Kubernetes

Kubernetes is a container management system. kubernetes is an open source container orchestration framework which was originally developed by Google . kubernetes helps you to manage applications that are made up of hundreds or maybe thousands of containers and helps it to manage them in different environments like physical machines virtual machines or Cloud environments or even hybrid deployment environments.

Kubernetes is an **open-source Container Management tool** that automates container deployment, container scaling, descaling, and container load balancing **(also called a container orchestration tool).**

Kubernetes also known as K8s was **built by Google. While Docker provides the lifecycle management of containers**, Kubernetes takes it to the next level by providing orchestration and managing clusters of containers. Kubernetes also manages the lifecycle of your container instances.

Kubernetes cluster :-- A cluster is a group of nodes; they can be physical servers or virtual machines that have the Kubernetes platform installed. the Kubernetes cluster consists of a Master node and a number of worker nodes.

A cluster is a set of nodes grouped together. This way even if one node fails you have your application still accessible from the other nodes. Moreover having multiple nodes helps in sharing load as well.

What is ContainerD :-- Kubernetes was originally built to orchestrate Docker containers specifically and there is no longer supports Docker directly, but supports the runtime component of Docker which is managed by ContainerD.

kubernetes CLI :-- Kubectl is the command line utility of Kubernetes. This is the tool or command you would use to operate the kubernetes cluster such as to view the status of the cluster, to provision application, to scale up, scale down, delete and many other things.

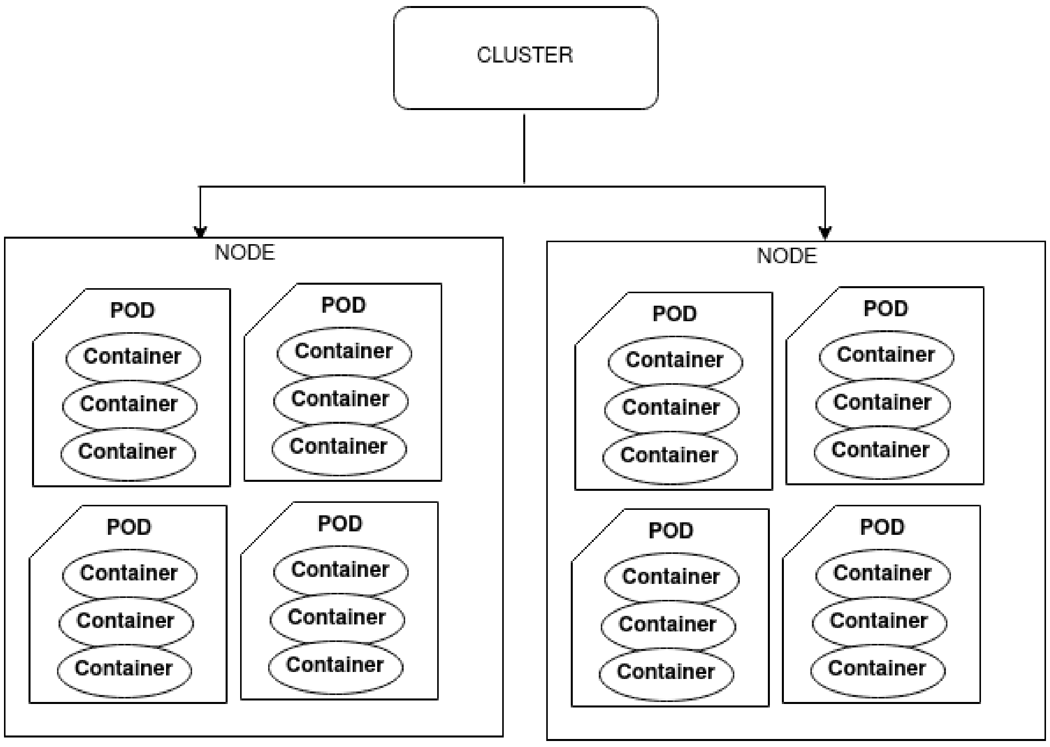
Kubernetes Pods :-- A pod is a collection of containers and its storage inside a node of a Kubernetes cluster. the Pod is the smallest unit of the kubernetes cluster. but usually in practice you're not creating pods or you're not working with the pods directly. **there is an abstraction layer over the pods that is called deployment.**

