Kubernetes

How to Become a Data Scientist at Microsoft? #scaler #datascience (youtube.com)

https://www.tutorialspoint.com/kubernetes/index.htm

- Google created this Kubernetes. It is an open source tool.
- Pre-requisite is docker. Called as k8s -- 8 letters between k and s.
- It is a **container orchestration tool**.
- Kubernetes creates cluster, deploy and manage clusters.
- By using Kubernetes we form a cluster.K8S schedules, runs and managers isolated containers.
- Convert isolated containers running on different hardware into a cluster.
- In AWS we have a service called EKS (Elastic Kubernetes service)

❖ Kubernetes is an open-source container management tool that automates container deployment, container scaling and load-balancing.
❖ It schedules, runs and manages isolated containers that are running virtual/physical/cloud machines.
❖ Supported by all cloud providers.

https://www.tutorialspoint.com/kubernetes/kubernetes_kubectl_commands.htm

Features of Kubernetes

1) Orchestration (clustering any no. of containers on different hardware)

2) Auto scaling

3) Auto healing (new containers in place of crashed containers similar to handling failover scenarios in docker swarm)

4) load balancing

5) Rollback (going to previous versions)

Kubernetes Architecture

❖ Kubernetes does not understand containers. Kubernetes can understand only pods.
❖ Pod is atomic (smallest) unit of deployment in Kubernetes.
❖ Pod consists of one or more docker containers.

- ❖ Pod runs on node. Node is controlled by Kubernetes master
- ❖ Node is also called minion.
- ❖ Cluster is combination of 1 master and multiple nodes.
- ❖ Kubernetes master is also called as control plane.

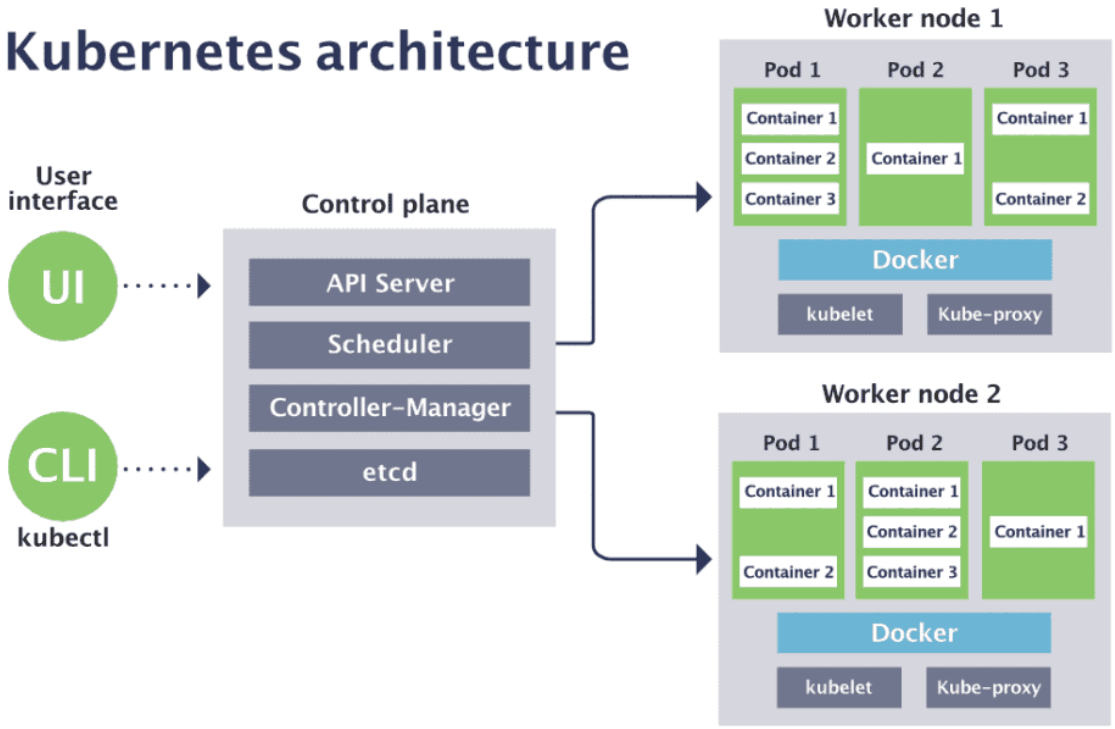
Pods

So far, we were running containers directly in docker. In Kubernetes we have Pods and we don't run containers directly. We always create pods and manage pods.

This is the smallest entity in the Kubernetes cluster. A pod can contain 1 or many containers.

We can have a pod running web service, database service, or the pod can be running multiple containers.

- Kubernetes Architecture
  - Master Node
    - API Server
    - ETCD Server
    - Kube Scheduler
  - Controller manager
    - Node Components
    - Kubelet
    - Pods
  - Overlay Network



Node :- Node is a working machine in k8s cluster that runs containerized applications. Kubectl sends the request to the API server.

- API server stores the information in the Etcd storage.
- The scheduler will pick up such information and if the information is like Create pod/container, it will find the right node based on the algorithms and will identify the worker node and then send the information to the Kubelet on that node.
- Kubelet will receive the information and do things like pulling images, running containers, assigning port, etc.

If we say we need 4 pods or replication of this container, then this request goes to the controller manager and the controller manager monitors/manages that and will make sure that 4 pods will be created inside the worker node.

Control plane makes global decisions about the cluster. For example, create new pods, create cloud load balancer

Master Node

Master Node is called the control plane and it has 4 further things, namely API Server, ETCD server, Scheduler and Controller Manager.

API Server

This enables all the communication b/w API; we are going to talk to Kube API Server only. It takes the request and sends it to other services.

We can use Kubectl CLI to manage the Kubernetes Cluster.

Kubectl sends the request to the API server and then API Server responds back.
kube api server acts like a receptionist. It receives the yaml file and passes the request to kube scheduler.

As a Devops engineer you create a yaml file (.yml) file. What this yaml file contains?

1) No of nodes you want?

2) Each node should have how many pods

3) Each pod should contain how many containers

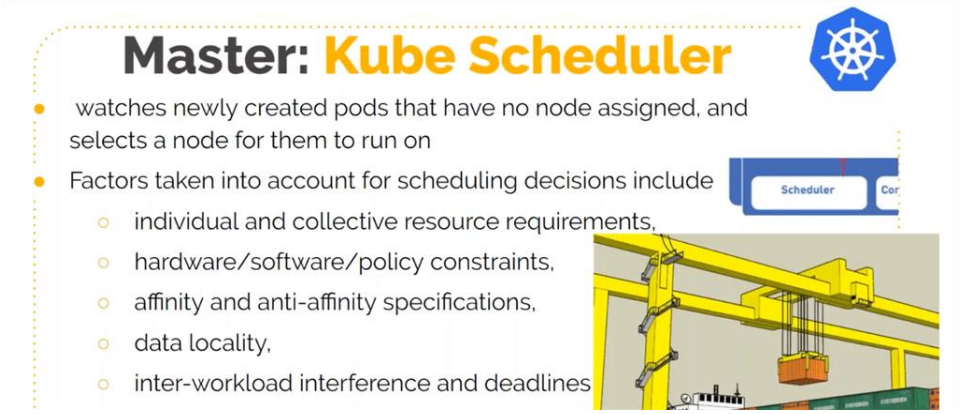
All the above information will be available in yaml file. This file is also called manifest file. This document should be provided to Kubernetes master.

