PLEASE JOIN HERE FOR FREE MATERIALS

LINK FOR TELEGRAM: CLICK HERE

LINK FOR INSTAGRAM : CLICK HERE

LINK FOR PAID MATERIALS: CLICK HERE

02-02-2-23

Kubernetes (k8s)
Kubernetes is an container cluster manager used for managing, monitoring, scaling the containerized applications on a clustered environment.

Kubernetes is used for deploying and running microservices based applications, to understand the crux and the way kubernetes works,
it would good to know the basics of microservices application and their architecture

There are 2 architectures in which an application can be build
1. Monolithic application architecture
2. Microservices application architecture

#1. Monolithic Application architecture
The Enterprise application which consists of several modules are built into one single sourcecode project and will be build and
packaged into one single deployable artifact is called "Monolithic application architecture"

There are few characteristics of Monolithic application:
1. the entire application is built into one single sourcecode, that includes all the modules or functional areas
2. A team of developers will be working across the functional areas or modules of the project
3. The whole application will be managed in single sourcecode management repository
4. The build system will produce an single deployable artifact
5. the whole application will be deployed and ran on a single enterprise application server environment

advantages:-
1. everyone in the team knows all the modules/functional areas of the system, since everyone works on end-end of the project
2. easy to build, package and deploy the application
3. scalability can be achieved easily

dis-advantages:-
1. since is an enterprise large application that comprises of lot of functionalities or modules and it is build into single sourcecode,
 the developers often find it very complex to work with entire system.
2. since the whole application is built into one single project
   2.1 To build the project it takes lot of time
   2.2 IDE are overloaded with lot of sourcecode files
   2.3 application servers takes more time in starting up

-===
===

03-2-2-2023

Kubernetes
Kubernetes is an container cluster manager, that takes care of running, monitoring and managing the containers on the cluster network of machines

There are 2 architectures on which applications are build
1. Monolithic application architecture
2. Microservices application architecture

1. Monolithic Application Architecture
An Enterprise large application that comprises of multiple

functional areas/modules are build together into one single
sourcecode and is packaged into an single deployable artifact is called "Monolithic application"

Characteristics:
1. The enterprise application with all the modules are functionalities are built into one single sourcecode project
2. The Team of developers will working across all the modules of the project
3. A single sourcecode repository will be used for versioning and collaborating the development
4. only one single deployable artifact will be produced out of the build process
5. The application will be deployed on enterprise application server

advantages:
1. anyone in the project can work on any of the modules of the system
2. easy to build, package and deploy the application
3. scalability is easy to achieve in monolithic applications

dis-advantages:-
1. since the application is very huge and is build into one single sourcecode, developers often find it very complex to understand
 and work on the entire system
2. because of huge sourcecode, the integrated development environments (ide) will be overloaded and quickly goes un-responsive
3. building and packaging of the application becomes heavy and takes more time, due to which development will be impacted
4. application servers takes more time in deploying and starting the application, because of large in nature, due to which
the productivity of the developer will be degraded

#2. Microservices application architecture
The Enterprise large application that comprises of several functional areas or modules are broken down into smaller services

===
04-02-2023

Microservices Application Architecture

The Enterprise large scale application that comprises of multiple functional areas or modules is broken down into smaller microservice applications which are independently deployable an loosely coupled applications are called "Microservices"

Characteristics:-

1. Each Microservice application is built into its own sourcecode project which are independent of other modules/services of the system
2. Each Microservice application/project has its own sourcecode repository
3. Each Microservice has its own database schema into which those performs persistence operations
4. Each Microservice application is built into an indepently deployable artifact, that is deployed on its own server runtime
5. Each Microservice is build by a team independent from other teams

===

05-02-2023

Kubernetes (k8s)
Kubernetes is an container cluster manager, it takes care of distributing containerized applications on a cluster of computers, scheduling, monitoring and scaling up the containerized applications

while we are working on containerized applications, running them on a cluster has lot of problems:

1. distributing the containerized applications on a network of computers is very difficult
2. Keeping tracking of which containers are running on which nodes of the cluster is very difficult
3. patching and upgrading the containers are very difficult, we need to identify on which nodes of the cluster these containerized applications are running, we need to manually ssh onto those nodes and stop the running containers and rollout the newer versions of them. and we need to determine the upgrade strategy to avoid downtime
4. monitoring and replacing an container in case of crash is very difficult
5. keeping track of the resource capacity and their utilization, and incase if a container is consuming huge system capacity on a node, which doesnt have sufficient resources, then migrating the container to another node that has enough capacity is very difficult
6. scaling up the applications on network cluster is difficult

To overcome the above problems/difficulties in managing and running the containerized applications on a network we need kubernetes.

Kubernetes Architecture

Kubernetes is an container cluster manager that takes careof scheduling, monitoring and managing the containerized applications on a network cluster of machines

The Kubernetes has 4 major components in it
1. Master Node
2. Worker Node
3. Kubectl
4. etcd

==
==
06-02-2023

Kubernetes (k8s)
Kubernetes is an container cluster manager that takes care of scheduling, monitoring and managing the containers over the cluster of machines.

There are 4 major components are there in kubernetes
1. Master Node or Control Plane
2. Worker Node
3. Kubectl
4. etcd

#1. Kubernetes Master Node
Kubernetes Master is the central component of the kubernetes cluster, he is even called as "Control Plane", the job of scheduling the pods,
monitoring and managing them across the worker nodes of the cluster will be taken care by "Kubernetes Master".
There are 3 sub-components are there in MasterNode
1.1 Api Manager
1.2 Scheduler
1.3 Controller Manager

1.1 Api Manager
The Api manager is the http endpoint which acts as an face or front-end to the kubernetes cluster/master. To perform any action or
an operation on the kubernetes cluster, the clients or we need to talk through the api manager only.
The Api manager is an endpoint built on http protocol and exposes itself to the world letting them interact with Kubernetes Master.

There are many ways we can talk to api manager
1. The Kubernetes has provided an CLI tool called "kubectl" using which we can talk to the api manager
2. we can invoke the httpendpoints exposed by the api manager directly
3. there are api libraries provided by the kubernetes itself, using them we can connect and interact with api manager

The api manager performs 2 major activities
1. the api manager upon receiving the request will authenticate whether user who is sending the request is an valid user or not
and checks whether the user is authorized to perform the requested operation or not

2. in addition to the above, it validates the kubernetes spec that is send aspart of the request before passing it to the scheduler for performing further operation

1.2 Scheduler

upon receiving and validating the request, the api manager to further process the request it handovers the request to the Scheduler.

The Scheduler is responsible for scheduling an pod for execution on the worker nodes of the cluster. The Scheduler will talks to the kubelet process that is running on each worker node of the cluster, checks does the workernode has enough capacity available for running the pod or not, if not available goes to the next worker node of the cluster until it finds one

upon identifying an workernode with enough capacity, the scheduler will handovers the request to the kubelet process of the workernode asking to bringup the pod on that node by allocating the requested resources

1.3 Controller Manager

The Controller Manager is an daemon/background process that is running on the kubernetes manager/control plane and ensures to bring the cluster to the desired state. It continous monitors and interacts with the kubelet process of the workernodes to identify the system is in desired state or not

There are 5 types of controllers are there

1. ReplicaSet

ReplicaSet controller ensures the desired number of replicas of a pod is distributed across different nodes of the cluster.

In the event of a crash, the ReplicaSet controller replaces that with an running pod on the cluster.

The ReplicaSet controller ensures the desired number of replicas are always running the cluster

2. DaemonSet

DaemonSet ensures a pod is running on all the nodes of the cluster. Incase if a new workernode has been added to cluster, the daemonset ensures the pod is broughtup onto the new worker node aswell.

3. DeploymentSet

DeploymentSet controller helps us in upgrading or patching the old pods with newer versions by supporting different deployment strategies

4. Service

service is an controller that discovers the running pods on the nodes of the cluster based on labels and register with them, so that it can distribute the request to the pod application by loadbalancing.

5. Job
Job controller helps us in running a job or script on a node of the cluster to
perform one-time operation

===
===
07-02-2023

Kubernetes (k8s)
Kubernetes is an container cluster manager that takes care of scheduling,
monitoring and managing the containers on the cluster of computers.

There are 4 components are there in kubernetes
1. master or control plane
2. workernode
3. kubectl
4. etcd

1. Master Node or Control Plane
it is the central component of the kubernetes cluster that takes of scheduing,
managing and monitoring the pods across the workernodes of the cluster
There are 3 more components are there in Master Node
1.1 Api Manager
acts an front-end or interface in communicating with the master or control plane,
api manager is build as an
rest/http endpoint through which people can interact with kubernetes master.
There are 2 major activities api manager performs
1.1.1 authentication and authorization
1.1.2 validating the request (specfile)

1.2 Scheduler
upon receiving the request, the api manager forwards the request to scheduler. The
scheduler does the job of
communicating with the kubelet process of the workernode across the cluster to
identify an workernode suitable
enough in running the pod and handovers the job of bringing up the pod on the
workernode to the kubelet process

1.3 Controller Manager
The Controller Manager is an daemon process, that always ensure for bringing the
cluster to the desired state.
There are 5 types of controllers are there
1.3.1 ReplicaSet Controller = desired no of replicas of a pod are running on the
cluster will be taken care by ReplicaSet controller
1.3.2 DaemonSet Controller = ensures an pod is running always across all the nodes
of the cluster
1.3.3 Service Controller = discovers the running pods on the cluster based on the

labels and groups them and
expose them over the cluster, it takes care of loadbalancing the request received across the pods registered with that service
1.3.4 DeploymentSet Controller = used for upgrading, patching the pods based on upgrade/rollout strategies
1.3.5 Job Controller = helps us in running one-time operation on the nodes of the cluster

#2. Worker Node
Kubernetes Worker Node is an physical server/virtual machine/cloud instance that is attached to the kubernetes
master/control plane on which the pods are scheduled for execution. On a workernode there could be one or more pods be scheduled for
execution based on the capacity of the worker node

There are 3 major components to be installed on the each workernode in order to register and schedule pods on them
1. container runtime
A workernode must be installed with an containerization engine, so that the containerized applications can be executed on them.
For eg.. we can install docker on the workernodes to run docker containers on them

2. kubelet
kubelet is an process that runs on each workernode of the cluster. The Kubelet process acts as an agent,
in letting the controlplane or master interact with the Workernodes of the cluster.
The Master Node schedules a pod for execution on the cluster by handovering the job to kubelet process only.
Additionally Kubelet process performs various activities like
1. kubelet gathers the information about the running pods and their status and reports them to the control plane or master whenever requested
2. The job of running the pods and bringing them up on a workernode is taken care by kubelet process only
3. through the help of kubelet process, the control plane can determine which workernode is suitable for running or scheduling a pod for execution

3. kubeproxy
kubeproxy enables the traffic to the external network for a pod in the cluster

#3. Kubectl
Kubectl is an cli tool provided by the kubernetes through which we can communicate with the controlplane or master.
it helps us in administering, monitoring and managing the kubernetes cluster

#4 etcd
etcd is an key/value pair database where all the kubernetes objects information will be stored on the etcd database only

===

08-02-2023

There are multiple ways we can setup an Kubernetes Cluster
1. on-premise with physical server machine
2. through virtual machines
3. Minikube
4. AWS EKS cluster

#4. AWS EKS Cluster
To work with AWS EKS Cluster we need to have an AWS Cloud Account, if not create one.

