

Why do we need Kubernetes When you have Docker already?

Docker is an image building, storing and container creation technology
But kubernetes is container orchestration technology.

At one point of time it is very difficult to manage docker containers spanning over multiple hosts. So we need Kubernetes for the below thi

- Auto Scaling

Networking

- High Availability

- Reliability

Self Healing

- Storage mechanism

What is the namespace in Kubernetes?

Namespace is a logical isolation in Kubernetes. Usually we will create one namespace for one project. So resources will be isolated to the namespace level.

What is the difference between Pod and Container?

Pod is the smallest deployable unit in Kubernetes. It can have multiple Containers. Containers in a pod share the same IP address, Volumes. This feature is useful when you want to keep multiple apps together with the same IP and volume, sidecar containers, proxies, etc.

How do you restrict Pods and Containers to use limited host resources?

We can limit the resources to the pod by mentioning limits and requests in YAML that can restrict containers not to use more CPU, memory.

```
apiVersion: v1
kind: Pod
metadata:
  name: resource
spec:
  containers:
  - name: nginx
    image: nginx
    resources:
      requests: # requests is a soft Rule
        memory: "64Mi"
        cpu: "250m" #1cpu=1000m
      limits: # limits is hard rule
        memory: "128Mi"
        cpu: "500m"
```

How can you restrict resources to Pod if the engineer forgets to mention them in Pod definition?

We have a resource called LimitRange in Kubernetes. As administrators we can restrict the resources at namespace level. So if the engineer forgets to mention limit range will be applied automatically.

```

apiVersion: v1
kind: LimitRange
metadata:
  name: cpu-resource-constraint
spec:
  limits:
  - default: # this section defines default limits
    cpu: 500m
    defaultRequest: # this section defines default requests
    cpu: 500m
    max: # max and min define the limit range
    cpu: "1"
    min:
    cpu: 100m
  type: Container

```

How to configure a health check in Pod?

We can configure using liveness probe and readiness probe.

Readiness is to check at the time of container creation whether pod is ready to take traffic or not

Liveness probe will kick in when readiness is done. It will be checked at a particular interval to check if the application is working properly or not.

```

apiVersion: v1
kind: Pod
metadata:
  name: goproxy
  labels:
    app: goproxy
spec:
  containers:
  - name: goproxy
    image: registry.k8s.io/goproxy:0.1
    ports:
    - containerPort: 8080
    readinessProbe: #check pod is ready to accept traffic or not
      tcpSocket:
        port: 8080
      initialDelaySeconds: 5 #It will be silent for 5 sec
      periodSeconds: 10 # Then it will check every 10sec until ready
    livenessProbe: #check application is running properly or not
      tcpSocket:
        port: 8080
      initialDelaySeconds: 15 #silent for first 15sec
      periodSeconds: 20 #Until pod is alive it checks every 20 sec

```

What is the image pull policy in Pod?

3 types of image pull policy are available.

IfNotPresent - If it is not present then it will pull, if available already it will not pull.

Always - whether image is available or not it pulls the image

Never - If present in host it can run the pod, otherwise error.

Keeping Always is the best way.

How can you provide env variables to the Pod in a better way?

We can create a configmap resource. We can refer to the configmap in the pod so that the entire config will be loaded at the time of container creation.

In future if we need to add more configurations, do the changes in configmap and restart the pod so that there is no need to make changes in pod definition and deployment.

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: course-config
data:
  course: docker
  duration: 30hr
```

```
apiVersion: v1
kind: Pod
metadata:
  name: config-pod
spec:
  containers:
  - name: nginx
    image: nginx
    envFrom:
    - configMapRef:
        name: course-config
```

What are the secrets in Kubernetes?

Secrets are confidential information like DB username, password, etc. we can store this info in Secret and refer it to a Pod definition like configmap. Secrets are base64 encoded in kubernetes. By default it is not secure, so we use third party solutions like AWS secrets manager, Hashicorp Vault,

```
apiVersion: v1
kind: Secret
metadata:
  name: mysecret
type: Opaque
data:
  username: c2l2YWt1bWFy
  password: c2l2YWt1bWFyMTIz
```

