KUBERNETES

- Kubernetes is a cluster manager for docker used for orchestration, service discovery and load balancing
- It is master and node concept
- Kubernetes minions run tasks passed to it by the Kubernetes master
- Kubernetes uses Etcd to communicate master and nodes
- Kubernetes having a cluster which it manages the containers in the server when severs get down and run the containers in the servers which was up and auto scales the containers in the server which is up

Components of Kubernetes

1.API Server

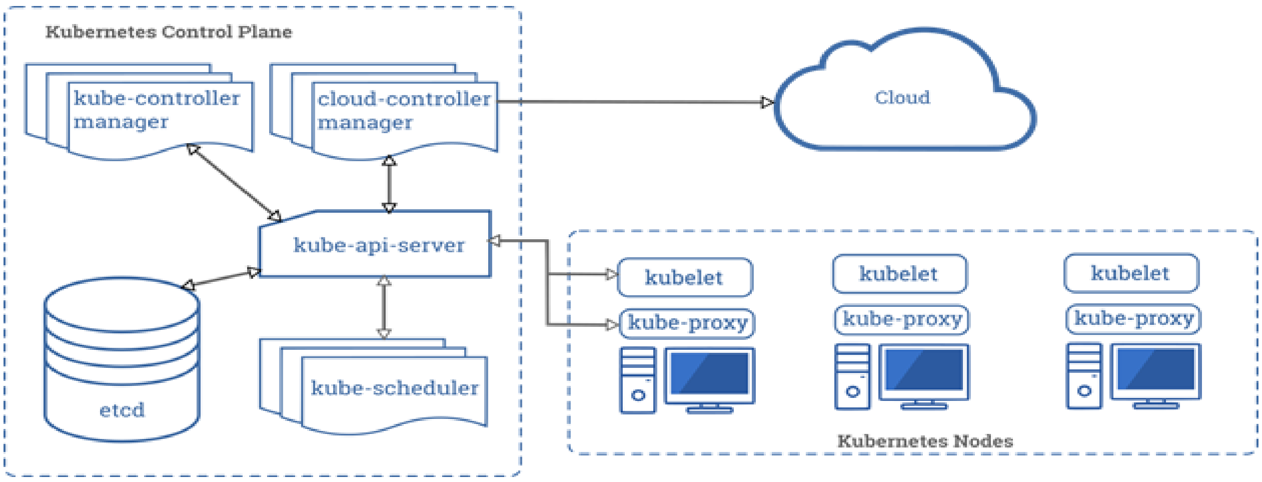
2.Etcd

3.kubelet

4.scheduler

5.controller

6.Container runtime



1.API Server (application interface programming Server)

It Acts as the front-end server for kubernetes(API sever is the only way we can talk with kubernetes)

there is command called kubectl by this we can talk to API severs in Yaml

here we can create multiple api server.(odd number)

2.Etcd

Etcd is a distributed reliable key value store to store all data to manage cluster

it is rap database, all information will be saved

3.kubelet

Kubelet is an agent to runs on each node of a cluster, Responsible to make sure containers are running in the pod.

4.scheduler

Scheduler is to distribute the work across all nodes or containers and assign pods to the node

5.Kube controller manager

Main brain behind the kubernetes. If some any containers go down automatically creates new container

6.Cloud Controller Manager

It will embed cloud-specific logic, it is used when we use in cloud platform

if you're working on on-premise this service will be not used

7.Kube-Proxy

it will maintain the network rules and nodes

8.Container runtime

Container run time in our case it is docker

Different ways to Install kubernetes

1.Development environment(minikube)

2.production environment(Kubernetes Operations(KOPS))

2.production environment(Kubernetes Operations(KOPS))