The aws cloudplatform has provided an service called EKS stands for "Elastic Kubernetes service",
 it is an managed service provided by aws platform to host kubernetes on aws.
The AWS Cloudplatform itself takes care of provisioning the master/control plane, workernodes and
setup the cluster with CNI network and install necessary components like containerization engine, kubelet, kubeproxy etc
There are lot of advantages of using EKS Cluster over the one-premise
1. we dont need to manually setup, configure the kubernetes cluster, rather with an click of a button
the AWS itself takes care of provisioning the kubernetes for us
2. Monitoring the kubernetes cluster and keeping track of the health and incase if any of workernodes
are crashed those are replaced with the healthier nodes will be taken care by the aws itself
3. The high availability of the cluster will be guaranteed since the workernodes are distributed across
the availability zones of the vpc, in addition we can take the advantages of making the application accessed to customer with low-network latency
4. The AWS Cloud platform itself takes care scale-out/scale-in the cluster capacity/size based on the
load of the cluster so that we never run out of cluster capacity
5. The HA of the master/control plane will be takencare by the aws cloud platform itself

How many ways we can provision an EKS Cluster on AWS Cloud?
There are 3 ways we can setup EKS Cluster on AWS Cloud
1. aws management console = we can manually configure and setup the EKS Cluster through console
2. ekscli interface = ekscli is an command-line tool provided by aws team, using which we can quickly setup the eks cluster on the cloud
3. through automation using terraform or ansible

Let us understand how to setup an EKS Cluster
#1. we need an vpc "hondaekscluster"
vpcname: "hondaekscluster"

cidr: 10.0.0.0/16

#2. now we need to create the subnets
There are 3 ways we can host the eks cluster
1. public = we can create the worker nodes on public subnet of the vpc = only for development purposes we can use but should avoided
2. private & public = eks masternode / controlplane will be provisioned on public subnet and the workernodes are provisioned on private subnet.
Typically used in organization env, where teams should able to access the kubernetes cluster for deploying and running
the applications on the cluster (non-production env)
3. private subnet = both master/workernodes should be provisioned on private subnet only (recommended for production usage).
To manage the cluster we need to use jumpbox

#3. to provision an eks cluster we need atlest 2 subnets under 2 different availability zones of an vpc (HA)
#4. when we are provisioning the workernodes under private subnet of an vpc, workernodes will not have access to the public network,
so they cannot pull the docker images from dockerhub to run pods on the container, so we need to must and should setup ecr (elastic container registry)
provided by aws cloudplatform, and publish the docker images on the ecr registry, so that the workernodes can pull the docker images from aws ecr.

===

09-02-2023

#1. setup the workstation
1. install openjdk-11-jdk

```
sudo apt update -y
sudo apt install -y openjdk-11-jdk
```

2. install apache maven

```
sudo apt install -y maven
```

3. install vscode
download the vscode binary ".deb" from the vscode downloads, by default it will be downloaded into ~/Downloads directory

```
cd ~/Downloads
sudo apt install -f ./code_....deb
```

4. docker
4.1

```
sudo apt install -y ca-certificates curl gnugp lsb-release
```

4.2 download and add the gpg docker key

```
sudo mkdir -m 0755 -p /etc/apt/keyrings
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o /etc/apt/keyrings/docker.gpg
```

4.3 add the docker registry into ubuntu sources.list.d

```
echo \
  "deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker.gpg] https://download.docker.com/linux/ubuntu \
  $(lsb_release -cs) stable" | sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
```

4.4 update the registry

```
sudo apt update -y
```

4.5 install docker

```
sudo apt install -y docker-ce
```

4.6 grant sriman access to the docker by adding him to the docker group

```
sudo usermod -aG docker $USER
```

exit the terminal and re-enter or restart

run the below command to verify sriman has access to docker

```
docker image ls
```

5. create the IAM User on aws cloud account

5.1. group: hondaopsgroup
attached policies:

1. AmazonEC2FullAccess
2. AmazonEKSClusterPolicy
3. AmazonEKSWorkerNodePolicy
4. AmazonEC2ContainerRegistryFullAccess
5. AmazonEKSServicePolicy
6. AmazonEKS_CNI_Policy

5.2 create a user and add him to the above group
username: bob
add: hondaopsgroup

5.3 generate the api keys for accessing
goto user, click on security credentials and click on generate accesskeys, once generated download them

5. install AWSCli on workstation

https://docs.aws.amazon.com/cli/latest/userguide/getting-started-install.html

```
curl "https://awscli.amazonaws.com/awscli-exe-linux-x86_64.zip" -o "awscliv2.zip"
unzip awscliv2.zip
sudo ./aws/install
```

6. create aws credentails using awscli tool to access the cloud account resources using cli

```
aws configure
```

prompts for
api key:
access key:
region: ap-south-1
output format: none

===

10-02-2023

#2. How to setup the Elastic Container Registry?
2.1
There are 2 types of Elastic Container Repositories are there
1. private repository = only the IAM user who has neccessary policies attached only can access the repository
2. public repository = any one can access, pull/publish images into the repository

2.2
per each docker image we want to publish we need to create on repository in ecr

2.3 to publish or pull images from these repositories we need to login into ecr registry. and in case of private repository
we need have AmazonEC2ContainerRegistryFullAccess to login or pull or push images

2.4 goto IAM User we have setup earlier and attach a policy to either user level or group level with policy: AmazonEC2ContainerRegistryFullAccess

2.5 goto elastic container registry and click on Get Starter or create repository
1. private repository and enter an name for repository
2. go into the repository and click on View Push Commands for login pull or push instructions

```
aws ecr get-login-password --region <regionName> | docker login --username AWS --password-stdin registryURL
```

How to provision the EKS Cluster?
1.

===
12-02-2023

How to provision an eks cluster?
There are 2 parts are there in setting up an eks cluster on aws cloud platform
1. control plane or master node
2. worker nodes

There are 3 topologies where we can choose one of them in setting up the eks cluster
1. both master/workernodes on public subnets = not recommended since the whole cluster is exposed to the world and poses security

2. master on public subnet and workernodes on private subnet = usually used in organizations
allowing the team of people to manage the cluster directly (not recommended for production usage)
3. both master/workernodes on private subnet only = highly recommended for production usage

we wanted to setup the eks cluster using option-#2

#1. create the vpc
vpcname: hondaeksvpc
cidr: 10.0.0.0/16

#2. create 2 public subnets and 2 private subnets, public subnets for control plane and private subnets for workernodes
we need to create 2 of them for each public/private subnets across the azs of the region, to ensure high-availability
2.1 hondaekspubsn1, 10.0.1.0/24
2.2 hondaekspubsn2, 10.0.2.0/24
2.3 hondaeksprvsn3, 10.0.3.0/24
2.4 hondaeksprvsn4, 10.0.4.0/24

#3. provision internet gateway and attach to the vpc
internet gateway name: hondaeksig
attach to vpc: hondaeksvpc

#4. route the public network traffic through internet gateway by creating routetable
routetable name: hondaigrt
subnet association: hondaekspubsn1, hondaekspubsn2
route: 0.0.0.0/0 -> hondaeksig

provision eks control plane or master node

1. The eks master node or control plane takes care of provisioning, setting up and installing the workernodes based on the configuration,
to let the eks cluster manage the workernode we need to attach an IAM Role to the master node while provisioning. So let us first setup the IAM Role
1.1 goto IAM Policies service and choose create role
1.2 within the create role, select the category type: EKSCluster
1.3 add the policy as EKSClusterPolicy into the Role and create it

2. goto Elastic Kubernetes Service and click on create cluster.
2.1 cluster name
2.2 choose the vpc
2.3 choose the way you want to host the cluster (master/controlplane : public subnet, workernodes: private subnet)
2.4 Attach Role we created above
and click on create cluster

upon creating the cluster, it setup or provision only control plane without workernodes, now navigate into the cluster and click on create nodegroup

setting up workernodes
1. to setup the workernodes we need create an IAM Role that should be attached to the workernode during provision.
goto IAM Policies and choose Role and create new
choose the type: EKSNodeGroupRole
policies:
1. AmazonEKSWorkerNodePolicy
2. AmazonEKSContainerRegistryReadOnly
3. AmazonEKS_CNI_Policy
create role

2. goto the eks cluster we have created above and click on add NodeGroup
we can think of a NodeGroup as equivalent to ASG. The NodeGroup takes care of provisioning the workernodes and attach to the EKS MasterNode.
here we can specify
1. shape of the workernode (t2.micro)
2. min, max, initial workernodes
3. scale-out threshold
4. subnets