ETCD Server

- Kube API Server stores all the information in Etcd and other services also reads and store the information in the Etcd storage**.**
- If we have multiple Kubernetes masters, then we can set up multiple ETCD Server clustered together syncing all the data.
- It should be backed up regularly.
- It stores the current state of everything in the cluster here at the ETCD server.
- Etcd is also called cluster store. It has the information of the complete cluster. It is used to store the data of master, node and containers. Data is stored in key-value pair.

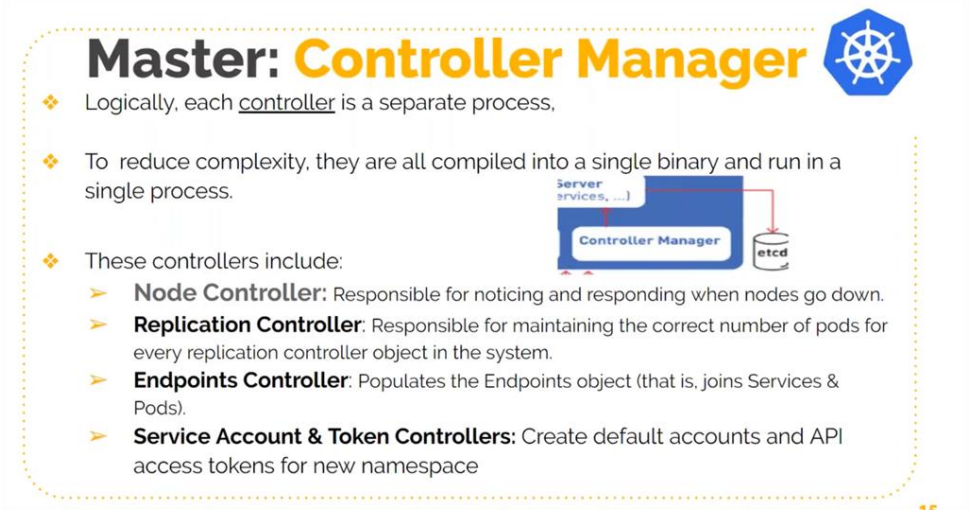
Kube Scheduler

It picks up the container and puts it on the right node based on different factors. kube scheduler will take the action. So kube scheduler will create pods and containers.



Controller Manager

All the controllers see that the desired state and actual state are same in the Kubernetes cluster.
There are different types of controllers and all are part of the Controller Manager.
Like →Node controller → Replication controller →Deployment controller →Namespace controller →Cloud Control Manager →Endpoint controller →PV-Protection controller →Service account controller →Cron Job →Job Controller
The node controller is responsible for checking the status of the node



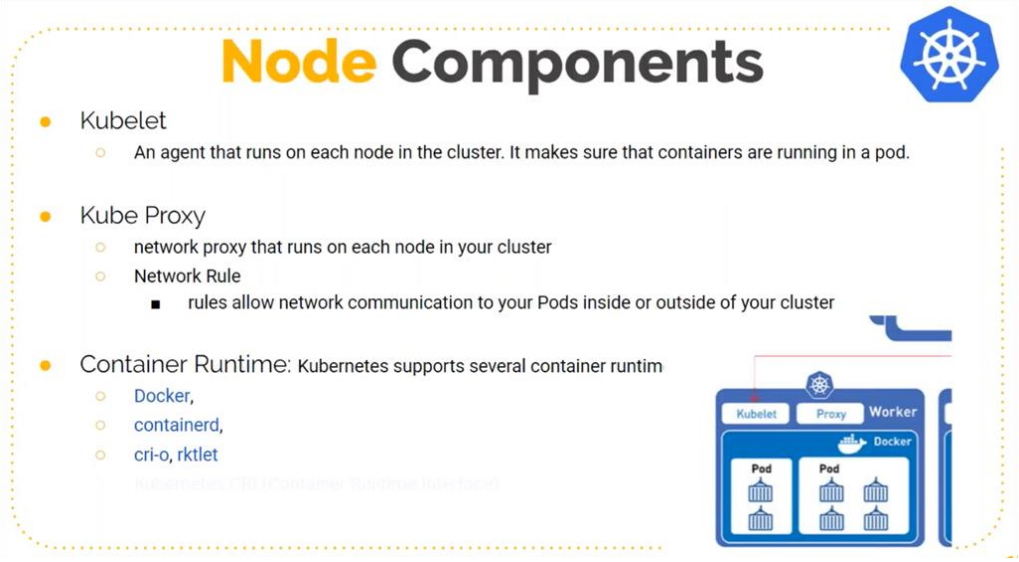
Node Components

There are 3 node components:

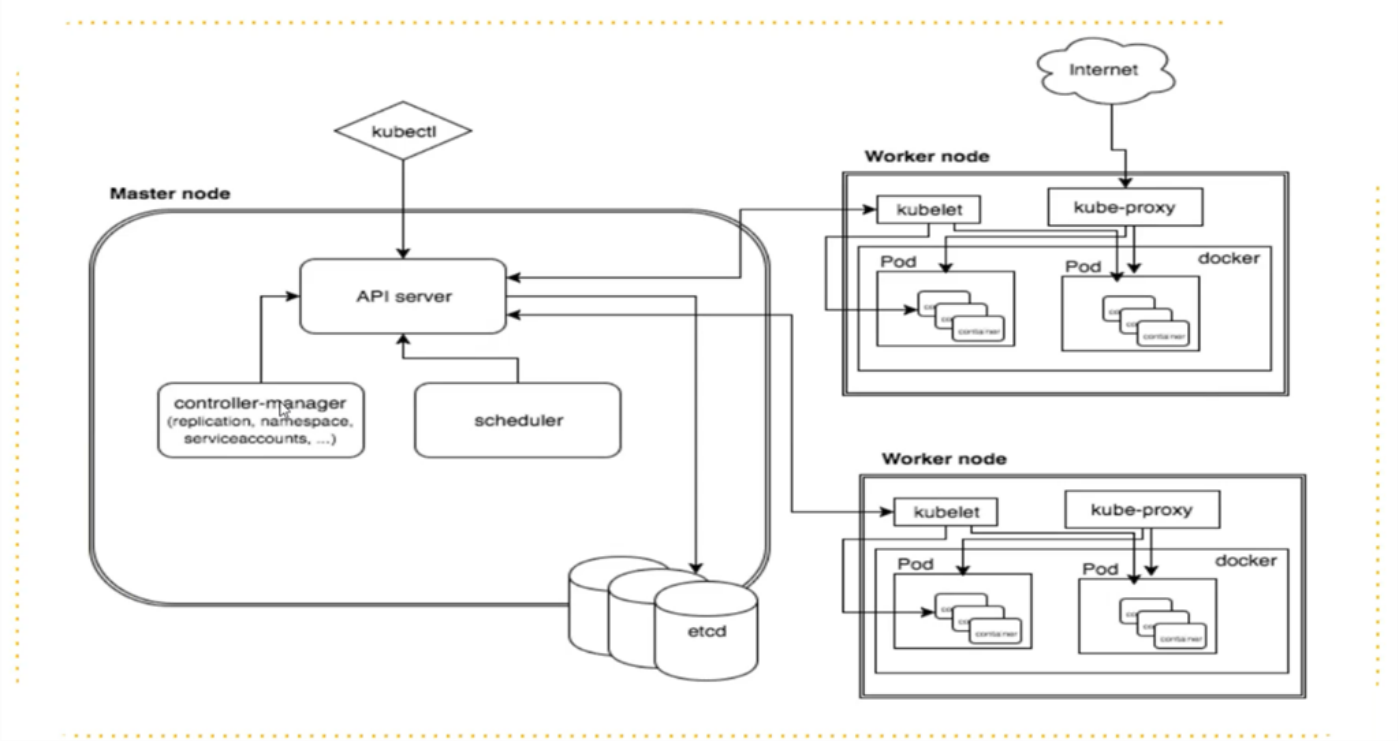
- Kubelet is the agent that listens to the request of master and is going to do all the heavy lifting.