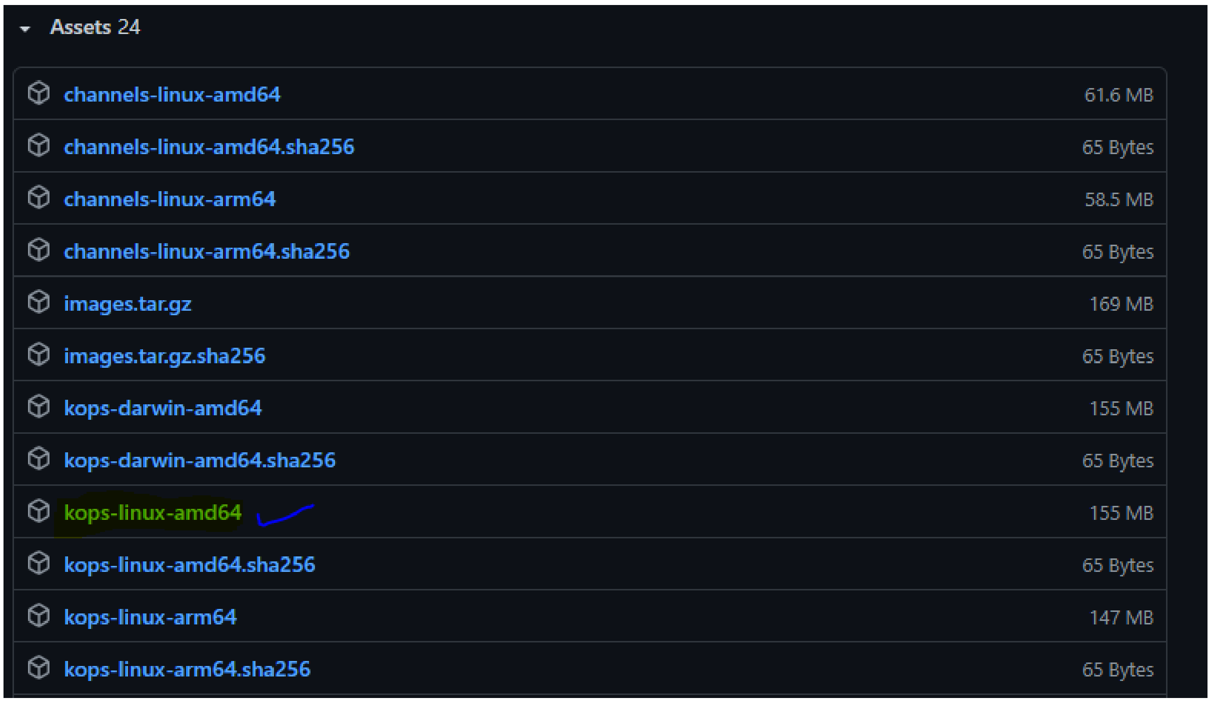
First, we need to create Domain Name and link to Route 53

create s3 bucket

Create one Mgmt Server(Ec2 instance)-t2.micro is enough and enter into the server

let's go to https://github.com/kubernetes/kops/releases

Installation of kops



Assets 24

channels-linux-amd64	61.6 MB
channels-linux-amd64.sha256	65 Bytes
channels-linux-arm64	58.5 MB
channels-linux-arm64.sha256	65 Bytes
images.tar.gz	169 MB
images.tar.gz.sha256	65 Bytes
kops-darwin-amd64	155 MB
kops-darwin-amd64.sha256	65 Bytes
kops-linux-amd64	155 MB
kops-linux-amd64.sha256	65 Bytes
kops-linux-arm64	147 MB
kops-linux-arm64.sha256	65 Bytes

copy the link address

```
wget <link>
chmod 700 kops-linux-amd64
mv kops-linux-amd64 kops
mv kops /user/local/bin/
kops version
```

or we can go to latest version also

Installation of Kubectl

download kubctl from browser

```
chmod 700 kubectl
mv kubectl /user/local/bin/
kubectl version
```

here KOPS is for k8s cluster Mgmt and KUBECTL for k8s Cluster ops

```
ssh-keygen
apt update && apt install unzip -y
```

AWS Cli Installation

next search in browser aws cli install ubuntu aws –version

Now let’s go to Aws account and go to IAM and create a user and

provide programmatical access and provide admin access (or search in internet for the permissions)

and copy the access key and Secret access key

now let’s back to ec2 server and

aws configure

and enter access key and Access key and us-east-1 and json

it will show all the buckets from this we will know ur connected to aws account

Creation of cluster, Master and nodes

kops create cluster –name=<route 53 domain> --state=s3://<bucket name> --zone=us-east-1a –node-count=2 --node-size=t2.micro –master-size=t2.small –master-volume-size 20 –node-volume-size 10 –dns-zone= <route 53 domain> --yes

here it will create the cluster with the specific given requirements here at the end of the cmd if we didn’t give any yes. it will not deploy cluster