The NodeGroup based on the above configuration takes care of provisioning and managing the workers automatically

How to setup the kubectl on ubuntu?
documentation:
https://kubernetes.io/docs/tasks/tools/install-kubectl-linux/#install-using-native-package-management

#1. install pre-requisite

```
1. sudo apt-get update
2. sudo apt-get install -y ca-certificates curl
3. sudo apt-get install -y apt-transport-https
```

#2. add gpgkey

```
sudo curl -fsSLo /etc/apt/keyrings/kubernetes-archive-keyring.gpg
https://packages.cloud.google.com/apt/doc/apt-key.gpg
```

#3. now add the kubernetes repository to the ubuntu sources.d or sources.list

```
echo "deb [signed-by=/etc/apt/keyrings/kubernetes-archive-keyring.gpg]
https://apt.kubernetes.io/ kubernetes-xenial main" | sudo tee
/etc/apt/sources.list.d/kubernetes.list
```

#4. update the repository and install kubectl

```
sudo apt update
sudo apt install -y kubectl
```

How to add the kubeconfig to workstation to access the eks cluster using kubectl run the below command.

```
aws eks update-kubeconfig --region <REGION> --name <ClusterName>
aws eks update-kubeconfig --region ap-south-1 --name HondaCluster
```

How to install kubectl on windows 10/11?
we just need to download the kubectl.exec and place it in the c:\minikube directory
1. goto c:\minikube\ directory, if we dont have create one manually and run the below command on powershell/command-prompt

```
curl.exe -LO "https://dl.k8s.io/release/v1.26.0/bin/windows/amd64/kubectl.exe"
```

2. set the PATH variables pointing to the c:\minikube directory
3. kubectl version

```
C:\Users\Sriman>kubectl version
WARNING: This version information is deprecated and will be replaced with the
output from kubectl version --short.
 Use --output=yaml|json to get the full version.
Client Version: version.Info{Major:"1", Minor:"26", GitVersion:"v1.26.0",
GitCommit:"b46a3f887ca979b1a5d14fd39cb1af43e7e5d12d",
GitTreeState:"clean", BuildDate:"2022-12-08T19:58:30Z", GoVersion:"go1.19.4",
Compiler:"gc", Platform:"windows/amd64"}
Kustomize Version: v4.5.7
Server Version: version.Info{Major:"1", Minor:"26", GitVersion:"v1.26.1",
GitCommit:"8f94681cd294aa8cfd3407b8191f6c70214973a4",
 GitTreeState:"clean", BuildDate:"2023-01-18T15:51:25Z", GoVersion:"go1.19.5",
Compiler:"gc", Platform:"linux/amd64"}
```

How to install minikube on windows 10/11 machine?
Running an kubernetes cluster on a physical server environment or in a virtual machine env requires huge system resources and would take lot of time in setting up on cluster. For local development and experimentation/learning we can use minikube installed

MiniKube is an down-sized version of kubernetes cluster provided by kubernetes team, that takes care of installing and running the kubernetes cluster on virtual machine with onesingle node/virtualmachine as master/workernode itself.
So that we can quickly run the cluster locally and work on local development
You can refer to the minikube start documentation:
https://minikube.sigs.k8s.io/docs/start/

before beginging installation of minikube, download and setup kubectl on the windows 10/11 machine

1. run the below command in powershell or windows command-prompt that downloads the minikube.exe and places under c:\minikube directory

```
New-Item -Path 'c:\' -Name 'minikube' -ItemType Directory -Force
Invoke-WebRequest -OutFile 'c:\minikube\minikube.exe' -Uri
```

```
'https://github.com/kubernetes/minikube/releases/latest/download/minikube-windows-amd64.exe' -UseBasicParsing
```

2. then add c:\minikube\ directory to the system path in environment variables. exit the terminal and reopen

3. minikube start --driver=virtualbox

to verify the minikube cluster status run the below command

```
kubectl get po -A
```

```
C:\Users\Sriman>kubectl get po -A
NAMESPACE     NAME                               READY   STATUS    RESTARTS
AGE
kube-system   coredns-787d4945fb-zm2ch           1/1     Running   1 (25m ago)
28m
kube-system   etcd-minikube                      1/1     Running   1 (25m ago)
28m
kube-system   kube-apiserver-minikube            1/1     Running   1 (25m ago)
28m
kube-system   kube-controller-manager-minikube   1/1     Running   1 (25m ago)
29m
kube-system   kube-proxy-znldh                   1/1     Running   1 (25m ago)
28m
kube-system   kube-scheduler-minikube            1/1     Running   1 (25m ago)
28m
kube-system   storage-provisioner                1/1     Running   1 (25m ago)
28m
```

Kubernetes Dashboard

Kubernetes has provided an management-console or a kubernetes cluster dashboard using which we can access all the kubernetes object through the web console like

- namespaces
- pods
- services
- deployments

etc

we can run the kubernetes dashboard on minikube using

```
minikube dashboard
```

this installs/configures the dasboard on minikube and opens the console on the browser.........

....

13-02-2023

Kubernetes Namespace

Namespaces are used for creating naming compartments or logical grouping of objects/resources on the kubernetes cluster. A kubernetes cluster would be shared across multiple teams to run various different projects on the cluster, so to avoid objects/resources of one project/team are accessed   by others we use kubernetes Namespaces

A kubernetes administrator can create multiple kubernetes users, associate them into groups. create an kubernetes namespace and grant access to the users/groups to those namespaces only.
In such a way per project/application/team, the administrator can create namespace associate users/groups into that namespace allowing them to run, manage modify the objects only within that namespace, so that we can avoid different people of different groups to see or manage resources of others, hence the cluster would be secured and protected.

By default aspart of the kubernetes install there are 4 namespaces are created

1. default   = every object that is created in kubernetes cluster will be placed by default under "default" namespace only. The default namespace is by default empty. all the users/groups of kubernetes cluster has access to the default namespace and should be sufficient for most of the usecases
2. kube-system = all the kubernetes system objects like api manager, scheduler and controller manager etc are placed under "kube-system" namespace only
3. kube-public = by default kube-public ns is empty, if we place any objects within the kube-public, those are accessible publicly to everyone without authentication

4. kube-node-lease = all the kubernetes leased objects are associated to this namespace

1. How to see the namespaces that are on the cluster?

```
kubectl get (namespaces or ns)
```

returns all the list of namespaces that are available on the cluster

2. How to create our own namespace in the kubernetes cluster?

```
kubectl create namespace namespacename
```

3. how to delete a namespace from the cluster?

```
kubectl delete namespace namespacename
```

4. How to see all the pods running/available on the cluster?

```
kubectl get pods
```

by default the kubectl will be running on "default" namespace so it shows all the objects like pods, deploymentset,
replicaset etc only in default namespace

if we want to see the objects/resources under a specific namespace then we need to use

```
kubectl get pods -n namespace
```

5. How to see all the objects on the cluster irrespective of the namespace?

```
kubectl get pods -A
```


14-02-2023

What is a namespace, what is the purpose of it in kubernetes cluster?
Namespaces are used for creating logical grouping of objects & resources within the kubernetes cluster, so that we can isolate the objects & resources belongs to one project or domain or business unit from other by sharing the same cluster across multiple teams/business units.

Kubernetes administrator can create users, groups associate them to an namespace granting access, so that users can access/manage resources/objects of that namespace only securing the cluster

There are 4 namespaces will be created by default aspart of kubernetes install
1. default = by default, default namespace is empty. all the objects/resources that are created would be placed by default under default namespace
2. kube-system = The kubernetes system objects like api manager, controller manager and scheduler are created under kube-system namespace
3. kube-public = by default empty, and the objects/resources placed here are publicly accessible without authentication
4. kube-node-lease = leased objects are placed in kube-node-lease

1. How to see all the namespaces within the cluster?

```
kubectl get ns or namespaces
```

2. how to create a new namespace?

```
kubectl create namespace namespacename
```

3. how to see objects/resources within a namespace?

```
kubectl get pods -n namespacename
```

4. how to delete a namespace?

```
kubectl delete namespace namespacename
```

5. how to see objects/resources from all the namespaces of the cluster?

```
kubectl get pods -A
```


How does the kubectl will be configured to access the kubernetes cluster?
during the time of kubernetes cluster install, the kubectl will be configured to point to the cluster. Incase of AWS EKS cluster
the aws cli has provided an cli command through which we can setup kubectl pointing to the AWS EKS cluster as

```
aws eks update-config --region regionName --cluster clusterName
```

The kubernetes cluster information will be stored by default under $HOME/.kube/config file. It has majorly 3 sections in it
1. clusters = clusters section lists all the kubernetes cluster and their certificate key using which it needs to communicate with the kubernetes cluster incase of https

2. users = all the users of different clusters using which we need to connect to the cluster

3. contexts = a context is a name given for the combination of cluster, user and namespace which would be specified to kubectl to be used in connecting and managing the cluster

```
$HOME/.kube/config
------------------
apiVersion: v1
kind: config
preferences: {}
clusters:
  - cluster:
    name: testcluster
                server: http://host:port
                certificate-authority-data: key
        - cluster:
    name: stagecluster
                server: http://host:port
                certificate-authority-data: key
users:
  - name: testclusteruser
          user:
      client-certificate-data:
      client-key-data:
  - name: stageclusteruser
          user:
      client-certificate-data:
      client-key-data:
contexts:
  - context:
    name: testcontext
                  cluster: testcluster
                  user: testclusteruser
                  namespace: default
        - context:
    name: stagecontext
                  cluster: stagecluster
                        user: stageclusteruser
                        namespace: ns1
current-context: testcontext
```

How to switch from one cluster to another cluster?
we need to change the current-context attribute in kube config file

There are 2 ways to change it
1. we can directly goto .kube/config file and modify it using text editor, which is not recommended
2. we can modify using kubectl cli commands

1. how to switch to an another context using kubectl?

```
kubectl config use-context stagecontext
```

