Kubernetes:

Orchestration framework and containers can be distributed into multiple nodes.

Components of this framework:

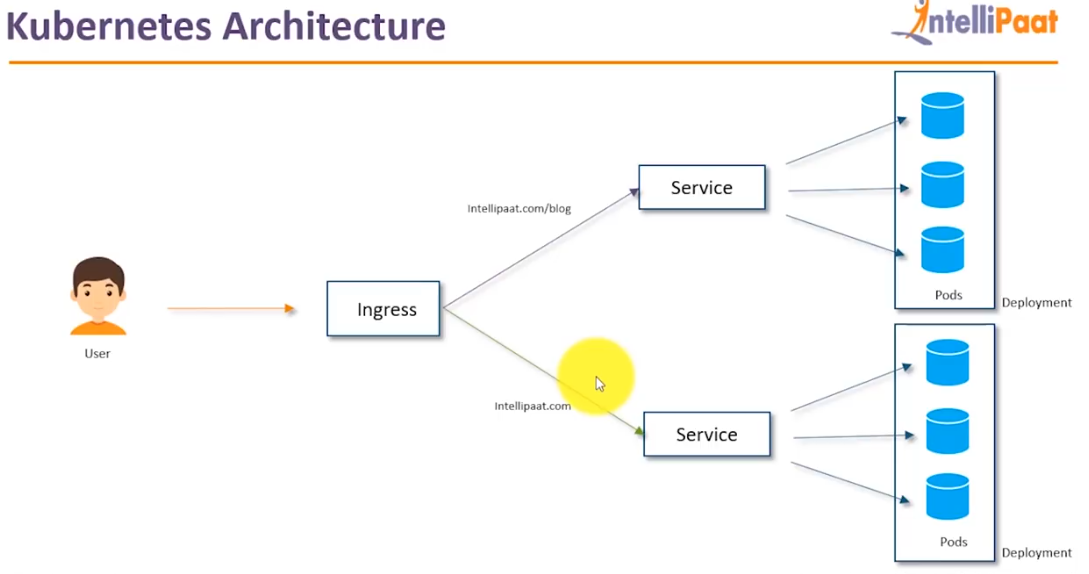
1. K8s master node
2. K8s worker node
3. dashboard
4. client (UI and CLI)

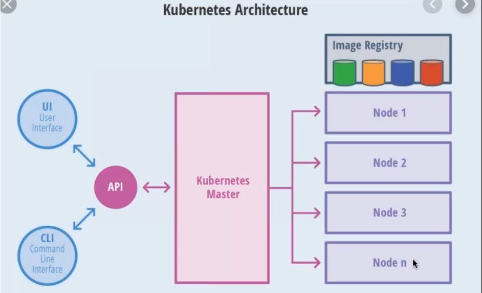
Out of k8s, docker swarm and messos, k8s is most popular.

we have different methods:

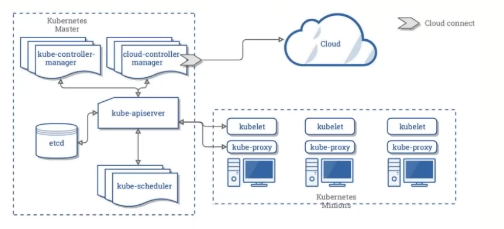
1. Opensource
2. AWS EKS
3. Azure AKS
4. GCP GKE
5. Redhat openshift

Kubernetes Architecture:





Each and every node will act like proxy servers.



Kube controller controls the services. Also it will monitor the containers whether they are up and running.

Kube scheduler will distribute tasks to the worker nodes.

Kube api server will take inputs as commands to start pods. These commands will be taken by api server and assign it to kube controller manager and kube controller manager will perform those tasks.

etcd will maintain key value pairs or certificates. Authentication and security.

The kubelet is responsible for maintaining a set of pods, which are composed of one or more containers, on a local system. Within a Kubernetes cluster, the kubelet functions as a local agent that watches for pod specs via the Kubernetes API server.

kube-proxy is a network proxy that runs on each node in your cluster, implementing part of the Kubernetes Service. concept. kube-proxy maintains network rules on nodes. These network rules allow network communication to your Pods from network sessions inside or outside of your cluster.

cloud controller manager used to connect with cloud environment and it comes in enterprise edition.

kubectl is the kubernetes utility which is used to interact with kubernetes master to create pod and execute services

**kubeadm** is like admin to initialize k8s master, wn.

**kubeadm init** is used to initialize k8s control plane

**kubeadm join** is used to join the k8s wn to master.

**kubeadm init** if you don’t have 2 core cpu then we get preflight errors.so use **kubeadm init --ignore-preflight-errors all**

First we install kubernetes utilities like kubectl,kubeadm,kubelet and kubernetes-cni. After that we use these utilities to setup k8s control plane.

once we issue kube init we get below instructions to execute

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

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 172.31.90.184:6443 --token mglt68.8mgum4kxc7252cr2 \

--discovery-token-ca-cert-hash sha256:eec414242a8d6d0ff892dea37f2674f0bf3defa8483ae01d8cf242431a6821b1 **===================================================**

Now we need to deploy pod network to the cluster. There are different pod networks available.

