1 - controller

2 - working nodes with red hat

step 1 -- install docker

step 2 -- install kunernetes

#####################----Containerd Installation---------- ############################

#Load Kernel modules at system startup

cat <<EOF | sudo tee /etc/modules-load.d/containerd.conf

overlay

br\_netfilter

EOF

# These normally reflect post server restart

# To reflect them immediately withour restarting the server

sudo modprobe overlay

sudo modprobe br\_netfilter

# Now set the kernel properties for the Kubernetes networking

cat <<EOF | sudo tee /etc/sysctl.d/99-kubernetes-cri.conf

net.bridge.bridge-nf-call-iptables = 1

net.ipv4.ip\_forward = 1

net.bridge.bridge-nf-call-ip6tables = 1

EOF

# Now to Apply these properties without reboot the system

sudo sysctl --system

sudo systemctl restart systemd-modules-load.service

# Install pre-requisites

sudo apt-get update && sudo apt-get install -y apt-transport-https ca-certificates curl gnupg lsb-release

# Install containerd

sudo mkdir -p /etc/apt/keyrings

curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o /etc/apt/keyrings/docker.gpg

echo \

"deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker.gpg] https://download.docker.com/linux/ubuntu \

$(lsb\_release -cs) stable" | sudo tee /etc/apt/sources.list.d/docker.list > /dev/null

sudo apt-get update

sudo apt-get install -y containerd.io

# Create default configuration file for containerd:

sudo mkdir -p /etc/containerd

# Generate default containerd configuration and save to the newly created default file:

sudo containerd config default | sudo tee /etc/containerd/config.toml

sudo sed -i 's/SystemdCgroup = false/SystemdCgroup = true/' /etc/containerd/config.toml

# Restart containerd to ensure new configuration file usage:

sudo systemctl enable containerd

sudo systemctl restart containerd

#--------------------------------------------

# Install Kubernetes

# Disable swap:

sudo swapoff -a

# Install dependency packages:

sudo apt-get update && sudo apt-get install -y apt-transport-https curl

#Download and add GPG key:

curl -s https://packages.cloud.google.com/apt/doc/apt-key.gpg | sudo apt-key add -

# Add Kubernetes to repository list:

cat <<EOF | sudo tee /etc/apt/sources.list.d/kubernetes.list

deb https://apt.kubernetes.io/ kubernetes-xenial main

EOF

# Update package listings:

sudo apt-get update

# Install Kubernetes packages (Note: If you get a dpkg lock message, just wait a minute or two before trying the command again):

sudo apt-get install -y kubelet kubeadm kubectl

# Turn off automatic updates:

sudo apt-mark hold kubelet kubeadm kubectl

# Initialize the Cluster

# Initialize the Kubernetes cluster on the control plane node using kubeadm (Note: This is only performed on the Control Plane Node):

#kubeadm init --pod-network-cidr 192.168.0.0/16

-------------------------------------------------------------------------------------------------> after initize the kubernites exit from root by $ exit

in home/ubuntu - copy and paste from above

# Install the Calico Network Add-On

# Only the control plane node, install Calico Networking:

#kubectl create -f https://raw.githubusercontent.com/projectcalico/calico/v3.26.1/manifests/tigera-operator.yaml

#curl https://raw.githubusercontent.com/projectcalico/calico/v3.26.1/manifests/custom-resources.yaml -O

#kubectl create -f custom-resources.yaml

#watch kubectl get pods -n calico-system

Run "kubectl apply -f [podnetwork].yaml" with one of the options listed at:

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

kubeadm join 172.31.1.47:6443 --token bqksvz.uj0hh5c8e35ryk5d \

--discovery-token-ca-cert-hash sha256:37fd88d05cb06fab227a8ee50cd7a2314c2328c255b21b38e79af14cc34d7c37

if the node showes not ready ----install some trivial CNI plugin, like flannel and resatrt kubelet

kubectl apply -f https://raw.githubusercontent.com/flannel-io/flannel/master/Documentation/kube-flannel.yml

if in worker nodes not joining to master--- reset the kubeadm

sudo kubeadm reset

Create a new token and join command to rejoin/add worker node

systemctl status kubelet

kubeadm token generate

kubeadm token create <generated token paste> --print-join-command

Now copy the join command output from the Master node and execute that to the worker node in root.