2. how to see all the contexts in kubeconfig?

```
kubectl config get-contexts
```

3. how to see the current-context in use?

```
kubectl config current-context
```

4. how to change a namespace of a kubernetes cluster

```
kubectl config set-context --current --namespace=ns1
```

```
-----------------------------------------------------------------------------------
-------------------------------------------------------------------
```

15-02-2023

How can we connect or manage multiple kubernetes cluster using kubectl?
the cluster information would be by default configured or stored in kubernetes workstation at $HOME/.kube/config. The kubeconfig file containers 3 parts
1. clusters = In the clusters section information about the kubernetes cluster and the SSL keys will be available
2. users   = The list of users pertaining to each cluster using whom we want to connect to the cluster along with secretkeys will be available
3. contexts = is a combination of cluster + user with namespace that should be used by kubectl to connect

.kube/config

```
apiVersion: v1
kind: config
preferences: {}
clusters:
        - cluster:
                name: cluster1
                        server: http://host:port
                        certificate-authority-data:
        - cluster:
                name: cluster2
                        server: http://host:port
                        certificate-authority-data:
users:
        - name: clusteruser1
                user:
                        client-certificate-data:
                        client-key-data:
        - name: clusteruser2
                user:
                        client-certificate-data:
                        client-key-data:
contexts:
        - context:
                name: devcontext
                        cluster: cluster1
                        user: clusteruser1
                        namespace: ns1
        - context:
    name: testcontext
                        cluster: cluster2
                        user: clusteruser2
                        namespace: ns2
current-context: devcontext
```

How to switch from one cluster to another cluster?
To switch from one cluster to another we need to modify current-context attribute in .kube/config file. There are 2 ways we can modify the .kube/config file
1. manually we can edit the file using text editor
2. use kubectl cli to modify the .kube/config file

1. How to see the current-context in which we are in?

```
kubectl config current-context
```

2. how to see the kubeconfig file

```
kubectl config view
(or)
cat ~/.kube/config
```

3. how to get all the contexts configured?

```
kubectl config get-contexts
```

4. How to switch from one context to a different context?

```
kubectl config use-context contextName
```

5. how to change the namespace in a context?

```
kubectl config --set-context --current --namespace=namespacename
```

How to add a new kubernetes cluster information in .kube/config file?
There are 2 ways we can add cluster information into .kube/config file
1. modify the .kube/config file manually and add the cluster information
2. use --set-cluster in kubectl

```
kubectl config --kubeconfig=config set-cluster development --server
```

```
http://host:port --insecure-skip-tls-verify
```

Kubernetes Objects

There are different types of resources are there in kubernetes

1. Pod
2. ReplicaSet
3. DeploymentSet
4. DaemonSet
5. Service
6. Job
7. Ingress
8. Loadbalancer

The kubernetes will represent all these resources information interms of objects and persist them in etc of the kubernetes cluster. all these objects are created under kube-system namespace. The state of the kubernetes cluster will be represented by the type/number of objects on the cluster

A kubernetes object is an record of intent, once we created an kubernetes object, the system constantly work to ensure the object exists or created on the cluster. Through the objects we are specifying the kubernetes system, what we want on the cluster

How to create the objects on the cluster?

These kubernetes objects can be created by writing resource spec or manifest file and should be passed as an input to the controlplane using kubectl.

The Control Plane reads the resource spec or manifest and creates an appropriate kubernetes object and stores in etcd

1. write a resourcespec based on the type of the object/resource we want to create
2. by using kubectl pass it as an input to control plane /master node
3. masternode validates the spec and reads/extracts the information and creates appropriate object and stores in etcd
4. the system constantly work to bring the desired state

The resource spec or manifest is an YAML file in which we describe the information about the object we want to create in terms of key/value pair and pass it as an input to control plane. The structure and contents of the spec file is defined by kubernetes itself for each type of resource.

But all of these spec files carries few common attributes irrespective of type object is:

1. apiVersion = which version of the kubernetes object we are using for creating
2. kind = type of object
3. metadata = used for defining labels for the object
3. namespace = under which namespace the object should be created
4. spec = the desired state of the object

How to create and manage these objects in kubernetes cluster?

The kubectl has provided sophisticated ways of creating, managing these objects on kubernetes cluster. There are 3 ways we can create these objects on the cluster.
1. imperative commands
2. imperative object configuration
3. declarative object configuration

1. imperative commands:
kubectl has provided handful commands to which we can pass arguments in creating various different types of kubernetes objects on the cluster. this avoid writing an specfile manually in creating objects.

```
kubectl run apache2 --image=apache2:latest --port=80
```

advantages:-
1. we can quickly create an object on the cluster without writing any manifest file and test it

dis-advantage:-
1. we dont have any spec file in hand, based on the inputs/arguments we passed, the kubectl creates the yml onfly and passes it to the control plane. if we want to modify or reuse in recreating the object at later point of time we dont have the spec with us

----

16-02-2023

What are kubernetes objects?
kubernetes object is an record of intent, through which we can tell what we wanted to create on kubernetes cluster. upon creating an object kubernetes cluster will tries to bring the system to the desired state based on object definition. all the kubernetes objects are persisted permanently aspart of etcd.
There are different types of objects are there
1. pods
2. replicaset
3. deploymentset
4. daemonset
5. job
6. service
7. ingress
etc

by writing an kubernetes resource specfile we tell the kubernetes to create an object on the cluster. The devops engineer or an kubernetes developer would write the resource specfile defining/describing the info about the object on workstation and using the help of kubectl and passes the resource spec/manifest to the control plane or kubernetes master. Kubernetes master validates the spec and creates an kubernetes object representing the spec and stores in the etc

The resource spec are yml files, and the structure of these yml files are specific to the type of the object we want to create and standardized by the kubernetes. based on the structure (key/value pairs) defined by the kubernetes we need follow and write the spec file inorder to get this validated by the cluster

There are few common attributes are there across all type of object specs which are defined as below.

1. apiVersion = defines the specfile version being used in defining the object
2. kind = type of object
3. metadata= used for attaching lables to the object for identification and retrieval
4. namespace = under which namespace object should be created
5. spec = specification of the object

How many ways are there in creating these objects on the cluster?

There are 3 ways are there

1. imperative commands
2. imperative object configuration
3. declarative object configuration

1. imperative commands

The kubectl has provided handful of commands taking arguments as input in creating various different kubernetes objects like pods, service, deployments etc. we dont need to write the resource specfile in creating these objects

```
kubectl run podName --image=imageName:tag --port=portNo
```

advantages:-

1. quickly create an object on the cluster

dis-advantage:-

2. since we dont have resource specfile we cannot reuse the object across the environments

2. imperative object configuration

In an imperative object configuration, for each type of resource we want to create, we need write an resource specfile describing the information about the object and passes it as an input to the kubectl asking him to perform the operation like create or delete.

apache2-pod.yml = in this specfile we described the pod information

```
kubectl create -f apache2-pod.yml
kubectl delete -f apache2-pod.yml
```

3. declarative object configuration

In order to run an application on kubernetes cluster, they could be bunch of objects we need to create on the cluster. So defining each object we need to write the respective resource specfile. upon writing these specfiles we need to create each object by running the spec keeping the order of dependencies. It takes lot of time and would be hard to memorize and execute the objects in the dependency order

Instead keep all the resources under one directory under the project like manifests / configs

```
airtelcare2
|-manifests
        |-pod.yml
        |-deployment.yml
        |-service.yml
```

pass the directory as an input to the kubectl asking to apply these manifests on the cluster

```
kubectl apply -f manifests/
kubectl delete -f manifests/
```

Let us try to understand how to create different kubernetes objects by writing the specfile

1. Pod

Pod is an smallest unit or entity in the kubernetes world in which one or more containers are kept together and executed. In general we may have multiple containerized applications that shares common dependencies like

1. network
2. resources file/mounts/volumes
3. lifecycle (start/stop)

In such case instead of having these 2 applications running as 2 independent containers, to reduce the effort of managing them togetherly easily kubernetes has introduced an logical container/unit called "pod", in which we can place one or more containers as an single unit and can manage them together like starting or stopping etc

How to write an resource specfile in running an pod on the cluster?

application binary -> Dockerfile -> image -> container registry -> pod specfile -> create/apply   #cluster

----

apache2 image

Apache2 image is already published and available on the container registry, so we can quickly write pod specfile in running an container ontop of the image on the cluster as below.

assumption: the image is already available in the registry

```
apache2-pod.yml
apiVersion: v1
```

```
kind: Pod
metadata:
        name: apache2pod
spec:
  containers: #defining the info about the containers we want to run inside this pod
          - name: apache2
                  image: apache:latest
                        ports:
        - containerPort: 80
                                        name: http
                                        protocol: tcp
```

```
kubectl create -f apache2-pod.yml
```

1. how to see the pods on the cluster

```
kubectl get pods -n namespace
```

2. how to create pods on the cluster

```
kubectl create -f pod.yml
```

3. how to see the information about the pod ?

```
kubectl describe pod podName
```

4. how to acces the logs of the running pod on the cluster?

```
kubectl logs podname
```

5. how to get more information about the pods on the cluster?

```
kubectl get pods -o wide -n namespace
```

6. how to expose a pod to the host using port-foward?

```
kubectl port-forward podName hostPort:containerPort
```

7. how to delete a pod?

```
kubectl delete -f pod.yml
or
kubectl delete podName
```

17-08-2023

What is a Pod?

Pod is an smallest entity or an unit within the kubernetes cluster, in which we run one or more container together inside it.There can be few containers that are dependent on each other interms of sharing

1. resources

2. file system / volumes
3. lifecycle
rather than having such containers managed independently we can place them inside one pod and manage it together easily.

