proxy_http
sudo a2enmod proxy_balancer
sudo a2enmod lbmethod_byrequests
```

```
/etc/apache2/sites-available/repo-rconnect.conf
<VirtualHost *:80>
ServerName www.repo.rconnect.org
ServerAdmin webmaster@localhost
ProxyPreserveHost on
ProxyPass / http://127.0.0.1:9091
ProxyPassReverse / http://127.0.0.1:9091
</VirtualHost>
```

How to install and use Jfrog Artifactory oss?
To install jfrog artifactory oss and use it we need apache2 with virtual domain configured with proxy pass should be setup.

steps for installing and configuring apache2 server.
----
1. sudo apt update -y
2. sudo apt install -y apache2

3. enable proxy modules in apache2 server
- sudo a2enmod proxy
- sudo a2enmod proxy_http
- sudo a2enmod proxy_balancer
- sudo a2enmod lbmethod_byrequests

4. create apache2 virtual host configuration file for proxy pass configuration
/etc/apache2/sites-available/
- create a file for eg.. repo-rconnect.conf with below contents

```
<VirtualHost *:80>
ServerName www.repo.rconnect.org
ServerAdmin webmaster@localhost
ProxyPreserveHost on
ProxyPass / http://127.0.0.1:9092/
ProxyPassReverse / http://127.0.0.1:9092/
</VirtualHost>
```

the port on which the jfrog artifactory ui is configured by us is "9092" always it should point to frontend port of jfrog artifactory
5. vim /etc/hosts
add domainname with loopback ip

```
127.0.0.1   repo.rconnect.org
```

6. sudo a2ensite repo-rconnect
7. sudo apt reload apache2
8. sudo apt restart apache2

Steps for setting up JFrog Artifactory OSS Repository.
#1 create /u01/app directory
mkdir -p /u01/app

#2 download JFrog Artifactory OSS software binary
https://jfrog.com/open-source/#artifactory
https://releases.jfrog.io/artifactory/bintray-artifactory/org/artifactory/oss/jfrog-artifactory-oss/[RELEASE]/jfrog-artifactory-oss-[RELEASE]-linux.tar.gz

#3 copy the tar.gz into app directory, gunzip and tar extract it

```
/u01/app$ gunzip jfrog-artifactory-oss-[RELEASE]-linux.tar.gz
/u01/app$ tar -xvf jfrog-artifactory-oss-[RELEASE]-linux.tar
```

it will extracts into below directory.
artifactory-oss-7.18.6

#4 change default port configuration
by default we dont have system.yml in artifactory-oss-7.18.6/var/etc. but there is a template file with name system.basic-template.yaml.
we should copy the contents of the file into system.yml under artifactory-oss-7.18.6/var/etc and add the below configuration at the end of the file

system.yml

```
## @formatter:off
## JFROG ARTIFACTORY SYSTEM CONFIGURATION FILE
## HOW TO USE: comment-out any field and keep the correct yaml indentation by deleting only the leading '#' character.
configVersion: 1
artifactory:
port: 9091
router:
entrypoints:
externalPort: 9092
```

#5 goto /u01/app/artifactory-oss-7.18.6/app/bin
./artifactoryctl start/stop to start and stop the artifactory oss server

#6 open the browser and type http://repo.rconnect.org -> should open the jfrog artifactory home page asking for changing the default admin user and password
default: admin/password

Creating Repositories for the project
once after the login into jfrog artifactory ui, click on administrator in the left panel. and expand repositories and click on repositories.

In general There are 2 types of repositories are required for maven projects.
1. artifact (just) repositories
The dependencies that we use as part of our project, those are jar dependencies usually. the artifact repositories are nothing but jar dependent repositories

when you build a java project, you produce the project as an published or distributable artifact, which is for eg.. rconnect.war. if we want to distribute this war within our project during different stages of delivery we need to publish into artifact repositories

2. plugin repositories
building blocks of maven build system, they perform actions as part of the maven build like maven-compiler-plugin compiling the source code, maven-jar-plugin for packaging the project as jar etc.

sometimes based on the requirement, in a project we might build a custom maven plugin of our own, how to distribute this maven-plugin during various stages of deployment and delivery of the project. publish the plugin into plugin repositories

While setting up repositories for the project in artifactory we need to create 2 types of repositories as described about
1. repositories
2. plugin-repositories
These can be distinguished only through name, underlying both the repositories are same.

To publish and distribute project artifacts of several stages of our project we need to create 2 repositories for each of the above
1. snapshots
2. release repositories

In JFrog there are 3 types of repositories are there
1. local repository = publishing the project artifacts so that it can be distributed across the delivery stages
2. remote repository = proxy the remote repositories
3. virtual repository = combination of local and remote, so that first time when we conduct the build virtualrepo redirect to remote and cache it locally, 2nd time onwards the builds will be done from local repo.

For a project we need to setup
repository, plugin repositories for both snapshot and releases of type local, remote, and virtual repo as shown below.

For eg.. lets a project rconnect we can setup below repositories

artifact repositories
----
local-repo-rconnect-snapshot
local-repo-rconnect-release

plugin repositories
----
local-pluginrepo-rconnect-snapshot
local-pluginrepo-rconnect-release

remote-repo-rconnect-snapshot
remote-repo-rconnect-release

virtualrepo-rconnect-snapshot
local-repo-rconnect-snapshot
remote-repo-rconnect-snapshot

virtualpluginrepo-rconnect-snapshot
local-pluginrepo-rconnect-snapshot
remote-repo-rconnect-snapshot

virtualrepo-rconnect-release
local-repo-rconnect-release
remote-repo-rconnect-release

virtualpluginrepo-rconnect-release
local-pluginrepo-rconnect-release
remote-repo-rconnect-release
----
after creating these repositories goto application tab on left navigation bar expand aritfacts and click on setup right top corner
select appropriate release, snapshot repositories as we created above, enter password in box and click on generate. will create settings.xml

when we run the maven build in our project
~/railworkspace
|-rconnect
|-pom.xml
mvn clean verify = by default maven build system goes to maven central repositories for resolving dependencies and plugins we defined in pom.xml.

to let the maven build goto artifactory repositories we setup now we need place settings.xml in ~/.m2 directory of the user

In the settings.xml we define repositories which are repositories and plugin repositories pointing to the jfrog repositories we created along with username and password to connect.

now after setting up the settings.xml in ~/.m2 directory if we perform maven build it starts pulling dependencies and plugins from our artifactory repositories only.

if we want to publish the build artifacts into local repositories of the artifactory we need to configure distributionManagement section under pom.xml, for different stages we can create different profiles pointing to snapshot and release repositories.