As given use below url to find different pod network (Networking and networking policy)

<https://kubernetes.io/docs/concepts/cluster-administration/addons/>

Now choose any pod network like calico and deploy the network to the cluster using below command

**kubectl apply -f https://docs.projectcalico.org/v3.10/manifests/calico.yaml**

once done issue below command to get all pods in network

**kubectl get pods --all -namespaces**

Now we need to initiate kubernetes dashboard by below command:

|  |  |  |
| --- | --- | --- |
|  | | |
| **kubectl apply -f** [**https://raw.githubusercontent.com/kubernetes/dashboard/v1.10.1/src/deploy/recommended/kubernetes-dashboard.yaml**](https://raw.githubusercontent.com/kubernetes/dashboard/v1.10.1/src/deploy/recommended/kubernetes-dashboard.yaml)  **We have complexity here where in v1 skip option is disabled for security reasons. So while initiating kubernetes dashboard instead of kubectl apply -f** [**https://raw.githubusercontent.com/kubernetes/dashboard/v1.10.1/src/deploy/recommended/kubernetes-dashboard.yaml**](https://raw.githubusercontent.com/kubernetes/dashboard/v1.10.1/src/deploy/recommended/kubernetes-dashboard.yaml) **create a file called** kubernetes-dashboard.yaml file and paste the contents of the url [**https://raw.githubusercontent.com/kubernetes/dashboard/v1.10.1/src/deploy/recommended/kubernetes-dashboard.yaml into the file** kubernetes-dashboard.yaml](https://raw.githubusercontent.com/kubernetes/dashboard/v1.10.1/src/deploy/recommended/kubernetes-dashboard.yaml%20into%20the%20file%20kubernetes-dashboard.yaml) and edit the following in vi editor  Locate the container -> args section under the Dashboard-Deployment section and add the following command line arguments:  --enable-skip-login  --disable-settings-authorizer  So final looks like below  spec:        containers:        - name: kubernetes-dashboard          image: k8s.gcr.io/kubernetes-dashboard-amd64:v1.10.1          ports:          - containerPort: 8443            protocol: TCP          args:            - --enable-skip-login            - --disable-settings-authorizer            - --auto-generate-certificates            # Uncomment the following line to manually specify Kubernetes API server Host            # If not specified, Dashboard will attempt to auto discover the API server and connect            # to it. Uncomment only if the default does not work.            # - --apiserver-host=http://my-address:port          volumeMounts:          - name: kubernetes-dashboard-certs            mountPath: /certs  save it and execute below  **kubectl apply -f ./kubernetes-dashboard.yaml** it will create dashboard and  Now check whether dashboard is initialized using below command  **Kubectl get pods --all -namespaces**  Initialize the proxy server using below command  **kubectl proxy --address 0.0.0.0 --accept-hosts '.\*' &**  Access the dashboard using below url  **http://<ip>:8001/api/v1/namespaces/kube-system/services/https:kubernetes-dashboard:/proxy/#!/overview?namespace=default**  **By doing above we will get skip option else we don’t get skip option.**  To authenticate create service account and bind cluster admin role to the service account:  **kubectl create serviceaccount cluster-admin-dashboard-sa** | | |
| **kubectl create clusterrolebinding cluster-admin-dashboard-sa \** | | |
|  | | **--clusterrole=cluster-admin \** |
|  | | **--serviceaccount=default:cluster-admin-dashboard-sa** |
|  | | |

Once done we need to get the token

**TOKEN=$(kubectl describe secret $(kubectl -n kube-system get secret | awk '/^cluster-admin-dashboard-sa-token-/{print $1}') | awk '$1=="token:"{print $2}')**

**echo $TOKEN > token**

**cat token**

**If token to join the nodes to master is lost use below command to get the token back.**

**kubeadm token create --print-join-command**

**========================================**

**Client configurations:**

create directory

**mkdir .kube**

under .kube create config

**cd .kube/**

**vi config**

Now go to master:

copy the contents on config to client config

**cat .kube/config**

pod contains multiple containers

To start pods:

**kubectl run ngnix-ex1 --image=ngnix --port=80 --env=”DOMAIN=Cluster”**

To expose as service:

**kubectl expose deployment nginx-ex1 --port=80 --name=nginxapp1 --type=LoadBalancer**

type can be ClusterIP, NodePort, LoadBalancer, or ExternalName. Default is 'ClusterIP'

rolling updates:

during patching or version upgrade if you have 10 pods we cannot upgrade all 10 at a time then application will go down if done. So in order to avoid this we go for rolling updates where on pods in taken upgraded and put back to cluster, like that we can do for all 10 pods.