#################### EKS cluster creatation ##################################

Generate AWS CLI credentials

Run the below 3 commands on CLI

$ export AWS\_ACCESS\_KEY\_ID=< from aws>

$ export AWS\_SECRET\_ACCESS\_KEY=< from aws >

$ export AWS\_DEFAULT\_REGION=us-east-1

Install eksctl cli tool:

https://docs.aws.amazon.com/eks/latest/userguide/eksctl.html

https://github.com/eksctl-io/eksctl/releases/tag/v0.150.0

create a file named aws.pub in your home directory .ssh with the contents of authroized\_keys from .ssh directory.

This file should contain the public key of the pem file you use to login to aws ec2 instances.

You can find this inside any existing ec2 instance already created using the key at authroized\_keys from .ssh directory

install eksctl

https://docs.aws.amazon.com/eks/latest/userguide/eksctl.html

wget <link>

tar zxf eksctl\_Linux\_amd64.tar.gz

rm eksctl\_Linux\_amd64.tar.gz

sudo mv eksctl /usr/bin

eksctl version

once eksctl is installed, run the below command

eksctl create cluster -f cluster.yaml

Install Ingress Controller from the website instructions:

https://aws.amazon.com/premiumsupport/knowledge-center/eks-access-kubernetes-services/

To download the kubeconfig file for the EKS cluster:

aws eks update-kubeconfig --name <cluster name> --region us-east-1

Once completed, delete the eks cluster

eksctl delete cluster --region=us-east-1 --name=basic-cluster --force

############### create pods in namespace #############################

kudectl create namespace test

166 kubectl create namespace test

167 kubectl get namespace

168 vi testpod.yaml

169 kubectl create testpod.yaml -n test (pod file <test.pod>)

170 kubectl apply -f testpod.yaml -n test

171 vi testpod.yaml

172 kubectl apply -f testpod.yaml -n test

173 kubectl get pods

174 kubectl get pods -n test

175 cd replicaSet

176 vi test-relicaSet.yaml(replicaset file)

177 cd ..

178 kubectl apply -f test-relicaSet.yaml -n test

179 pwd

180 ls

181 cd replicaSet

182 ls

183 kubectl apply -f test-relicaSet.yaml -n test

184 kubectl get pods -n test

185 kubectl get pods -o wide test

186 kubectl get pods -o wide

187 kubectl get pods -n test -o wide

=> testpod.yaml file

# Section 1 - API Version

apiVersion: v1

# Section 2 - Which type of Object we want to create

kind: Pod

# Section 3 - Meta data of the object we are creating

metadata:

name: sample-test-pods (name of the pod )

labels:

app: sample

env: test

# Section 4 - The actual options we need for the object we want to create

spec:

containers:

- name: sample-test-container

image: nginx:latest

=>test-replicaSet.yaml file

apiVersion: apps/v1

kind: ReplicaSet

metadata:

name: test-app (name of replica pod )

spec:

replicas: 3

selector:

matchLabels:

app: sample-test-pods (original pod name )

template:

metadata:

labels:

app: sample-test-pods (original pod name )

spec:

containers:

- name: nginx-rs

image: nginx:latest

resources:

limits:

cpu: 200m

memory: "105Mi"

kubeadm join 172.31.47.87:6443 --token ugk9x9.3cuytxx7phra33dz \

--discovery-token-ca-cert-hash sha256:d5a2a9085b7604635aa41450ec108efd1de4bf3cb98ae5999efddaa33a1547e4

---------------------------------------------------------------------------------------------------------

--> Amazon EKS cluster consists of two primary components:

The Amazon EKS control plane

Amazon EKS nodes that are registered with the control plane

The Amazon EKS control plane consists of control plane nodes that run the Kubernetes software, such as etcd and the Kubernetes API server. The control plane runs in an account managed by AWS, and the Kubernetes API is exposed via the Amazon EKS endpoint associated with your cluster. Each Amazon EKS cluster control plane is single-tenant and unique, and runs on its own set of Amazon EC2 instances.

All of the data stored by the etcd nodes and associated Amazon EBS volumes is encrypted using AWS KMS. The cluster control plane is provisioned across multiple Availability Zones and fronted by an Elastic Load Balancing Network Load Balancer. Amazon EKS also provisions elastic network interfaces in your VPC subnets to provide connectivity from the control plane instances to the nodes (for example, to support kubectl exec logs proxy data flows).