```
<profiles>
<profile>
<id>snapshot</id>
<distributionManagement>
<repository>
<id>snapshots</id>
<uri>local-repo-rconnect-snapshot</uri>
</repository>
</distributionManagement>
</profile>
<profile>
<id>release</id>
<distributionManagement>
<repository>
<id>central</id>
<uri>local-repo-rconnect-release</uri>
</repository>
</distributionManagement>
</profile>
</profiles>
```

```
mvn clean deploy -P snapshot/release
```

How can we distribute docker images that are build out of the project, to various different stages of our application development and delivery. how can we achieve ci/cd for an application?
The one way we can achieve is through docker hub repository, the developer build container images can be published in docker hub repository, so that it can be pulled at various different stages of the development/delivery phase and use it.

But there are several problems with this approach.
1. by default all the images that are published into docker hub are public and anyone can use it
2. only one single private repository is permitted for one account and that is of non-commercial usage by docker hub
3. pushing and pulling the container images onto the docker hub takes lot of time due to the internet connection and bandwidth speeds and delays the delivery of application.

So, instead of using docker hub repository, we can setup our own internal organizational container registries and there are several container registry products are available in opensource and commercial as well.
We can host these container registries within organization, publish and distribute project container images internal and for implementing ci/cd.

JFrog artifactory fcr = it is mandatory to have domain name associated to do dns resolution while pushing and pulling container images from the container registry. If we are hosting the container registry locally, then we need to atleast setup virtualhost with /etc/hosts domain name resolution.

JFrog Container Registry supports 3 types of image repositories
1. local repository = distributed container images we produced out of the project locally
2. remote repository = proxy of the docker hub repository
3. virtual repository = combination of both local and remote repository

8081/8082

What is Kubernetes (k8s), what is the purpose of it?
Kubernetes (k8s) is a container cluster manager

background about micro-services?
In general an application comprises of several modules. Modules are nothing but grouping related functionalities together, so that those can be planned and developed easily on similar grounds.

For eg.. in an RConnect application we were developing it has several functionalities
Features:-
book ticket
cancel ticket
print ticket
ticket status
check availability of berths
trains between stations
running status of train
booking history

Each of them we need to study or understand the requirement and should develop, if we can group related features together as a module, we can plan all the related requirements and code it together, which helps us in saving development time and helps us in arriving to the better design of those features. out of the above features we can derive 3 modules

ticket management module
- book ticket
- plan journey

availability
- trains running between stations
- check berth availability in trains
- train running status
- pnr status

account
- cancel ticket
- history
- print

In our project we have 3 modules and is developed by team of developers based on the project planning. Now a project can be developed based on 2 architectures
1. Monolithic application architecture
2. MicroService Layer architecture

#1 Monolithic application architecture
The project is developed as a single unit/codebase which comprises of all the modules inside it and would be delivered as a single deployable artifact.

There are 2 Architectures in which an Application can be developed.
1. Monolithic Application Architecture
All the Features/Modules are being developed as part of the Single Project source and would be delivered and deployed as a single deployable artifact.

When we build the application using the build tools like maven or gradle they produce a war/ear which can be deployed on the server.
advantages:-
1. every developer in the team knows everything as the entire source code of the project is at one single place
2. easy to deploy, just build and copy the artifact
3. can achieve scalability easily

dis-advantages:-
1. Developers often find very complex in understand building the system as all the modules are being place at one single place
2. The build, deployment and staring up the application takes more time which delays the development and delivery of the application
3. Overloaded Ide as the huge source code of the project cannot be handled by ide, due to poor response of the ide, delay the development will takes place
4. Release Management would be very difficult
- as all the modules are being developed as a single source, we cannot release an module independent of other modules in the project. we have wait for until their completion.
5. Patching and Upgrades are difficult and imapcts the availablity of the whole system
6. We can achieve only Horizantal scaling, in a monolithic system, even though the traffic comes into the system for a specific module. Due to this hardware cost in scaling up system is going to be very high.

2. Microservice Application Architecture

We can follow 2 architectural styles in developing an application.
1. Monolithic application architecture
2. MicroServices Layed Architecture.

MicroService Architecture
In MicroService Architecture the whole system is being de-composed of several smaller individual services which can be deployable independent of the other.
The Services are identified and decomposed based on multiple technics.
1. based on domain
2. based on responsibility
3. based on functionality

These Service component we build are distributed and interoperable components which can be accessed over the network by any other applications build on any langauge and any platform.

What is a MicroService?
MicroService is an Interoperable and Distributed (Network) Business component, which will take the data from the Http protocol and send the data back to the Client Program/Application over the network using Http Protocol.

Design and Deployment considerations
----
1. Each service should be independently deployable and completely loosely coupled
2. Each individual service is built out of its own project source code
3. There are mulitple Independent development teams working on building each of these services parallely since these are independent.
4. Each Service should have its own database on which it is storing/accessing the data, to achieve highest level loose coupling.
5. Each service should run on its own container and should be deployed independently

----
Kubernetes

What is kubernetes?
Kubernetes is a Container Cluster Manager, which takes care of scheduling, monitoring and running the containerized applications on the Worker Nodes of Cluster

In a Kubernetes architecture there are 3 parts of it.
#1 Kubectl
#2 Kubernetes Master/Control Plane
#3 Worker Nodes

#1 Kubernetes Master Node / Control Plane
Kubernetes Master Node is also called Kubernetes ControlPlane, which is the central component of the Kubernetes Cluster, which takes care of scheduling and managing the containerized applications on the network cluster. It has to be installed on the Central Machine of the Network, which has access to the Machines connected to the network.

There are 3 major components are there in Kube Master.
1. Api Manager
We can access the Kubernetes Controlplane and schedule/manage/monitor applications by talking to the Api Manager.

Api Manager is a Front-End to the Kubernetes Cluster, through which we can talk to the MasterNode. It is build-on Rest technology so that we can access the Master Node remotely through the help of Api Manager.
Api Manager receives the instructions from the Client remotely and passes those instructions to the Scheduler/Controllers, asking to perform the operations and returns the response back to the Client

When the client submitted an manifest asking to run the pod on the worker nodes to Api Manager, he will perform certain activities before accepting the request like
- validate the manifest that is supplied by the client
- is the operation requested by the client is allowed or not

if any of these checks failed, the without accepting the request, api manager with return an error to the client, otherwise the request will be accepted and passed to the relevant components of the Kube Master for processing.

2. Scheduler
Upon receiving an manifest from the Client, once after Api Manager has validated and verified the operation, he passes the manifest to Scheduler asking to apply the manifest.
Scheduler will goes to each Worker Node and talks to the kubelet process asking whether the work node has enough computing capacity to run the pods or not, he reaches all the work nodes until he finds one suitable worker node in the cluster.
Then handovers the pod manifest to the kubelet process asking to run the pod.

3. Controller Manager
