

Kubernetes

- We already know about the containers from docker, As we used run applications and services in containers
- Here as a real time example, our jio cinema streaming the IPL matches these are running on container, when users increased watching we need our containers also increased to handle the load, but as a person we don't know when the load will increases and decreases
- To Solve this problem google worked on this orchestration and developed Kubernetes
- Before docker Google and few companies used to run their ecosystems on containers, Google had their own orchestration softwares like Omega and Borg build by them.
- Google started writing an open source software which does this orchestration(automatically scaling containers,made container to run on production) using language called as GO. That is Kubernetes
- It is open source software
- There are some other softwares similar to k8's in market like AWS ECS, even though k8 got popular since google was made it and it is using for 15 years.
- The features which are there in k8's are eminent compared to others since it is leader
- K8s started with supporting Docker and it became an enhancement software Docker, when it got bigger they ignored Docker and made to support any kind of containerisation tools.
- Advantages:
 - In k8s whenever you create anything it has label
 - In the world of docker we create container similarly in the world of k8s we create Pods.
 - you can create clones using Replica sets
 - it as SERVICES to speak one part of application with another part of application
 - It has Persistent Volume
- What is Cluster:
 - There some server which runs containers so that our application runs these servers called as Nodes, this creates clusters
 - To run these clusters we have some servers called as Master Nodes.
 - Cluster is combination of two types of servers
 - Master Nodes (Servers which manages Nodes)
 - Nodes (Manages container servers)
- As a User we speak with Master Node, this will speak with nodes to make our job
- To interact with Master Node we have two ways

- through Kubectl
- programming libraries
- Kubectl(command line tool):
 - we will write these files using language called YAML
 - These are manifest files
- Programming languages:
 - JSON files

Architecture :

- ON the Master NODE :
- when ever we are using kubectl we are speaking with kube api server this will speak with etcd(memory of k8s cluster[a key value store]) api server will store that information in etcd.
- to the k8s we always ask what we want, this information is stored in etcd memory, say like you have asked to run nop application on three pods, Scheduler will do this work for us.
- when ever you want to create anything new scheduler is responsible for this.
- At first scheduler will check which node fits for you application to run efficiently among all the nodes it has and it creates a container inside and it starts running app inside.
- Controller Manager is responsible for maintaing the needs perfectly on what you have asked
- say like you have asked for run an application in three pods controller manager always cheks whether the application is running on three pods, for some reason one of the server is down at this time we have 2 pods runnig now controller manager speaks with the api server and stores the information that we have only two pods running not three then api server assigns a job to scheduler, scheduler will figure out and try to run one more pod, this property is called self healing.
- these all components are called as control plane components because these control k8s clusters
 - kube api server
 - etcd (memory of k8s)
 - controller manager
- ON THE NODE :
- on the nodes we have three componets
 - kubelet (it is an agent of k8s)
 - it won't take any decision on its own but it will executes whatever the command which comes from control plane
 - Docker run time (you can replace with any container technology)
 - kube proxy (handles networking)

Overview of k8s components

- Pods :
 - pod is a atomic unit of creation in k8s
 - Pod will have containers in it
 - All the containers in the pod share the same namespaces
 - Each pod gets a unique ip address in the container
 - pod specification will have details about running containers
 - if a container suddenly stops in a pod, k8s will start the container automatically this process is called self healing process
- Replica Set:
 - Replica sets are controller objects (they control certain things)
 - Replica sets create multiple replicas of pods
 - Desired state will be in about pods and number of replicas
- Deployment:
 - This controller helps in performing zero downtime deployments
 - deployment internally creates replicaset and replica sets create pods create container
 - Deployment enables rollouts and rollbacks
- Labels:
 - A label is a key value attached to any k8s object
 - labels are query mechanisms in k8s
- Service:
 - service is used to expose pods with matching labels to
 - other pods
 - externally to access outside world.

CRI (Container Runtime Interface)

- CRI is an interface for k8s to communicate with container runtimes
- it is middle ware for k8s and container runtimes
- when you install k8s, along with k8s the CRI which supports docker is also gets installed, they felt k8s getting heavy because of it and they removed, k8s 1.23 version we will get docker cri included later it was removed
- Moby is a company which maintains k8s now, this company writes a CRI for docker
- OCI(open container initiative):
 - docker image which is built using docker technology can run in other container technologies because it is built with OCI
- after 1.24 version, you need to install docker cri to use docker in k8s

Finally : * For k8s to speak with any containerisation tool it needs a CRI * it is not that k8s stopped supporting docker, but earlier to speak with docker k8s developed CRI for docker , now it is removed. * later from 1.24 other opensource such as cri-dockerd and mirantis written this component

CNI (container Network Interface)

- This is used for networking implementation in the k8s cluster
- This can also be implemented by any one
- There are many CNI's available.
- it is similar to CRI but CNI is for Networking

CSI (Container storage Interface)

- CSI interface helps in creation, updation and management of volumes in various storage sources.
- When you are installing k8s you need to mention which CRI and CNI you need use, later if you are going to use storage you need mention CSI k8s doesn't have any default CRI or CNI or CSI.

