**Kubernetes:**

* Open-source container orchestration(systemically) as a collection of small, independently deployable services.
* Highly available and scaling (means multiple copies) and managements of container application.
* Scheduling capacity.
* In docker server we call docker host
* In Kubernetes we call it as worker node.
* What is Cluster of multiple nodes is called worker node.

**Micro Services:**

* Splitting the services(combination)
* Perform functionality. (well defined API and protocol)
* Micro services are deployed in container
* Docker micro service is called container and Kubernetes it called as pod.

Container ----🡪pod---🡪Kubernetes

**Architecture Kubernetes:**

**Cluster: Master and Node are mandatory.** (Collection of nodes)

1. Master node
2. Multiple worker nodes

**Master node: (control Plane)**

1. Cube API server
2. Controller manager
3. Schedular
4. Etcd

Can use multiple master node but need high availability.

Master nodes can be on deployment in cloud environment we call as

1. **Aws**🡪 elastic Kuber (EKS)
2. **Google** 🡪 google Kuber engine (GKE)
3. **Microsoft** 🡪 Azure Kuber services (AKS)

On premises -🡪 we are responsible of master node with multiple and high availability.

**Cube API server:**

* Acts and medicator b/w master and node (check what is asked and perform the action accordingly)
* authorization and authorisation

**Etcd Database:**

* (kubelet related data is stored) Data related to cluster means nodes highly availability.

**Scheduler:**

* Distribute the newly created nodes (data) based on the availability of nodes. (calculates the data and pass data to that node)

**Controller Manager:**

* Control the all the components in master.

Scheduler

API Server

Control manger

Etcd Database

**Worker node components:**

1. Kubelet
2. K-proxy
3. Container platform

**Kubelet Agent:**

* always shared the information to master upto data.
* have data related to node like space and memory usage, transaction.

**K-proxy**

* Communication n/w container eatable.
* Choose the best path to communication.

**Container platform:** container need to install in the container platform.

**GIT:**

**Inside the Folder:**

git init

git commit -m "first commit"

git branch -M main

git status

git add .

git status

git commit -m "first commit"

git push -u origin main

**…or create a new repository on the command line**

echo "# Kubernetes" >> README.md

git init

git add README.md

git commit -m "first commit"

git branch -M main

git remote add origin https://github.com/srilatha71/Kubernetes.git

git push -u origin main

**…or push an existing repository from the command line**

git remote add origin https://github.com/srilatha71/Kubernetes.git

git branch -M main

git push -u origin main

**Install Kubectl and eksctl and Docker in AWS**

**Kubectl:**

<https://docs.aws.amazon.com/eks/latest/userguide/install-kubectl.html#linux_amd64_kubectl>

**eksctl:**

<https://eksctl.io/installation/>

**Aws config:**

Account setting 🡪security credentials 🡪create security credentials

**cluster :**

<https://eksctl.io/getting-started/>

**Code for Cluster:**

apiVersion: eksctl.io/v1alpha5

kind: ClusterConfig

metadata:

name: expense-1

region: us-east-1

managedNodeGroups:

- name: ng-1

instanceType: m5.large

desiredCapacity: 10

spot: true

**Advantages:**

ASG -ec2 instance (SG) (IAM)

Expose.

**Workload Components:**

**Pods**-🡪 command line -🡪 pod commands

* Collection one or more container.

**Developments:**

* collection of Pods (copies)

**Services or ingress:**

* n/w communication pods and developments

**Configmaps:**

* configuration

**Secrets:**

* For patch work

**Namespace:**

* Logic parties like dev, test, prod

**Persistent Volumes:**

* Storage

**Statefulsets and more:**

* Database container to setup.