Controller Manager is a daemon process that continously works in background on a kubernetes controlplace, which ensures always we reach to the desired state of manifest definition.

There are 5 types of Controllers are there in Kubernetes
1. ReplicaSet = ensures desired number of replicas are running
2. DeploymentSet = rolling updates
3. DaemonSet = ensures the pods are distributed across the worker nodes on the cluster
4. Service = Offers networking Services to pods running across the worker nodes enabling them to communicate and expose to external world
5. Job = We want to run an external program on all the pods running across the worker nodes of the cluster we can use Job Controller

#2 What is Pod?
Pod is a smallest unit/component in the entire Kubernetes Cluster, where one or more containers can be combined and executed together.
In general a container under execution requires certain resources like
- networking
- storage
- publish ports etc

There can an application that might be distribtued into multiple containers which has dependency on each other. To run the application we need to run both the containers together as there are dependent. These containers might have common resource requirement in sharing networking, storage etc.

How can we have these dependent containers running together and manage them?
place them or package them into one pod, so that they share the common lifecycle and resources across them can co-exist together.

in-short a pod is an abstraction of multiple containers which has to be executed together. At a minimal a pod should have atleast one container inside it running.

From the above on a kubernetes cluster we can schedule the pods for execution, wherein within the pods the containers would be running(it could be one or more containers inside the pod)

#3 Worker Nodes
Worker Node is a Machine which can be a virtual machine, physical computer or a cloud instance, that is configured to work with Kuberntes Master plane. The pods are scheduled to be running on Worker Nodes.

Every worker node should be installed and configured with 3 components
1. containerization engine = To run containers on Worker Nodes we need a containerization engine like docker, containerd, apache Mesos. So the kubelet can create containers and run them on the containerization engine

2. kubelet (restapi) = kubelet is a process running on the each worker node, providing the information about the pods and worknode to the kubernetes master. we can think of a kubelet is an end-process that takes care of perform actions on the worker node

3. kube-proxy = is a networking process that exposes the pods to the external work.

#4 kubectl
Its an CLI tool a client will use to communicate with api manager in passing the instructions to the kubernetes asking to manage the cluster.
kubectl requires the information about the kubernetes cluster which is configured in kubeconfig.yml file and placed under $HOME\.kube directory.

#5 etcd = its a storage for the kubernetes cluster, where all the pod manifest and kubernetes cluster information is stored in etcd. (simple key-value pair storage unit)

ODWA123052021 = [s&W6(

Kubernetes (k8s) = container cluster manager, that takes care of scheduling, monitoring and managing the containers on a cluster environment.
There are 3 components are there
1. Kubectl = its an CLI tool used by the kube administrator in interacting with Control Plane.

2. Master Node, Kubernetes Control place
- The Master Node is the engine of the Kubernetes, he takes care of scheduling, managing the pods on the kubernetes Worker nodes.
There are 3 parts are there in Kubernetes Master/Control Plane
2.1 Api Manager = Rest Endpoint interface for the clients to access the Master/Control Plane. Kubectl cli tool interacts with Api Manager to manage the kubernetes cluster.

2.2 Scheduler = Takes care of identifying the Nodes and scheduling the Pods to execute.

2.3 Controller Manager = These are daemon process that executes on the Kubernetes cluster and ensure the designed state of the System is always reached. There are 5 types of Controller Managers are there

1. ReplicaSet
2. Deployment
3. DaemonSet
4. Job
5. Service

3. Worker Node = Worker nodes are the group of machines (virutal/physical/cloud) that are connected to the Kubernetes Master, and the pods are scheduled to run on any of these worker nodes of the kubernetes cluster.
What are the components must be installed on Worker Nodes?
1. kubelet
2. kubeproxy
3. containerization engine

etcd = is a database for MasterNode/Controlplane to store the data

What is a Pod?
Pod is the smallest unit of the Kubernetes Cluster in which one or more containers are packages and ran together.
There can be multiple containers who has dependency in terms of lifecycle or resources which needs to be executed together. Such containers of that dependencies can be combined together and can get executed on kubernetes cluster as pod.
By default even we have one single container to execute still it should be packages/run inside the pod only.

Kubernetes is a declarative container cluster manager tool, we only provide configuration information asking to run the containers, what has to be done to run the pods on the cluster would be decided by Kubernetes itself and makes best effort in running them.

How to declare the desired state definitions in kubernetes?
Kubernetes defines an Yaml specification for defining the various different objects we would want kubernetes cluster to manage.

CNI = Container Network Interface is a specification which has standardized the networking aspects of interconnecting the containers. He has provided libraries and plugins for writing our own implementations of CNI Network. Kubernetes has built-in support for CNI.

Kubernetes has been built, centrally considering the networking as a key requirement without networking aspect there is no way we can achieve collaborative application deployment. There are 4 aspects of kubernetes networking

1. Container to Container communication = package containers into pod and use localhost
2. Pod-to-Pod communication
3. Pod-to-Service communication
4. Service to the External communication

There are multiple Kubernetes CNI implementations are there

1. calico
2. canal
3. Flannel
4. Kube-Router
5. Kubenet
6. AWS VPC
7. Weave

Kubernetes has been build from the ground by keeping networking as the important aspect of the System. There are 4 ways of communication to be established in a kubernetes cluster
1. container-to-container communication
2. pod-to-pod communication
3. pod-to-service communication
4. service-to-external communication

To the basic lets learn pod networking
#1 2 containers which are part of the same pod can directly communicate with eachother using localhost:port of the container applications.

#2 pod to pod communication
Whenever we create a pod in kubernetes cluster, by default a pod will be generated with an ip address automatically.
The basic networking model kubernetes enforces between the pods is:
1. pod on a node can communicate with all pods on all the nodes without an NAT
2. agents on a node can communicate with all the pods on that node only.
so this helps us to treat pods as virtual machines and help us in treating a pod as vm and can use it.

CNI = Container Network Interface is a specification and a standard defined for inter connecting containers together.
kubernetes supports CNI networking model, so that any vendor can provide implementation of k8s cni networking and can be installed and used as part of kubernetes network.

There are lot of k8s CNI implementations are provided by vendors, few of the popular network implementations:
1. calico
2. flannel
3. weave
4. cannal
5. aws vpc
6. kube router

During kubernetes cluster install, after installing the master node before adding worker nodes to the kubernetes cluster, we need to install one of the k8s cni networking implementations. then the master-> workernode communication will be enabled when we add worker node.

Fundamental to installation kubernetes on ubuntu virtual machine.
1. Kubernetes Master Node/Control install on one ubuntu virtualmachine (if we want we can take 2 kubernetes master also for HA).
2   - cpus
2gb - ram

2. Atleast to setup a kubernetes cluster we need 2 Worker Nodes, in a production like env it is recommended to have 5 worker nodes in kubernetes cluster. So for each of the worker node we need a virtual machine.
1   - cpu
1gb - ram
to make kubernetes master and worker node communicate with each other we need to establish internal network and to let these machines communicate with external network we need NAT

networking
we have to configure for each of these virtual machines 2 network adapters
1. internal network
2. nat networking
recommendations to setup:
- the network with static ip address which will not change
- the hostname should be set with computer name

The best way to bring all these computers with static ip address is use vagrant.

Steps in Installing and Configuring the Kubernetes Cluster:-

#1
Create a vagrant configuration with 1 Master and 2 Worker Nodes as shown below with private_network and static ip configuration.

Vagrantfile

