

## K8s Interview Questions and Answers

### 1. Can you please explain the architecture of Kubernetes.

**Ans:** According to the Architecture of kubernetes, there are some components that exist on master and worker node as well. Such as **Kube Controller Manager, Kube API Server, Kube-Scheduler and ETCD on master node**. And there are three components on worker node which is **Container Engine, such as Docker, Kubelet and Kube Proxy**.

So If we want to give a brief about each of the component. The first one **Kube Controller**. It make sure that if your pod has some problem it has been failed due to some reason or if your node goes down. So to control those kind of things kube controller will be used.

1. **KubeAPI Server** – that exposes the Kubernetes API. So this is the API server where your worker nodes will be communicating with. So whenever you run.  
**Kubectl get nodes**  
**Kubectl get pods**

All of those commands first reach the KubeAPI server. So this is like a gateway or the entrypoint for the entire kubernetes cluster.

2. **ETCD** – It is a key value pair which is used to store cluster related information.  
**Such as What nodes exist in the cluster, what pods should be running, which nodes they are running on.**

So whenever you create something with **kubectl create / kubectl run** will create an entry in the **ETCD**.

3. **Kube Scheduler** – That watches newly created pods that have no node assigned, and it selects nodes for them to run.

## ON Worker Node

1. **Kubelet** – It is like an agent that runs on each node in the cluster. It makes sure that containers are running in a pod.
2. **Kube Proxy** – It is basically to have communication like, for example if you have multiple machines and you will have some networking solutions installed right such as **Calico or Flannel**. So to have proper communications between pods **kube-proxy** is responsible.
3. **Container Engine** – Which is responsible for running containers.  
Such as **Docker**.

## What is Ingress?

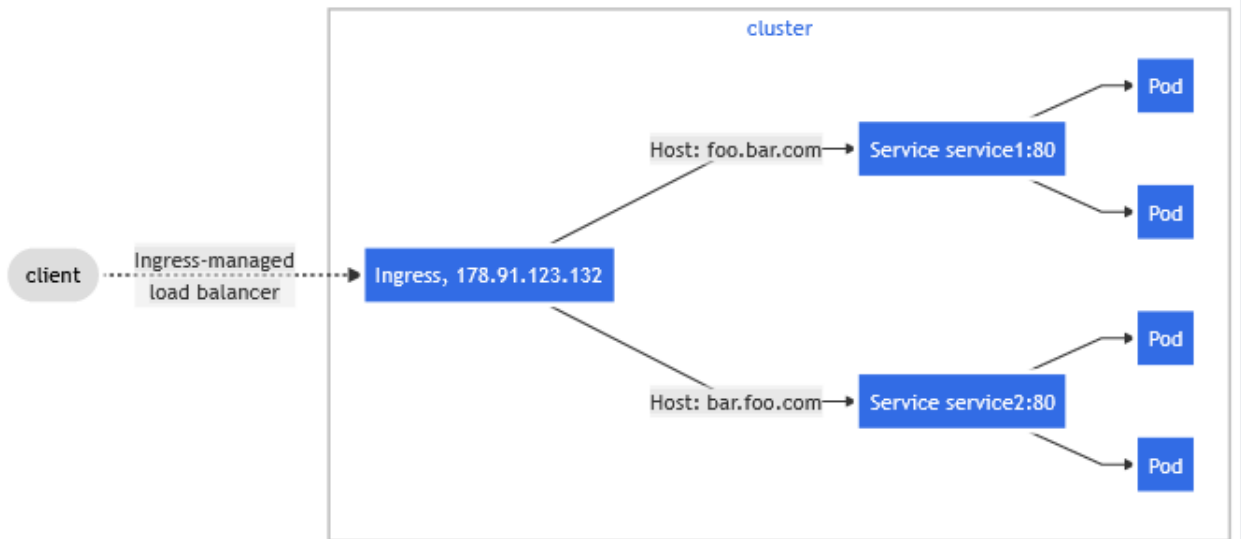
**Ans:**

Ingress is a resource of kubernetes, it allows us to access the multiple services from over the internet using single load balancer.