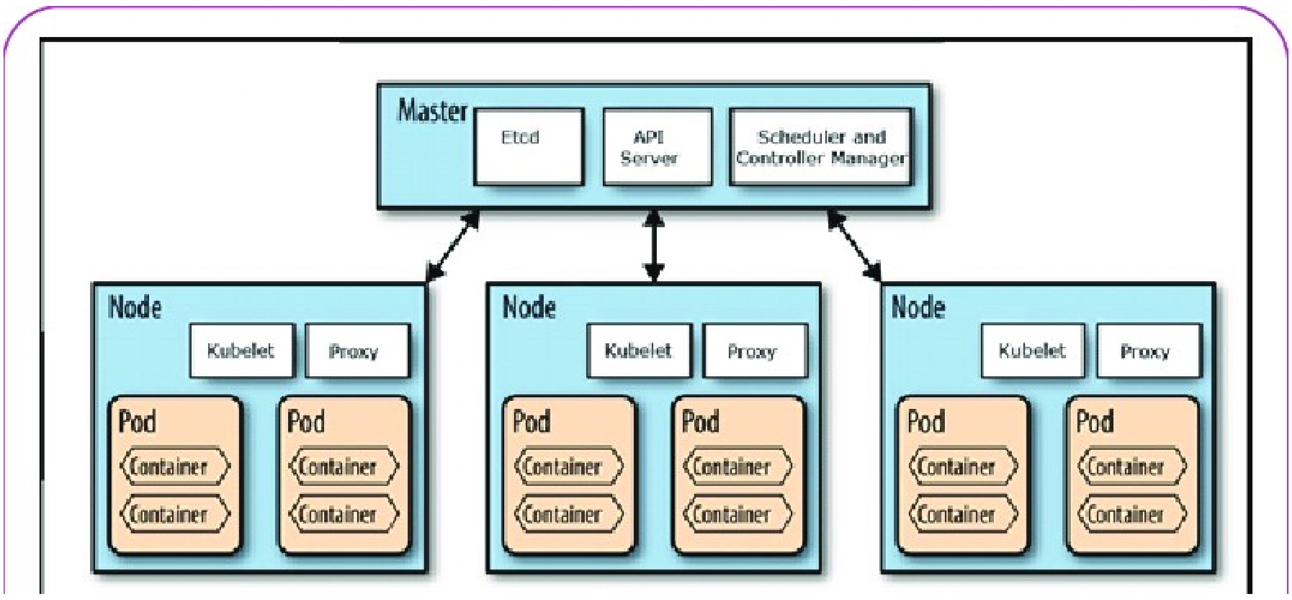
with kubernetes our ultimate aim is to deploy our application in the form of containers on a set of machines that are configured as worker nodes in a cluster. However, kubernetes does not deploy containers directly on the worker nodes. So a **Pod has a one-to-one relationship with a node.** A pod can only run one node at a time.

The containers are encapsulated into a Kubernetes object known as PODs. A POD is a single instance of an application. A POD is the smallest object, that you can create in kubernetes. To create more instances of application you create more Pods.

Typically an application instance running as a container has a 1-to-1 relationship with a Pod. However the 1-to-1 relationship is not a strict rule. The Pod consists of one or more Docker containers. This is the basic unit of the Kubernetes platform and an elementary piece of execution that Kubernetes works with. A pod is the smallest unit that exists in Kubernetes. A specific pod can have one or more applications.

Note :- Cluster is group of nodes and pod is group containers and it stores in the node under the cluster.





Cluster is Group of Nodes and there are two types of nodes
1. one is Master Node
2. Second is Worker node
3. master node can have multiple worker nodes
4. Nodes are nothing but Physical or Virtual machies
   (in Kubernetes ,we called them node).
5. Pod is Group of Container and it found in node.

