# Kubernetes

Kubernetes in an open source container management tool hosted by Cloud Native Computing Foundation (CNCF). This is also known as the enhanced version of Borg which was developed at Google to manage both long running processes and batch jobs, which was earlier handled by separate systems.

Kubernetes comes with a capability of automating deployment, scaling of application, and operations of application containers across clusters. It is capable of creating container centric infrastructure.

Features of Kubernetes

Following are some of the important features of Kubernetes.

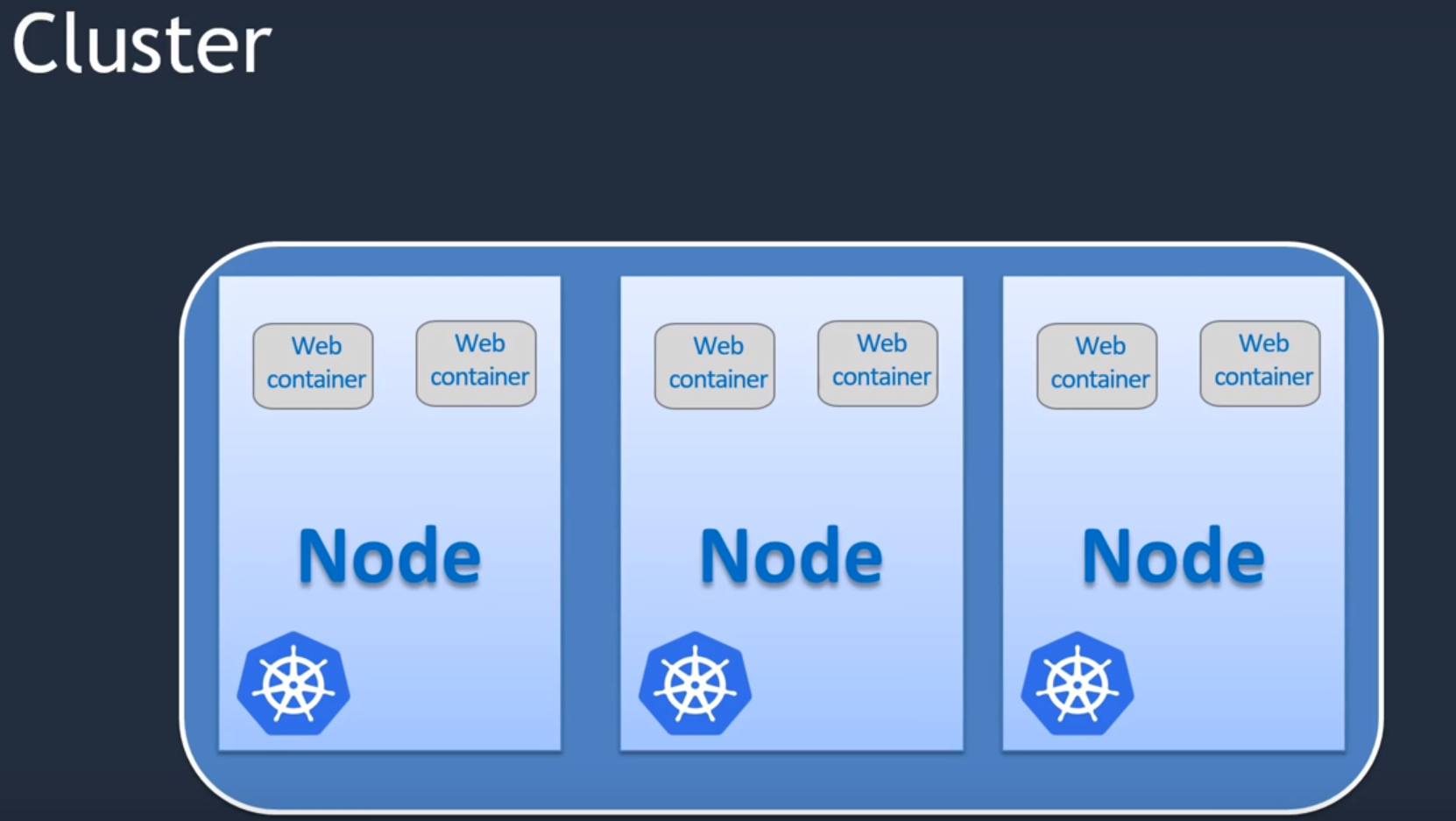
* Continues development, integration and deployment
* Containerized infrastructure
* Application-centric management
* Auto-scalable infrastructure
* Environment consistency across development testing and production
* Loosely coupled infrastructure, where each component can act as a separate unit
* Higher density of resource utilization
* Predictable infrastructure which is going to be created