K8s INstallation options

- Desktops:
 - minikube
 - kind
- On-prem servers
 - kubeadm
 - kubespray
- Cloud based(k8s as a service)
 - AKS (azure)
 - EKS (Aws)
 - GKE (google)

JUN 11

Kubernetes setup :

- Kubeadm setup:
 - This is a setup where need to have physical or virtual machines with connectivity between them ready i.e we can use this setup on-premises as well as virtual machines
- Managed K8's:
 - This is k8's as a service offered by cloud providers

Kubeadm

- we need to take two vms which are in same network , we name one of them as Master and other two by Node 1, Node 2.
- Allow all the ports on both vms
- Install the container runtime we, i am going to install docker for now [docker install link](#)
- add current user to docker group `sudo usermod -aG docker azureuser` in both machines

- logout and login again.
- Now install kubeadm, kubelet, kubectl








```
sudo apt-get update
sudo apt-get install -y apt-transport-https ca-certificates curl gpg



curl -fsSL https://pkgs.k8s.io/core:/stable:/v1.31/deb/Release.key | sudo gpg --
dearmor -o /etc/apt/keyrings/kubernetes-apt-keyring.gpg

echo 'deb [signed-by=/etc/apt/keyrings/kubernetes-apt-keyring.gpg]
https://pkgs.k8s.io/core:/stable:/v1.31/deb/ /' | sudo tee
/etc/apt/sources.list.d/kubernetes.list

sudo apt-get update
sudo apt-get install -y kubelet kubeadm kubectl
sudo apt-mark hold kubelet kubeadm kubectl
```

- now install docker cri [refere here for docker cri](#)

 cri-dockerd_0.3.11.3-0.debian-bookworm_amd64.deb	9.51 MB	Mar 8
 cri-dockerd_0.3.11.3-0.debian-bullseye_amd64.deb	9.51 MB	Mar 8
 cri-dockerd_0.3.11.3-0.ubuntu-bionic_amd64.deb	9.51 MB	Mar 8
 cri-dockerd_0.3.11.3-0.ubuntu-focal_amd64.deb	9.51 MB	Mar 8
 cri-dockerd_0.3.11.3-0.ubuntu-jammy_amd64.deb	10.6 MB	Mar 8
 Source code (zip)		Mar 8
 Source code (tar.gz)		Mar 8

 2  1 3 people reacted

- here we choose Jammy

```
wget https://github.com/Mirantis/cri-dockerd/releases/download/v0.3.11/cri-
dockerd_0.3.11.3-0.ubuntu-jammy_amd64.deb
```

- now do this

```
sudo dpkg -i cri-dockerd_0.3.11.3-0.ubuntu-jammy_amd64.deb
```

- now switch to root user

```
sudo -i
```

- now if you do `kubeadm init` it won't work because, we have two cri's installed, execute it and read the info. we need to say which one to use

```
in Root mode
kude
kubeadm init --cri-socket unix:///var/run/cri-dockerd.sock
```

- After this you will get successfull message

```
[addons] Applied essential addon: kube-proxy

Your Kubernetes control-plane has initialized successfully!

To start using your cluster, you need to run the following as a regular user
:

mkdir -p $HOME/.kube
sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
sudo chown $(id -u):$(id -g) $HOME/.kube/config

Alternatively, if you are the root user, you can run:

export KUBECONFIG=/etc/kubernetes/admin.conf

You should now deploy a pod network to the cluster.
Run "kubectl apply -f [podnetwork].yaml" with one of the options listed at:
https://kubernetes.io/docs/concepts/cluster-administration/addons/

Then you can join any number of worker nodes by running the following on each
as root:

kubeadm join 10.0.0.5:6443 --token mi35xm.7wwwvh78wae248luy \
--discovery-token-ca-cert-hash sha256:6bc922059897732541a4c9484b7a2d
5098ce97223baeb48741e155b39ee32ce0
root@vm-1:~# exit
logout
azureuser@vm-1:~$ mkdir -p $HOME/.kube
sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
sudo chown $(id -u):$(id -g) $HOME/.kube/config
azureuser@vm-1:~$ kubectl get nodes
NAME          STATUS    ROLES    AGE   VERSION
root@vm-1:~$
```

Till now for both servers above is common, from now things will change

For Node servers:

- here in this message we see `kubeadm join ...` script we need to execute this script in the servers which we need to join with our master node.

```
kubeadm join 10.0.0.5:6443 --token mi35xm.7wwvh78wae248luy \
--discovery-token-ca-cert-hash
sha256:6bc922059897732541a4c9484b7a2d5098ce97223baeb48741e155b39ee32ce0
```

here we need to define **which** cri to be used and execute it because **for** above script there is nothing mentioned

THIS SHOULD BE EXECUTED IN ROOT MODE

```
kubeadm join 10.0.0.5:6443 --token mi35xm.7wwvh78wae248luy \
--discovery-token-ca-cert-hash
sha256:6bc922059897732541a4c9484b7a2d5098ce97223baeb48741e155b39ee32ce0 --cri-
socket "unix:///var/run/cri-dockerd.sock"
```

- that's it for Node server, now we should initialize the kubeadm in master server and check nodes.