There are two ways to create POD.
1. imperative way
command :-- kubectl run nginx
NOTE :-- Now this is called as the imperative way to create a POD.

2. declarative way :-- we create the (.yml) file to create the POD in declarative way. The more preferred approach is the declarative way, where you create a YAML file with the specifications of the object.

This approach enables version control, CI/CD and sharing these with others and collaborating together.

Kubernetes uses YAML files as input for the creation of objects such as PODs, Replicas, Deployments, Services etc.
All of these follow similar structure. A kubernetes definition file always contains 4 top level fields.

(apiVersion, kind, metadata, spec) These are top level or root level properties

apiVersion :-- This is the version of the kubernetes API we're using to create the object. **apiVersion is different for each kind.**

KIND	**VERSION**
Pod	v1
Service	v1
ReplicaSet	apps/v1
Deployment	apps/v1

kind:-- The kind refers to the type of object we are trying to create. Remember (kind) is case sensitive.**(Pod, Service, ReplicaSet, Deployment)** are kind.

metadata :-- The metadata is data about the object like its name, labels etc. so metadata is nothing but one dictionary , which gives information of object such as name, labels, type. labels is separate dictionary within metadata dictionary.

Example :--
```
metadata:
 name: myapp-pod
 labeles:
   app: myapp
   type: front-end
```

Important use of (Label metadata) :-- example there are 100s of PODs running a front-end application, and 100's of them running a backend application or a database, it will be DIFFICULT for you to group these PODs once they are deployed. If you label them now as front-end, back-end or database, you will be able to filter the PODs based on this label at a later point in time.

spec : -- Spec is a dictionary so add a property under it called containers, which is a list or an array.
spec field specifies the pod and its desired state (such as the container image name for each container within that pod).
so (spec) is a dictionary of (container and image)s, you can define multiple containers in spec.

What is node :-- A node is a machine – physical or virtual – on which kubernetes is installed. A node is a worker machine and this is were containers will be launched by kubernetes.

Kubernetes Nodes are the Worker or master machines where the actual work happens. A node is a work horse in Kubernetes' architecture. It may be a virtual or physical machine, depending on your infrastructure. A worker node runs the tasks as instructed by the Master node.

WHAT IS REPLICASET :-- 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. And the replicaset brings the failed one back to ensure a pre-defined number of replicas are always running.

The replicaset helps us run multiple instances of a single POD in the kubernetes cluster thus providing High Availability.

A ReplicaSet is a process that runs multiple instances of a Pod and keeps the specified number of Pods constant and provides High Availability to avoid fault tolerance issue.

Another reason we need replicaset is to create multiple PODs to share the load across them. A replicaset, spans across the entire cluster. And a replicaset can deploy a pod on any node in the cluster. It monitors the number of pods in the cluster and ensures enough is deployed at all times. So a Pod has a one-to-one relationship with a node. A pod can only run on one node at a time.

WHAT IS DEPLOYMENT :-- when newer versions of application is released, you would like to UPGRADE your application instances seamlessly. when you upgrade your instances, you may want to upgrade them one after the other. And that kind of upgrade is known as Rolling Updates.

Important Note :-- Suppose one of the upgrades you performed resulted in an unexpected error and you are asked to undo the recent update. You would like to be able to rollBACK the changes that were recently carried out.

The deployment provides us with capabilities to upgrade the underlying instances seamlessly using rolling updates, undo changes, and pause and resume changes to applications running on the cluster.

The Deployment is responsible for creating and updating instances of your application.A Deployment is a higher level of abstraction; it manages ReplicaSets when doing Pod orchestration, creation, deletion, and updates. A Deployment provides declarative updates for Pods and ReplicaSets.

The Deployment is a kind of control structure that takes care of the spinning up or down of Pods. A Deployment takes care of the state of a Pod or group of pods by creating or shutting down replicas.

WHAT IS SERVICES :-- A service is responsible for enabling network access to a set of pods.