One of the key components of Kubernetes is, it can run application on clusters of physical and virtual machine infrastructure. It also has the capability to run applications on cloud. **It helps in moving from host-centric infrastructure to container-centric infrastructure.**

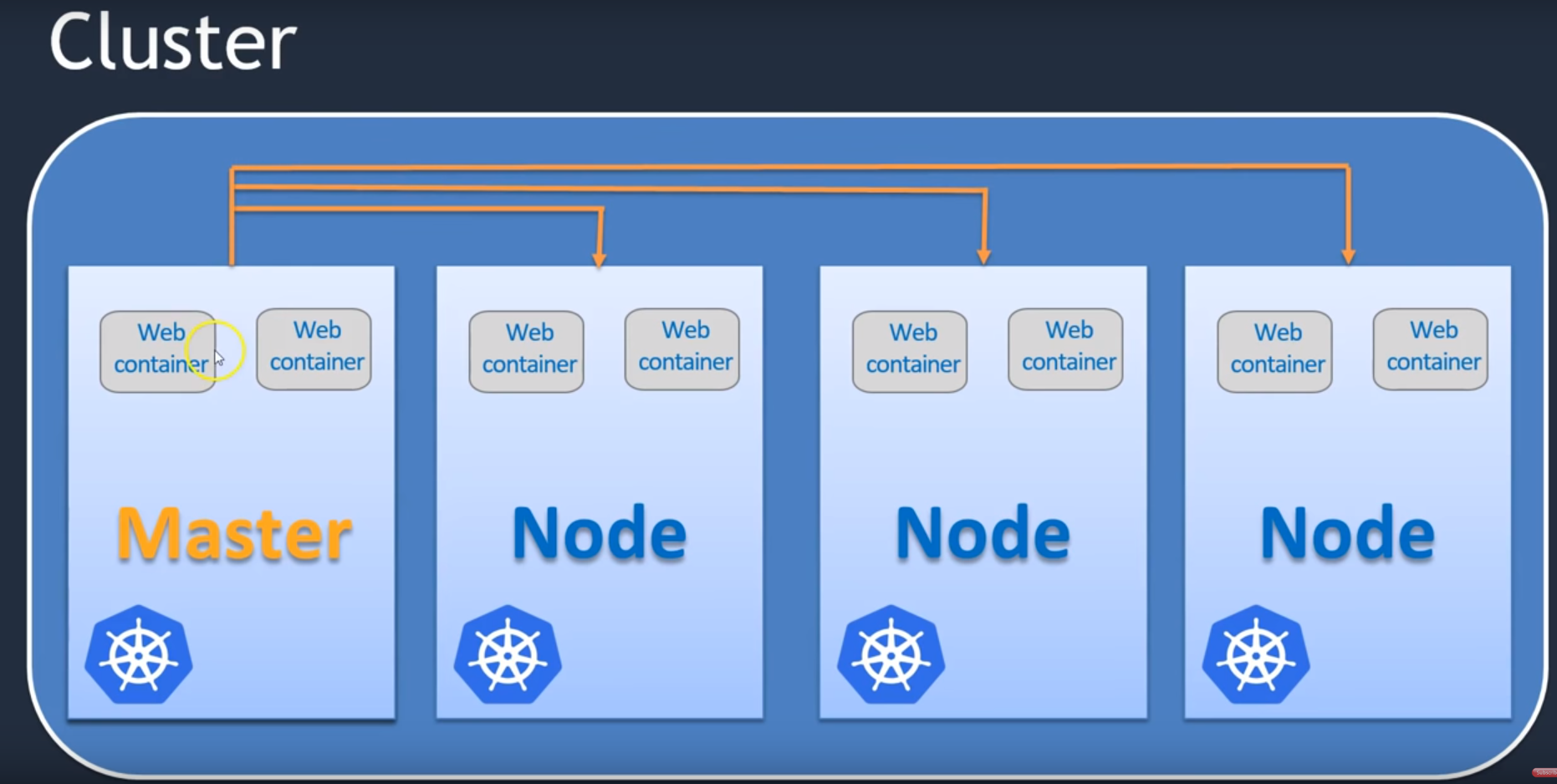
Kubernetes components | Kubernetes Master and Node components