```
Vagrant.configure(2) do | config |
config.vm.define "kubemaster" do | kubemaster|
kubemaster.vm.box = "ubuntu/hirsute64"
kubemaster.vm.network "private_network", ip: "192.168.10.14"
#kubemaster.vm.network "forwarded_port", guest: 8081, host: 8081
kubemaster.vm.hostname = "kubemaster"
kubemaster.vm.provider "virtualbox" do | vb |
vb.cpus = 2
vb.memory = 2048
vb.name = "kubemaster"
end
end
%w{workernode1 workernode2}.each_with_index do |nodename, index|
config.vm.define nodename do | workernode |
workernode.vm.box = "ubuntu/hirsute64"
workernode.vm.network "private_network", ip: "192.168.10.#{index + 15}"
workernode.vm.hostname = nodename
workernode.vm.provider "virtualbox" do | vb |
vb.cpus = 2
vb.memory = 1024
vb.name = nodename
end
end
end
end
```

#2 bridge traffic to iptables is enabled for kube-router [master node and worker nodes]
/etc/ufw/sysctl.conf
```
net/bridge/bridge-nf-call-ip6tables = 1
net/bridge/bridge-nf-call-iptables = 1
net/bridge/bridge-nf-call-arptables = 1
```

#3 edit /etc/hosts file make an entry of three nodes with ip and node for 3 virtualmachines
kubemaster
/etc/hosts

192.168.10.14 kubemaster
192.168.10.15 workernode1
192.168.10.16 workernode2

Below configuration is required for routing the requests through the kube-proxy process to the services of your cluster:-

additional entry in each host within /etc/hosts
kubemaster node
192.168.10.14 kubemaster kubmaster.local

workernode1 node
192.168.10.15 workernode1 workernode1.local

workernode2 node
192.168.10.16 workernode2 workernode2.local

#4 by default in ubuntu-18.04 swap space will be enabled.
sudo -i
swapoff -a = restart again will be enabled
permanently turn off the swap
/etc/fstab = goto the file and comment /swap/d entry
to check swap status you have to type swap status = shows all the swap partitions on the machine

#5 sudo apt update -y
sudo apt upgrade -y

#6 sudo apt install -y ebtables ethtool
eithernet bridge tables
eithernet tools

#In addition we need to install docker on all the machines.

#Now to begin installing the kubernetes cluster we need 3 packages to be install across the cluster both on kubernetes master node and worker node also.

kubectl = cli tool
kubeadm = to install kubernetes on master node, and to join worker nodes to master node
kubeadm tool has been provided by kubernetes engineers.
kubelet = kubelet is a process to let kubernetes master and worker nodes to communicate with each other

#1 install docker on both kubernetes master and worker nodes as well.
why on master node, by default pods are not scheduled on master node, so we dont need docker on master node. but we can taint master node to schedule pods for development or test environments so in such case master also requires docker container.

sudo apt install -y docker.io apt-transport-https curl

#2 add gpg key to enable kubernetes repositories to be added to ubuntu for installing kubernetes.
be in root user and run the below 2 commands
sudo su - = switch to root user
curl -s https://packages.cloud.google.com/apt/doc/apt-key.gpg | apt-key add -
cat <<EOF >/etc/apt/sources.list.d/kubernetes.list
deb http://apt.kubernetes.io/ kubernetes-xenial main
EOF

sudo apt update = will cache the kubernetes repositories
sudo apt install -y kubelet kubeadm kubectl

#3 enable external dns
goto /etc/systemd/resolved.conf
DNS=8.8.8.8 = google dns
service systemd-resolved restart

Now let us setup kubernetes master/Controlplane. Run the below commands only on KubeMaster node only.

#1 kubeadm init = creates or installs or initialize a kubernetes cluster on a machine. we need to run kubeadm init as below

sudo kubeadm init --apiserver-advertise-address=192.168.10.14 --pod-network-cidr=10.244.0.0/16

The above command creates kubernetes master/controlplane, it generates kubeconfig file which we can use it for talking to master thru kubectl

the pod network cidr is the one through which we are asking kubernetes to create a kubernetes cluster network setup with cidr ip address range starts from 10.244.0.0/16. So all pods that are created within the kubernetes cluster will be given the ip address within that range only.

Since the pods are running a different network, either the kubernetes master node or worker nodes cannot directly access the pods or any of kubernetes objects. kubernetes pods/objects are isolated from the world.
Any 2 pods within the Kuberneters cluster can communicate with each other since those are all on the same cluster network.

The cidr notation that is supported by most of cni implementations are within the range of /16 subnetmask only. This has been enforced due to the below reason.

In one Kubernetes Cluster we can add 5000 worker nodes. and in a production like deployment it is recommended to have 5 worker nodes minimum.
within the kubernetes cluster of 5000 worknodes we can run 150000 pods

on-premise we have to add = 5 nodes
but in aws eks cloud environments we can configure nodepool and can specify to start with 2 and a max 10
2 ec2 instances will be provisioned and added as work nodes
there after based on the demand eks cluster will add the worknodes automatically based on the load upto at max 10 as specified.

/etc/kubernetes/admin.conf = generates the cluster information.
The kubectl looks for kubeconfig file under default location as $HOME/.kube/config

after running the kubeadm init command it generates the output with kubectl setup and kubeadm join
#1 kubectl join commands are shown below.
mkdir -p $HOME/.kube
sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
sudo chown $(id -u):$(id -g) $HOME/.kube/config

$HOME/.kube/config
----
apiVersion: v1
clusters:
- cluster:
certificate-authority-data:key
server: https://192.168.10.14:6443
name: kubernetes
contexts:
- context:
cluster: kubernetes
user: kubernetes-admin
name: kubernetes-admin@kubernetes
current-context: kubernetes-admin@kubernetes
kind: Config
preferences: {}
users:
- name: kubernetes-admin
user:
client-certificate-data: key
client-key-data: key

run these commands to configure kubectl with kubeconfig file to connect to kubernetes cluster.

we can verify kubectl setup by running the below command
kubectl get pods -o wide --all-namespaces

*********************** DONT RUN KUBEADM JOIN untill we setup cni networking
****
#2 after running the above command it generates in the output kubeadm join with token sha as below. we need to run this on worker nodes to add him to the master.
kubeadm join 192.168.10.14:6443 --token 2lrfll.2xvu7u5xi18djdfb \
--discovery-token-ca-cert-hash
sha256:53f41da08dd2acca9ad1eb61bc0133bfc919eafd2c4d4b28d16edc45981e8706

after 1-2 hours the above kubeadm join command will be expired, to regenerate in adding workernodes later we have to use the below command
kubeadm token create --print-join-command

we can see the existing tokens in the list
kubeadm token list
dont add worker nodes to the master now, before this we need to install/enable cni network.

#3 fix kubelet ip by changing kubelet.service.d/10-kubeadm.conf (all the nodes)
sudo vi /etc/systemd/system/kubelet.service.d/10-kubeadm.conf
"Environment KUBELET_EXTRA_ARGS=--node-ip=192.168.10.14"

#4 download kube-flannel.yml file for enabling cni network of implementation flannel. (only on master)

curl https://raw.githubusercontent.com/flannel-io/flannel/master/Documentation/kube-flannel.yml -o kube-flannel.yml
modify the below snippet as shown
net-conf.json: |
{
"Network": "10.244.0.0/16",
"Backend": {
"Type": "vxlan"
}
}
kubectl create -f kube-flannel.yml

sudo apt install etcd-client
sudo systemctl daemon-reload
sudo systemctl restart kubelet

************************ at the last install multicaste package on all the nodes ***************
sudo apt install -y avahi-daemon libnss-mdns

----
worker node setup
#1 fix kubelet ip by changing kubelet.service.d/10-kubeadm.conf

sudo vi /etc/systemd/system/kubelet.service.d/10-kubeadm.conf
"Environment KUBELET_EXTRA_ARGS=--node-ip=workernode ip"

#2 run kubeadm join
sudo kubeadm join 192.168.10.14:6443 --token 2lrfll.2xvu7u5xi18djdfb \
--discovery-token-ca-cert-hash
sha256:53f41da08dd2acca9ad1eb61bc0133bfc919eafd2c4d4b28d16edc45981e8706

sudo systemctl daemon-reload
sudo systemctl restart kubelet

************************ at the last install multicaste package on all the nodes ***************
sudo apt install -y avahi-daemon libnss-mdns
----
after setting up master and worker nodes do vagrant reload

we can check how many nodes are there with kubernetes cluster using below command
kubectl get nodes

we can check the status of the workernodes and thier readiness probe by listing all the pods as below.
kubectl get pods -o wide --all-namespaces