- Suppose if it gets a request that it needs to launch suppose X no of pods. So Kubelet is going to fetch the image, run the container from the image, etc.

- There are different add-ons that can be added to the Kubernetes, like monitoring for container resources to log at the cluster level, or we can use third-party tools like Splunk, etc.



- Kubelet -- is also called as agent, as it listens to kubernetes master. kube-scheduler component communicates to kubelet. kubelet communicates to container engine (docker) so that containers are created. Note: Containers are created in pods
- kube proxy -- Maintains network rules on nodes, implementing Kubernetes Service abstraction by forwarding requests to the appropriate pods based on IP and port.
- Container Runtime: kubectl doesn't run container directly. It uses container runtime where we need to install software responsible for running containers, such as Docker, containerd on every single node.



Container Orches Containerization

Dockerswarm ----------> Docker

Kubernetes ---------- Docker / XYZ (means it can be used with other containerization tools if required)

Kubernetes Terminology

In docker Swarm, Manager machine takes the load.

In Kubernetes Manager is called as Master.

Kubernetes master does not take up the load. It only distributes load to slaves/ nodes.

Nodes are also called Minion. Minions combined together called as cluster.

Smallest Object that kubernetes can create is pod. Within the pod, we have the container.

Kubernetes commands are always triggered using kubectl.

AWS, is expensive

Freeways to work on kubernetes is katakoda Goto https://www.katacoda.com/

Learn --- --- Kubernetes Introduction -- Start Course -- Launch Multinode cluster -- Start Scenario

We have one more site

https://labs.play-with-k8s.com/ using which we can practice Kubernetes.

But, both the options will be slow.

++++++++++++++++++++++++++++++++

We learn kuberntes on GCP, as AWS is expensive.

Sign up to GCP account using gmail credentials. (Free trial comes with USD 300)

https://cloud.google.com/

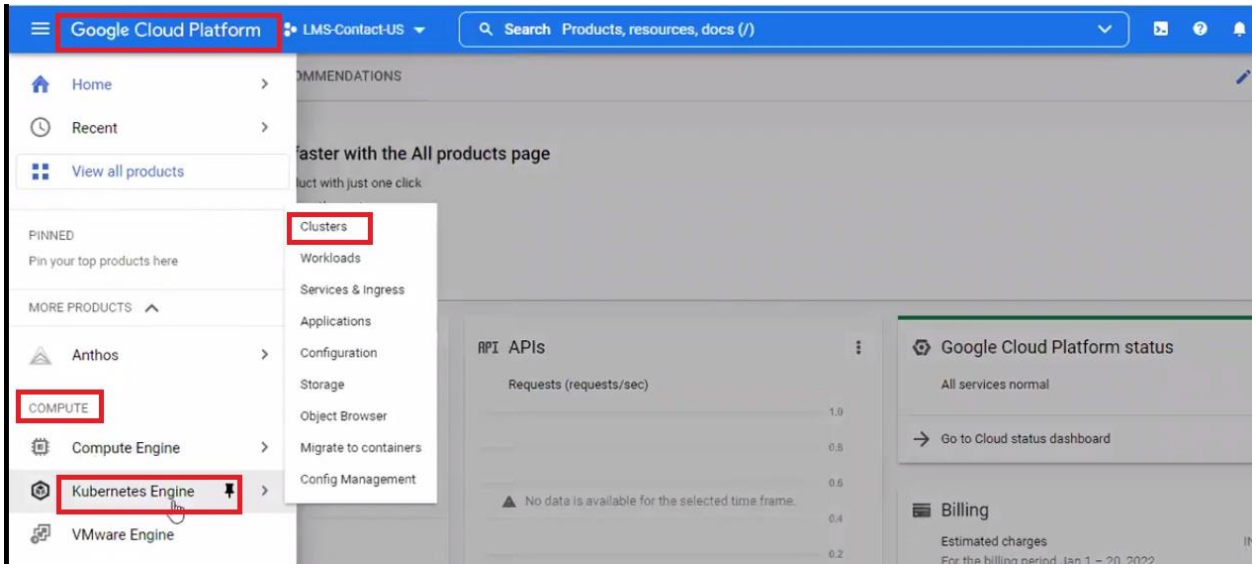
Sign in using gmail

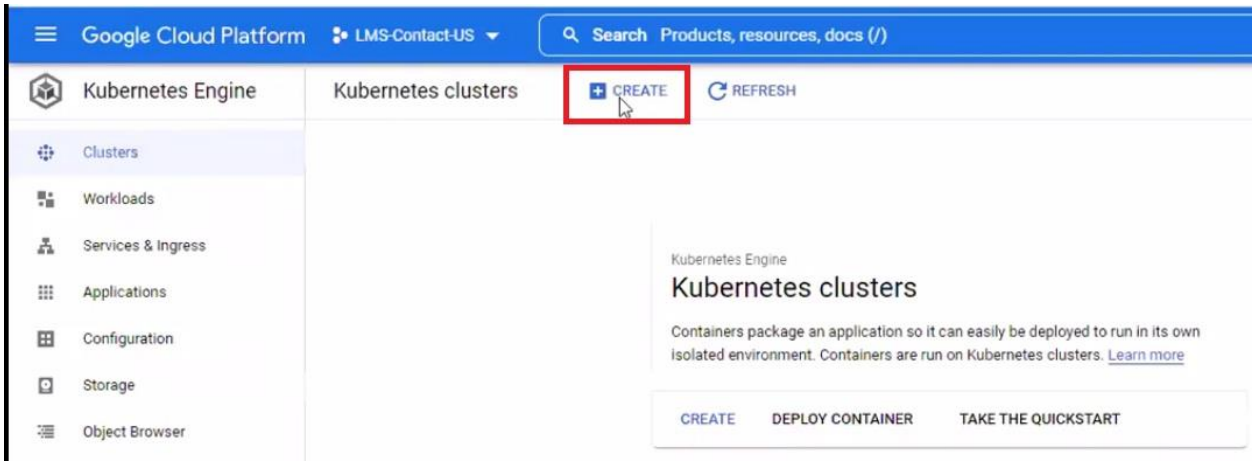
Creating Cluster

Click on console

You will enter into google cloud platform console

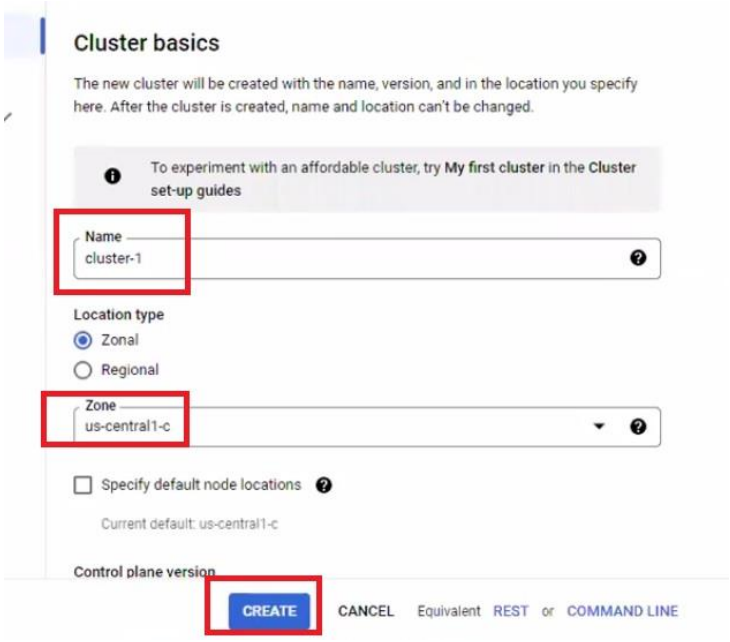
Navigation Menu --- Kubernetes Engine -- Clusters -- Create cluster -- Create