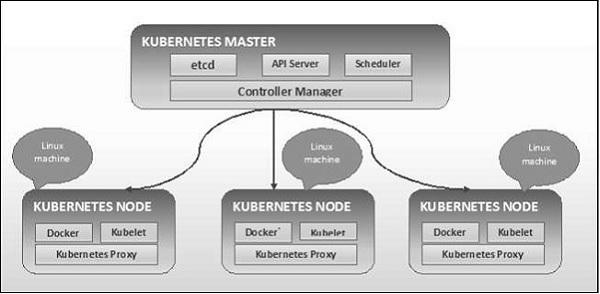
Earlier we used to have a Node which can run or hold containers- 

**Now Clustering is for High Availability**



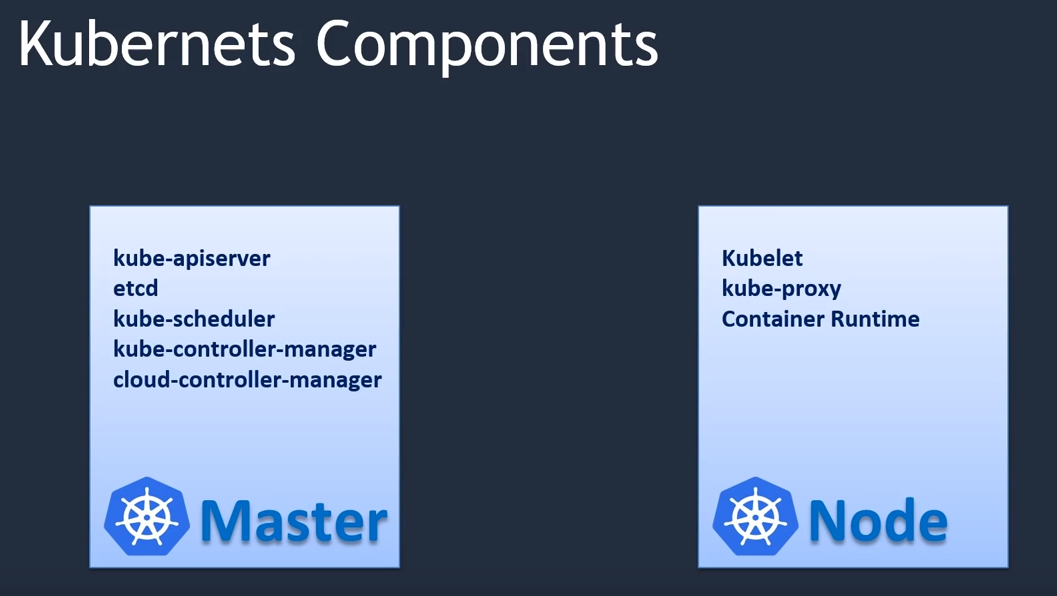
Now the question is **WHO IS GOING TO MANAGE THE CLUSTER. In below architecture, Cluster is managed by Master node**