```
NAMESPACE    NAME                          READY   STATUS    RESTARTS   AGE   IP            NODE
NOMINATED NODE   READINESS GATES
kube-system   coredns-558bd4d5db-2p4vk          1/1     Running   1          22h   10.244.0.207
kubemaster   <none>           <none>
kube-system   coredns-558bd4d5db-p7qgz          1/1     Running   1          22h   10.244.0.208
kubemaster   <none>           <none>
kube-system   etcd-kubemaster                   1/1     Running   2          22h   192.168.10.14
kubemaster   <none>           <none>
kube-system   kube-apiserver-kubemaster         1/1     Running   2          22h   192.168.10.14
kubemaster   <none>           <none>
kube-system   kube-controller-manager-kubemaster   1/1     Running   2          22h   192.168.10.14
kubemaster   <none>           <none>
kube-system   kube-flannel-ds-n2qpt             1/1     Running   2          22h   192.168.10.14
kubemaster   <none>           <none>
kube-system   kube-flannel-ds-sbpkm             1/1     Running   0          70s   192.168.10.16
workernode2   <none>           <none>
kube-system   kube-flannel-ds-xjlvd             1/1     Running   2          22h   192.168.10.15
workernode1   <none>           <none>
kube-system   kube-proxy-5txdk                  1/1     Running   2          22h   192.168.10.14
kubemaster   <none>           <none>
kube-system   kube-proxy-p57zh                  1/1     Running   0          70s   192.168.10.16
workernode2   <none>           <none>
kube-system   kube-proxy-v5x7s                  1/1     Running   2          22h   192.168.10.15
workernode1   <none>           <none>
kube-system   kube-scheduler-kubemaster         1/1     Running   2          22h   192.168.10.14
kubemaster   <none>           <none>
```

To remove cluster setup
sudo kubeadm reset = delete the kubernetes cluster install.
sudo rm -rf $HOME/.kube
sudo rm -rf /etc/cni

to remove worker node on cluster
kubectl delete node nodename1
sudo kubeadm reset
sudo rm -rf $HOME/.kube
sudo rm -rf /etc/cni