In Kubernetes, a Service is a method for exposing a network application that is running as one or more Pods in your cluster. service enables communication between applications within a kubernetes cluster. It provides an endpoint for other services. service enables connectivity between applications within a kubernetes cluster as well as to expose applications outside the cluster.

A service is used to provide a network connection to one or more Pods.
Kubernetes service features are
1. Services are persistent and permanent
2. They provide discovery
3. They offer load balancing
4. They expose a stable network IP address
5. They find Pods to group by usage of labels

There 3 types of service

1. ClusterIP :-- An internal fixed IP known as a ClusterIP and This is the default ServiceType.
ClusterIP exposes the Service on a cluster-internal IP. It makes the Service only reachable within the cluster, not from the outside.

2. NodePort :-- The NodePort service serves as the external entry point for incoming requests for your app.
A node port exposes the service on a static port on the node IP address.

3. LoadBalancer : -- Exposes the service externally using a cloud provider’s load balancer. NodePort and ClusterIP services,
to which the external load balancer will route, are automatically created.

What is Namespace :-- Kubernetes Namespace is a mechanism that enables you to organize resources. It is like a virtual cluster inside the cluster.Namespace helps to organize resources such as pods, services, and volumes within the cluster.
we can create differente namepsace For **(Dev, Prod, UAT)** environment.

What is Autoscaling :-- With horizontal Pod auto scaling, Kubernetes automatically scales the number of Pods in a deployment or ReplicaSet based on observed CPU utilization.

Example command of Autoscaling :--
kubectl autoscale deployment rest-example --cpu-percent=50 --min=1 –max=10

What is API versioning :-- There are three API levels in the Kubernetes API specification: **alpha, beta, and stable.**

alpha :-- The alpha version level is disabled by default, as with the other software, an alpha version should be considered as buggy and not production ready.

Beta :-- The beta level totally different from the alpha level of the API, code is tested (it still may have some bugs, as it is still not the stable release).

Stable :-- The stable level of the API is a tested, production-ready software.

Container and Orchestration.

Containers running in the same Pod share the same common network namespace,disk, and security context.the communication over localhost is recommended between containers running on the same Pod. Each container can also communicate with any other Pod or service within the cluster. The controlplane is another node with Kubernetes components installed in it. The controlplane watches over the nodes in the cluster and is responsible for the actual orchestraCon of containers on the worker nodes.

Kubernetes WorkFlow :--

In Kubernetes will have Mainly Two types of Nodes **(1 is Master Node & 2 is Worker Node).** In Master node will have (4 components)

1. Api Server :- Api Server is nothing but Cluster Gate way. all user request will go through this Api Server.

2. Schedular :- Schedular will get user request from API Server , and then Schedular checks Which (Worker node) is free and send that request to this (Worker node).

3. Controller Manager (Control Plane) :-- Controller Manager keeps the eye of (Worker node) , it checks the which (Worker node) is unhealthy / crashed and it tries to recover it and creates new (Worker node).

4. ETCD :-- ETCD is Kubernetes cluster database in (key value) pair, it keeps the all information like [(Worker node) is unhealthy / crashed]

When you install Kubernetes on a System, you are actually installing the following components.

(An API Server. An ETCD service. Controllers and Schedulers.)

The API server acts as the front-end for kubernetes. The users, management devices,Command line interfaces all talk to the API server to interact with the kubernetes cluster.

Next is the ETCD key store. ETCD is a distributed reliable key-value store used by kubernetes to store all data used to manage the cluster. This is where information about the nodes in the cluster, the application running on the cluster are stored.

The controllers are the brain behind orchestration. They are responsible for noticing and responding when nodes, containers or endpoints goes down. The controllers makes decisions to bring up new containers in such cases.

The scheduler is responsible for distributing work or containers across multiple nodes. It looks for newly created containers and assigns them to Nodes.

On the worker nodes you have the kubelet which is the agent that runs on each node in the cluster. The agent is responsible for making sure that the containers are running on the nodes as expected.

You also have the kube-proxy that is responsible for maintaining rules on the nodes.