------------------------------------------------------------------------------------------------------------------------

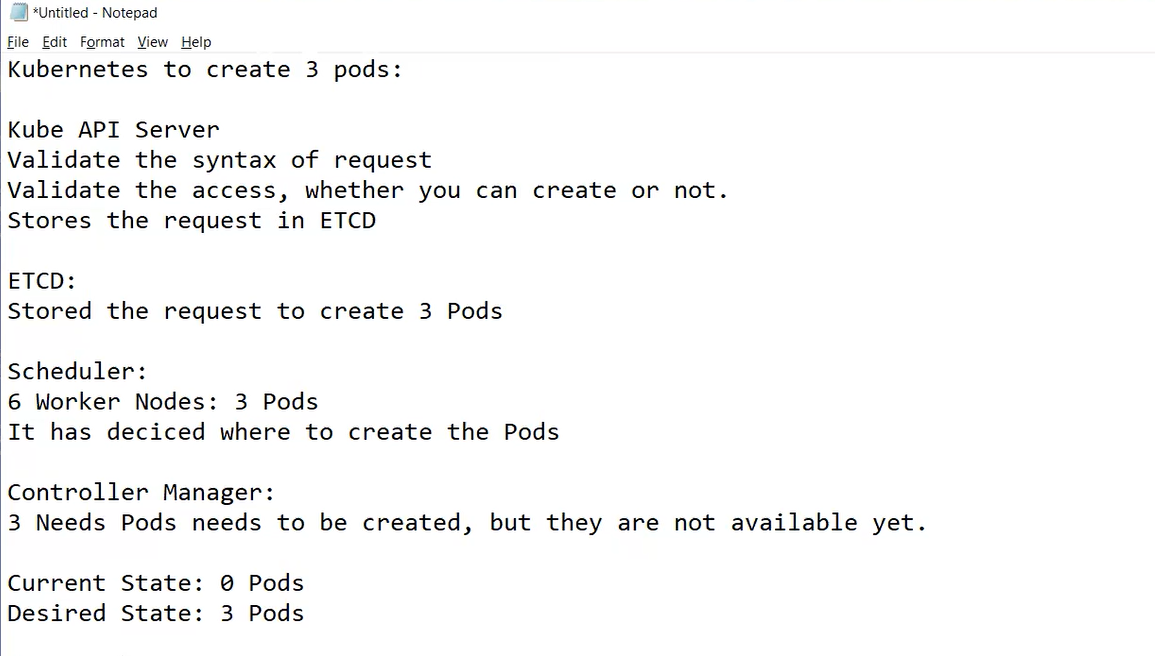
--> What is rolling update deployment?

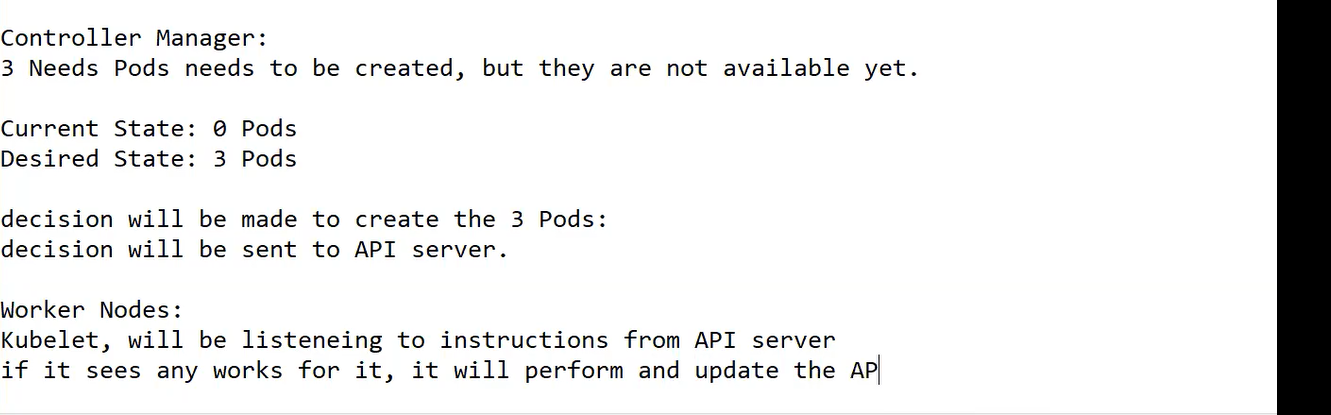
A rolling deployment is a deployment strategy that slowly replaces previous versions of an application with new versions of an application by completely replacing the infrastructure on which the application is running.

--> What is the Kubernetes rollout update strategy?