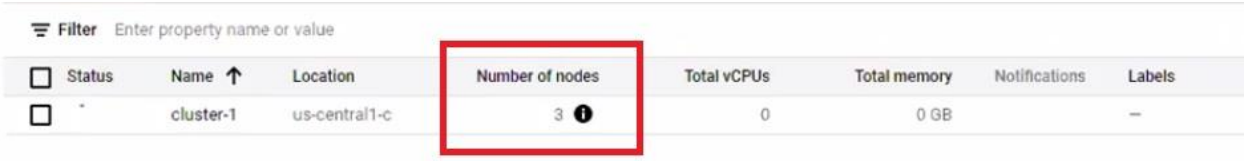


Select the standard

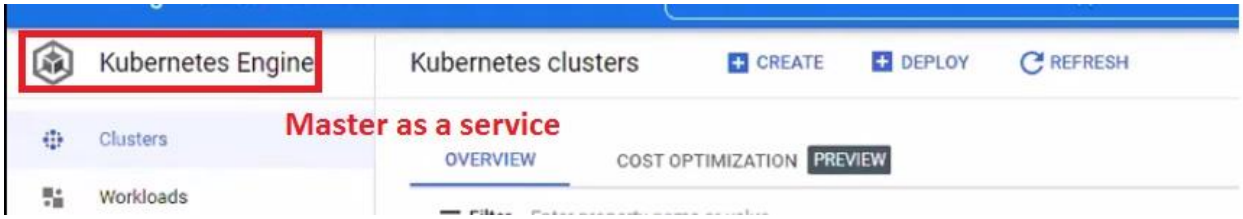
Give cluster name and location



Observation: Cluster size is 3. By default, it creates 3 node clusters.

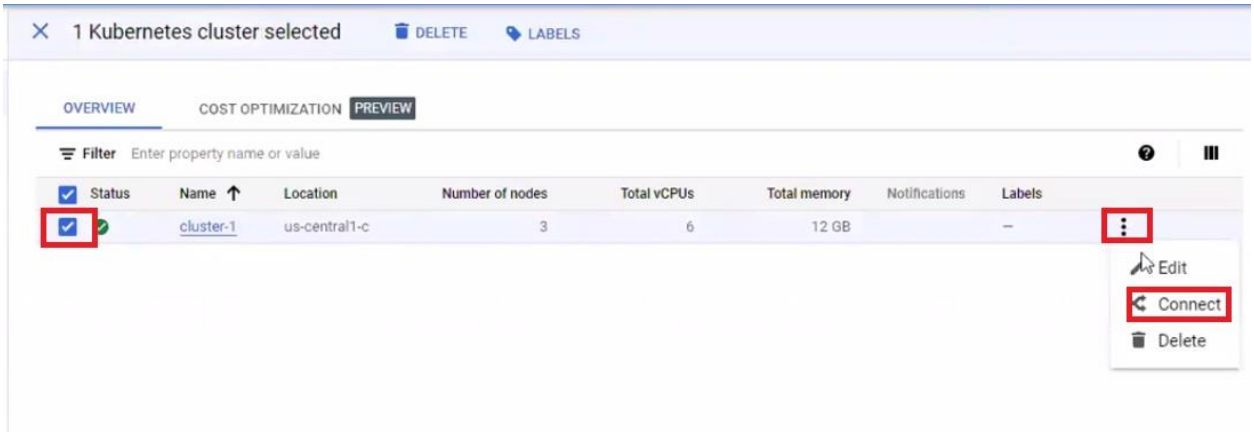


Master Machine is not provided as a Linux server. It is given as a service. As it is a service, it never fails. So, we do not need to worry about master. (SAAS Software As A Service)

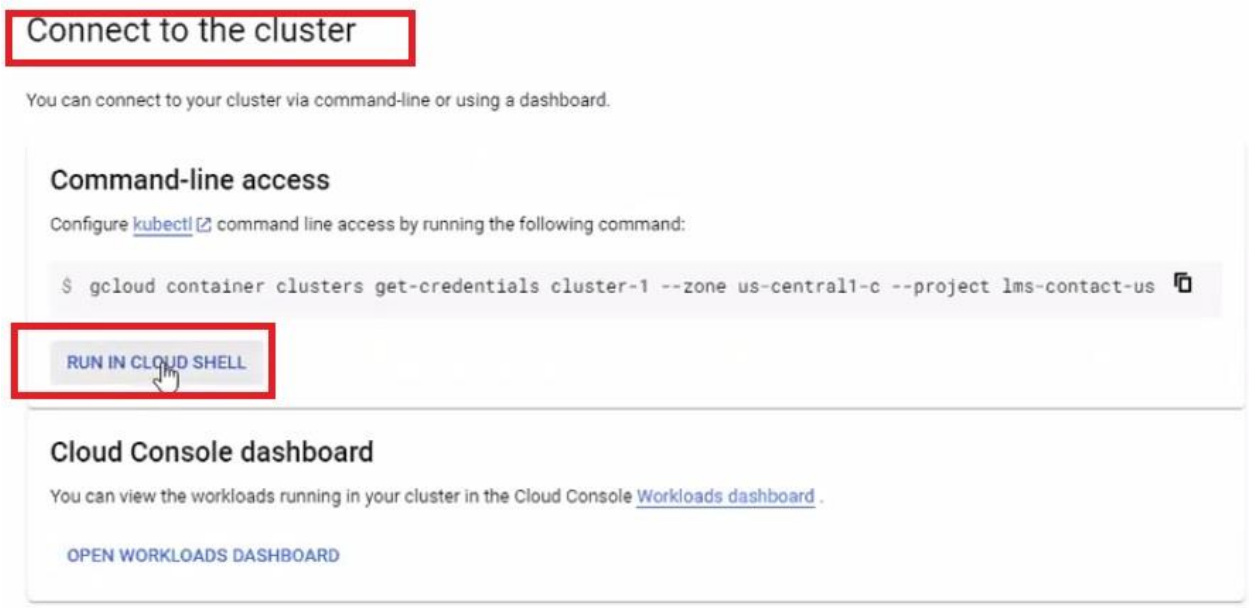


To connect to the cluster

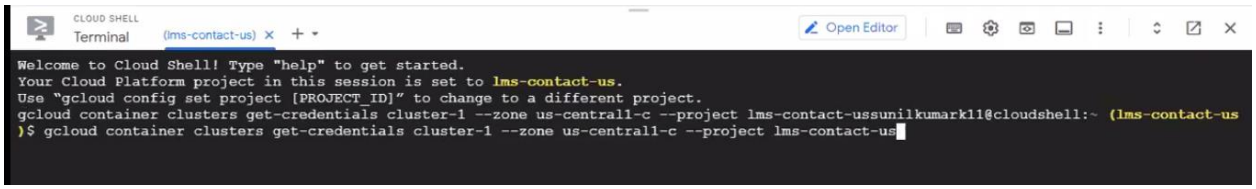
Select the cluster we need to connect. Click on 3 dots at the end of cluster and click on connect