On the worker nodes you also have container runtime that is responsible for running containers.

Practicle :--

how to execute (.yml) file to create pod ?

how to execute (.yml) file to create pod ?

Example :-- (.yml) file

pod-defination.yml

```
apiVersion: v1
kind: Pod

metadata:
 name: myapp-pod
 labeles:
   app: myapp
   type: front-end
spec:
 containers:
 - name: nginx-container
   image: nginx
```

how to execute (.yml) file to create pod ?

command :-- kubectl create -f pod-defination.yml

explanation :--
kubectl create -f <FOLLOWED BY THE FILE NAME>

or you can use apply command.
command :-- kubectl apply f pod-defination.yml

Once we create the pod, how do you see it?
command :-- kubectl get pods

How To see detailed information about the pod?
command :-- kubectl describe pod myapp-pod

how to create replicas

Example :-- (.yml) file

```
apiVersion: apps/v1
kind: ReplicaSet
metadata:
 name: myapp-replicaSet
 labeles:
   app: myapp
   type: front-end
spec:
  template:
    metadata:
     name: myapp-replicaSet
     labeles:
```

how to execute (.yml) file to create replicaset ?

command :-- kubectl create -f replicaset-defination.yml

explanation :--
kubectl create -f <FOLLOWED BY THE FILE NAME>

or you can use apply command.
command :-- kubectl apply f replicaset-defination.yml

How to increase the replicaset run time.

Answer :-- you just modify the replicas

example :--

```
        app: myapp
        type: front-end
    spec:
    containers:
    - name: nginx-container
      image: nginx

  replicas: 3
  selector:
    matchLabels:
      app: front-end
  strategy:
    type: RollingUpdate
```

```
replicas: 6
  selector:
    matchLabels:
      app: front-end
  strategy:
    type: RollingUpdate
```

and run below command

command :-- kubectl apply -f replicaset-defination.yml
explanation :-- above command will create 3 more pods

other wise ,Without making any change in replicas you can run another command

command :-- kubectl scale --replica s= 6 -f replicaset-defination.yml

other replicaset command

Once we create the replicaset, how do you see it?
command :-- kubectl get replicaset

How to delete replicaset
command :-- kubectl delete replicaset myapp-replicaSet

how to create deployment

Example :-- (.yml) file

deployment-defination.yml

```
apiVersion: apps/v1
kind: ReplicaSet
metadata:
 name: myapp-deployment
 labeles:
   app: myapp
   type: front-end
spec:
  template:
    metadata:
     name: myapp-pod
     labeles:
        app: myapp
        type: front-end
    spec:
    containers:
    - name: nginx-container
      image: nginx

  replicas: 3
  selector:
    matchLabels:
      app: front-end
```

how to execute (.yml) file to create deployment ?

command :-- kubectl create -f deployment-defination.yml
explanation :--
kubectl create -f <FOLLOWED BY THE FILE NAME>

or you can use apply command.
command :-- kubectl apply f deployment-defination.yml

other deployment command

Once we create the deployment, how do you see it?
command :-- kubectl get deployment

How to delete deployment
command :-- kubectl delete deployment myapp-deployment

How to see all created objects at once ?
command :-- kubectl get all

```
  strategy:
    type: RollingUpdate
```

how to create service & deployment file and create relation.

how to create deployment

Example :-- (.yml) file

Example :-- deployment-defination.yml

```
apiVersion: apps/v1
kind: ReplicaSet
metadata:
 name: myapp-deployment
 labeles:
   app: myapp
   type: front-end
spec:
  template:
    metadata:
     name: myapp-pod
     labeles:
        app: myapp
        type: front-end
    spec:
    containers:
    - name: nginx-container
      image: nginx

  replicas: 3
  selector:
    matchLabels:
      app: front-end
  strategy:
    type: RollingUpdate
```

how to create Service

Example :-- (.yml) file

Example :--   service-defination.yml

```
apiVersion: v1
kind: Service
metadata:
  name: redis-db
spec:
   type: ClusterIP
   ports:
     - targetPort: 6379
       port: 6379

   selector:
        app: myapp
        type: front-end
```

kubectl command

```
kubectl version
kubectl --help
kubectl get nodes
kubectl get nodes -o wide
```

DOCKER :-- docker is an open source and virtualization software platform used to build design and manage applications.
A Docker container image is a lightweight, standalone, executable package of software that includes everything needed to run an application.
docker container is an executable software package that includes all the dependencies such as frameworks libraries etc that are required to execute an application.