Let's say if you deployed 10 services and you are going to exposed 10 services as LoadBalancer so what will happen it will create 10 load balancer in cloud environment so it would be very costly, but if you have deployed ingress controller on kubernetes, so you can create ingress resource and access multiple services using single load balancer.

**Ingress** can provide features like **Load Balancing, SSL Termination, Namebased virtual hosting,**

1. **Name based virtual hosting** – it is used to access multiple services or websites at the same IP Address.



You must have an ingress controller to satisfy an Ingress. Only creating an Ingress resource has no effect.

The following Ingress tells the backing load balancer to route requests based on the [Host header](#).

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: name-virtual-host-ingress
spec:
  rules:
  - host: foo.bar.com
    http:
      paths:
      - pathType: Prefix
        path: "/"
        backend:
          service:
            name: service1
            port:
              number: 80
  - host: bar.foo.com
    http:
      paths:
      - pathType: Prefix
        path: "/"
        backend:
          service:
            name: service2
            port:
              number: 80
```

**For example:** If you specify the **website01.example.com**, so the request should be forwarded to **service1**.

And if you specify the **website02.example.com** then request should be forwarded to **service2**.

- 2. SSL Termination** – you can secure your ingress controller by implementing **SSL termination** from **HTTP to HTTPS**. If you want to implement it on ingress resource then you need to first create the **secret** with **TLS keys**. And that secret you will specify in **ingress resource**, where you will write the parameters like **tls**, **hosts**, **host header name** for example : <https://website01.example.com> And **secret name** that you have created using **TLS certificate**.

```

apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: tls-example-ingress
spec:
  tls:
  - hosts:
    - https-example.foo.com
    secretName: testsecret-tls
  rules:
  - host: https-example.foo.com
    http:
      paths:
      - path: /
        pathType: Prefix
        backend:
          service:
            name: service1
            port:
              number: 80

```

### 3. Explain kubeconfig file of Kubernetes?

**Ans: In the kubeconfig file there are three primary fields.**

**1. Cluster field** – In the Cluster Field has details related to the URL of your Kubernetes Cluster and it's associated information. In the first section of the cluster has information of your kubernetes master and name of the cluster. There can be two fields also. **Certificate-authority-data, OR insecure-skip-tls-verify:true** which means the **TLS certificate** for this cluster will not be verify.

**2. Context Field** – **Context basically groups all the information together**

**3. Users Field** – User field contains authentication specific information like **username, passwords.**

There can be different type of authentication mechanism such as username, password, certificates and tokens.

So in order for your kubectl to connect to your cluster it will need to authenticate to it. So your kubectl takes the information associated with the cluster. Takes the information related to the authentication. And then authenticates with one of the cluster.

## What is the difference between Readiness probe and Liveness probe.

**Readiness Probe** – is used to determine whether pod is ready to accept the traffic or not

**Liveness probe** checks the status of the container (whether it is running or not). If the liveness probe **fails**, then automatically container move on with its **restart** policy.

```
apiVersion: v1
kind: Pod
metadata:
  labels:
    test: liveness
    name: liveness-http
spec:
  containers:
  - name: liveness
    image: k8s.gcr.io/liveness
    args:
    - /server
    livenessProbe:
      httpGet:
        path: /healthz
        port: 8080
        httpHeaders:
        - name: Custom-Header
          value: Awesome
      initialDelaySeconds: 3
      periodSeconds: 3
```