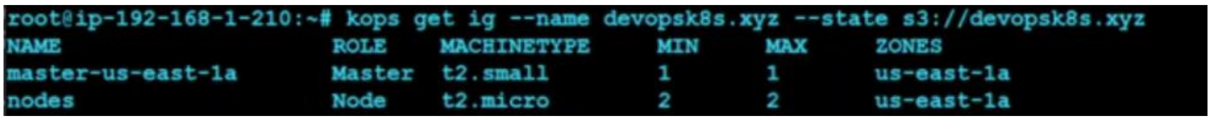
kops delete cluster --name=<route 53 domain> --state=s3://<bucket name> --yes

this will the delete the cluster, here we need to provide name and state

kops get cluster --state=s3://<bucket name>

it will show the cluster name and cloud and zones

kops get ig –name <route 53 domain> --state s3://<bucket name>



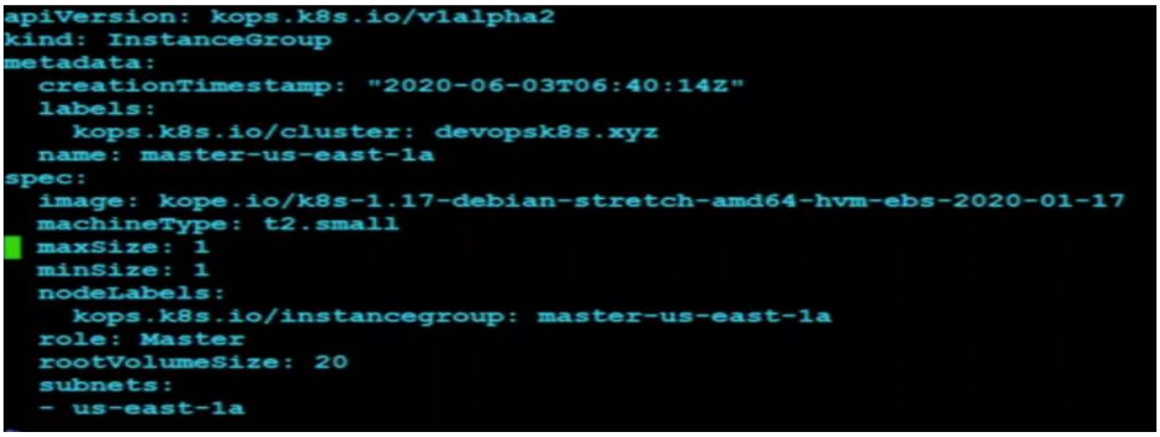
```
root@ip-192-168-1-210:~# kops get ig --name devopsk8s.xyz --state s3://devopsk8s.xyz
NAME                    ROLE    MACHINETYPE    MIN    MAX    ZONES
master-us-east-1a       Master  t2.small       1      1      us-east-1a
nodes                   Node    t2.micro       2      2      us-east-1a
```

it will show the how many masters and nodes are running and which type of instance type it is using

here ig =instance group

kops edit ig –name<route 53 domain> <master name> --state=s3://<bucket name>

here we can edit the master… here we need to give master name as we get in above screenshot, when we execute this cmd all the content of master will be opened and we can modify



```
apiVersion: kops.k8s.io/v1alpha2
kind: InstanceGroup
metadata:
  creationTimestamp: "2020-06-03T06:40:14Z"
  labels:
    kops.k8s.io/cluster: devopsk8s.xyz
  name: master-us-east-1a
spec:
  image: kope.io/k8s-1.17-debian-stretch-amd64-hvm-ebs-2020-01-17
  machineType: t2.small
  maxSize: 1
  minSize: 1
  nodeLabels:
    kops.k8s.io/instancegroup: master-us-east-1a
  role: Master
  rootVolumeSize: 20
  subnets:
  - us-east-1a
```

here we can edit all the data e.g.: how many master you need, how much volume you need….