**Pods:**

* Collection of containers.
* To provide high availability we use deployments
* To create deployments, we use replicasets or statesets

**Eg: YAML**

**apiVersion:** v1

**Kind:** pod or deployment or repliceset

**Metadata:**

**Name:** my-pod (like pod name , type of application, last modified like base information)

**Spec:** (container deliates)

**Containers:**

* Name : my-container
* Image: nginx:latest
* Ports:
* Container port: 8080

Op:

Name: my-prod1

Namespace: default

Priority: 0

Service Account: default

Node: minikube/192.168.49.2

Start Time: Sun, 16 Feb 2025 11:00:32 +0000

Labels: <none>

Annotations: <none>

Status: Running

IP: 10.244.0.5

**Deployments:**

* Defines no of container images and no of replicas and other configuration parameters.
* Self healing capacity

**Eg:**

**apiVersion:** apps/v1

**Kind :** deployment

**Metadata:**

**Name:** my-pod (like pod name , type of application, last modified like base information)

**Spec:** (container deliates)

**Replicas:** 3

**Selector:**

matchLables:

app: my-app

**templates:**

metadata:

labels: (specific the name and map to service)

app: my -app

**spec:**

Containers:

* Name: my-container
* Image: nginx:latest
* Ports:
* Containerport: 8080

**Replicasets:**

* No of copies
* Managing the lifecycle of pods.
* Highly availability and scalability

**Eg:** same as above code but kind : Replicaset

**From windows PowerShell:**

1. **Kubectl apply -f .\<code\_file\_Name>:**
2. **Kubectl get pods:**
3. **Kubectl describe pods .\<code\_file\_Name>:**
4. **Kubectl edit pods:**

open the file and edit mode like notepad

1. **Kubectl get pods**
2. **Kubectl delete pod <code\_file\_Name>:**

It deletes and create new pod.

O/p: get details of Ip address.

1. **Kubctel describe svc my-app-cip**

O/P: IP: 10.0.3.20

Endpoint: 10.3.4.19, 10.23.4.67, 10.3.5.8.50

1. **Kubctel get pods – o wide:**

O/P: show list of all 3 ip addresses.

Kubectl get pods

Kubectl get deployments

Kubectl get replicate

**Kubectl apply -f .\<code\_file\_Name>:**

* Its read the file and create deployment

**Kubectl delete -f .\<code\_file\_Name>:**

* Its deleted deployment

**Kubectl get pods:**

* Shows the pod is running state

**Kubectl describe pod <created pod name>:**

* All information of the pod

**Egs:** all event, start, stop and running IP address.

**Kubectl get nodes:**

It shows the no of worker nodes

**Kubectl get pods:**

Its shows no of pods created

**Kubectl apply -f .\sample\_deploy.yaml:**

Deployment.apps/my-deployment created

**Kubectl get deploy:**

Its shows no of deploy created

**Kubectl describe replicaset <created pod name>:**

* All information of the replicaset (which doesn’t have control by)

Note : in Really time we don’t use replicate set

**Services:** Its standard IP (app =A). Its baind to specific IP address.

It’s a consistent way to access and communicate with set of pods.

Its act as stable endpoints for accessing the pods, enabling inter-pod communication and load balancing.

Eg: each pod will have ip address and enabling inter-pod communication and load balancing.

**Types of service:**

1. Cluster Ip-🡪 within cluster to inter communicate (b/w clusters) and secured connection.
2. Nodeport -🡪 exposes service on a specific port on each node in the cluster. Using cluster IP addresses. (using in testing purpose)
3. LoadBalancer -🡪 distribute traffic to the service. Using publicly accessible Ip address
4. ExternalName-🡪 DNS name allowing the service to redirect request to external endpoints.

Label Pod (App: A)

Nodeport -----🡪 Service (App: A) ------🡪 pod (App: A)

(load Balancer) pod(App: A)

1. Cluster IP :
2. **Kubectl get service**

**O/P: Name Type Cluster-IP External-IP Ports Age**

Kub cluster 18.54.67.83 <none> 443/port 43

1. **Kubectl apply -f <code\_file\_Name>:**

**Note : code file as deployment and service**

**O/p:** my\_cip : can see deployment and service

1. **Kubctel get deployment**

O/p: available pod details like :3

1. **Kubctel get pods -o wide:**

O/p: get details of Ip address.