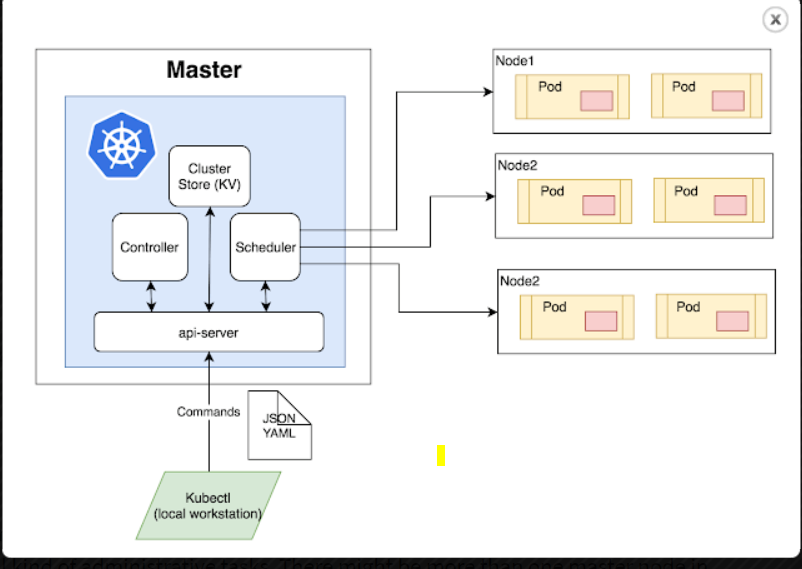
**Kubernetes Components –**

1. **KUBE-APISERVER –** front end for Kubernetes. User, management interface and CLI interact with apiserver to talk with Kubernetes cluster.
2. **ETCD –** distributed reliable, **key value store**. Kubernetes stores all the data in the etcd to manage the cluster. It will store information like **how many Master we have and how many worker we have**. All these are stored in ETCD. ETCD is available in all the node in the cluster/
3. **Kube scheduler –** actually distribute the work in the multiple containers which are there across the cluster.
4. **Kube-controller-manager –** To identify if there is any failure in the nodes; It will check if any manager and worker node is up and running.
5. **Kubelet –** this will run in all the nodes and it will make sure that all the containers are up and running.
6. **Kube-proxy** – this is at network level which is used to forward network connections to particular Service. **Kube-proxy**: The **Kube-proxy** is an implementation of a network proxy and a load balancer, and it supports the service abstraction along with other networking operation. It is **responsible** for routing traffic to the appropriate container based on IP and port number of the incoming request.
7. **Container Runtime** – underline software which is used to run our containers. In our case we are using **docker** for containers.



# Kubernetes Setup

Reference - <https://kubernetes.io/docs/setup/production-environment/tools/kubeadm/install-kubeadm/>



## **Create 2 servers in AWS using ubuntu image - Ubuntu Server 18.04 LTS**

Below Ports need to be opened while creating Kubernetes Cluster

### **Control-plane node(s)**

| Protocol | Direction | Port Range | Purpose | Used By |
| --- | --- | --- | --- | --- |
| TCP | Inbound | 6443\* | Kubernetes API server | All |
| TCP | Inbound | 2379-2380 | etcd server client API | kube-apiserver, etcd |
| TCP | Inbound | 10250 | Kubelet API | Self, Control plane |
| TCP | Inbound | 10251 | kube-scheduler | Self |
| TCP | Inbound | 10252 | kube-controller-manager | Self |

### **Worker node(s)**

| Protocol | Direction | Port Range | Purpose | Used By |
| --- | --- | --- | --- | --- |
| TCP | Inbound | 10250 | Kubelet API | Self, Control plane |
| TCP | Inbound | 30000-32767 | NodePort Services† | All |

## Install using kubeadm

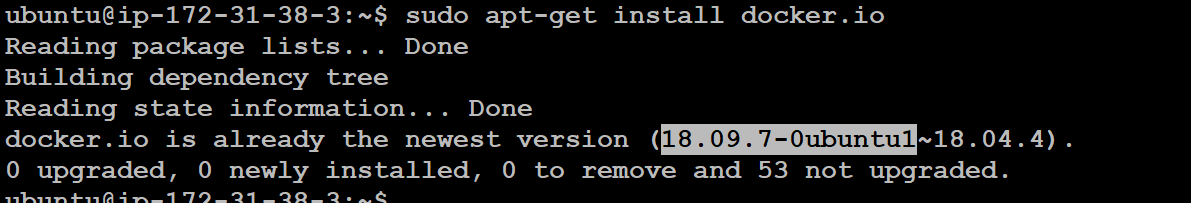
### **Install docker engine**

Run it in All the nodes.

sudo apt-get update ## Update the apt-get

sudo apt-get install docker.io -y ## Installing Docker Engine or **Container Runtime**

systemctl start docker # Start the Docker



### **Installing kubeadm, kubelet, kubectl**

You will install these packages on all of your machines:

**Kubeadm (Kubernetes administration tool)**: the command to bootstrap the cluster. Create the Cluster Manually

**kubelet**: the component that runs on all of the machines in your cluster and does things like starting pods and containers.