kops edit ig –name<route 53 domain> <node name> --state=s3://<bucket name>

here we can edit nodes

kops update cluster –name <route 53 domain> --yes –state s3://<s3 bucket name>

after editing we need to update the cluster to make a changes deployed

vi .bashrc

    export NAME=devopsk8s.xyz

    export KOPS_STATE_STORE=s3://<s3 bucket name>

    alias <any name>='kubectl'

source .bash

here we need to go to bottom of page add this steps, because if we add these here it will take kops name and state will be default, no need to specify every time in the cmd like before and enter source .bash to execute this

kops get cluster

from here no need of providing name and state because we added as the default

kops validate cluster

it will show weather cluster is ready or not

ssh -i .ssh/id_rsa admin@<Ip address of master or node>

from here we can connect to master or node through ssh connection

cat .kube/config

it will show all the information of cluster

Creating a Basic pod

kubectl run <pod name> --image=index.docker.io/<repository name>/<image name>:v1

it will create the pod by giving repository name and image from docker hub here index.docker.io = docker hub

here run cmd will process deployment and we can add --replicas=3(it will create replicas)

kubectl get pods

here it will show all pods

kubectl describe pod <podname>

it will give detail description of a pod and what is happening in background

kubectl get pod <podname> -o yaml

it will show detail description of a pod with ip address, restarts, status….etc. in yaml format

Kubelet expose pod <podname> --port=8000 --target-port=80 --type=Nodeport

it will expose the node to specific port

kubectl get svc

it will show the node on which port it is exposed

kubectl edit svc <Node Name>

it will edit service of specific node

kubectl delete svc <Node Name>

it will delete service for specific port

kubectl get all

it will the list of pod

kubectl exec -it <podname> -- ls -al

it will show all directories in pod

kubectl exec -it <podname> -- bash

we can enter into the specific pod

kubectl delete pod <podename1> <podname2>

it will delete the specific pods

kubectl create -f pod.yaml

it will create the pod with the specific yaml file content

kubectl apply -f pod.yaml

if any changes to be done on the created pod we need to give apply in the cmd

kubectl ns <name>

it will create a namespace for the given

kubectl get pods -l env=dev

here in yaml we can mention labels, e.g.; env=dev,prod…etc. so while executing the cmd we can mention the label name(env=dev) it will sort the specific pods with that label

kubectl get pods --show-labels

it will all labels of every pod

kubectl label node <nodename> app-

here it will remove the label of the pod, If you're running a replicas then a new replica will be created

kubectl get rc

it will show the replication controller pods

kubectl delete rc <ReplicationControllerName>

it will delete the all pods within the Replication controller name

kubectl expose rc <ReplicationControllerName> --port=8000 --target-port=80 --type=NodePort

it will expose the replication controller to specific port

kubectl expose deploy <DeployName> --port=8000 --target-port=80 --type=NodePort

it will expose the Deploy to specific port

kubectl rolling-update <ReplicationControllerName> <New Name> --image=index.docker.io/sreeharshav/testcontainer:v1

here rolling-update is for changing the version of application without any downtime… it will upgrade one by one pod

kubectl rolling-update <ReplicationControllerName> <New Name> --rollback

here it will take back to old version

Deployment Advantages:

1. it is uses replicasets and replicasets automatically perform the rollingupdates
2. Rollback is easy as we can record the deployments
3. we can use liveness and readiness probes to improve application availability.
4. We can Pause & resume the deployment which useful canary update.

kubectl set image deployment <Deployment Name> nginx=<repository name>/<image name>:v1

this will replace our deployed image to the new image, but it was not best practice we need to change image in yaml

kubectl describe deploy <Deployment Name>

it will all details of deployment… eg: image used for deployment

kubectl scale deployment <deployment Name> --replicas=5

it will increase the replicas

kubectl rollout history deployments <deployment Name> --revision=2 here we should give which revision…. so it will show the changes done in the specific revision