1. **Kubctel get service:**

**O/P: Name Type Cluster-IP External-IP Ports Age**

Kub cluster 18.54.67.83 <none> 443/port 43

My-cip cluster 18.54.67.16 <none> 80/port 43

1. NodePort:
2. **Kubctel get service:**

**O/P: Name Type Cluster-IP External-IP Ports Age**

Kub cluster 18.54.67.83 <none> 443/port 43

My-cip cluster 18.54.67.16 <none> 80/port 43

Node nodeport 23.45.65.23 <none> 80:4322/TCP 49

3.load Balancer:

1. **Kubctel get service:**

**O/P: Name Type Cluster-IP External-IP Ports Age**

Kub cluster 18.54.67.83 <none> 443/port 43

My-cip cluster 18.54.67.16 <none> 80/port 43

Node nodeport 23.45.65.23 43.56.87.35 80:4322/TCP 49

Load loadbalancer 23.45.65.23 <none> 80:3672/TCP 49

**Ingress:**

It act as layer 7 (Application layer) and route to different services based on Http/https routes and rules.

It transforms the load to different application based on request.

Ingress Resources (check in list)



User request ---🡪 ingress controller ---🡪 service -app

Nginx is popular ingress controller.

Eg:

We have code for kind: deployment and service –cluster IP

To start working on ingree:

Create naginx ingress first -🡪Kubectl create namespace ingress-nginx

To deploy ingree controller -🡪 github they maintain repository link .yaml file to download it

It will create all resources: like pods, service, deployment, replicatset, job etc.,

To see all resources in ingress : kubctel get all -n ingress-nginx

Run the code to see the deployment and service cluster ip.

Ingress \_resource:

After base url: /nginx(/|$)(.\*)

o/p: welcome to nginx

www after: /httpd(/|$)(.\*)

o/p: hit words message

if it base url: /(.\*)

o/p: base url

use apply -f <file name>

kubectl get ingress

if not

kubectl get service

based on that get

kubectl get ingress

Address: 34.135.200.115

Use the above same ip address : /nginx : it opens the welcome page.

Kubectl get service -n ingress-nginx

Load balancer-🡪External: 34.135.200.115

Cluster 🡪 none

Advantage: using single load balancer using ingress (based on the path)

ConfigMaps and secrets:

Raw format for Pod:

Kind: pod

Apiversion: v1

Metadata:

Name : mutli-container

Spec:

**Label:**

Kind: pod

Apiversion: v1

Metadata:

Name: labels

Labels:

Project: expense

Module: backend

Environment: dev

Spec:

Containers:

* Name: nginx
* Image :nginx

**Annotations:**

Kind: pod

Apiversion: v1

Metadata:

Name: annotations-demo

Annotations:

Imageregistry: “<https://hub.docker.com>”

buildURL: “https://jenkins.com”

Spec:

Contains:

* Name: nginx
* image :nginx

ports:

* containerPort: 80

**Labels:** limited characters (tags)

**Annotations:** more data with URLs (based on depends)

**Environment:**

ENV in image defition:

Env in Dockerfile should rebuild if you change

Env in manifest no need to rebuild, just restart is enough

Kind: pod

Apiversion: v1

Metadata:

Name: Environment

Spec:

Contains:

* Name: nginx
* image :nginx

env:

-name : course

Value: devops

-name: trainer

Value: “Sivakumar reddy”

**Resource utilization:**

If something goes wrong in loop, it will occupy entire host resource.

1. Softlimit
2. Hardlimit

Kind: pod

Apiversion: v1

Metadata:

Name: limits

Spec:

Containers:

* Name: nginx
* image :nginx

resources:

requests:

cpu: 100m

memory: 68Mi

limits:

cpu: 120m

memory: 100mi