**Kubectl(Kubernetes Controller)**: the command line utility to talk to your cluster.

Follow This for Full reference - <https://v1-17.docs.kubernetes.io/docs/setup/production-environment/tools/kubeadm/create-cluster-kubeadm/>

**Run it in All the nodes.** Create a shell script and run it

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

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

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

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

EOF

sudo apt-get update

sudo apt-get install -y kubelet kubeadm kubectl

sudo apt-mark hold kubelet kubeadm kubectl

### Initiate the Master Node

root@ip-172-31-38-3:~# kubeadm init --pod-network-cidr=192.168.0.0/16 --apiserver-advertise-address=<your\_server\_private\_ip> --ignore-preflight-errors=NumCPU

W0308 07:53:56.603304 8542 validation.go:28] Cannot validate kube-proxy config - no validator is available

W0308 07:53:56.603497 8542 validation.go:28] Cannot validate kubelet config - no validator is available

[init] Using Kubernetes version: v1.17.3

[preflight] Running pre-flight checks

[WARNING NumCPU]: the number of available CPUs 1 is less than the required 2

[WARNING Service-Docker]: docker service is not enabled, please run 'systemctl enable docker.service'

[WARNING IsDockerSystemdCheck]: detected "cgroupfs" as the Docker cgroup driver. The recommended driver is "systemd". Please follow the guide at https://kubernetes.io/docs/setup/cri/

[preflight] Pulling images required for setting up a Kubernetes cluster

[preflight] This might take a minute or two, depending on the speed of your internet connection

[preflight] You can also perform this action in beforehand using 'kubeadm config images pull'

[kubelet-start] Writing kubelet environment file with flags to file "/var/lib/kubelet/kubeadm-flags.env"

[kubelet-start] Writing kubelet configuration to file "/var/lib/kubelet/config.yaml"

[kubelet-start] Starting the kubelet

[certs] Using certificateDir folder "/etc/kubernetes/pki"

[certs] Generating "ca" certificate and key

[certs] Generating "apiserver" certificate and key

[certs] apiserver serving cert is signed for DNS names [ip-172-31-38-3 kubernetes kubernetes.default kubernetes.default.svc kubernetes.default.svc.cluster.local] and IPs [10.96.0.1 172.31.38.3]

[certs] Generating "apiserver-kubelet-client" certificate and key

[certs] Generating "front-proxy-ca" certificate and key

[certs] Generating "front-proxy-client" certificate and key

[certs] Generating "etcd/ca" certificate and key

[certs] Generating "etcd/server" certificate and key

[certs] etcd/server serving cert is signed for DNS names [ip-172-31-38-3 localhost] and IPs [172.31.38.3 127.0.0.1 ::1]

[certs] Generating "etcd/peer" certificate and key

[certs] etcd/peer serving cert is signed for DNS names [ip-172-31-38-3 localhost] and IPs [172.31.38.3 127.0.0.1 ::1]

[certs] Generating "etcd/healthcheck-client" certificate and key

[certs] Generating "apiserver-etcd-client" certificate and key

[certs] Generating "sa" key and public key

[kubeconfig] Using kubeconfig folder "/etc/kubernetes"

[kubeconfig] Writing "admin.conf" kubeconfig file

[kubeconfig] Writing "kubelet.conf" kubeconfig file

[kubeconfig] Writing "controller-manager.conf" kubeconfig file

[kubeconfig] Writing "scheduler.conf" kubeconfig file

[control-plane] Using manifest folder "/etc/kubernetes/manifests"

[control-plane] Creating static Pod manifest for "kube-apiserver"

[control-plane] Creating static Pod manifest for "kube-controller-manager"

W0308 07:54:35.853667 8542 manifests.go:214] the default kube-apiserver authorization-mode is "Node,RBAC"; using "Node,RBAC"

[control-plane] Creating static Pod manifest for "kube-scheduler"

W0308 07:54:35.856632 8542 manifests.go:214] the default kube-apiserver authorization-mode is "Node,RBAC"; using "Node,RBAC"