How to write a pod manifest file in creating and running a pod?
apache2-pod.yml

```
apiVersion: v1
kind: Pod
metadata:
        name: apache2pod
namespace: default
spec:
        containers:
                - name: apache2
                  image: ubuntu/apache2:latest
                        ports:
                                - name: apache2port
                                        containerPort: 80
                                        protocol: TCP
```

----

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

repositoryName/image:tag
techsriman/sailor:1.0

18-02-2023

Readiness Probe and Liveness Probe
Upon placing an request for executing a pod the kubernetes makes best effort of choosing an node on which the pod will scheduled for execution. during the time of bringing up the pod, kubernetes will read the podspec file and identifies the image on which the pod container should be created and checks to see is the container image is available locally.

If the container image is not available locally, then it pulls the image from container registry onto that workernode and creates the pod and runs it. the moment the pod has been started kubernetes reports the pod under running state

but the underlying application running inside the pod may not be started or while running the underlying application might go into unresponsive state due to resource availability, stuck threads etc. even then also the kubernetes reports the pod status as running and available. The kubernetes doesnt know the real/underlying status of the application that is running inside pod, due to which it would be able to identify such faulty applications

In general kubernetes takes care of replacing an pod that is not working or crashed, but in this scenario since kubernetes dont know the statusof the underlying application, it would assume the pod is working and leave it unresponsive

To let the kubernetes identify such faulty or unresponsive applications running inside the pod and replace them we need to provide the health information about the application to the kubernetes through readinessProbe and livelinessProbe

Kubernetes supports 2 types of checks to be performed on a pod application
1. readinessProbe
upon scheduling a pod for execution the kubernetes pulls the image and runs the pod and reports the statusof the pod as running, but the underlying application that is running inside the podcontainer could be still under starting state and may not be ready for accessing or accepting the requests. even then also kubernetes assumes the application has been started and every request received for the application from the outside world would be routed to the application which will results in failure.

To avoid this problem kubernetes has introduced readinessProbe, using which kuberneters determines when the actual application is available for accessing. The developer of the application has to expose an http endpoint and configure the information about this endpoint in podspec configuration file, letting kubernetes understand, it has to access this endpoint to verify the readiness of the application.

upon bringing up the pod, the kubernetes will perform the readinessCheck by periodically hitting or accessing the readiness Application Endpoint we configured in spec file, until the application has reported its availability, kubernetes will not route the request to the pod instance of the application. upon the application reports its availability, kubernetes marks the pod ready for scheduling the request and would stop performing readinessProbe

2. livelinessProbe
while the pod is running on the cluster, there could be a chance due to resource issues or applciation failures the application running inside the pod may become unresponsive. kubernetes is not aware of the such unresponsive pods, so that it routes the incoming requests for the pod application to these unresponsive pods as well which will eventually leads to failures.

if the kubernetes can some how identify such unresponsive running pods on the cluster, it can terminate them and recreate another pod on the cluster, which can be done through livelinessProbe

the application developer has to expose an http endpoint, that can be used by the kubernetes to periodically verify the application is running/responsive or not. while write the podspec file the kubernetes developer has to configure the livelinessProbe information so that kubernetes can perform this check for us

The livelinessProbe would be started only upon the readinessProbe has been reported as successful and continued to perform the checks until the pod has been reported as failed or manually terminated

We have an application urotaxi (java web application), in this application we exposed #2 endpoints (readiness/liveness). The url for accessing these endpoints are below
1. /actuator/health/readiness
2. /actuator/health/liveness

now while writing the podspec file in running the application as pod on kubernetes cluster, we need to configure both readinessProbe and livenessProbe configuration as below.

```
urotaxi-pod.yml
---------------
apiVersion: v1
kind: Pod
metadata:
        name: urotaxipod
        labels:
                version: 1.0
spec:
        containers:
                - name: urotaxi
                        image: techsriman/urotaxi:1.0
                        ports:
                                - name: tomcatport
                                        containerPorts: 8089
                                        protocol: TCP
                        readinessProbe:
                                httpGet:
                                        path: /actuator/health/readiness
                                        port: 8080
                                initialDelaySeconds: 5
                                timeoutSeconds: 10
                                failureThreadShold: 3
                        livenessProbe:
                                httpGet:
                                        path: /actuator/health/liveness
                                        port: 8080
                                initialDelaySeconds: 5
                                timeoutSeconds: 10
                                failureThreshold: 3
```


19-02-2023

Resource declarations
whenever an pod application is running on the kubernetes cluster, it is going to consume cpu/memory during execution. The amount of cpu/memory it consumes depends on various factors like
1. hardware capacity of the machine on which we are running the application
2. how much amount of user traffic is coming to the application
3. the amount of data the application is processing

How can we know the cpu/memory utilization of an application?
The performance testing team puts the application for evaluation for deriving the beanch mark metrics of the cpu/memory usage levels of the application. The team puts an virtual load on the application based on the average number of users always going to access the application and derives the cpu/memory consumption.

whenever we scheduled an pod for execution, the kubernetes scheduler takes care of identifying an appropriate worknode on the cluster which has sufficient cpu/memory resources available for running the pod and handovers the pod execution to the kubelet process of the node.
So to the kubernetes scheduler determine the right workernode   to be used in running the pod, we need to define or declare the resource specification aspart of the pod spec file. we define the minimal requirements in running an pod based on which the workernode will be choosen, incase if the pod application is requesting more than the resources requested, the kubelet process will try to accomodate the resources if available on the worker node.

In case if the workernode doesnt have sufficient resource capacity, the pod will be terminated and would rescheduled to execute on a workernode which has appropriate capacity. so we need to define the resource specification in the pod spec for execution

roadster-pod.yml
```
apiVersion: v1
kind: Pod
metadata:
        name: roadsterpod
spec:
        containers:
                - name: roadster
                        image: techsriman/roadster:1.0
                        ports:
                                - name: tomcatport
                                        containerPort: 8080
                                        protocol: TCP
                        readinessProbe:
                                httpGet:
                                        path: /roadster/actuator/health/readiness
                                        port: 8080
                                initialDelaySeconds: 10
                                timeoutSeconds: 10
                                failureThreshold: 3
                        livenessProbe:
                                httpGet:
                                        path: /roadster/actuator/health/liveness
                                        port: 8080
                                initialDelaySeconds: 10
                                timeoutSeconds: 10
                                failureThreshold: 3
                        resources:
                                requests:
                                        cpu: "500m"
                                        memory: "512Mi"
                                limits:
                                        cpu: "1000m"
                                        memory: "1024Mi"
```

20-02-2023

what are resource declarations, why do we need to use them?
Resource declarations are the way through which we specify the cpu and the memory requirements and limits for an pod to be scheduled and execute on the cluster. Based on the resource specifications we specified, the scheduler chooses the workernode of enough capacity to run the pod on the cluster.

There are 2 types of metrics we can specify
1. cpu = how many number of cpus to be allocated
2. memory = how much amount of memory to be assigned

These limits will be specified to the individual container of a pod, not at the pod level. so that the sum of inidivual container resource specifications would be considered as the final resource limits in scheduling and running a pod on the cluster

There are 2 types of limits are there
1. requests = requests indicates the initial capacity to be allocated in running the pod
2. limits = the max capacity a pod can be give for execution

In case if we have not specified the "limits" metrics then it considers the max cpu/memory the memory to be allocated as
1. no limit, and allocate how much ever has been requested
2. while creating an namespace, the administrator can set a max default limit of the resources to be allocated to the pods, which would be applied for all the

containers running inside the namespace, given if limits not specified at the pod level

In case if we have not specified the "requests" resource spec for cpu/memory of a pod, the it would considered by default same as "limits" spec only
It is always advised to specify the resource declarations within the pod spec file to better manage the pod on the cluster

----

What are the different states in which a pod can exists in kubernetes cluster?
A pod in kubernetes cluster can exists in 5 different states, which is also reffered as "pod lifecycle"

1. pending = when we send a request for creating an pod, the api manager upon accepting the request would send the request to the scheduler for creating the pod, at this moment the pod state is reported as "pending"
2. running = atleast one of the container inside the pod has been started and readinessProbe on the pod has been passed, then the pod is reported as running
3. succeeded = all the containers within the pod has been exited with an exitcode as zero, then the pod is reported as succeeded
4. failed = when atleast one of the container inside the pod has been exited with non-zero exitcode then the pod is reported as failed
5. crashloopbackoff = when a pod is repeatedly failing for execution after a successive restarts, then to avoid further scheduling of the same pod for execution, kubernetes marks the state of the pod as "crashloopbackoff" indicating the pod should not be scheduled for futher execution, since it is repeatedly failing

----

Working with Labels and Annotations in Kubernetes
Labels:
Labels are arbitary key/value pairs we can attach to the an kubernetes object, these are used for identifying and accessing the objects over the cluster. For an kubernetes object we can assign any number key/value pair labels but, they should appear only once and should be unique

We can define labels for the kubernetes objects in 2 ways

1. we can declare the labels in the spec or manifest files directly
2. we can attach/detach the labels dynamically at runtime

As specified earlier we can attach labels for different types of objects like

1. pod
2. replicaSet
3. deploymentSet
4. Service

etc

#1 we can declare the labels in the spec or manifest files directly
How to declare labels in the pod specfile while creating the pod?

roadster-pod.yml

```
apiVersion: v1
kind: Pod
metadata:
        name: roadsterpod
        labels:
                app: roadster
                version: 1.0
                env: stage
spec:
        containers:
                - name: roadster
                        image: techsriman/roadster:1.0
                        ports:
                                - name: tomcatport
                                        containerPort: 8080
                                        protocol: TCP
```

1. How to see all the labels attached to an object?

```
kubectl get pods --show-labels
```

2. we can query or search for objects based on the labels using -l switch?

```
kubectl get pods -l key=value
```

#2. how to dynamically attach lables to the object on the cluster?

```
kubectl label pods podName key=value
```

Annotations

Annotations are used for attaching arbitary information which is non-identifier data to an kubernetes objects. These are only used as documentation helpers which we can read or used through kubernetes metadata api.