Rolling Update

```
---
  apiVersion: extentions/v1beta1
  kind: Deployment
  metadata:
    name: nginx-deployment
    labels:
      app: nginx
  spec:
    minReadySeconds: 20
    replicas: 3
    selector:
      matchLabels:
        maxsurge: 1
        maxUnavailable: 0
      type: RollingUpdate
    template:
      metadata:
        labels:
          app: nginx
      specs:
        container:
        - name: nginx
          image: sreeharshav/testcontainer:v1
          ports:
          - containerPort: 80

---
  kind: Service
  apiVersion: v1
  metadata:
    name: myservice
  spec:
    selector:
      app: nginx
    type: NodePort
    ports:
    - name: name-of-the-port
      port: 8000
      targetPort: 80
```

Probes: Health checks

1.Rediness Probe => check the POD is ready to service the requests then only provide traffic

2.Liveness Probe => Check the POD is responding, if not restart it here KUBELET will perform the probes.

```
      specs:
        container:
        - name: nginx
          image: nagasiva/testcontainer:v1
          ports:
          - containerPort: 80
          readinessProbe:
            initialDelaySeconds: 30
            PeriodSeconds: 5
            timeoutSeconds: 10
            successThreshold: 1
            failureThreshold: 3
            httpGet:
              path: /
              Port: 80
          livenessProbe:
            initialDelaySeconds: 60
            PeriodSeconds: 5
            timeoutseconds: 10
            successThreshold: 1
            failureThreshold: 1
            httpGet:
              path: /
              Port: 80
```

kubectl get events

it will show all the updates done on the node or pod and it will show where readiness and liveness failed

Types of services

here we have three services:

i. Node port
ii. Cluster Ip
iii. None(Headless Service)
iv. Load Balancer
v. ingress

I. **Nodeport**: It is a service which exposes the node directly, it can be accessed from internet (It is to communicate with Single node)
II. **Cluster IP**: It will enable the communication between node to node internally within the cluster, it will be not exposed to internet, by default if we create service cluster ip will be taken (from frontend application to backend)

III. **None(Headless Service)**: it is service which it can given to one backend pod like databases, it is used when you don’t need load balancer(it is not exposed to internet)
IV. **Load Balancer**: A Load Balancer service is the standard way to expose a service to the internet, but it can be used for only one service
V. **Ingress**: it is similar to load Balancer but it is used for Different Services

NodePort:

kubectl expose pod <pod name> --port=8000 --target-port=80 --type=nodeport

```yaml
apiVersion: v1
kind: Service
metadata:
  name: mysql
  labels:
    name: mysql
spec:
  type: NodePort
  ports:
    - port: 3036
      nodePort: 30036
      name: http
  selector:
    name: mysql
```

Cluster IP:

here no need of giving “type” by default it will take as Custer IP

kubectl expose deploy <deployment name> --port=8000 --target-port=80

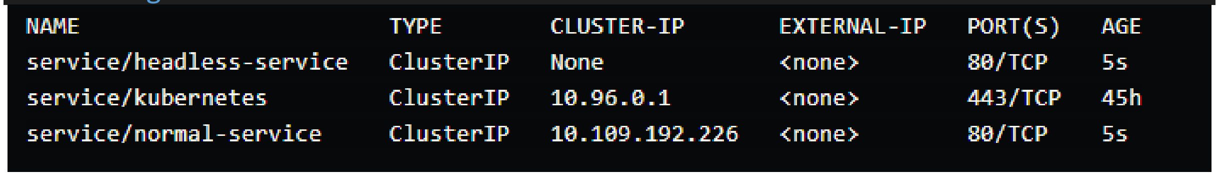
```yaml
apiVersion: v1
kind: Service
metadata:
  name: nginx-service
spec:
  selector:
    app: webserver-nginx
  ports:
    - protocol: TCP
      port: 80
      targetPort: 8080
```

None(Headless Service):

kubectl expose <deployment name> --port=8000 --target-port=80 --cluster-ip=None

```yaml
apiVersion: v1
kind: Service
```

```
metadata:
  name: headless-service
spec:
  clusterIP: None # <-- Don't forget!!
  selector:
    app: api
  ports:
    - protocol: TCP
      port: 80
      targetPort: 3000
```



```
NAME                       TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)   AGE
service/headless-service   ClusterIP   None            <none>        80/TCP    5s
service/kubernetes         ClusterIP   10.96.0.1       <none>        443/TCP   45h
service/normal-service     ClusterIP   10.109.192.226  <none>        80/TCP    5s
```