```

apiVersion: v1
kind: Pod
metadata:
  name: goproxy
  labels:
    app: goproxy
spec:
  containers:
  - name: goproxy
    image: k8s.gcr.io/goproxy:0.1
    ports:
    - containerPort: 8080
    readinessProbe:
      tcpSocket:
        port: 8080
      initialDelaySeconds: 5
      periodSeconds: 10
    livenessProbe:
      tcpSocket:
        port: 8080
      initialDelaySeconds: 15
      periodSeconds: 20

```

## What is the difference between Deployments and StatefulSets.

**Statefulsets** is used for Stateful applications such as databases MongoDB, MySQL, each replica of the pod will be using its own Volume, If you are deleting the statefulsets it will not delete the volumes associated with the Statefulsets. Basically it is used when an application need a stable pod hostname (instead of podname-randomstring).

**For Example:** podname-0, podname-1, podname-2. (And when a pod gets rescheduled it will keep that identity).

**Deployment** is a resource to deploy a stateless application like frontend, backend, if using a PVC, all replicas will be using the same Volume.

## What is Daemon Sets?

DaemonSet is a controller similar to ReplicaSet that ensures that the pod runs on all the nodes of the cluster. If a node is added/removed from a cluster, DaemonSet automatically adds/deletes the pod.

## What is Autoscaling?

The Horizontal Pod Autoscaler automatically scales the number of Pods in replica set or based on observed CPU utilization.

# Autoscaling

---

- This is a pod that you can use to test autoscaling:

```
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: hpa-example
spec:
  replicas: 3
  template:
    metadata:
      labels:
        app: hpa-example
    spec:
      containers:
        - name: hpa-example
          image: gcr.io/google_containers/hpa-example
          ports:
            - name: http
              containerPort: 80
          resources:
            requests:
              cpu: 200m
```

Autoscaling





```
apiVersion: autoscaling/v1
kind: HorizontalPodAutoscaler
metadata:
  name: hpa-example-autoscaler
spec:
  scaleTargetRef:
    apiVersion: extensions/v1beta1
    kind: Deployment
    name: hpa-example
  minReplicas: 1
  maxReplicas: 10
  targetCPUUtilizationPercentage: 50
```

## Explain EndPoints?

Endpoint tracks the IP Address of the pod, so that service can send traffic to the pod. When we create the service the endpoints will also be created automatically. When a service selector matches a pod label, that IP Address is added to your endpoints.

```
apiVersion: v1
kind: Endpoints
metadata:
  name: my-service
subsets:
  - addresses:
      - ip: 192.0.2.42
    ports:
      - port: 9376
```

## What is the difference between ConfigMaps and Secrets.

ConfigMaps is used to store application configuration like environment variables, configuration files. We can mount configmaps as volume.

Below is the command to create configmap from literal.

```
C:\Users\Zeal Vora\Desktop\k8s\Storage>kubectl create configmap dev-config --from-literal=app.mem=2048m
configmap/dev-config created
C:\Users\Zeal Vora\Desktop\k8s\Storage>_
```

Create a file as per below and create configmap of this files.

File name: dev.properties

app.env=dev

app.mem=2048m

app.properties=dev.env.url

```
C:\Users\Zeal Vora\Desktop\k8s\Storage>kubectl create configmap dev-properties --from-file=dev.properties
configmap/dev-properties created
C:\Users\Zeal Vora\Desktop\k8s\Storage>_
```

## How to mount configmaps as volume.

```
pod-configmap.yaml
1  apiVersion: v1
2  kind: Pod
3  metadata:
4    name: configmap-pod
5  spec:
6    containers:
7    - name: test-container
8      image: nginx
9      volumeMounts:
10     - name: config-volume
11       mountPath: /etc/config
12   volumes:
13   - name: config-volume
14     configMap:
15       name: dev-properties
16   restartPolicy: Never
```

```

C:\Users\Zeal Vora\Desktop\k8s\Storage>kubectl apply -f pod-configmap.yaml
pod/configmap-pod created

C:\Users\Zeal Vora\Desktop\k8s\Storage>kubectl get pods
NAME                READY   STATUS    RESTARTS   AGE
configmap-pod       1/1     Running   0           17s

C:\Users\Zeal Vora\Desktop\k8s\Storage>kubectl exec -it configmap-pod bash
root@configmap-pod:/# cd /etc/config
root@configmap-pod:/etc/config# ls
dev.properties
root@configmap-pod:/etc/config# cat dev.properties
app.env=dev
app.mem=2048m
app.properties=dev.env.url
root@configmap-pod:/etc/config#

```

## What is Secrets?

Secret is used to store sensitive information in an encrypted format. like password, token, keys.

## What are the types of secrets we can create?

1. Generic
  - !) File (--from-file)
  - !!) Directory
  - !!!) Literal Value
2. Docker Registry
3. TLS

## Create generic secrets.