roadster-pod.yml

```
apiVersion: v1
kind: Pod
metadata:
        name: roadsterpod
        labels:
                app: roadster
                version: 1.0
                env: stage
        annotations:
                license: GPL License
                Warranty: product comes under limited warranty
spec:
        containers:
                - name: roadster
                        image: techsriman/roadster:1.0
                        ports:
```

```
                                  - name: tomcatport
                                          containerPort: 8080
                                          protocol: TCP
```

we can see the annotations attached to an object using
kubectl describe pod podName

22-02-2023

configmaps- config-secret

ConfigMaps
What are ConfigMaps and what is the purpose of ConfigMaps?
Every software application uses configuration information pertaining to an external resource/system it is using to perform operations like an
1. database management system
2. enterpise cache (rediscache, memcache)
3. remote filesystem
4. an external api provided by an vendor
etc

How can the software application can maintain or use this configuration within their application?
There are many ways of maintaining the configuration information
#1
The application can directly hardcode the configuration values within their programs itself inorder to communicate and perform operations on the external resources/systems. writing the configuration values directly within the sourcecode of the application is not recommended, because we run into lot of problems as described below.
dis-advantages:
1. whenever there is a change in the configuration values, the developer has to modify the sourcecode, build/package and redeploy the application which takes huge amount of time and rework for a configuration change
2. from one env to another env, the configuration values of the external resources would be different, so while moving the application across the env, we need to modify the sourcecode which again runs into the same problem we discussed above

#2
To avoid the above problem in maintaining the configuration information, we need to externalize the configuration values into an external files like properties, yaml or xml files. The developer has to write these configuration values within these configuration files and read those values in the programs to perform operations on the external resources.
advantages:
1. since the configuration values are placed in non-program files, a change in configuration needs to modify the configuration files which does require rebuild/repackaging or redeploying the applications. So the configuration changes

can be easily reflected
2. for different environments we have different configuration values, so we can create multiple configuration files pertaining to each env and we can run the application against those configurations
dis-advantages:-
1. incase if multiple replicas of the application has been deployed across the nodes of the cluster for high availability and scalability, then maintaining these configurations and changing the configuration values across all the instances will be difficult

#3 centralize the configuration and distribute it across all the instances of the application

In addition it is adviced to design the application to accept the configuration values from the environment variables, so that we can easily override the configurations while lauching the application

How to run such applications that accepts the configuration values from an external source like env variables aspart of the kubernetes cluster?

23-02-2023

ConfigMaps
An Application has to be designed to read the configuration values as an input through environment variables so that while running the application the devops engineer can pass these values as an input by configuraing them as environment variables.

How to lauch or run an application which accepts the configuration interms of env variables on a kuberneters infrastructure?
The env variables should be seeded into the container, while lauching the container so that these env variable values will be available as an input to the application.
In case of kubernetes we are not creating/lauching the containers rather the controlplane takes care of creating the container, so we need to specify which what env/values with which the container application has to be launched by writing them in spec file

If we write these env variable with values in the pod specfile, there are few problems are there
1. everytime there is a change in the value, we need to modify the pod specfile, which is an unnecessary maintainance
2. the same env variables with values may have to be reused across different applications running on the cluster, so if we write the env variables with values locally in the pod spec file these will get duplicated across the applications so a change in these values inccurs huge efforts and time

Instead of writing these configuration values in pod spec file place them inside

the configmap object in kubernetes cluster. These configuration values we placed inside the ConfigMap can be accessed as inputs into the pod application in 3 ways
1. environment variables = we can pass these configMap values as environment variables into the pod application by refferring them in pod specfile
2. command-line arguments = we can pass these configMap values as commandLine-arguments while launching the application
3. Through ConfigMap api = The containerized application, the developer can write the code in reading the values from ConfigMap object stored on the kubernetes cluster (not recommended)

How to work with ConfigMap?
In order to use the configuration values from the ConfigMap, we need to create the ConfigMap object with keys/values on the cluster. For creating the ConfigMap object we need to write ConfigMap spec file as below.

coronaguidelines-configmap.yml

```
apiVersion: v1
kind: ConfigMap
metadata:
        name: guidelinesconfigmap
        labels:
                app: corona
data:
        oxygenLevels: 85
        quarantine: 20
        liters: 5
        temparatureLevels: 99 - 100
```

with these properties we created an ConfigMap object in kubernetes cluster, now we need to pass these properties in running the pod on the cluster as below

corona-pod.yml

```
apiVersion: v1
kind: Pod
metadata:
        name: coronapod
        labels:
                app: corona
spec:
        containers:
                - name: corona
                        image: techsriman/corona:1.0
                        ports:
                                - name: tomcatport
                                        containerPort: 8080
                                        protocol: TCP
                        env:
                                - name: guidelines.oxygenLevels
                                        valueFrom:
                                                configMapKeyRef:
                                                        name: guidelinesconfigmap
                                                        key: oxygenLevels
                                - name: guidelines.liters
                                        valueFrom:
                                                configMapKeyRef:
                                                        name: guidelinessconfigmap
                                                        key: liters
                                - name: guideliness.temparatureLevels
                                        valueFrom:
                                                configMapKeyRef:
                                                        name: guidelinessconfigmap
                                                        key: tempartureLevels
```

===
===

27-02-23

What are ConfigMaps, why do we need to use them?
ConfigMap is an kubernetes object in which we can store key/value pair data inside it and we can use the ConfigMap data inside the applications that are running in the pod containers.
There are 3 ways we can use the data inside the pod applications
1. environment variables / command-line arguments to the program
2. application programs can use Kubernetes apis to access ConfigMap object that is stored on the cluster
3. We can mount ConfigMap with ConfigurationFile as volumes into the container through podspec

3. How to pass ConfigMap with configuration file as a volume in the podspec?

```
corona-configmap.yml
apiVersion: v1
kind: ConfigMap
metadata:
  name: coronaconfigmap
data:
  corona.properties |
    oxygenLeves=85
    liters=5
    quarantine=20
    tempartureLevels=99 - 102
```

Now while writing the podspec file we need to mount the properties file as an volume mount into the container, so that developers while building the application will have the logic for reading the file from the mountLocation

```
corona-pod.yml
apiVersion: v1
kind: Pod
```

```
metadata:
  name: coronapod
spec:
        container:
                - name: corona
                        image: techsriman/corona:1.0
                        ports:
                                - name: tomcatport
                                        containerPort: 8081
                                        protocol: TCP
                        volumeMounts:
                                - name: coronavolume
                                        mountPath: /config
                                        readOnly: true
  volumes:
                - name: coronavolume
                        configMap:
                                name: coronaconfigmap
                                items:
                                        - key: corona.properties
                                                path: "corona.properties"
```

here we defined an volume "coronavolume" populated with ConfigMap data (corona.properties) inside it. Then we are mounting the volume inside the container of a pod using volumeMounts under a specific directoryLocation /config

lets connect tommorow looks like it is taking time

==
==

02-03-2023

Config Secrets

Kubernetes secrets let us store and manage sensitive informaton like passwords, ssh keys, encryption keys etc that are required as an input by an software application. We can store these secrets/credentials directly aspart of podspec or in a configMap as well, but storing these secrets in pod spec or configMap makes them insecure. everyone can read the information/secrets we stored in podspec or configMap and can grab the access to the end systems. Instead it is recomended to store such sensitive information in ConfigSecret

Note:-
By default when we store the credential information in ConfigSecret it will not encrypt the data while storing, rather it encodes the data into Base64 encoding and will be stored. That means we can read the values back into plain-text format and

hence these are not by default secured.
So kubernetes ConfigSecrets are stored in HashiCorp Vault by integrating kubernetes with vendor Vaults

While storing the sensitive data within the ConfigSecret we can attach type information to help us identify what type of secret we are storing in ConfigSecret. It is not mandatory to attach type information while storing a ConfigSecret but it is recommended so that we can easily understand what type it is while accessing. By default while storing if we dont specify the type, it treats the type as "opaque"

Kubernetes has provided built-in secret types, we can use these secret types while defining our own secrets
1. opaque = arbitary data
2. kubernetes.io/service-account-token = The service account token is an system secret or kubernetes secret
3. kubernetes.io/dockercfg = serialized format of docker config file
4. kubernetes.io/dockerconfigjson = serialized format of docker config json file
5. kubernetes.io/basic-auth = username/password
6. kubernetes.io/ssh-auth = ssh keys
7. kubernetes.io/tls = ssl keys or public/private encryption keys

we can use the ConfigSecret in 3 ways like ConfigMaps
1. we can pass the secrets as environment variables or commandline arguments
2. we can mount the ConfigSecret as files in the pod container
3. The kubelet process itself uses these secrets for connecting to the docker container registry for pulling the images

How to create an ConfigSecret object for storing the database username and password?

airtel2-configsecret.yml

```
apiVersion: v1
kind: Secret
metadata:
        name: airtel2dbconfigsecret
type: kubernetes.io/basic-auth
stringData:
        username: root
        password: root
```

airtel2-pod.yml

```
apiVersion: v1
kind: Pod
metadata:
        name: airtel2pod
spec:
        containers:
                - name: airtel2
                image: techsriman/airtel2:1.0
                ports:
                        - name: tomcatport
                                containerPort: 8081
                                protocol: TCP
                env:
                        - name: "spring.datasource.username"
                                valueFrom:
                                        secretKeyRef:
                                                name: airtel2dbconfigsecret
                                                key: username
                  - name: "spring.datasource.password"
                                valueFrom:
                                        secretKeyRef:
                                                name:airtel2dbconfigsecret
                                                key: password
```

==

04-03-2023

There are 4 components are there in kubernetes cluster
1. Control Plane
2. Worker Nodes
3. Kubectl
4. etcd

Within the Control Plane there are 3 components are there
1. Api Manager
2. Scheduler
3. Controller Manager

There are 5 types of controllers are there
1. ReplicaSet Controller
2. DeploymentSet
3. DaemonSet
4. Job
5. Service

1. ReplicaSet Controller
Pod is the smallest entity within the kubernetes cluster where multiple containers are kept together and are executed within a pod. these container may want to share common resources like FileSystem or has a common lifecycle to be packaged and run inside one pod.