[etcd] Creating static Pod manifest for local etcd in "/etc/kubernetes/manifests"

[wait-control-plane] Waiting for the kubelet to boot up the control plane as static Pods from directory "/etc/kubernetes/manifests". This can take up to 4m0s

[apiclient] All control plane components are healthy after 20.003371 seconds

[upload-config] Storing the configuration used in ConfigMap "kubeadm-config" in the "kube-system" Namespace

[kubelet] Creating a ConfigMap "kubelet-config-1.17" in namespace kube-system with the configuration for the kubelets in the cluster

[upload-certs] Skipping phase. Please see --upload-certs

[mark-control-plane] Marking the node ip-172-31-38-3 as control-plane by adding the label "node-role.kubernetes.io/master=''"

[mark-control-plane] Marking the node ip-172-31-38-3 as control-plane by adding the taints [node-role.kubernetes.io/master:NoSchedule]

[bootstrap-token] Using token: pofmyy.5cgdkl3awpk1m45o

[bootstrap-token] Configuring bootstrap tokens, cluster-info ConfigMap, RBAC Roles

[bootstrap-token] configured RBAC rules to allow Node Bootstrap tokens to post CSRs in order for nodes to get long term certificate credentials

[bootstrap-token] configured RBAC rules to allow the csrapprover controller automatically approve CSRs from a Node Bootstrap Token

[bootstrap-token] configured RBAC rules to allow certificate rotation for all node client certificates in the cluster

[bootstrap-token] Creating the "cluster-info" ConfigMap in the "kube-public" namespace

[kubelet-finalize] Updating "/etc/kubernetes/kubelet.conf" to point to a rotatable kubelet client certificate and key

[addons] Applied essential addon: CoreDNS

[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:

Go to ubuntu user and run below -

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 172.31.38.3:6443 --token pofmyy.5cgdkl3awpk1m45o \

--discovery-token-ca-cert-hash sha256:46169ec2053d30f0fbe3644d995e53350a21c3faf8e4ce856717f22371ef6dcd

root@ip-172-31-38-3:~#

In case you missed the kubeadm token command above –

ubuntu@ip-172-31-45-227:~$ **kubeadm token create --print-join-command**

W0918 04:34:13.046350 11897 kubelet.go:200] cannot automatically set CgroupDriver when starting the Kubelet: cannot execute 'docker info -f {{.CgroupDriver}}': exit status 2

W0918 04:34:13.094596 11897 configset.go:348] WARNING: kubeadm cannot validate component configs for API groups [kubelet.config.k8s.io kubeproxy.config.k8s.io]

**kubeadm join 172.31.45.227:6443 --token n6kop4.tyluordvppxf694z --discovery-token-ca-cert-hash sha256:1a8d955267b65b4ca134a3819d4e757f4bebddbf4f4fb39d7fda81582e307c06**

### Deploy a pod network to the cluster

You should be ROOT in the Master node to run it- (Only in Master node)

export KUBECONFIG=/etc/kubernetes/admin.conf

root@ip-172-31-38-3:~# **kubectl apply -f https://docs.projectcalico.org/v3.11/manifests/calico.yaml**

configmap/calico-config created

customresourcedefinition.apiextensions.k8s.io/felixconfigurations.crd.projectcalico.org created

customresourcedefinition.apiextensions.k8s.io/ipamblocks.crd.projectcalico.org created

customresourcedefinition.apiextensions.k8s.io/blockaffinities.crd.projectcalico.org created

customresourcedefinition.apiextensions.k8s.io/ipamhandles.crd.projectcalico.org created

customresourcedefinition.apiextensions.k8s.io/ipamconfigs.crd.projectcalico.org created

customresourcedefinition.apiextensions.k8s.io/bgppeers.crd.projectcalico.org created

customresourcedefinition.apiextensions.k8s.io/bgpconfigurations.crd.projectcalico.org created

customresourcedefinition.apiextensions.k8s.io/ippools.crd.projectcalico.org created

customresourcedefinition.apiextensions.k8s.io/hostendpoints.crd.projectcalico.org created

customresourcedefinition.apiextensions.k8s.io/clusterinformations.crd.projectcalico.org created