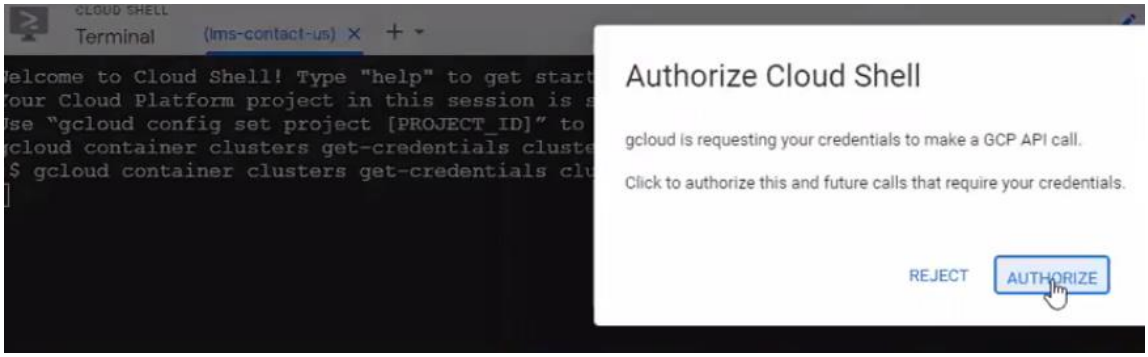
In GCP, Cloud Shell is the terminal, used to connect to the cluster.



Command will be copied automatically. Click Enter



Click authorize



To see list of nodes →kubectl get nodes (we can see the nodes)

```
No resources found in default namespace.
sunilkumark11@cloudshell:~ (lms-contact-us)$ kubectl  get nodes
NAME                                          STATUS   ROLES    AGE     VERSION
gke-cluster-1-default-pool-1aa095b2-16h9      Ready    <none>   6m28s   v1.21.5-gke.1302
gke-cluster-1-default-pool-1aa095b2-fmsh      Ready    <none>   6m28s   v1.21.5-gke.1302
gke-cluster-1-default-pool-1aa095b2-p42q      Ready    <none>   6m27s   v1.21.5-gke.1302
```

To see list of nodes along with ip addresses→kubectl get nodes -o wide

```
sunilkumark11@cloudshell:~ (lms-contact-us)$
sunilkumark11@cloudshell:~ (lms-contact-us)$ kubectl get nodes -o wide
NAME                                          STATUS   ROLES    AGE   VERSION           INTERNAL-IP   EXTERNAL-IP      OS-IMAGE
                KERNEL-VERSION   CONTAINER-RUNTIME
gke-cluster-1-default-pool-1aa095b2-16h9      Ready    <none>   20m   v1.21.5-gke.1302   10.128.0.2    35.232.233.254   Container-Optimized OS
 from Google   5.4.144+         containerd://1.4.8
gke-cluster-1-default-pool-1aa095b2-fmsh      Ready    <none>   20m   v1.21.5-gke.1302   10.128.0.3    34.134.212.241   Container-Optimized OS
 from Google   5.4.144+         containerd://1.4.8
gke-cluster-1-default-pool-1aa095b2-p42q      Ready    <none>   20m   v1.21.5-gke.1302   10.128.0.4    35.239.154.103   Container-Optimized OS
 from Google   5.4.144+         containerd://1.4.8
```

Kubernetes uses various types of objects.

1 **Pod:** This is a layer of abstraction on top of a container. This is the smallest object that Kubernetes can work on. In the pod, we have the container. kubectl commands will work on the pod and pod communicates the instructions to the container.

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

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

4. **Secrets**: This is used for passing encrypted data to the pods.

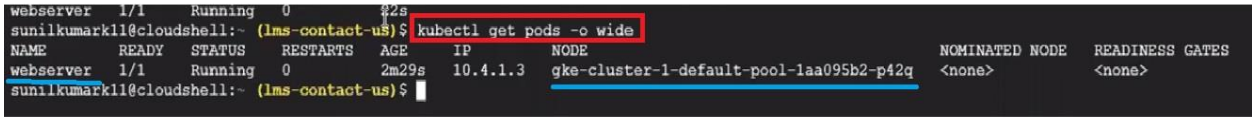
5. **ReplicaSet** / **Replication Controller**: This is used for managing multiple replicas of a pod to perform activities like load balancing and autoscaling.

6. **Deployment:** This is used for performing all activities that a ReplicaSet can do. It can also handle rolling updates.

Create Cluster. Open cloud shell terminal.
Command to create a pod→kubectl run --image tomcat webserver (Webserver is pod name)
To see list of pods→kubectl get pods

```
sunilkumark11@cloudshell:~ (lms-contact-us)$ kubectl run --image tomcat  webserver
pod/webserver created
sunilkumark11@cloudshell:~ (lms-contact-us)$ kubectl get pods
NAME        READY   STATUS              RESTARTS   AGE
webserver   0/1     ContainerCreating   0          9s
sunilkumark11@cloudshell:~ (lms-contact-us)$
```

To see on what node pod is running →kubectl get pods –o wide



Along with pod name we can see on what node it is running

If we do not specify replicas, it creates only one replica.

To delete the pod→ kubectl delete pods webserver (webserver is pod name)

Kubernetes performs container orchestration by using definition files. Definition files are .yaml files

Definition file, will have 4 top level elements

- **apiVersion** - Which version of the Kubernetes API you're using to create this object
- **kind** - What kind of object you want to create
- **metadata** - Data that helps uniquely identify the object, including a name string, UID, and optional namespace
- **spec** - What state you desire for the object

apiVersion:
Depending on kubernetes object we want to create, there is corresponding code library we want to use. apiVersion referes to code library

Kind	apiVersion
Pod	v1
Service	v1
NameSpace	v1
Secrets	v1
ReplicaSet	apps/v1
Deployment	apps/v1

Kind:
Refers to kubernetes object which we want to create.
Ex: Pod, Replicaset, service etc

Metadata:
Additional information about the kubernets object
like name, labels etc

Spec:
Contains docker container related information like image name,
environment variables, port mapping etc.
++
Connect to cluster by using cloud shell.

$ mkdir *directoryname* →*creates a directory*
$ cd *directoryname*
$ls right now directory is empty

Ex1: Create a pod definition file to start nginx in a pod. Name the pod as nginx-pod, name the container as appserver.
Command→**vim pod-definition1.yml or cat > pod-definition1.yml**

```
sunilkumark11@cloudshell:~/samplefiles (lms-contact-us)$
sunilkumark11@cloudshell:~/samplefiles (lms-contact-us)$ vim pod-definition1.yml
```

Now copy the below details
---
apiVersion: v1
kind: Pod
metadata:
name: nginx-pod
labels:
author: sunil
type: reverse-proxy
spec:
containers:
- name: appserver
image: nginx

:wq (to save and exit)

Command to run the definition file→kubectl create –f filename.yaml
Pod is created.