```
apiVersion: v1
kind: Pod
metadata:
  name: secret-pod
spec:
  containers:
  - name: nginx
    image: nginx
    envFrom:
    - secretRef:
        name: mysecret
```

etc.

What is Service in Kubernetes? How many types are there?

Pods are naturally ephemeral in Kubernetes. So Pod IP will be changed every time it is created. To achieve pod to pod communication we can't rely on IP addresses. Solution is service.

Service can have name and Port when a pod is created it can go attached to service itself. It acts as a service mesh.

Multiple pods can be attached to service based on label selectors. Service acts as Load balancer between pods.

3 types of services available.

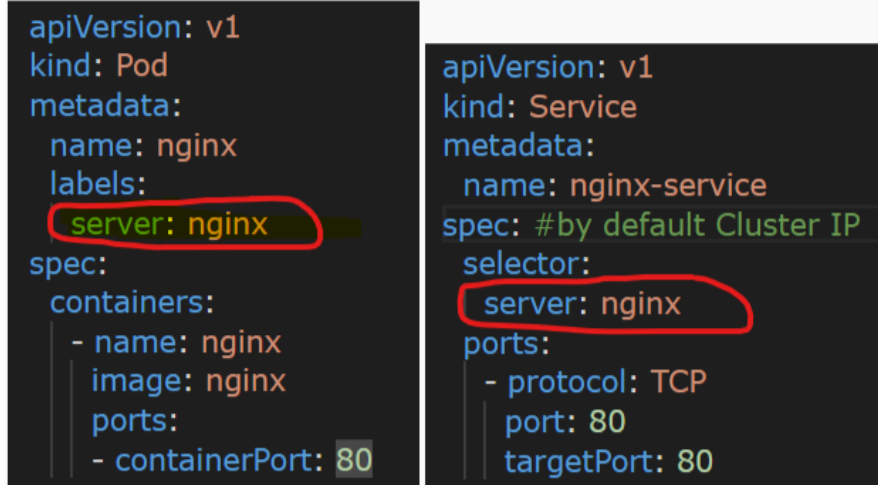
Cluster IP: Default type. It can get one IP address. We can configure cluster IP service to achieve pod to pod communication, But it can't be accessed over the internet.

Node Port: Node Port by default creates cluster IP in background. When we say NodePort a port will be opened on each and every node. This port will be redirected to the cluster IP. Node Port can be accessed over the internet.

Load Balancer: We can create Load Balancer through cloud providers like AWS, GCP, Azure, etc.

Load Balancer by default creates NodePort and Cluster IP in the background.

Load Balancer -> NodePort -> Cluster IP



```
apiVersion: v1
kind: Pod
metadata:
  name: nginx
  labels:
    server: nginx
spec:
  containers:
  - name: nginx
    image: nginx
    ports:
    - containerPort: 80
```

```
apiVersion: v1
kind: Service
metadata:
  name: nginx-service
spec: #by default Cluster IP
selector:
  server: nginx
ports:
  - protocol: TCP
    port: 80
    targetPort: 80
```

What is Replica Set in Kubernetes?

By default we can increase the number of Pods, If you want multiple instances of Pods to serve high traffic we can use Replica Set. It guarantees the declared no of pods will run always. It is the most important feature of high availability and autoscale.

When using ReplicaSet we no need to have separate Pod, it is included under template section

```

apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: nginx
  labels: # these are labels related to replica set resource
    app: nginx
    tier: frontend
spec:
  # modify replicas according to your case
  replicas: 3
  selector:
    matchLabels: #this is the syntax replica set uses to find the pods
      tier: frontend
  template: # pod template, labels are related to pod
    metadata:
      labels:
        tier: frontend
    spec:
      containers:
        - name: nginx
          image: nginx:alpine

```

What is Deployment in Kubernetes?

Deployment is the most important resource in Kubernetes to maintain the applications. It ensures zero downtime of application. Deployment creates ReplicaSet in background to maintain the desired number of replicas

We can use deployment for stateless applications. Finally Deployment is the highest resource in Kubernetes to maintain

Availability - Make sure desired no of replicas are always available

Scalability - Can be used to scale at runtime based on traffic.

Maintainability - We can do the updates with new image versions.

```

apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  annotations:
    deployment.kubernetes.io/change-cause: "Upgraded to version 2.0"
  labels:
    app: nginx
spec:
  replicas: 10
  selector:
    matchLabels:
      app: nginx
  strategy:
    type: RollingUpdate
    rollingUpdate:
      maxUnavailable: 2 # we asked for 10, we are okay if 9 are running at the time of upgrade
      maxSurge: 2 # we asked for 10, we are okay at the time of upgrade 11 are running
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
        - name: nginx
          image: nginx:1.22.1
          ports:
            - containerPort: 80

```

What is a Daemon Set in Kubernetes?

The Daemon set is similar to Deployment. But there is only one difference.

Deployment makes pods available in any node.

But DaemonSet makes sure one replica of Pod runs in each and every node in the Kubernetes cluster.

This feature is helpful for Cluster level administration like collecting the metrics, logs of all nodes, etc.

```
apiVersion: apps/v1
kind: DaemonSet
metadata:
  name: fluentd-elasticsearch
  namespace: kube-system
  labels:
    k8s-app: fluentd-logging
spec:
  selector:
    matchLabels:
      name: fluentd-elasticsearch
  template:
    metadata:
      labels:
        name: fluentd-elasticsearch
    spec:
      containers:
        - name: fluentd-elasticsearch
          image: quay.io/fluentd_elasticsearch/fluentd:v2.5.2
          resources:
            limits:
              memory: 200Mi
            requests:
              cpu: 100m
              memory: 200Mi
          volumeMounts:
            - name: varlog
              mountPath: /var/log
      volumes:
        - name: varlog
          hostPath:
            path: /var/log
```