```

C:\Users\Zeal Vora\Desktop\k8s\Security>kubectl get secret
NAME                TYPE                               DATA   AGE
default-token-n1mkm  kubernetes.io/service-account-token 3        14d

C:\Users\Zeal Vora\Desktop\k8s\Security>kubectl create secret generic firstsecret --from-literal=dbpass=mypassword123
secret/firstsecret created

C:\Users\Zeal Vora\Desktop\k8s\Security>

```

```

C:\Users\Zeal Vora\Desktop\k8s\Security>kubectl get secret
NAME                                TYPE                                DATA  AGE
default-token-n1mkm                kubernetes.io/service-account-token  3      14d
firstsecret                        Opaque                              1      11s

C:\Users\Zeal Vora\Desktop\k8s\Security>kubectl describe secret firstsecret
Name:                                firstsecret
Namespace:                          default
Labels:                              <none>
Annotations:                         <none>

Type: Opaque

Data
====
dbpass: 13 bytes

C:\Users\Zeal Vora\Desktop\k8s\Security>

```

```

C:\Users\Zeal Vora\Desktop\k8s\Security>kubectl get secret firstsecret -o yaml
apiVersion: v1
data:
  dbpass: bXlwYXNkd29yZDEyMw==
kind: Secret
metadata:
  creationTimestamp: "2019-09-04T18:55:18Z"
  name: firstsecret
  namespace: default
  resourceVersion: "1835505"
  selfLink: /api/v1/namespaces/default/secrets/firstsecret
  uid: f107584d-9a68-4478-8667-733030bdc7d3
type: Opaque

C:\Users\Zeal Vora\Desktop\k8s\Security>

```

## Create secrets from file.

Create a file credentials.txt and write the db password into this file.

```
# echo "dbpassword" > credentials.txt
```

Now create secrets from this file.

```

C:\Users\Zeal Vora\Desktop\k8s\Security>kubectl create secret generic secondsecret --from-file=./credentials.txt
secret/secondsecret created

C:\Users\Zeal Vora\Desktop\k8s\Security>

```

Create a secret using yaml file.

```
secret-data.yaml
apiVersion: v1
kind: Secret
metadata:
  name: thirdsecret
type: Opaque
data:
  username: ZGJhZG1pbG==
  password: bXlwYXNzd29yZDEyMw==
```

```
C:\Users\Zeal Vora\Desktop\k8s\Security>kubectl apply -f secret-data.yaml
secret/thirdsecret created
```

```
C:\Users\Zeal Vora\Desktop\k8s\Security>kubectl get secret
```

NAME	TYPE	DATA	AGE
default-token-nlkm	kubernetes.io/service-account-token	3	14d
firstsecret	Opaque	1	7m42s
secondsecret	Opaque	1	3m
thirdsecret	Opaque	2	9s

```
C:\Users\Zeal Vora\Desktop\k8s\Security>
```

Create secrets using string data, if you create secrets using stringdata the value of username and password will be encoded automatically you don't need to encode it and put in yaml.

```

secret-data.yaml      secret-stringdata.yaml
apiVersion: v1
kind: Secret
metadata:
  name: stringdata
type: Opaque
stringData:
  config.yaml: |-
    username: dbadmin
    password: mypassword

```

```

C:\Users\Zeal Vora\Desktop\k8s\Security>kubectl apply -f secret-stringdata.yaml
secret/stringdata created

```

```

C:\Users\Zeal Vora\Desktop\k8s\Security>kubectl get secret

```

NAME	TYPE	DATA	AGE
default-token-n1mkm	kubernetes.io/service-account-token	3	14d
firstsecret	Opaque	1	9m28s
secondsecret	Opaque	1	4m46s
stringdata	Opaque	1	10s
thirdsecret	Opaque	2	115s

```

C:\Users\Zeal Vora\Desktop\k8s\Security>

```

## What is Taint and Tolerations?

**Taints** are used to repel the pods from a specific node. If you applied taints on a node you cannot schedule pods on that node unless it has a matching toleration.

`kubectl taint nodes node1 mysize=large:NoSchedule` → to apply the taints.

`kubectl taint nodes node1 mysize=large:NoSchedule-` → to remove the taints.



