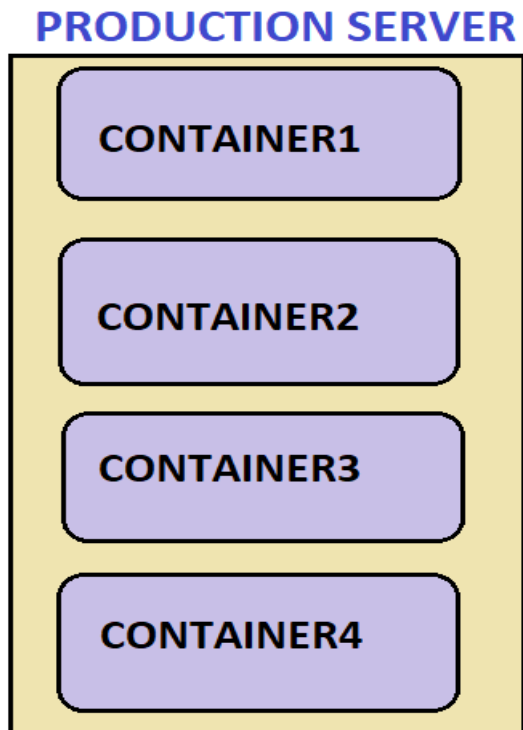# KUBERNETES

PROBLEMS WITH CONTAINERS

1.Containers could not communicate with other.

2.Containers had to be deployed appropriately.

3.Containers had to be managed properly.

4.Auto scaling was not possible.

5.Distributing traffic is still challenging.

Container Orchestration

It automates the deployment, management, scaling and networking of container.

Container orchestration tools

1.Kubernetes

2.Docker swarm

3.Mesos

What is Kubernetes?

Kubernetes is an open-source container management tool which automates container deployment, container scaling and de-scaling and container load balancing.

Kubernetes will package your website files as containers or pods which you can run on any instances in your kubernetes cluster.

It works with all cloud vendors like public, hybrid and on-premises.

Kubernetes is written in Golang, it has huge community support because it was first developed by Google and later donated to CNCF [cloud native computing foundation].

Features of Kubernetes

1.Automated scheduling: kubernetes provides an advanced scheduler to launch containers on cluster nodes.

2.Self-healing capabilities: it provides rescheduling, restarting and replacing the containers that are dead.

3.Load Balancing: kubernetes can scale up and scale down the application as per the requirement.

4.Community Support: Kubernetes has a large and active community with frequent updates, bug fixes, and new features being added.

KUBERNETES ARCHITECTURE
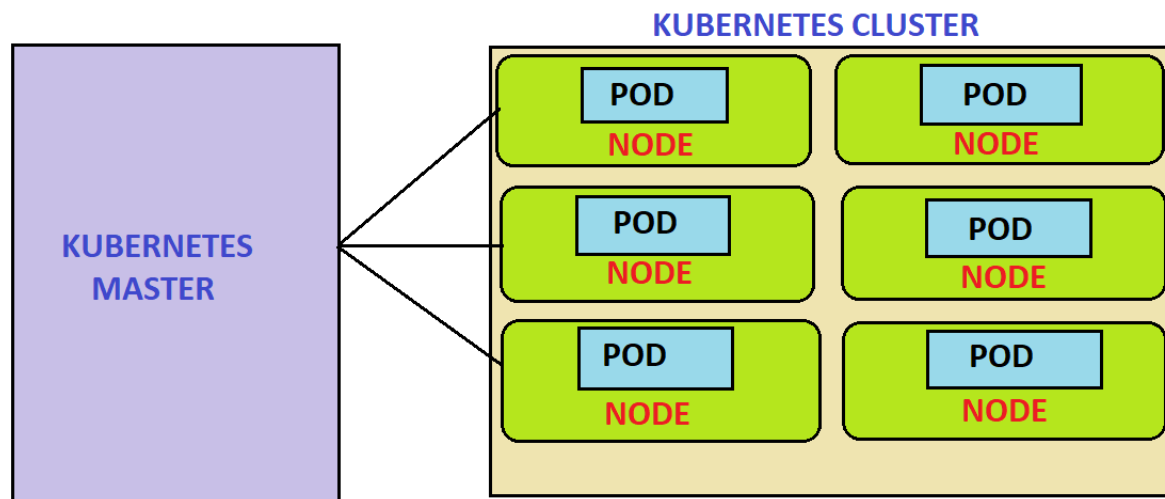
Kubernetes master: It will manage or maintain the kubernetes cluster and its nodes.