If we had a modified the file and then we run kubectl apply -f filename.yaml to apply the configuration defined in the YAML file to the cluster

++++++++++++++++++++++++++++++++

Replication Controller:

The replica set and the replication controller's key difference is that the replication controller only supports equality-based selectors whereas the replica set supports set-based selectors.

This is a high-level object used for handling multiple replicas of a specific pod. Here we can perform load balancing and scaling.

- ✓ Replication Controller is one of the key features of Kubernetes, which is responsible for managing the pod lifecycle.
- ✓ It is responsible for making sure that the specified number of pod replicas are running at any point of time.
- ✓ It is used in time when one wants to make sure that the specified number of pod or at least one pod is running. It has the capability to bring up or down the specified no of pod.
- ❖ A Replication Controller is similar to a process supervisor, but instead of supervising individual processes on a single node, the Replication Controller supervises multiple pods across multiple nodes.

Replication Controller uses keys like replicas, template" etc in the "spec" section.

In template section we can give metadata related to the pod and also use another spec section where we can give containers information.

Ex: Create a replication controller for creating 3 replicas of httpd

Command→ vim replication-controller.yml

```
---
apiVersion: v1
kind: ReplicationController
metadata:
 name: httpd-rc
 labels:
  author: sunil
spec:
 replicas: 3
 template:
  metadata:
   name: httpd-pod
   labels:
    author: sunil
  spec:
   containers:
    - name: myhttpd
      image: httpd
      ports:
       - containerPort: 80
         hostPort: 8080
```

:wq

hostPort: 8080 → to open the port 8080
kubectl delete --all pods (To delete all the existing pods)
kubectl get pods (No pods available)

Open the port
---
gcloud compute firewall-rules create rule21 --allow tcp:8080 (external command to open port if not done at the time of creating)

to create replica→ kubectl create -f replication-controller.yml

kubectl get pods (We should get 3 pods), kubectl get pods -o wide (Observation, 3 pods are distributed in 3 nodes), kubectl get nodes -o wide

Suppose we delete a pod by →kubectl delete podname, replica controller will try to create another pod before it gets deleted. Thus it does autohealing.

To view the following →kubectl get pods -w

```
aveerama-mac:~ aveerama$ kubectl get pods -w
NAME                                READY   STATUS    RESTARTS   AGE
nginx-deployment-9456bbbf9-hkkmw    1/1     Running   0          114s


nginx-deployment-9456bbbf9-hkkmw    1/1     Terminating   0          2m
nginx-deployment-9456bbbf9-64vrl    0/1     Pending       0          0s
nginx-deployment-9456bbbf9-64vrl    0/1     Pending       0          0s
nginx-deployment-9456bbbf9-64vrl    0/1     ContainerCreating   0          0s
nginx-deployment-9456bbbf9-hkkmw    0/1     Terminating         0          2m1s
nginx-deployment-9456bbbf9-hkkmw    0/1     Terminating         0          2m1s
nginx-deployment-9456bbbf9-hkkmw    0/1     Terminating         0          2m1s
nginx-deployment-9456bbbf9-64vrl    1/1     Running             0          2s
```

Take external IP (Public IP) of any node
34.122.234.70:8080

34.134.16.68:8080

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

++++++++++++++++++++++++++

ReplicaSet

Pod is the smallest Kubernetes object, which we worked on. Next Level is replication controller.
ReplicaSet is similar to replication controller.
In ReplicaSet, we have an additional field in spec section called as "selector" field.

- This selector uses a child element called "matchLabels", where it will search for pods based on a specific label name, and adds them to the cluster.

Ex: Create a replicaset file to start 4 tomcat replicas and then perform scaling

vim replica-set.yml

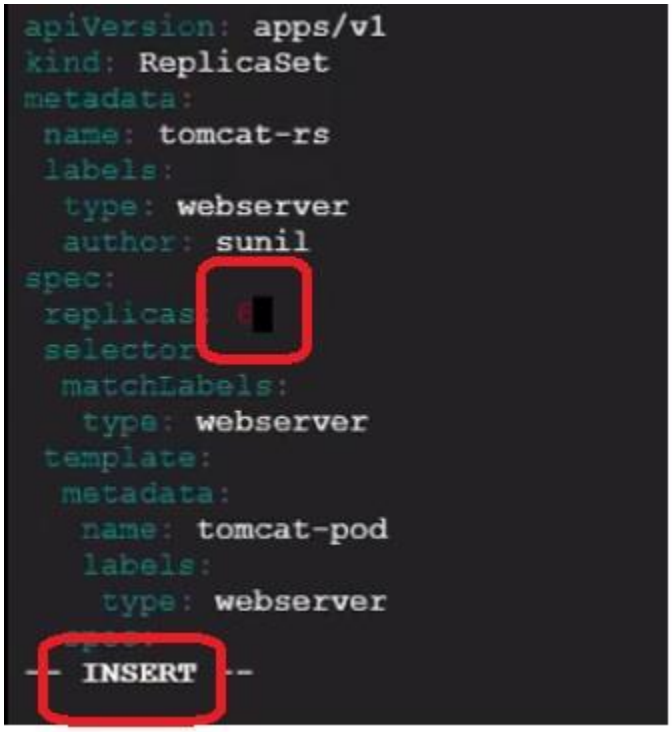
```
---
apiVersion: apps/v1
kind: ReplicaSet
metadata:
 name: tomcat-rs
 labels:
  type: webserver
  author: sunil
spec:
 replicas: 4
 selector:
  matchLabels:
   type: webserver
 template:
  metadata:
   name: tomcat-pod
   labels:
    type: webserver
  spec:
   containers:
    - name: mywebserver
      image: tomcat
      ports:
       - containerPort: 8080
         hostPort: 9090

 :wq
```

kubectl create -f replica-set.yml
kubectl get pods (We should get 4 pods)
kubectl get replicaset

```
pod-definition1.yml  replica-set.yml  replication-controller.yml
sunilkumark11@cloudshell:~/samplefiles (lms-contact-us)$ kubectl get pods
No resources found in default namespace.
sunilkumark11@cloudshell:~/samplefiles (lms-contact-us)$ kubectl create -f replica-set.yml
replicaset.apps/tomcat-rs created
sunilkumark11@cloudshell:~/samplefiles (lms-contact-us)$ kubectl get pods
NAME              READY   STATUS              RESTARTS   AGE
tomcat-rs-msmtt   0/1     ContainerCreating   0          12s
tomcat-rs-nkgp8   0/1     Pending             0          12s
tomcat-rs-pf481   0/1     ContainerCreating   0          12s
tomcat-rs-plnz4   0/1     ContainerCreating   0          12s
sunilkumark11@cloudshell:~/samplefiles (lms-contact-us)$ kubectl  get replicaset
NAME         DESIRED   CURRENT   READY   AGE
tomcat-rs    4         4         3       30s
sunilkumark11@cloudshell:~/samplefiles (lms-contact-us)$
```

Let's perform scaling from 4 pods to 6 pods

Option 1: We can open the definition file and make changes in the code from 4 to 6 in replicas field.
vim replica-set.yml



Now, we should not use create commands, we should use replace command.

kubectl replace -f replica-set.yml
kubectl get pods (We should get 6 pods)

Option 2:

kubectl scale --replicas=2 -f replica-set.yml
kubectl get pods (We should get 2 pods)
++