```

apiVersion: v1
kind: Pod
metadata:
  name: pod5
spec:
  containers:
  - image: coolgourav147/nginx-custom
    name: firstcontainer
    imagePullPolicy: Never
  tolerations:
  - effect: "NoSchedule"
    key: "mysize"
    operator: "Equal"
    value: "large"

# kubectl taint node worker01 mysize=large:NoSchedule

```

```

gaurav@manager:~$ kubectl apply -f pod.yml
pod/pod5 created
gaurav@manager:~$ kubectl get pods -o wide

```

NAME	READY	STATUS	RESTARTS	AGE	IP	NODE	NOMINATED	NODE	READINESS	GATES
pod1	1/1	Running	0	7m48s	10.244.2.14	worker02	<none>		<none>	
pod2	1/1	Running	0	6m52s	10.244.2.15	worker02	<none>		<none>	
pod3	1/1	Running	0	6m42s	10.244.2.16	worker02	<none>		<none>	
pod4	1/1	Running	0	6m33s	10.244.2.17	worker02	<none>		<none>	
pod5	1/1	Running	0	8s	10.244.1.13	worker01	<none>		<none>	

```

gaurav@manager:~$

```

You can see here pod5 has been deployed on worker01, because it has matching tolerations.

## What is NoExecute in Taint and Tolerations?

When you apply the taint **NoExecute**, on node, pods those are already running on that node will be terminated. And you can not schedule pods on that node.

`kubectl taint nodes worker02 mysize=small:NoExecute` → to apply the taints.

```

gaurav@manager:~$ kubectl taint node worker02 mysize=small:NoExecute
node/worker02 tainted
gaurav@manager:~$ kubectl get pods -o wide

```

NAME	READY	STATUS	RESTARTS	AGE	IP	NODE	NOMINATED	NODE	READINESS	GATES
pod1	0/1	Terminating	0	27m	<none>	worker02	<none>		<none>	
pod2	0/1	Terminating	0	26m	10.244.2.15	worker02	<none>		<none>	
pod4	0/1	Terminating	0	26m	10.244.2.17	worker02	<none>		<none>	
pod5	1/1	Running	0	20m	10.244.1.13	worker01	<none>		<none>	

When you apply the below configurations, pod will tolerate the node till 60 seconds only, after 60 seconds it will be in terminated.

```
name: firstcontainer
imagePullPolicy: Never
tolerations:
- effect: "NoExecute"
  key: "mysize"
  operator: "Equal"
  value: "large"
  tolerationSeconds: 60

# kubectl taint node worker01 mysize=large:NoSchedule
gaurav@manager:~$
```

---

```
Every 2.0s: kubectl get pods -o wide
```

NAME	READY	STATUS	RESTARTS	AGE	IP	NODE
pod9	0/1	Terminating	0	3m57s	10.244.1.17	worker01

## Which are the deployment strategies available in Kubernetes?

- 1) Recreate
- 2) Rolling Update
- 3) Blue/Green
- 4) Canary

**Recreate:** In the Recreate Strategy, if you are deploying version2 application, it will delete version1 pods first, then it will create pods of version2.

**RollingUpdate:** Rolling update is used when you want to deploy your new version without having downtime to your application.

**For example:** When we deploy new instance of the applications, a new pod will be created and the existing pod will be destroyed, after the new pod up and running and this happens for every **ReplicaSet**.

**maxUnavailable:** 0 (How many pods you want to unavailable of the version1 while deploying version2.



## What is Role?

Role is used to grant access to resources within a single namespace.

## What is Role Binding?

And role binding is used to grant the permissions defined in a role to a user or set of users.

## Practicle:

```
bash-4.2# kubectl --context=zeal-context get pods
NAME                                READY   STATUS    RESTARTS   AGE
nginx-7bb7cd8db5-5lnf7             1/1     Running   0           20h
nginx2                              1/1     Running   0           3h25m
bash-4.2# kubectl --context=zeal-context get pods --namespace teama
Error from server (Forbidden): pods is forbidden: User "zeal" cannot list resource "pods" in API group "" in the namespace "teama"
bash-4.2#
```

We can see in above screenshot that user zeal not able to get the pods in teama namespace.

So let's create the role that will grant access to user zeal to create pods in teama namespace.