Load Balancer:

kubectl expose deploy <deployment name> --port=8000 --target-port=80 --type=LoadBalancer

```
apiVersion: v1
kind: Service
metadata:
  name: my-cip-service
spec:
  type: ClusterIP
  selector:
    app: metrics
    department: sales
  ports:
  - protocol: TCP
    port: 80
    targetPort: 8080
```

Ingress:

we need to create NGINX Ingress Controller

```
$ kubectl apply -f https://raw.githubusercontent.com/kubernetes/ingress-ngi
nx/controller-v0.44.0/deploy/static/provider/cloud/deploy.yaml
```

```
 apiVersion: networking.k8s.io/v1

kind: Ingress
metadata:
  name: simple-fanout-example
spec:
  rules:
  - host: foo.bar.com
  http:
    paths:
    - path: /foo
      pathType: Prefix
      backend:
        service:
          name: service1
          port:
            number: 4200
```

kubectl apply -f <.yaml>

it will apply the service

kubectl get ing

it will show all ingress services

Name Spaces:

- Allowing teams or projects to exist in their own virtual clusters without fear of impacting each other's work.
- Enhancing role-based access controls (RBAC) by limiting users and processes to certain namespaces.
- Enabling the dividing of a cluster's resources between multiple teams and users via resource quotas.
- Providing an easy method of separating development, testing, and deployment of containerized applications enabling the entire lifecycle to take place on the same cluster.

First need to download kubens from internet

https://github.com/ahmetb/kubectx/releases

take the latest version

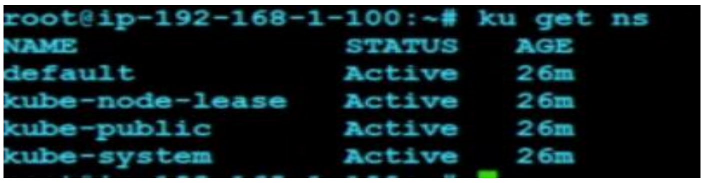
and unzip the file

mv kubens /usr/local/bin/

move the folder to specific directory

kubectl get ns

It will show all namespaces list



```
root@ip-192-168-1-100:~# ku get ns
NAME              STATUS   AGE
default           Active   26m
kube-node-lease   Active   26m
kube-public       Active   26m
kube-system       Active   26m
```

if you didn’t create any namespace it will go to default namespace

kubectl create ns <namespace name>

it will create a namespace with the given name

kubectl delete ns <namespace name>

it will delete the namespace

kubens <Name of Namespace>

this command will convert given specific namespace to Default

kubectl apply -f pod.yaml --namespace=test

It will create pod with the specific test namespace

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: mypod
  namespace: test
  labels:
    name: mypod
spec:
  containers:
  - name: mypod
    image: nginx
```

kubectl describe ns <name>

It will show all details of the namespace like-status, resource quota, limit range resource.

ResorceQuata:

It is for limiting the total resource consumption of a namespace

limit range:

It is for managing constraints at a pod and container level within the project

ResorceQuata:

```yaml
apiVersion: v1
kind: ResourceQuota
metadata:
  name: compute-resources
spec:
  hard:
    pods: "4"
    requests.cpu: "1"
    requests.memory: 1Gi
    limits.cpu: "2"
    limits.memory: 2Gi
```

kubectl apply -f <yaml file of Resorcequotas> --namespace=<Name of namespace>

it will create the Resorcequota for the specific namequotas

kubectl get resorcequotas

it will show all resorcequotas created

kubectl delete resorcequotas <name>

it will delete the specific quota

limit range:

```yaml
apiVersion: "v1"
kind: "LimitRange"
metadata:
  name: "core-resource-limits"
spec:
  limits:
    - type: "Pod"
      max:
        cpu: "2"
        memory: "1Gi"
      min:
        cpu: "200m"
        memory: "6Mi"
    - type: "Container"
      max:
        cpu: "2"
        memory: "1Gi"
      min:
        cpu: "100m"
        memory: "4Mi"
      default:
        cpu: "300m"
        memory: "200Mi"
      defaultRequest:
        cpu: "200m"
        memory: "100Mi"
      maxLimitRequestRatio:
        cpu: "10"
```