Kubernetes cluster: It is a collection of nodes.

Node: A node is a virtual or physical machine on which kubernetes is installed.

A node is a worker machine and that is where containers are launched by kubernetes.

Pod: A pod is a collection of one or more containers and it is smallest unit of kubernetes.

Components of Kubernetes

1.api server

2.etcd

3.Kubelet

4.Controller

5.scheduler

1.api server: kubernetes API server receives the command which are sent by users. After receiving, it validates the requests, process and then executes it. The resulting state of cluster will be saved in etcd as key value.

2.etcd: it is distributed key value storage which is used to store cluster data.

3.kubelet: it is an agent that runs on each node in cluster.

Agent is responsible for making sure that containers running on the node as expected.

4.Controllers: it will make decision to bring up new containers, in such case, the container runtime is the underlying software that is used to run containers.

5.Scheduler: scheduler in a master node schedules the task to worker nodes. It is a process that is responsible for assigning the pods to the available nodes.

CREATION OF POD IN KUBERNETES CLUSTER

KUBERNETES CLUSTER
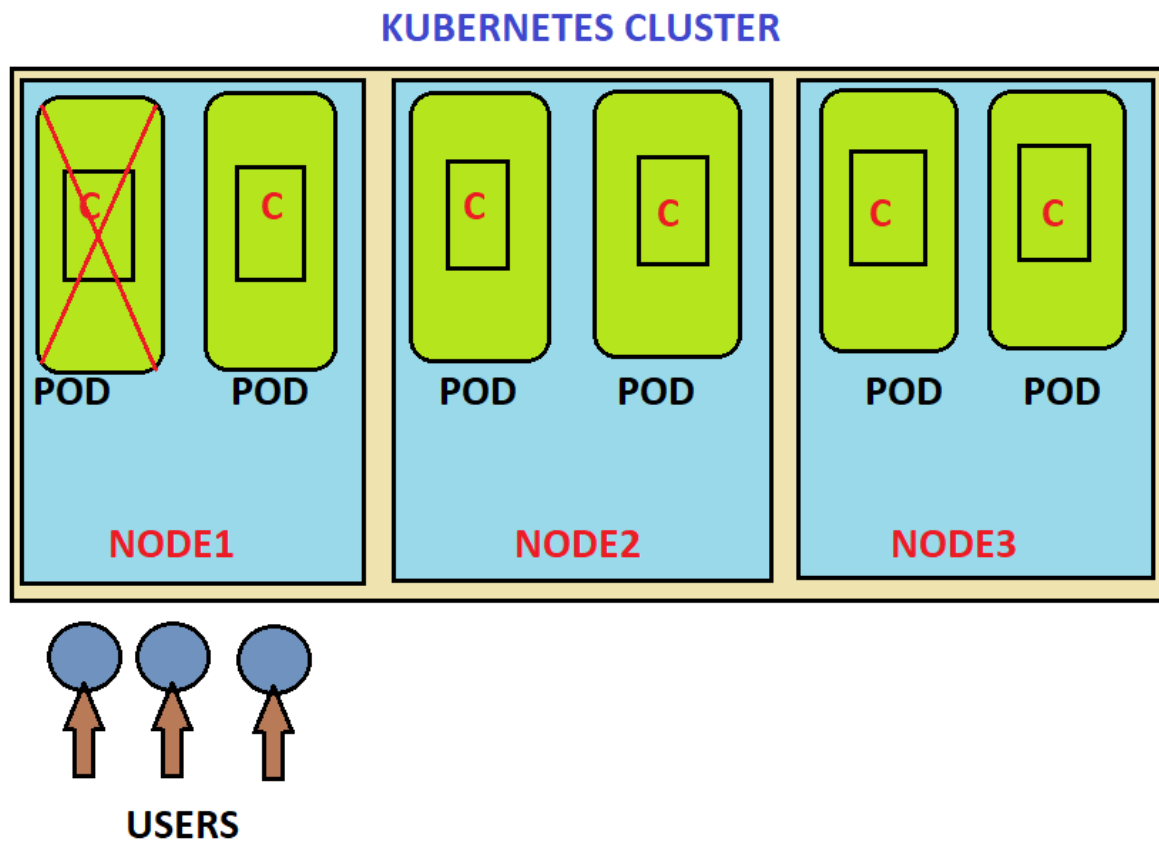
Pod is a collection of containers and its storage inside a node of a kubernetes cluster.