```
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  namespace: teama
  name: pod-reader
rules:
- apiGroups: [""]
  resources: ["pods"]
  verbs: ["get", "watch", "list", "create"]
```

```
C:\Users\Zeal Vora\Desktop\k8s\Security>kubectl apply -f teama-role.yaml
role.rbac.authorization.k8s.io/pod-reader created

C:\Users\Zeal Vora\Desktop\k8s\Security>kubectl get role --namespace teama
NAME          AGE
pod-reader    12s

C:\Users\Zeal Vora\Desktop\k8s\Security>
```

```
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: read-pods
  namespace: teama
subjects:
- kind: User
  name: zeal
  apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: Role
  name: pod-reader
  apiGroup: rbac.authorization.k8s.io
```

```
C:\Users\Zeal Vora\Desktop\k8s\Security>kubectl apply -f teama-rolebinding.yaml
rolebinding.rbac.authorization.k8s.io/read-pods created
```

```
C:\Users\Zeal Vora\Desktop\k8s\Security>
```

Now try to access the pods using zeal user.

```
bash-4.2# kubectl --context=zeal-context get pods --namespace teama
NAME                                READY   STATUS    RESTARTS   AGE
nginx-7bb7cd8db5-fgkjc             1/1     Running   0           148m
bash-4.2#
```

## Cluster Role and Cluster Role Binding

### What is Cluster Role?

Cluster Role and ClusterRoleBinding is used to grant permissions at the cluster level resources like pods, secrets, service, and in all the namespaces.

### Practicle:

Try to access pods at cluster level using zeal user you will get forbidden.

```
bash-4.2# kubectl --context=zeal-context get pods
Error from server (Forbidden): pods is forbidden: User "zeal" cannot list resource "pods" in API group "" in the namespace "default"
bash-4.2#
```

Now create clusterrole and clusterrolebinding to access pods in all the namespaces of cluster.

### # vim clusterrole.yaml

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  # "namespace" omitted since ClusterRoles are not namespaced
  name: cluster-pod-reader
rules:
- apiGroups: [""]
  resources: ["pods"]
  verbs: ["get", "watch", "list"]
```

### # kubectl apply -f clusterrole.yaml

**# vim clusterrolebinding.yaml**

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: read-secrets-global
subjects:
- kind: User
  name: zeal
  apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: ClusterRole
  name: cluster-pod-reader
  apiGroup: rbac.authorization.k8s.io
```

**# kubectl apply -f clusterrolebinding.yaml**

Now try to access pods in all the namespace.

```
bash-4.2# kubectl --context=zeal-context get pods
NAME                                READY   STATUS    RESTARTS   AGE
nginx-7bb7cd8db5-5lnf7             1/1     Running   0           20h
nginx2                              1/1     Running   0           3h48m
bash-4.2# kubectl --context=zeal-context get pods --namespace teama
NAME                                READY   STATUS    RESTARTS   AGE
nginx-7bb7cd8db5-fgkjc             1/1     Running   0           167m
```

You can see above zeal user can access the pods in all the namespaces.

## What is Resource Quota?

There are two types of Resource Quota.

1. Compute Resources
2. Object Count

**Compute Resources** quota is used to set limit of CPU and memory on a namespace level.

**Object Count** is used to set limit of objects to be deploy within a namespace.

### Practicle:

```
kubectl create namespace myspace
```

```
cat <<EOF > compute-resources.yaml
apiVersion: v1
kind: ResourceQuota
metadata:
  name: compute-resources
spec:
  hard:
    requests.cpu: "1"
    requests.memory: 1Gi
    limits.cpu: "2"
    limits.memory: 2Gi
    requests.nvidia.com/gpu: 4
EOF
```