We can create a pod in kubernetes cluster through manifest or pod specfile. There are few characteristics of a pod are there

1. one pod manifest creates one pod on the cluster in running state
2. pod will not survive by crash = If a pod has been crashed due to any reason it will not be replaced with another pod

In addition:
3. if we want to run #10 pods out of the same pod specfile (with same containers inside it), we need to create #10 pod manifest files with different pod names and create the pods manually, which is an difficult job

4. upon creating the #10 pods, we need to monitor them and ensure always those are running, incase if any one of the replica of the pod has been crashed, we need to take care of replacing them within another pod

So managing the multiple replicas of a pod and replacing them incase of crash by monitoring them is difficult job, So to overcome this problem kubernetes has provided ReplicaSet Controller

ReplicaSet Controller:
A ReplicaSet Controller can be imagined as an Reconcilation loop, where the ReplicaSet controller loops through all the workernodes of the cluster to identify whether the desired number of Replicas of a pod are running on the cluster or not. if the desired number of replicas are not met, then the ReplicaSet Controller talks to the Scheduler in bringing up the pods on the cluster to meet the desired state. If already the desired number of replicas are met, it goes into monitoring state to see if any pods has crashed over the course of time, it can replace them with another pod

To create an ReplicaSet Controller we need to write an ReplicaSet Specfile similar to an pod spec. within the ReplicaSet manifest/spec we need to embedded the podspec specifying how many replicas of this pod should be runing. without a podspec a replicaset spec will be meaningless

Let us consider we have an #4 workernode cluster, in which we created an replicaSet controller with desired number of replicas as #3, the kubernetes control plane or ReplicaSet controller makes best effort in distributing the #3 replicas of a pod on #3 different workernodes of the cluster for high availability, but there is no guarantee that always the replicas are running on uniquenodes of the cluster

From the above we can understand we always write the podspec inside the replicaset spec with replicas to bring up desired number of pods on the cluster

sailor-replicaset.yml

```
apiVersion: apps/v1
kind: ReplicaSet
metadata:
        name: sailorreplicaset
        labels:
                app: sailor
spec:
        replicas: 2
        selector: [through which we specify which pods of the specified replias to be running]
                matchLabels:
                        app: sailor
                        version: 1.0
template:
        metadata:
                labels:
                        app: sailor
                        version: 1.0
        spec:
                containers:
                        - name: sailor
                                image: techsriman/sailor:1.0
                                ports:
                                        - name: tomcatport
                                                containerPort: 8080
                                                protocol: TCP
```

==

05-03-2023

Deployment

Deployment is another way of deploying the application onto the kubernetes cluster and releasing it to the customers. Using Deployment controller any changes to the pod template can be rolledout in a controlled way

There are several deployment strategies are supported by Deployment controller
1. recreate   = terminate the old version and release the new one
2. ramped   = release a new version on a rolling update fashion one after the another
3. blue/green   = release a new version alongside of the old version then switch the traffic to newer version
4. canary   = release a new version to a subset of users then proceed to a full rollout
5. a/b testing = release a new version to a subset of users in a precse way (HTTP headers, weight etc). A/B testing is a technic for making business decisions based on statistics. It doesn works out of box with kubernetes, we need to do extra setup to implement it like (istio, traefik, nginx/haproxy etc)

Deployment controller applies the strategy of rolling or releasing the pod replicas on the cluster, by which we can understand without a replicaSet there is no deployment controller exists to manage the releases. So within the Deployment spec always we embed the ReplicaSet spec

speed-deployment.yml

```
--------------------
apiVersion: apps/v1
kind: Deployment
metadata:
        name: speeddeployment
spec:
        replicas: 2
        selector:
    matchLabels:
      app: speed
        strategy:
    type: Recreate
        selector:
                matchLabels:
                        app: speed
        template:
                metadata:
                        labels:
                                app: speed
                spec:
                        containers:
                                - name: speedcontainer
                                        image: techsriman/speed:1.0
                                        ports:
                                                - name: tomcatport
                                                        containerPort: 8080
                                                        protocol: TCP
```

once we created the deployment with the above deployment spec, we can apply the spec to create deployment on the kubernetes

```
kubectl create -f speed-deployment.yml
```

#1. how can we see the deployments on the cluster?

```
kubectl get deployments
```

#2. describe deployments

```
kubectl describe deployment deploymentName
```

when we create an deployment, kubernetes internally creates an replicaSet in rolling out the pods based on the deployment

```
-----------------------------------------------------------------------------------
------------------------------------------------
```

How to rollout the newer version of the application?

1. make the changes in the application, and have these changed build, tested and packaged and distributed it as docker image into the docker container repository the image name should be bumped up when we are releasing the newer version of the application

2. upon the new version of the image is published and ready, we need edit the deployment we have already created for that application on the cluster

There are 2 ways we can modify the deployment
1. using kubectl set command
2. updating the specfile on the cluster

1.

```
kubectl set image deployment/deploymentName container=newImage:newVersion
```

for eg..

```
kubectl set image deployment/speeddeployment speedcontainer=techsriman/speeddep:2.0
```

2. run the below command that opens the existing deployment spec on the cluster

```
kubectl edit deployment deploymentName

kubectl edit deployment speeddeployment
```

#3. how to see the revision history of the rollout changes?
revision history means the modifications done on a deployment, we can see them using the below command

```
kubectl rollout history deployment/deploymentName
```

everytime when we modify the deployment, kubernetes internally creates a new replicaSet controller with the latest changes and modifies the old replicaSet replicas to 0 to rollout the newer pods. the way the old and new replicaSet controller will be changes is depends on the strategy we choosen.

strategies:
1. Recreate
The recreate strategy means terminate all the running instances then recreate them with newer version

```
spec:
        replicas: 2
        strategy:
                type: Recreate
```

advantages:-
        1. application state entirely renewed
        2. no need of additional infrastructure to be created or planned to rollout the newer version of the application
        3. cost of releasing the newer version is less
dis-advantages:-
        1. downtime and the amount of time the application will be un-available depends on the time it takes to boot the instances

===

06-03-2023

Deployment Controller
Deployment controller is another way of deploying the application onto the kubernetes cluster. Through deployment controller any changes in the pod template can be rolledout in a controlled way

For eg.. if we have an pod application of version: v1 running on the kubernetes cluster, and we have an updated version of the application: v2 is available. To update the existing application of v1 version on cluster with v2 version the Deployment controller helps us in releasing by providing various different deployment strategies. There are 5 deployment strategies are supported by Deployment Controller

1. Recreate
2. Ramped
3. blue/green
4. canary
5. a/b testing

#1. Recreate
In Recreate the existing pods on the cluster will be terminated and the new version of the pods will be rolled onto the cluster.

advantage:-

    1. the new version can be rolledout at one shot
    2. no need of additional infrastructure to be planned for making an release

advantage:-

    1. downtime, and the downtime depends on how long the new version of the application takes time to boot up

#2. Ramped [Slow Rollout] [Rolling Update]
A Ramped deployment updates the pods in a rolling update fashion, a second replicaSet will be created with the new version of the application, then the number of replicas of the old version is decreased and the new version is increased util the desired number replicas are reached.

upon creating 2 replicaSet controllers, it will reduces the number of replicas of the old ReplicaSet to 1 less than the total for eg.. if it is 3 then it makes as 2. Then updates the new ReplicaSet controller with replicas = 1
upon the new version of the pod readinessCheck has been passed again it goes to old ReplicaSet and decreases the replicas by 1 and increments the replicas for new ReplicaSet

speed-deployment.yml

```
appVersion: apps/v1
kind: Deployment
metadata:
        name: speeddeployment
spec:
        replicas: 3
        strategy:
                type: RollingUpdate
                rollingUpdate:
                        maxSurge: 2         # how many pods can be added at a time
                        maxUnavailable: 0         # maxUnavailable pods can
existing during this rolling update
```

advantages:-

1. version is slowly released across the instances
2. no downtime of the application

dis-advantage:-

1. rollout/rollback can take more time
2. no control over the traffic
3. supporting multiple api versions is very hard

How to rollback to the specific version in the rollout history?

```
kubectl rollout undo deployment/deploymentName --to-revision=revisionNo
```

How to scale the deployment

```
kubectl scale deployment/deploymentName --replicas=4
```


#3. Blue/Green Deployment

Unlike the Ramped version where the pods on updated from old to the new version in rollout fashion, incase of blue/green deployment along with the blue version of the pods the green version of the pods also available along side.
After testing the new version of the pods, we update the Service object

For supporting blue/green deployment strategy, we need to create one more deployment spec to rollout the new version of the application and once it is been tested, the repoint the service to point to the latest pods and delete/scale-down the older deployment

advantages:-

1. instant rollout/rollback
2. zero downtime for the application
3. avoids versioning issues, because it changes the entire cluster state at one go

dis-advantage:-

1. requires more infrastructure for every release
2. handling stateful applications will be hard

===

===

07-03-2023

how to pause and resume the deployment?
if we want to perform multiple updates/changes to the pod, and wanted to deploy all of these changes at once then we need to pause and resume the deployemnt

1. Pause

```
kubectl rollout pause deployment/deploymentName
```

upon applying the changes we can resume the deployment using the below command

```
kubectl rollout resume deployment/deploymentName
```

1. history
2. scale
3. to-revision
4. rollout undo
5. pause
6. resume

out of box kubernetes deployment controller supports only 2 deployment strategies
1. Recreate
2. RollingUpdate

For realtime application deployments, these 2 strategies are not sufficient, because each of them has their own downsides. So people uses different strategies of their own in rolling out the changes to the pod application on to the cluster

3. blue/green = release a new version of the pod applications alongside the old versions and upon completing the testing, switch the traffic to new version and obselete the older application

4. canary = the canary strategy is also similar to blue/green only, the only difference is we release the new version of the application to the subset of users, then based on feedback we rollout the full version

5. a/b testing = release a new version to the subset of users in precise way / controlled way. For eg if the user is sending the request with a specific request header or cookie, then send the user to new version of the application, other let the user access to access the older version