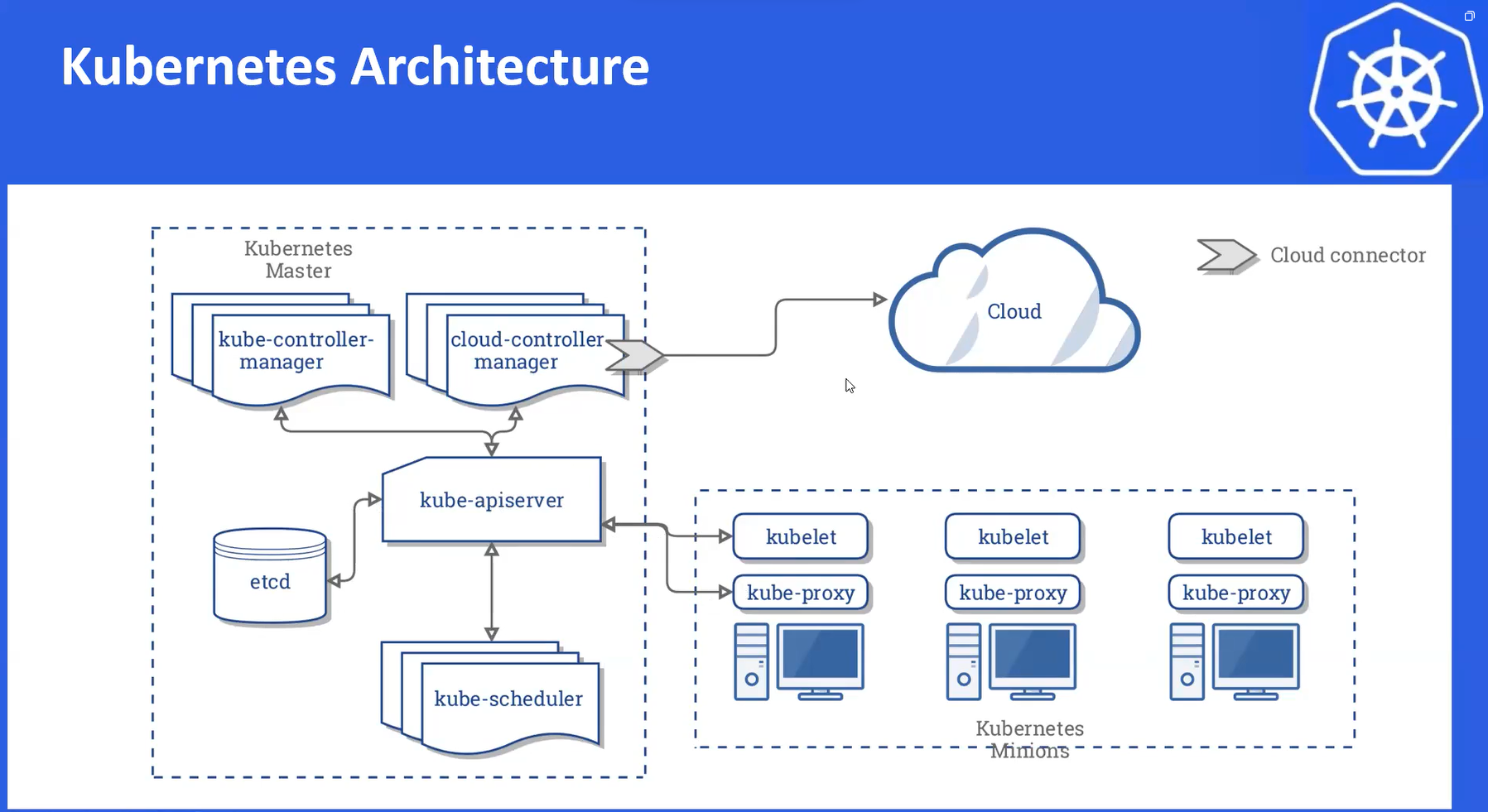
Rolling deployment is the default deployment strategy in Kubernetes. It lets you update a set of pods with no downtime, by incrementally replacing pod instances with new instances that run a new version of the application.

-----------------------------------------------------------------------------------------------------------------------------





Kubelet will create the pods.



Why we use docker in Kubernetes?

Kubernetes: it cannot create a container itself.it needs the help of a container runtime .

Pod vs container:

Kubernetes creates a abstraction on top of container , to manage it through Kubernetes and add all the features of Kubernetes to pod.