```
kubectl create -f ./compute-resources.yaml --namespace=myspace
```

```
cat <<EOF > object-counts.yaml
apiVersion: v1
kind: ResourceQuota
metadata:
  name: object-counts
spec:
  hard:
    configmaps: "10"
    persistentvolumeclaims: "4"
    pods: "4"
    replicationcontrollers: "20"
    secrets: "10"
    services: "10"
    services.loadbalancers: "2"
EOF
```

```
kubectl create -f ./object-counts.yaml --namespace=myspace
```

```
kubectl get quota --namespace=myspace
```

NAME	AGE
compute-resources	30s
object-counts	32s

```
kubectl describe quota compute-resources --namespace=myspace
```

Name:	compute-resources	
Namespace:	myspace	
Resource	Used	Hard
-----	----	----
limits.cpu	0	2
limits.memory	0	2Gi
requests.cpu	0	1
requests.memory	0	1Gi
requests.nvidia.com/gpu	0	4

```
kubectl describe quota object-counts --namespace=myspace
```

Name:	object-counts	
Namespace:	myspace	
Resource	Used	Hard
-----	----	----
configmaps	0	10
persistentvolumeclaims	0	4
Pods	0	4
replicationcontrollers	0	20
secrets	1	10
services	0	10
services.loadbalancers	0	2

## What is Resource Limit?

Resource Limit is used to set cpu and memory limit to pods. When we create a pod by default it uses the whole cpu and memory, so to apply limit on pod we can use resource limit.

### Practicle:

```
apiVersion: v1
kind: Pod
metadata:
  name: quota-mem-cpu-demo
spec:
  containers:
  - name: quota-mem-cpu-demo-ctr
    image: nginx
    resources:
      limits:
        memory: "800Mi"
        cpu: "800m"
      requests:
        memory: "600Mi"
        cpu: "400m"
```



## What is the difference between PV and PVC.

PersistentVolume (PV) is a piece of storage and it is also a cluster resource that has been provisioned by an administrator or dynamically provisioned using storage class.

PersistentVolumeClaim (PVC) is a request for storage by a user for the application to use 10GB.

In real life scenario, PV is whole cake and PVC is piece of cake (But you can have a whole cake if there are no other people to eat (just like if there are no other application to use you can use whole PV )).

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: block-pv
spec:
  capacity:
    storage: 10Gi
  accessModes:
    - ReadWriteOnce
  volumeMode: Block
  persistentVolumeReclaimPolicy: Retain
  fc:
    targetWWNs: ["50060e801049cfd1"]
    lun: 0
    readOnly: false
```

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: block-pvc
spec:
  accessModes:
    - ReadWriteOnce
  volumeMode: Block
  resources:
    requests:
      storage: 10Gi
```

## **What are the different types of Volumes?**

EBS

OpenEBS

NFS

GlusterFS

Cinder

## **What is Storage Class?**

Storageclass is a Kubernetes object that stores information about creating a persistent volume for your pod. With a storageclass, we don't need to create a persistent volume separately before claiming it.

**Below are the storage classes available.**

[AWS EBS](#)

[Azure Disk](#)

[OpenStack Cinder](#)

[GCE PD](#)

[NFS](#)

OpenEBS

Local

## What is Security Context?

When you run a container, it runs with the UID 0, means root user, In-case of container breakouts, attacker can get root privileges to your entire system. So to avoid this we can use security context which allows us to run container as non-root user.

```
pod-securitycontext.yaml

apiVersion: v1
kind: Pod
metadata:
  name: pod-context
spec:
  securityContext:
    runAsUser: 1000
    runAsGroup: 3000
  containers:
  - name: sec-ctx-demo
    image: busybox
    command: [ "sh", "-c", "sleep 1h" ]
```

```
C:\Users\Zeal Vora\Desktop\k8s\Security>kubectl apply -f pod-securitycontext.yaml
pod/pod-context created
```

```
C:\Users\Zeal Vora\Desktop\k8s\Security>kubectl get pods
NAME                                READY   STATUS    RESTARTS   AGE
busybox-5d847dfc85-87lcg           1/1     Running   1           2m45s
pod-context                         1/1     Running   0           9s
```