```
curl https://rawgit.com/coreos/flannel/master/Documentation/kube-flannel.yml \
| sed "s/amd64/arm/g" | sed "s/vxlan/host-gw/g" \
> kube-flannel.yaml
```

jfrog repository username/password= admin/Jfrog#123

#1 to connect to jfrog repository from docker client we need to configure on all the 3 machines.
create a file daemon.json

```
sudo vi /etc/docker/daemon.json
{
"insecure-registries" : ["reg.rconnect.org"]
}
```

after adding the file we should restart the docker

```
sudo systemctl restart docker
```

We are going to write the code on Kubernate Master from the local windows environment.
#1 In VS Code Editor install
Remote SSH plugin

#2 From the leftside navigation bar select bottom 2 Remote Explorer
In the dropdown menu select SSH Targets

#3 configure ssh targets
create a ssh_config file under c:\Users\Sriman\.ssh\config with the below contents

config

```
Host rconnect
HostName 192.168.10.14
User vagrant
Port 22
IdentityFile D:
\work\master\devops\20200610\kubecluster\.vagrant\machines\kubemaster\virtualbox\private
_key
```

What is Kubernetes?

Kubernetes is an opensource container orchestrator engine. With the adoption of docker and microservices based application development,
how do we manage the container infrastructure?
Managing the containers on an infrastructure and monitoring them is very difficult. This is where kubernetes container orchestrator comes in, it takes care of deploying multiple containers either by themself or part of multiple hosts across the infrastructure. It has few features.
1. Provision Hosts
2. Starts Containers on WorkerNode
3. Be able to restart failing containers
4. Ability of linking containers together so that they can communicate (Container Networking)
5. It exposes the containers to the outside world
6. Scaling up/down the cluster
7. Rolling updates

What is a pod?
The smallest unit in the Kubernetes Architecture which abstracts and runs multiple containers inside it, if there any 2 containers whose lifecycle is dependent on each other and shares common resources between them, package or bundle those 2 containers into one single pod and run on kuberneters cluster.

Kubernetes Namespaces
Namespaces are meant for creating naming compartments, so that a resource created in one namespace will be accessible with that namespace and cannot be visible to others. Resources created in a namespace need to be unique within a Namespace. Namespaces cannot be nested. and each resource only can be in one namespace only.

Kubernates administrator has access to all the namespaces of the cluster. he can create kubernetes developers and can grant access to specific namespaces within the cluster, such a way we can control access to the kubernetes cluster resources.

How to see all the namespaces in the kubernetes cluster?
```
kubectl get namespace
```

By default as part of the kubernetes install, it will create 4 namespaces
1. default = by default is empty and all the objects the user create will fall into default namespace. For most of the cluster install default namespace is sufficient.
2. kube-system = the namespace for objects created by kubernetes system (api-manager, controller-manager etc)
3. kube-public = by default it doesnt have any objects, but the objects placed in kube-public namespace can be accessible to anyone without authentication
4. kube-node-lease = This namespace for the leased objects associated with each node

How to create our own namespace in Kubernetes cluster?
```
kubectl create namespace <namespaceName>
```

How to see the namespace in which we are in?
```
kubectl config view –minify | grep namespace:
```

How to switch to another namespace from default namespace?
```
kubectl config set-context –current –namespace=<namespacename>
```

Multiple Clusters

Lets say within our organization we have multiple kubernetes clusters are available/setup. For eg.. 1 for development cluster and other for production cluster. we would want from workstation connect to these clusters using kubectl command to manage the Resources across these clusters.

How can we configure kubectl to switch between the clusters?
We need to create our own kube configuration defined with multiple clusters and their contexts as shown below.