----

Service
Service is adding networking to the pods that are running on the kubernetes cluster. By default when we create a pod on the node of a cluster, the pod has few characteristics:
1. The pod will be accessible within the Node of the cluster and cannot be accessed by any other pods that are running on other nodes of the cluster
2. The pod will be assigned with emphermal ip address, which would be renewed upon

a pod restart

but we wanted the pod applications to be
1. accessed across the nodes of the cluster, so that other pods can access our pod
2. we want the pod applications to be exposed over an fixed ip address so that incase of a pod crash, the other applications using our pod will not be impaced upon recovery
3. we want to loadbalance the requests that are received by multiple instances of a pod
4. we want the pod applications to be exposed and made them accessible to the public world

all these can be achieved through Service
There are 5 types of Services are supported by Kubernetes Cluster
1. ClusterIP
2. NodePort
3. Loadbalancer
4. Ingress
5. Headless Service

==
08-03-2023

Service
Service is adding an network to the pods of the kubernetes cluster. By default a pod is accessible within the node on which it has been created and has few more characteristics
1. A pod will not be recovered upon crash
2. empheramal ip address is assigned to the pod, so that upon crash and recovery an new ip address will be assigned
but we wanted to the pod to be
1. accessed by other pods on the cluster without worrying about the ip address being changed
2. pod should be exposed to the external world
3. the traffic should be load balanced across the pod replicas
we can achieve all these things through the help of Service

There are 5 types of services are there
1. ClusterIP
2. NodePort
3. Loadbalancer
4. Ingress
5. Headless Service

#1. ClusterIP Service
whenever we create an service in kubernetes cluster, by default the service type is cluster ip if we dont specify any type. If we want out pod application to be exposed within the cluster and make it accessible over fixed ip and loadbalance the

traffic we need to use ClusterIP Service.

The clusterIP Service will be assigned withan ip address within the cluster range, using which the other pods of the cluster can access the pod.

usually we expose backend applications like microservices to the frontendapplications or database pods to the java applications on the cluser using ClusterIP Service.

#2. NodePort
NodePort service the name itself indicates we open a Port on each WorkerNode through which we receive traffic and forward the to the targetPort on which the pod application is running. We can directly access an Pod running on a workernode using the fixed ip and port number using NodePort Service.

It is not meant for loadbalancing and distributing the traffic to the pods that are running across the nodes of the cluster.

Usually it is not recommended to user NodePort Service, since it exposes the Port of a WorkerNode over fixed ip address that creates an security breach. The purpose of NodePort is to directly acces the application that is running on a workernode during development/testing phases only

Note: To have an pod application running on a worker node to be directly accessible to the public world, the WorkerNode should have public ip address.

==

09-03-2023

NodePort
NodePort is used for exposing an pod application that is running on a workernode of the cluster directly to the external world over an fixed port. The name itself tell us, it opens a port on the workernode through which it makes the underlying pod application running on the node to be made accessible to the world.

===
10-03-2023

NodePort
NodePort Service is used for exposing an pod application that is running on an workernode directly to the external world. NodePort opens an port on the workernode (within the range: 30000 - 32767) and forwards the request to the targetPort on which the Pod application is running.

When we create an NodePort Service, The NodePort will be created across all the Nodes of the cluster opening an port by itself. The traffic recieved on the WorkerNode NodePort will be forwarded to NodePort Service Port, there by which be forwarded to the targetPort on which the application is running on a pod

There are 3 Ports are opened in NodePort Service
1. NodePort = opens a port on the WorkerNode allowing the external network traffic
2. Port = Service Port to which the Traffic will be forwarded upon receiving the external traffic on the NodePort, now the service does the discovery of the pods based on selector to forward the request to the targetPort
3. TargetPort = port on which the pod applications are running

The default username/password for login into minikube server is
docker/tcuser
ssh docker@minikube

Persistence Volume and Persistent Volume Claim
How to run stateful applications on kubernetes pods?
The Stateful applications generates the data during their execution, that is by default written onto the container write layer of the container of the pod in which it is running. Incase of a crash, the data written by the container will not be survived.

How can we persist the data that is generated by the pod application that is running inside the container of a Pod, so that in the event of crash the data would be retained?
That is where kubernetes has introduced Persistent Volume and Persistent Volume Claim

Persistent Volume = is a storage defined on the kubernetes cluster and it is created and maintained by the kubernetes administrator
Persistent Volume Claim = is a request for that storage to be consumed by the devops engineer that can be used aspart of an pod application

There are few attributes we need to defined while creating an persistent volume:
1. storageClassName = indicates the type of storage to be created on the cluster
2. accessMode:

```
ReadOnly           = only the pods can read the data from this volume
ReadWriteOnce      = only one pod can write the read/write the data at one time
ReadWriteMultiple  = Multiple pods are allowed to read/write at the same time
```

3. capacity = storage size to be assigned for that persistent volume

How to define an persistent volume?

```
pv.yml
-------
apiVersion: v1
kind: PersistentVolume
metadata:
        name: pv1
spec:
        storageClassname: pvClass
        capacity:
                storage: 2Gi
        accessMode:
                - ReadWriteMultiple
        hostPath:
                path: /u01/data

pvc.yml
-------
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
        name: pvc1
spec:
        storageClassname: pvClass
        accessMode:
                - ReadWriteOnce
        resources:
                requests:
                        storage: 1Gi

speed-pod.yml
-------------
apiVersion: v1
Kind: Pod
metadata:
        name: speedpod
spec:
        containers:
                - name: speedcontainer
                        image: techsriman/speeddep:2.0
                        ports:
                                - name: tomcatport
                                        containerPort: 8080
                                        protocol: TCP
                        volumeMounts:
                                - name: speedvolume
                                        mountPath: /u01/app

        volumes:
                - name: speedvolume
                        persistentVolumeClaim:
                                claimName: pvc1
```

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

11-03-2023

Persistent Volume and Persistent Volume claims

Persistent Volume = it is a storage that is defined on the kubernetes cluster and

it is created by the kubernetes administrator reserving the storage location on the worker nodes
Persistent Volume Claim = is a request for that storage to be consumed by the devops engineer aspart of the pod application

There are 3 attributes we need define to create an persistent volume
1. storageClassName
2. accessMode
3. capacity

```
mysql-pv.yml
---------------
apiVersion: v1
kind: PersistentVolume
metadata:
        name: mysqlpv
spec:
        storageClassName: mysqlStorageClass
        capacity:
                storage: 2Gi
        accessMode:
                -ReadWriteMultiple
        hostPath:
                path: /u01/data #location on the workernode the volume should be created
```

to use the part of the storage we need to create an pvc

```
mysql-pvc.yml
-------------
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
        name: mysqlpvc
spec:
        storageClassName: mysqlStorageClass
        accessModes:
                - ReadWriteOnce
        resources:
                requests:
                        storage: 1Gi
```

```
mysql-pod.yml
-------------
apiVersion: v1
kind: Pod
metadata:
        name: mysqlpod
spec:
        container:
                - name: mysqlcontainer
                        image: mysql:8.0
                        ports:
                                - name: mysqlport
                                  containerPort: 3306
                                        protocol: TCP
                        volumeMounts:
                                - name: mysqlvolume
                                        mountPath: /u01/mysql
        volumes:
                - name: mysqlvolume
                        persistentVolumeClaim:
                                claimName: mysqlpvc
```

==
==

13-03-2023

Ingress
Ingress is an another type of Service in kubernetes, that is used for exposing the pod applications to the external world.
Ingress is an another controller of the kubernetes, that receives the request from the external world and routes it to the Service component.
by default withan kubernetes or minikube install Ingress controller will not be available, we need to enable the ingress explicitly

```
minikube addons enable ingress
```

There are different ingress controller component providers are there in the market one such provider is Nginix
These are internally httpd servers which receives the request over an domainName and proxy the request to the backend
now we need to write an ingressService that receives request over a domain
or host: covido.org
then forward the request to the ClusterIP Service
(clusterip)
covido-service.yml

```
-------------------
apiVersion: v1
kind: Service
metadata:
  name: covidoclusteripservice
spec:
  type: ClusterIP
  selector:
    app: covido
    version: v1
  ports:
   - port: 8080
     target: 8080
```

```
covido-ingress.yml
----------------------
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
        name: covidoingress
        annotations:
                niginx.ingress.kubernetes.io/rewrite-target /$1
spec:
        rules:
                - host: covido.org
                        http:
                                paths:
                                        - path: /
                                                pathType: Prefix
                                                backend:
                                                        service:
                                                                name: 
covidoclusteripservice
                                                                port:
                                                                        number: 
8080
```

http://covido.org/index

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

15-03-2023

Job
Job is used for performing an operation on the kubernetes cluster. A Job creates one or more Pods and executes util the specified number of executions are successful

```
numbers-job.yml
apiVersion: batch/v1
kind: Job
metadata:
        name: numbersJob
spec:
        template:
                metadata:
      name: numberspod
                spec:
                        containers:
                                - name: numberscontainer
                                        image: ubuntu:20.03
                                        command:
                                                - "bin/bash"
                                                - "-c"
                                                - "for i in 1 2 3 4 5 6 8 9 0 ; do
echo $i ; done"
                                        restartPolicy: Never
```

DaemonSet

```
nginx-daemonset.yml
--------------------
apiVersion: apps/v1
kind: DaemonSet
metadata:
        name: nginxdaemonset
spec:
        selector:
                matchLabels:
                        name: nginx
        template:
                metadata:
                        labels:
                                name: nginx
                spec:
                        containers:
                                - name: nginx
                                        image: nginx
```

==
==

19-03-2023

Premetheus can be installed on minikube using helm charts

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

helm install prometheus  prometheus-community/prometheus
```

then expose the prometheus server over nodeport

```
kubectl expose service prometheus-server --type=NodePort --target-port=9090
--name=prometheus-server-ext
service/prometheus-server-ext exposed
```

to access the prometheus dashboard run the below command

```
minikube service prometheus-server-ext
```