Ex 2:
---
vim pod-definition2.yml

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

:wq

Command to run the definition file
kubectl create -f pod-definition2.yml

To get the list of pods
kubectl get pods

To get the list of pods along with IP address and which node the pod is running
---

kubectl get pods -o wide

TO get more details about the pod
---
kubectl describe pods postgres-pod

or

kubectl describe pods postgres-pod | less

Ex3:

vim pod-definition3.yml

```
---
apiVersion: v1
kind: Pod
metadata:
 name: jenkins-pod
 labels:
  author: sunil
  ci: cd
spec:
 containers:
  - name: myjenkins
    image: jenkins/jenkins
    ports:
     - containerPort: 8080
       hostPort: 8080
```

:wq

How to open the port?
---
gcloud compute firewall-rules create rule35 --allow tcp:8080
gcloud compute firewall-rules create rule2 --allow tcp:9090
((kubectl expose deployment myapp --port=8080 --type=LoadBalancer --name=myapp-service))

kubectl create -f pod-definition3.yml

kubectl get pods -o wide

Take a note on the node in which the pod is running.

gke-cluster-1-default-pool-9fb99245-q1nm

TO get the list of nodes
---
kubectl get nodes -o wide

Take the external IP of the node

35.223.183.189:8080
34.68.242.87:8080

Open browser (chrome)

35.223.183.189:8080   (we should get the jenkins page)
++++++++++++++++++++

Deployment Object

In Kubernetes a deployment is a method of launching a pod with containerized applications and ensuring that the necessary number of replicas is always running on the cluster.

This is also a high-level object which can be used for scaling, load balancing and performs rolling updates.
Deployment is a method of converting images to containers and then allocating those images to pods in the Kubernetes cluster. This also helps in setting up the application cluster which includes deployment of service, pod, replication controller and replica set. The cluster can be set up in such a way that the applications deployed on the pod can communicate with each other.

Create a deployment file to run nginx 1.7.9 with 3 replicas.
Later perform a rolling update to nginx 1.9.1

vim deployment.yml

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

:wq

kubectl get all (we have one default service running)

```
sunilkumark11@cloudshell:~/samplefiles (lms-contact-us)$ kubectl get all
NAME                    READY   STATUS    RESTARTS   AGE
pod/jenkins-pod         1/1     Running   0          7m52s
pod/postgres-pod        1/1     Running   0          11m
pod/tomcat-rs-msmtt     1/1     Running   0          19m
pod/tomcat-rs-pf481     1/1     Running   0          19m

NAME                 TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)   AGE
service/kubernetes   ClusterIP   10.8.0.1     <none>        443/TCP   41m

NAME                            DESIRED   CURRENT   READY   AGE
replicaset.apps/tomcat-rs       2         2         2       19m
sunilkumark11@cloudshell:~/samplefiles (lms-contact-us)$
```

kubectl create -f deployment.yml
TO check, if the deployment is created or not
---
kubectl get deployment (we can see 1 deployment object)

```
deployment.apps/nginx-deployment created
sunilkumark11@cloudshell:~/samplefiles (lms-contact-us)$ kubectl get deployment
NAME               READY   UP-TO-DATE   AVAILABLE   AGE
nginx-deployment   3/3     3            3           11s
```

kubectl get pods (we should get 3 pods)
We can anyways perform scaling, apart from that we can perform rolling updates.
kubectl get all (we get all the objects)

```
sunilkumark11@cloudshell:~/samplefiles (lms-contact-us)$ kubectl get all
NAME                                    READY   STATUS    RESTARTS   AGE
pod/jenkins-pod                         1/1     Running   0          9m40s
pod/nginx-deployment-7778fb954b-2zkcf   1/1     Running   0          41s
pod/nginx-deployment-7778fb954b-rhdz4   1/1     Running   0          41s
pod/nginx-deployment-7778fb954b-w755t   1/1     Running   0          41s
pod/postgres-pod                        1/1     Running   0          13m
pod/tomcat-rs-msmtt                     1/1     Running   0          20m
pod/tomcat-rs-pf481                     1/1     Running   0          20m

NAME                 TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)   AGE
service/kubernetes   ClusterIP   10.8.0.1     <none>        443/TCP   43m

NAME                               READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/nginx-deployment   3/3     3            3           42s

NAME                                          DESIRED   CURRENT   READY   AGE
replicaset.apps/nginx-deployment-7778fb954b   3         3         3       42s
replicaset.apps/tomcat-rs                     2         2         2       20m
sunilkumark11@cloudshell:~/samplefiles (lms-contact-us)$
```

When we create a deployment object, a replica set will be created. Take a note of the full name of the deployment object
deployment.apps/nginx-deployment

To perform rolling update

Syntax: kubectl set image deployment/deployment_name container_name=new_image_name:tag

kubectl --record deployment.apps/nginx-deployment set image deployment.v1.apps/nginx-deployment nginx=nginx:1.9.1
We get a message (image updated)

```
sunilkumark11@cloudshell:~/samplefiles (lms-contact-us)$ kubectl --record deployment.apps/nginx-deployment  set image deployment.v1.apps/nginx-deployment nginx=nginx:1.9.1
Flag --record has been deprecated, --record will be removed in the future
deployment.apps/nginx-deployment image updated
deployment.apps/nginx-deployment image updated
sunilkumark11@cloudshell:~/samplefiles (lms-contact-us)$
```

The process of updating an image in a Kubernetes deployment is typically referred to as "rolling update" or "rolling deployment."

kubectl get pods
when performing rolling update an extra pod will be created
To know more about pod
---
kubectl describe pods podname
nginx-deployment-6fdc797dc6-qrlqb

kubectl describe pods nginx-deployment-6fdc797dc6-qrlqb | less
we can see as Image: nginx:1.9.1
:q
(It will take some time)
kubectl get pods

++

Service Object

Kubernetes services connect a set of pods to an abstracted service name and IP address. Services provide discovery and routing between pods

We have 3 pods in a cluster with respective ip address for communication. Suppose a pod in a cluster gets deleted. As replication controller/replication set is present pod will be recreated with

a different ip address. So, it will be difficult for user to communicate with that newly created pod. What if this the case with other pods? So, we do use service here in the deployment which creates communication b/w users and pods irrespective of ip addresses. This is done with the help of labels and selectors. Instead of ip address of pods, all pods will be given name and we can reach out to pod through labels. Even when a pod gets recreated, it will be under same label.

Labels and selectors are crucial for various Kubernetes operations, such as:

Selecting Pods for Services.
Defining Replica Set or Deployment selectors to manage Pods.
Grouping Pods for monitoring or management purposes.
Applying policies or access control based on labels.

Service object is used for **network load balancing** and port mapping.
Load balancing in Kubernetes involves distributing incoming network traffic across multiple pods (instances of your application) to ensure optimal resource utilization, high availability, and reliability.
Service discovery in Kubernetes involves automatically locating and connecting to the appropriate service endpoints (Pods) that provide a particular functionality or service within the cluster.
Service Object uses 3 ports
1. Target port - It is pod or container port
2. port - Refers to service port.
3. hostPort - Refers to host machine port to make it accessible from external network.

Service Objects are classified into 3 types

1. clusterIP: This is default type of service object used in Kubernetes and it is used when we want the pods in the cluster to communicate with each other and not with external network.

2. nodePort: This is used, if we want to access the pods from an external network and it also performs network load balancing. i.e. Even if a pod is running on a specific slave, we can access it from another slave in the cluster.