```
mkdir -p $HOME/kubeenvs
touch config

apiVersion: v1
kind: Config
preferences: {}

clusters:
- cluster:
name: development
- cluster:
name: production
```

users:
- name: developer
- name: productionengineer

contexts:
- context:
name: development
- context:
name: production

goto $HOME/kubeenvs
kubect config --kubeconfig=config set-cluster development --server=kubermasternode:port --insecure-skip-tls-verify
it will add server ip to the config file.

by default kubectl looks for config file under $USER/.kube/config. but if we want kubectl to use the current config file we need to set KUBECONFIG env variables pointing to the newly created config file.

export KUBECONFIG=/home/vagrant/kubenvs/config
now kubectl refers to the current config file we have created.

How to switch between clusters by chaning the context?
kubectl config set current-context kubernetes-admin@kubernetes

kubectl config = is a command for changing the properties of the config file.

Kubernetes Objects

There are several types of resources are there in kubernetes cluster as described below
pods
deployments
replicaSets
daemonSets
job
service

How are these represented and maintained in Kubernetes Cluster, that is where objects comes into picture.
Kubernetes objects are persistent entities in the Kubernets System. Kubernetes uses these objects to represent the state of the cluster.
What does a kubernetes objects represents:-
- What containerized applications are running (on which nodes)
- The Resources available to those application
- Policies binding the application

A kubernetes object is a record of intent once created the kubernetes system will constantly work to ensure the object exists. By creating an kubernetes object, we are telling to the kubernetes system what we want on the kubernetes cluster.

pod spec (yaml) -> kubectl (config) -> kubemaster(cluster) -> object (etcd) (persisted)

These objects can we creates using Resource spec:
Resource spec is an .yaml in which you need to define the kubernate object with attributes and values, and pass this yaml file asking kubernetes cluster to create and manage the object.

Required Fields:-
apiVersion - which version of the kubernate object you are using
kind = type of the object
metadata = used to define lables
namespace = this object should be created under which namespace
spec = desired state of the object

How to create and manage these objects in Kubernetes Cluster?
The kubectl command supports different ways of create and manage the kubernates objects. There are 3 technics are there.

1. Imperative commands = kubectl has provided handful commands through which we can creates various different types of objects by passing attributes of objects as parameters to the command. we can avoid writing .yaml file.

kubectl run apache2 --image=apache2 --exposedPort=8080:8080
kubectl create deployment apache2 --image=apache2

advantages:-
1. quick at development
dis-advantages:-
1. we dont have object definitions stored to create in other env

2. Imperative Object configuration = In imperative object configuration, the kubectl we specify the oepration (create, replace, delete). the object information is described in an .yaml file and passed to kubectl.
kubectl create -f apache2-pod.yaml
kubectl delete -f apache2-pod.yaml

3. Declartive Object configuration = there can be group of objects to be created on kubernetes cluster to achieve the desired state. for each of these objects we need to write spec (.yaml) files and should apply in a specific order. instead group them into directory like config/ and pass that to the kubectl along with operation to be performed by passing directory as an input
kubectl apply -f configs/
kubectl delete -f configs/

There are different types of Resources are there in Kubernate Cluster, among them to the basic is Pod.

Pod = Smallet unit of entity in kubernetes cluster wherein one or more containers are packaged and ran together.

How to run a Pod?
To run a pod on the kubernetes cluster we need to describe the information about the containers and images with which the containers has to be created. In a pod containerized applications are running we need describe the ports on which the containerized application is running as well.

healthcare application (web application):-

package this application as an docker image (delivered by the developer)

Dockerfile
```
FROM reg.rconnect.org/k8s-virtual-repo/ubuntu:21.04
ENV JAVA_HOME=/u01/middleware/jdk-11
ENV PATH=${PATH}:${JAVA_HOME}/bin
RUN mkdir -p /u01/middleware
RUN mkdir -p /u01/app

WORKDIR /u01/middleware
ADD https://download.java.net/openjdk/jdk11/ri/openjdk-11+28_linux-x64_bin.tar.gz .
RUN gunzip openjdk-11+28_linux-x64_bin.tar.gz
RUN tar -xvf openjdk-11+28_linux-x64_bin.tar
RUN rm -rf openjdk-11+28_linux-x64_bin.tar

WORKDIR /u01/app
COPY target/covido-1.0.jar .
CMD [ "java", "-jar", "/u01/app/covido-1.0.jar"]
```

healthcare-pod.yaml
```
-------------------
apiVersion: v1
kind: Pod
metadata:
name: healthcarepod
spec:
containers:
- image: reg.rconnect.org/k8s-virtual-repo/healthcare:1.0
name: healthcare
ports:
- containerPort: 8081
name: http
protocol: TCP
```

What is Kubernetes, why do we use it?
kubernetes is an container orchestrator takes care of provisioning nodes, create/run containers and manages/monitors them with in a cluster.

Features:-
1. provision a node
2. create and run container
3. scale up/down
4. rolling updates
5. networking for containers
6. load balancing

Kubernetes Namespaces
Namespaces are used for creating naming compartments, so that objects created in one namespace will not be visible in others.

There are 4 namespaces created by default as part of kubernetes install
1. default = default namespace (no objects)
2. kube-system = all kubernetes system objects are created under this namespace
3. kube-public = global access
4. kube-node-lease = for leased objects

```
kubectl get namespace
kubectl create namespace <namespacename>
```

kubectl config = used for modifying the kubeconfig file attributes
```
kubectl config set-context –current –namespace=rconnectns
```

kubectl config view = show me the current kubeconfig am using, here we can grep namespace
```
kubectl config view | grep namespace:
```

How to manage multiple kubernetes clusters?

create our own kubeconfig file defining multiple clusters information.
```
apiVersion: v1
kind: Config
preferences: {}
clusters:
- cluster:
name: development
- cluster:
name: production

users:
- name: developer
- name: productionengineer

contexts:
- context:
name: developer@development
cluster:
user:
namespace:
- context:
```

name: productionengineer@production
cluster:
user:
current-context: developer@development

kubectl config --kubeconfig=config set-cluster developer --server=http://host:port --insecure-skip-tls-verify

What is the default config the kubectl uses?
$USER/.kube/config is the default kubeconfig file it uses

how to switch to an different kube configuration?
export KUBECONFIG=location of the file

Kubernates objects

kubernetes objects are persisted entities in kuberneters system. they represent
- resources
- containerized applications
- policies
they define the state of the cluster. by creating an object in kubernetes cluster we define the desired state of the cluster.

There are 3 ways we can create and manage a kubernetes object
1. imperative commands = kubectl has provided commands to directly create objects

2. imperative object configuration = we define the object information in terms of .yaml file declaratively.

3. declarative object configuration = place all the resource specs in a directory and pass as an input to kubectl rather than individual spec.

What are the Required Fields in a Resource declaration?
apiVersion:
kind:
metadata:
namespace:
spec:

What are pods, what is the purpose of them?
A pod in kubernetes is a smallest unit , in which one or more containers are defined and executed together. There are some containers that dependent on each other interms of lifecycle and resources being used, in such case we can combine and run together those containers in a pod.

We can create a Pod spec defining the details of container image we want kubernetes to run on the cluster as below.

healthcare-pod.yaml
----
apiVersion: v1
kind: Pod
metadata:

```
name: healthcarepod
spec:
containers:
- image: reg.rconnect.org/k8s-virtualrepo/healthcare:1.0
name: healthcare
ports:
- containerPort: 8081
name: tomcatPort
protocol: TCP
```

we create a pod spec and when apply the pod spec only one pod will be created on the cluster. whenever a pod has been created, kubernetes cluster will allocates an ip address for the pod from pod-cidr range. By default we cannot access the pod outside the cluster, it can be used by other pods running within the same cluster across any of nodes as well.

Pods are temporary, when those gets created an ip address will be assigned, upon termination/ destroyed the ip addresses assigned to the pod are released. everytime when we create a pod, a new ip address within the range of pod cidr will be allocated.

we can create pod using the below command if we have pod spec.
```
kubectl create -f healthcare-pod.yaml
```

we can get the status of the pod using
```
kubectl get pods -n=rconnectns
```

we can get more details of the pod using
kubectl get pods -o wide = -o stands for output | wide = more information
from here we can get ip address of the pod and node on which it is scheduled to run

we can get details of the pod and the events happening write from the begining to its creation using
```
kubectl describe pod podname
```

we can see the logs emitted by the pod application using
```
kubectl logs podname
```
kubectl logs -f podname = equal to tail on the pod log

How to delete a pod?
```
kubectl delete -f healthcare-pod.yml
pod "healthcarepod" deleted
```
(or)
```
kubectl delete pod healthcarepod
```

----
if we want to run multiple instances of the pod, we cannot achieve it, we need to create multiple pod secs and manage it. From this we can understand we cannot run multiple replicas of a pod using pod spec.
In case if we want multiple replicas, it is difficult to achieve it through pod spec, we need to create multiple pods specs representing the container information

Pods will not survive crash. in the event of a crash of a pod, kubernetes will not try to bring up the pod back, it is lost.

Pod will not retain the data that has been generated durings its execution once it has terminate.

When we create pod, during the execution due to several reasons like thread stuck issues or heap memory issues due to which an application running within the pod goes non-responsive and may not work. but still the health pod will be showing running only within kubernetes cluster. how to handle these situations.

Kubernetes has provided livenessProbe for a pod, apart from this kubernetes has provided another probe called readinessProbe.
1. livenessProbe = once the pod is ready and available for serving the requests, at regular interval time kubelet process will try to verify the pod state is healthy or not by doing an livenessProbe we configured. if livenessProbe has been failed for some threadshold number of times kubernetes marks the pod as unhealthy and will be destroyed automatically.

2. readinessProbe = In general once the pods are created using the pod spec looks the pod is ready to be used, but in practical, they might be an application server or a program still under initialization state and trying to startup, at this moment if kubernetes makes the pod live to world, then all the requests routed during the time of startup of the application will fail. So how to solve this problem?

readinessProbe has been introduced = we can configure an readinessProbe in pod spec letting kubernetes use it to verify whether the pod (underlying application with in the pod) is fully initialized or not before marking it to be Running. So once the pod readinessProbe was success then only kubernetes will expose it. There is a threadshold limit in no of times readinessProbe might fail, in case the pod will marked as failed and terminated.

How to configure the readinesss and livenessProbe?
The developer while developing the application will configure http endpoints in case of http application exposing to check readinessProbe and livenessProbe. Now in the pod spec kubernetes developer has to configure the endpoints to let the kubernetes probe or check for it as shown below.

healthcare-pod.yaml
```
apiVersion: v1
kind: Pod
metadata:
name: healthcarepod
spec:
containers:
- image: reg.rconnect.org/k8s-virtual-repo/healthcare:2.0
name: healthcare
ports:
- containerPort: 8081
name: http
protocol: TCP
livenessProbe:
httpGet:
path: /actuator/health/liveness
port: 8081
initialDelaySeconds: 15
timeoutSeconds: 5
failureThreadshold: 3
readinessProbe:
httpGet:
path: /actuator/health/readiness
```

```
port: 8081
initialDelaySecond: 10
timeoutSeconds: 5
failureThreadshold: 3
```

Resource declarations
While running a pod the application within the pod requires some amount of Resources like cpu and memory, unless we help the kubernetes in understanding the resources required, kubernetes may not select an appropriate worknode suitable for running the pod.

To help kubernetes to understand Resource requirements for a pod application to run we need to write Resource declaration in the pod.
There are 2 types of Resource declarations are there
1. requests = the amount of capacity the pod requests from the kubenetes to run, minimal capacity required for the pod to run. So that while the Scheduler choosing the worker node, checks the node capacity does it can allocate requested resources are not based on the pod will be scheduled on the worker node.

2. limits = max capacity pod can consume during its runtime. if the pod is requesting more than the limit capacity and if the resources are available on that worker node, the kubelet allocates in case if the resources are not available on the pod beyond the limit the pod will be terminated, so that scheduler can schedule pod on the higher capacity node.

```
containers:
- image: healthcare:2.0
name: healthcare
ports:
- containerPort: 8081
name: http
protocol: TCP
resources:
requests:
cpu: "500m"
memory: "128Mi"
limits:
cpu: "1000m"
memory: "256Mi"
```

> Create a Role (eksclusterrole)
EKS-Cluster->permission
AmazonEKSClusterPolicy

IAM Authenticator
Kubectl
Aws CLI

```
aws eks --region us-east-1 update-kubeconfig --name eks-Cluster
```

EKS Worknodes
IAM
-> Create ROLE eks
amazonekscnipolicy
amazoneksworknodepolicy
ec2containerpolicy