```
C:\Users\Zeal Vora\Desktop\k8s\Security>kubectl exec -it pod-context sh
```

```
/ $ ps
PID   USER     TIME  COMMAND
   1   1000     0:00  sleep 1h
   6   1000     0:00  sh
  11   1000     0:00  ps
/ $ cd /tmp
/tmp $ touch test.txt
/tmp $ ls -l
total 0
-rw-r--r--    1 1000    3000          0 Oct 19 08:32 test.txt
/tmp $
```

# KUBERNETES TROUBLESHOOTING

## 1. OOMKilled

If you run *kubectl get pods* and see that some pods are being restarted, the next thing to do is to check the reason. You can do this by using the following command:

*kubectl describe pod myPodName -n myNamespace*

You may see a message that looks something like this:

```
State:          Running
  Started:      Sun, 16 Feb 2020 10:20:09 +0000
Last State:     Terminated
  Reason:       OOMKilled
  Exit Code:    137
  Started:      Sun, 16 Feb 2019 09:27:39 +0000
  Finished:     Sun, 16 Feb 2019 10:20:08 +0000
Restart Count: 7
```

When a pod reaches its memory limit it restarts. You can see the restart count when you run the *describe* command. The obvious solution is to increase the memory setting. This can be done by running *kubectl edit deployment myDeployment -n mynamespace* and **editing the memory limit**.

## 2. Sudden Jumps in Load / Scale

If the application traffic is suddenly increases. You can scale your replicas by executing a command.

```
# kubectl scale deployment myDeployment --replicas=5
```

If you use an autoscaler (e.g. Horizontal Pod Autoscaler) then this process can be automated.

## 3. Rollbacks

If you deployed a version 2 on cluster, and there is issue with version2 and you want to rollback to previous version that time you can rollback your deployment.

You can check the history of deployment by executing.

```
kubectl rollout history deployment myDeployment
```

The output will look similar to this:

```
kubectl rollout history deployment myDeployment
deployment.extensions/myDeployment
REVISION  CHANGE-CAUSE
1          <none>
2          <none>
3          <none>
4          <none>
```

The most recent deployment has the highest number. You can do rollback using a command like this:

```
kubectl rollout undo deployment myDeployment --to-revision=3
```

## 4. Logs

You can check the logs of pod.

## CrashloopBackoff

**When the problem is CrashLoopBackOff (your pod is starting, crashing, starting again, and then crashing again)**

There can be many reasons for this error.

1. Your **Dockerfile** doesn't have a **Command** (CMD), so your pod will immediately exit after starting.
2. You have used the same port for two containers inside the same pod. All containers inside the same pod have the same ip address; they are not permitted to use the same ports. You need a separate port for every container within your pod.
3. Kubernetes **can't pull the image** you have specified, and therefore keeps crashing.
4. Run *kubectl logs podName* to get more information about what caused the error.

## Issues

**I have deployed Jenkins on cluster, container was trying to create a file in /var/Jenkins\_home/ copy\_reference\_file.log. And it is getting permission denied.**

```
[mohammed@db-test-k8master ~]$ kubectl logs -f jenkins-5cdc87888f-988ws -n corestack-test
```

**touch: cannot touch '/var/jenkins\_home/copy\_reference\_file.log': Permission denied**

**Can not write to /var/jenkins\_home/copy\_reference\_file.log. Wrong volume permissions?**

```
[mohammed@db-test-k8master ~]$
```

**I am getting this issue because of Jenkins is running as Jenkins user when I change the user from Jenkins to root in deployment file issues has been fixed. I added the below parameters in deployment file.**

**securityContext:**

**runAsUser: 0**

**runAsGroup: 0**

**fsGroup: 0**

## **Worker Node Failure Reason.**

**OutOfDisk** -> You can check the Disk size if it full please increase.

**MemoryPressure** -> You can check the memory if it is full.

**DiskPressure** -> You can check the disk space

**PIDPressure**

You can check the kubelet service by executing.

`Journalctl -u kubelet`

Also you can check the certificates of worker node if it is expired by executing below command.

`Openssl x509 -in /var/lib/kubelet/worker-1.crt -text`