Any given pod can be composed of multiple, tightly coupled containers or just a single container (a more common use case).

Commands used for creating pods

1.To deploy the pod inside a node

  kubectl run pod_name - -image=image_name

ex: kubectl run my-pod - -image=Jenkins/jenkins

2. To list the created pods

 kubectl get pods

3.To describe the pod details

 kubectl describe pod pod_name

 4.To get wide information of pod

  kubectl get pods -o wide

YAML IN KUBERNETES

YAML [yet another markup language]

It acts as an input for creation of objects such as pods, replicas and deployments.

A kubernetes definition file always contains 4 top level fields. The apiVersion, kind, metadata and spec. These are top level or root level properties. Think of them as siblings, children of the same parent. These are all REQUIRED fields, so you MUST have them in your configuration file.

Components of YAML

1.apiVersion

2.kind

3.metadata

4.spec

1.apiVersion: This is the version of the kubernetes API we're using to create the object. Depending on what we are trying to create we must use the RIGHT apiVersion. For now, since we are working on PODs, we will set the apiVersion as v1.

2.kind: The kind refers to the type of object we are trying to create, which in this case happens to be a POD. So, we will set it as Pod. Some other possible values here could be ReplicaSet or Deployment.

3.metadata: The metadata is data about the object like its name, labels etc.

4.spec: Spec is a dictionary so add a property under it called containers.

Creation of pod using yaml

```
apiVersion: v1
kind: Pod
metadata:
    name: myapp-pod
    labels:
        app: myapp
        type: front-end
spec:
  containers:
      - name: nginx-container
        image: nginx
      - name: jenkins-container
        image: jenkins/jenkins
```

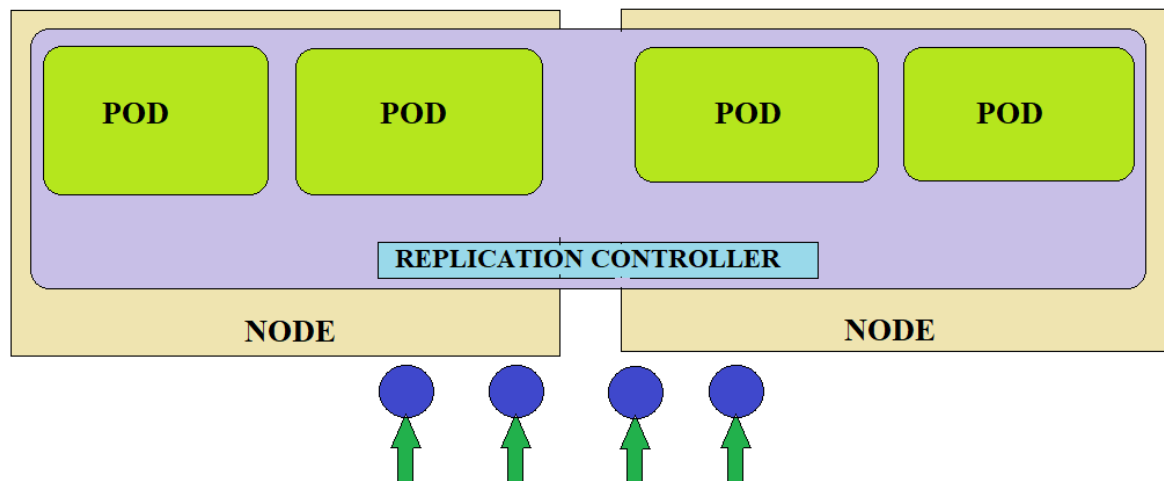1.To create pod from above file

```
kubectl create -f pod_filename.yaml
```