Docker Architecture :--- docker is a client server app.
docker daemon serves your application to create deploy shutdown etc and the docker client communicates with the daemon to manage its services.
client and server are found in docker engine. docker engine or docker is a client server application that builds and executes containers using docker components.

Docker is a virtualization tool just like a virtual machine.

Docker image :--
1. docker image is a template of instructions that are used to create containers.
2. image is a read only pattern that is used to build a container.
3. an image is built using a file called dockerfile.
4. docker image has multiple layers, whenever a developer builds a container a new docker layer is automatically created on top of image layers and it is called as a container layer.
5. a docker image provides a separate read or write layer to each container and whenever a user makes a change to the container that gets reflected upon the specific container layer alone.

What is Docker container :- a running instance of an image is a container and from one image you can run multiple containers.
Docker containers help us to create isolated environments on our systems to run applications completely isolated from each other.

Docker registry :--
1. docker registry is a service to the host and it distributes docker images amongst users
2. a repository is a collection of docker images.
3. docker registry has public and private repositories.

Docker Swarm :-- Docker is used to create, distribute and run isolated containers, while Docker Swarm is used to manage and orchestrate a cluster of Docker containers.
Docker Swarm is a container orchestration tool for clustering and scheduling Docker containers.
Docker Swarm lets developers join multiple physical or virtual machines into a cluster.

DOCKER FILE directives

1. FROM
2. WORKDIR
3. COPY
4. RUN
5. CMD
7. ARG
8. ENV
9. EXPOSE
10. VOLUME
11. MAINTAINER
12. ADD
13. ENTRYPOINT
14. LABEL
15. USER

Containers are completely isolated environments, as in they can have their own processes or services, their own network interfaces, their own mounts, just like Virtual machines, except that they all share the same OS kernel.

What is docker image :-- Image as a read-only template which is a base foundation to create a Docker container. Images are created using a series of commands, which we used inside the Dockerfile.

What is Docker Container :-- A running instance of an image is called a container. Docker launches them using the Docker images as read-only templates. Containers are runtime instances of an image. They can be running or stopped. You can have multiple containers of the same image running.
the container is a runtime of an image. It will contain your Java application altogether with all needed dependencies, such as JRE or an application server.