3. LoadBalancer: This is similar to nodePort. It is used for external connectivity of a pod and also network load balancing and it also assigns a public ip for all the nodes combined together.

++++++++++++++++++++++

vim pod-definition1.yml

We will be creating a service object for the labels used in pod-definition1.yml

kubectl create -f pod-definition1.yml

As we know pod will be created using the above command.

We want to create service object for the above pod

Ex: Create a service definition file for port mapping on nginx pod

vim pod-definition1.yml

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

:wq

kubectl create -f pod-definition1.yml

Observation: Along with the pod, service object gets created.
This service object is type clusterIP. Hence cannot be accessed from external network.

gcloud compute firewall-rules create rule3 --allow tcp:30008

vim service1.yml

```
---
apiVersion: v1
kind: Service
metadata:
 name: nginx-service
 labels:
  author: sunil
spec:
 type: NodePort
 ports:
  - targetPort: 80
    port: 80
    nodePort: 30008
 selector:
  author: sunil
  type: reverse-proxy
...
```

:wq

kubectl create -f service1.yml

Now, the nginx pod is accessible externally.

kubectl get nodes -o wide

As we have created nodePort, we should able to access from any node.

Take external_IP from anynode

34.66.234.81:30008 (We should be able to access nginx)

34.123.230.145:30008

If service object is not created, we use to identify in which node the pod is running, take that node IP, from that node IP, we used to access that application.

(Note: We need to open 30008 port in cluster)

++++++++++++++++++++++

In Kubernetes, pods can indeed be created using various resources like Deployment, DaemonSet, StatefulSet, and Job. Each of these resources serves a different purpose in managing and deploying applications.

Deployment: Deployments are a higher-level resource used for managing a set of identical pods, typically for stateless applications. They provide features like rolling updates, scaling, and automated rollbacks.

DaemonSet: DaemonSets ensure that a copy of a specific pod is running on all (or a subset of) nodes in the cluster. They are often used for system daemons or logging agents that need to run on every node.

StatefulSet: StatefulSets are used to manage stateful applications that require stable, unique network identifiers, persistent storage, and ordered deployment and scaling. Examples include databases like MySQL or stateful caching systems like Redis.

Job: Jobs create one or more pods and ensure that a specified number of them successfully terminate. They are used for batch processing, running a task to completion, and then terminating.

Each of these resources provides different features and behaviors tailored to different types of applications and workload requirements in Kubernetes.

What Is Kubernetes DaemonSet and How to Use It? (kodekloud.com)

What is Kubernetes StatefulSets? (komodor.com)

Jobs | Kubernetes

Parallel Processing using Expansions | Kubernetes

Kubernetes Project

This is a python based application which is used for accepting a vote (voting app).
This application accepts the vote and passes it to temporary db created using redis. From redis, the data is passed to worker application created using dotnet. Dotnet based application analyses the data and stores it in permanant database created using postgres.
From postgres database, results can be seen on an application created using node JS.

Have a look at the project Architecture.

Redis and postgres pod needs to assigned as cluster IP.
As cluster Ip is used for internal communication.

Voting App and Result App needs to be assigned as loadbalance type.

We need to create 5 definition files.

These 5 images related to this project is available in hub.docker.com
Using those images, we will create pods.
We need to create 5 pod definition files
We need to create 4 service files

We will be creating these definition files using pycharm.

vim voting-app-pod.yml

```
---
apiVersion: v1
kind: Pod
metadata:
  name: voting-app-pod
  labels:
    name: voting-app-pod
    app: demo-voting-app
spec:
```

```
  containers:
    - name: voting-app
      image: dockersamples/examplevotingapp_vote
      ports:
        - containerPort: 80
...


:wq


vim result-app-pod.yml

apiVersion: v1
kind: Pod
metadata:
  name: result-app-pod
  labels:
    name: result-app-pod
    app: demo-voting-app
spec:
 containers:
   - name: result-app
     image: dockersamples/examplevotingapp_result
     ports:
       - containerPort: 80
...


:wq



vim worker-app-pod.yml


---
apiVersion: v1
kind: Pod
metadata:
```

```
  name: worker-app-pod
  labels:
    name: worker-app-pod
    app: demo-voting-app
spec:
  containers:
   - name: worker-app
     image: dockersamples/examplevotingapp_worker
...


:wq


vim redis-pod.yml

---
apiVersion: v1
kind: Pod
metadata:
  name: redis-pod
  labels:
    name: redis-pod
    app: demo-voting-app
spec:
  containers:
   - name: redis
     image: redis
     ports:
      - containerPort: 6379
...


:wq



vim postgres-pod.yml

---
```

```
apiVersion: v1
kind: Pod
metadata:
  name: postgres-pod
  labels:
    name: postgres-pod
    app: demo-voting-app
spec:
  containers:
    - name: postgres
      image: postgres:9.4
      ports:
        - containerPort: 5432
...
```

We are done with 5 pod definiton files.
We need to create 4 service definiton files.

vim redis-service.yml

```
---
apiVersion: v1
kind: Service
metadata:
  name: redis-service
  labels:
    name: redis-service
    app: demo-voting-app
spec:
  ports:
    - port: 6379
      targetPort: 6379
  selector:
    name: redis-pod
    app: demo-voting-app
```

:wq

Note: As we have not specified the type of service object, by default it creates service object to type Cluster_IP

vim result-app-service.yml

```
apiVersion: v1
kind: Service
metadata:
  name: result-service
  labels:
    name: result-service
    app: demo-voting-app
spec:
  type: LoadBalancer
  ports:
    - port: 80
      targetPort: 80
  selector:
    name: result-app-pod
    app: demo-voting-app
```

:wq

The above two service objects are of type load balancer. we can access it from the external network.

Open gitbash from the location where all the definition files are saved.

```
$ git init
$ git add .
$ git commit -m "a"
```

Open github ---> create new repository

Repository name - kuber_project

upload the files from local repository to remote repository using the two commands

$ git remote add XXXXX
$ git push XXXX

We should able to see the definition files in github repository (Total 9 files)

We need to download the 9 files into kubernetes cluster.

Login to GCP console
Create kubernetes cluster
Connect to the cluster

Get the repository URL in github.

$ git clone rep_url

$ git clone https://github.com/sunildevops77/kube_project_durga.git

(Observation all the definition files will be downloaded)

$ cd kuber_project
$ ls (we get the files)

$ kubectl create -f voting-app-pod.yml

$ kubectl get pods (we should get one pod)

$ kubectl create -f redis-pod.yml

$ kubectl create -f worker-app-pod.yml

$ kubectl create -f postgres-pod.yml

$ kubectl create -f result-app-pod.yml

Now, we need to run service definition files

$ kubectl create -f voting-app-service.yml

$ kubectl create -f redis-service.yml

$ kubectl create -f postgres-service.yml

$ kubectl create -f result-app-service.yml

To get all the information

$ kubectl get all

We can see 5 pods and 4 services created.

Observation: worker pod also failed.

++++++++++++++++++++++++++++

These images are coming from community called dockersamples
Connection between workerpod and postgresPod is creating issues.

Go to service&ingress option in the kuberneted dashboard
We can see the four services, which are created.

Click on endpoint (IP) of voting Application
Click in URL, we get Voting App (CATS / DOGS) -- This is python based App.

Click on endpoint (IP) of result Application

Click in URL, we get Result App

++++++++++++++++++++++++++++++++++++