2.To list the created pods

kubectl get pods

3.To delete the pods

kubectl delete pod pod_name

REPLICATION CONTROLLER

So, what is a replica and why do we need a replication controller? Let's go back to our first scenario where we had a single POD running our application. What if for some reason, our application crashes and the POD fails? Users will no longer be able to access our application. To prevent users from losing access to our application, we would like to have more than one instance or POD running at the same time. That way if one fails we still have our application running on the other one. The replication controller helps us run multiple instances of a single POD in the kubernetes cluster thus providing High Availability.

Another reason we need replication controller is to create multiple PODs to share the load across them.

To create Replication Controller

```
apiVersion: v1
kind: ReplicationController
metadata:
  name: myapp-rc
  labels:
    app: myapp
    type: frontend
spec:
  template:
      metadata:
          name: myapp-pod
          labels:
              app: myapp
              type: frontend
        spec:
          containers:
              - name: nginx-container
                image: nginx
```

1. To create replication controller from above file

   kubectl create -f replicationcontroller_filename.yaml

2. To get list of replication controller

   kubectl get replicationcontroller

REPLICASET

ReplicaSet in the Kubernetes is used to identify the particular number of pod replicas are running at a given time. It replaces the replication controller because it is more powerful and allows a user to use the "set-based" label selector.

To create Replicaset

```
apiVersion: apps/v1
kind: ReplicaSet
metadata:
    name: my-replicaset
    labels:
        app: myapp
spec:
   template:
     metadata:
           name: nginx2
           labels:
              app: myapp
       spec:
          containers:
            - name: nginx-cont
              image: nginx
     selector:
        matchLabels:
           app: myapp
   replicas: 3
```

1. To create replicaset from above file

   kubectl create -f replicaset_filename.yaml

2. To get list of created replicaset

   kubectl get replicaset

3. To scale up the pods

   kubectl scale replicaset my-replicaset - -replicas=6

4. To delete replicaset

   kubectl delete replicaset my-replicaset

   note: it also deletes the pod running inside it.

DEPLOYMENTS

A Kubernetes Deployments tells kubernetes how to create or modify instances of the pods that hold a containerized application. Deployments can help to efficiently scale the number of replica pods, enable the rollout of updated code in a controlled manner, or roll back to an earlier deployment version if necessary.

To create deployments

```
apiVersion: apps/v1
kind: Deplyment
metadata:
    name: myapp-deployment
    labels:
        type: front-end
        app: nginx
spec:
    template:
        metadata:
            name: d-pod
            labels:
                app: myapp
            spec:
                containers:
                        -name: nginx-cont
                         image: nginx
        selector:
            matchLabels:
                app: myapp
                type: front-end
```

1. To create deployments

   kubectl create deployment_filename.yaml

2. To get list of created deployments

   kubectl get deployments

3. To set the different image for the existing container

   kubectl set image deployment/myapp-deployment
   nginx=nginx:1.9.1

4. To rollback the changes of deployment

   kubectl rollout undo deployment/myapp-deployment

5.To see all created objects in kubernetes at once

   kubectl get all

INSTALLATION OF KUBERNETES

Pre-requisites:

1.Create AWS instance with ubuntu OS.

2.instance type: t2.medium (or) t2.large.

3.enable the port 8080 and 6443 in security groups.

For installation refer installation of kubectl.pdf