Kubernetes Vs Docker

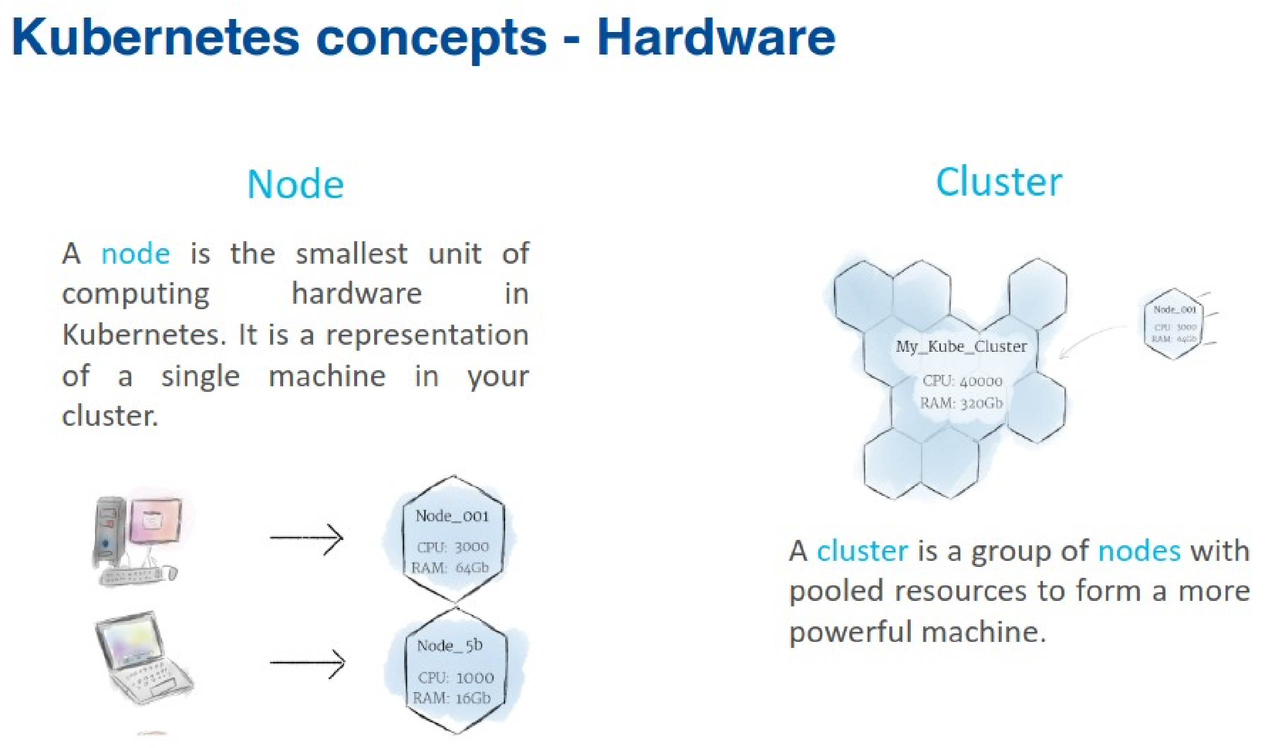
Kubernetes	**Docker**
1. Kubernetes is a container management system.	1. Docker is a container runtime
2. Kubernetes supports auto scaling and	2. docker does not support auto scaling.
3. Kubernetes has High fault tolerance	3. docker has low fault tolerance
4. Kubernetes is a platform for running and managing containers from many container runtimes.	4. Docker Swarm is also an orchestration tool.

what are Containers?

• Containers provide a standard way to package your application's code, configurations, and dependencies into a single object. Containers share an operating system installed on the server and run as resource-isolated processes.

• Quick, reliable, and consistent deployments, regardless of environment.

Container orchestra[on defines tools that automate the deployment, management, scaling, networking, and availability of container-based applications.



Kubernetes concepts - Hardware

Node

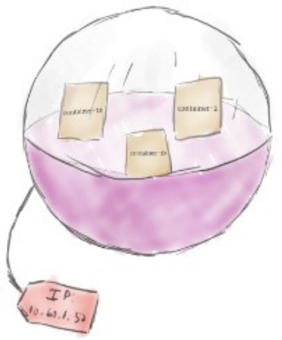
A node is the smallest unit of computing hardware in Kubernetes. It is a representation of a single machine in your cluster.

Cluster

A cluster is a group of nodes with pooled resources to form a more powerful machine.

Kubernetes concepts - Software

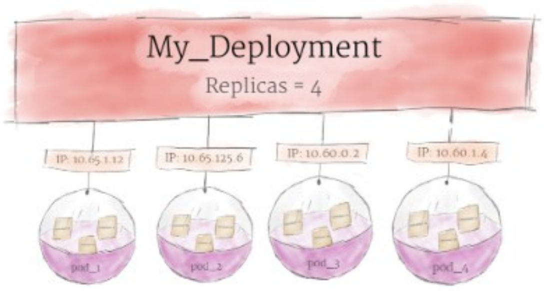
Pods



A Pod represents a single instance of an application in Kubernetes

Might consist of either a single container or a small number of containers that are tightly coupled and that share resources.

Deployments



When a deployment is added to the cluster, it will automatically spin up the requested number of pods, and then monitor them. If a pod dies, the deployment will automatically re-create it.