For Master Node :

- now come back to normal user `exit`
- take the piece of code given and execute it

```
mkdir -p $HOME/.kube
sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
sudo chown $(id -u):$(id -g) $HOME/.kube/config
```

- now check the nodes using `kubectl get nodes`
- this shows nodes are not ready
- to start them we need CNI to be there. we know two things are necessary in k8's, 1.CNI and 2. CNI
- again here there are many CNI available, we are going with Weave
- go to this link [refer link](#)

```
kubectl apply -f https://reweave.azurewebsites.net/k8s/v1.29/net.yaml
```

- To run the nodes

```
kubectl get nodes -w
```

- check again `kubectl get nodes` it should be running now

How Kubectl works

- kubectl is the tool which interacts with API server for the inputs which we give
- kubectl will have communication with API server over https.
- kubectl will use a configuration file, this file having the information of certificates that belongs to API servers.

Script:

```
apiVersion: v1
kind : Pod
metadata:
  name: hello-pod
spec:
  containers:
    - name: hello-container
      image: nginx:latest
      ports:
        - containerPort: 80
```

```
apiVersion: v1
kind: Pod
metadata:
  name: activity-1
  labels:
    app: nop
    env: dev
    purpose:
spec:
```

Kubernetes Pods

- Pod have three types of containers
 - init containers
 - containers
 - ephemeral containers
- init container:
 - these are executed sequentially(series one up on other)
 - one must be completed to execute next
- containers:
 - here we run the application or application component
 - the created containers will all starts parallelly
 - first container in a spec is referred as car rest are referred as side cars

- all containers in a Pod share same network namespace i.e localhost. they can communicate each other through local host
- it is good practice that defining max cpu utilization in manifest file, this makes scheduler to find effective node to run application.

Labels:

- whenever you create a k8's object you need try to create labels
- labels have two things to define name:value
- you can define multiple labels
- these labels are used to querying the k8's objects through selectors
- Selectors:
 - equality based:
 - we have two options `equals` and `not equals`.
 - set based:
 - this gives more comprehensive way to select, we have multiple operators such as
 - in
 - not in
 - exists

Resources and Limits

- Using this we can specify the k8's resources limits
- Requests: refer to lower boundary
- limits: refer to upper boundary
- these values will be defined based on the performance test conducted on the app like load testing.
- you should give max cpu value necessarily else you should get this values from architect this effects on billing.
- kubernetes billing is mainly depend on this cpu values.

```
---
apiVersion: v1
kind: Pod
metadata:
  name: activity-1
  labels:
    app: nop
    env: dev
    purpose: practice
spec:
  containers:
    - name: nop
      image: aravindh146/nopcommerece:v1.0
      ports:
        - containerPort: 5000
```

```
    protocol: TCP
  resources:
    limits:
      cpu: "500m"
      memory: "256Mi"
    requests:
      cpu: "250m"
      memory: "64Mi"
```

- Use killer coda website to practice k8's
- create a folder and create a file inside it with name activity1.yaml paste above code
- go to shell cd to the folder
- run command `kubectl apply -f activity1.yaml`
- run command `kubectl get pods -o wide` . this will shows the creating pods
- run command `kubectl get po` to get pods
- run command `kubectl describe pod activity-1`
- Rcmd `kubectl get pods -o wide` this shows ip address of pods. where each pods get an ip address
- `kubectl get pods --show-labels` this displays labels for pods

** Note: we can create k8's pods with imperative commands.**

- `kubectl run --image nginx app1` create a pod through commands
- select pods by labels:
 - `kubectl get pods -l "env=dev"`
 - `kubectl get pods -l "run"`
- Delete Pods:
 - `kubectl delete -f activity1.yaml`
 - `kubectl delete pod activity-1`

init Containers

- Order of creation is
 - init containers one by one
 - containers all at one shot

Activity

```
apiVersion: v1
kind: Pod
metadata:
  name: activity-2
  labels:
    env: dev
```

```
  purpose: testing
spec:
  initContainers:
    - name: initmaincar
      image: alpine
      args:
        - sleep
        - 10s
    - name: initsidecar
      image: alpine
      args:
        - sleep
        - 11s
  containers:
    - name: maincar
      image: nginx
      ports:
        - containerPort: 80
    - name: sidecar
      image: alpine
      args:
        - sleep
        - 1d
```

- using above code we are going to create 4 containers in pod.
- 2 init containers and 2 containers
- first init containers will get executed and next containers will start executed
- take the script and apply it for creation pod.
- execute this after pod is running `kubectl exec activity1 -- pwd`
- `kubectl exec activity1 -c sidecar -- uname` will speak about distribution.
- `kubectl exec activity1 -c sidecar -it -- /bin/sh ->` for alpine
- `kubectl exec activity1 -c maincar -- /bin/sh ->` for bash