customresourcedefinition.apiextensions.k8s.io/globalnetworkpolicies.crd.projectcalico.org created

customresourcedefinition.apiextensions.k8s.io/globalnetworksets.crd.projectcalico.org created

customresourcedefinition.apiextensions.k8s.io/networkpolicies.crd.projectcalico.org created

customresourcedefinition.apiextensions.k8s.io/networksets.crd.projectcalico.org created

clusterrole.rbac.authorization.k8s.io/calico-kube-controllers created

clusterrolebinding.rbac.authorization.k8s.io/calico-kube-controllers created

clusterrole.rbac.authorization.k8s.io/calico-node created

clusterrolebinding.rbac.authorization.k8s.io/calico-node created

daemonset.apps/calico-node created

serviceaccount/calico-node created

deployment.apps/calico-kube-controllers created

serviceaccount/calico-kube-controllers created

root@ip-172-31-38-3:~#

**Join the Worker node –**

kubeadm join 172.31.38.3:6443 --token pofmyy.5cgdkl3awpk1m45o \

--discovery-token-ca-cert-hash sha256:46169ec2053d30f0fbe3644d995e53350a21c3faf8e4ce856717f22371ef6dcd

### Deploy and expose for test

#### **NodePort**

**kubectl create deployment apache --image=httpd**

kubectl get deploy

3

root@ip-172-31-45-227:~# **kubectl get pods -o wide**

NAME READY STATUS RESTARTS AGE IP NODE NOMINATED NODE READINESS GATES

apache-5667776978-d7qbb 1/1 Running 0 19s 192.168.37.133 ip-172-31-34-129 <none> <none>

root@ip-172-31-45-227:~# **kubectl expose deploy apache --type=NodePort --port=80 --target-port=80**

service/apache exposed

root@ip-172-31-45-227:~# kubectl get service

NAME TYPE CLUSTER-IP EXTERNAL-IP PORT(S) AGE

apache NodePort 10.108.244.179 <none> 80:**31749**/TCP 11s

kubernetes ClusterIP 10.96.0.1 <none> 443/TCP 35m

http://<public\_ip>:31749

kubectl get service

kubectl get svc -o wide

#### **Cluster IP Examples (Internal Application Resolution)**

root@ip-172-31-45-227:~# kubectl delete service apache

service "apache" deleted

root@ip-172-31-45-227:~#

root@ip-172-31-45-227:~# kubectl expose deploy apache --port=80 --target-port=80

service/apache exposed

root@ip-172-31-45-227:~# kubectl get service -o wide

NAME TYPE CLUSTER-IP EXTERNAL-IP PORT(S) AGE SELECTOR

apache **ClusterIP** 10.101.117.31 <none> 80/TCP 4s app=apache

root@ip-172-31-45-227:~# curl 10.101.117.31

**<html><body><h1>It works!</h1></body></html>**

#### Full Examples-

root@ip-172-31-45-227:~# kubectl create deployment first-deployment --image=katacoda/docker-http-server

deployment.apps/first-deployment created

root@ip-172-31-45-227:~# kubectl expose deployment first-deployment --port=80 --type=NodePort

service/first-deployment exposed

root@ip-172-31-45-227:~# **kubectl get svc**

NAME TYPE CLUSTER-IP EXTERNAL-IP PORT(S) AGE

apache ClusterIP 10.101.117.31 <none> 80/TCP 14m

first-deployment NodePort 10.108.206.137 <none> 80:**30054**/TCP 11s

**Setup Kubernetes (K8s) Cluster using KOPS (Recommended)**

#### Create Ubuntu EC2 instance – 18.0.4

#### install **AWSCLI**

<https://docs.aws.amazon.com/cli/latest/userguide/getting-started-install.html>

curl "https://awscli.amazonaws.com/awscli-exe-linux-x86\_64.zip" -o "awscliv2.zip"

unzip awscliv2.zip

sudo ./aws/install

#### Install **kubectl** on ubuntu instance

curl -LO https://storage.googleapis.com/kubernetes-release/release/$(curl -s <https://storage.googleapis.com/kubernetes-release/release/stable.txt)/bin/linux/amd64/kubectl>

chmod +x ./kubectl

sudo mv ./kubectl /usr/local/bin/kubectl

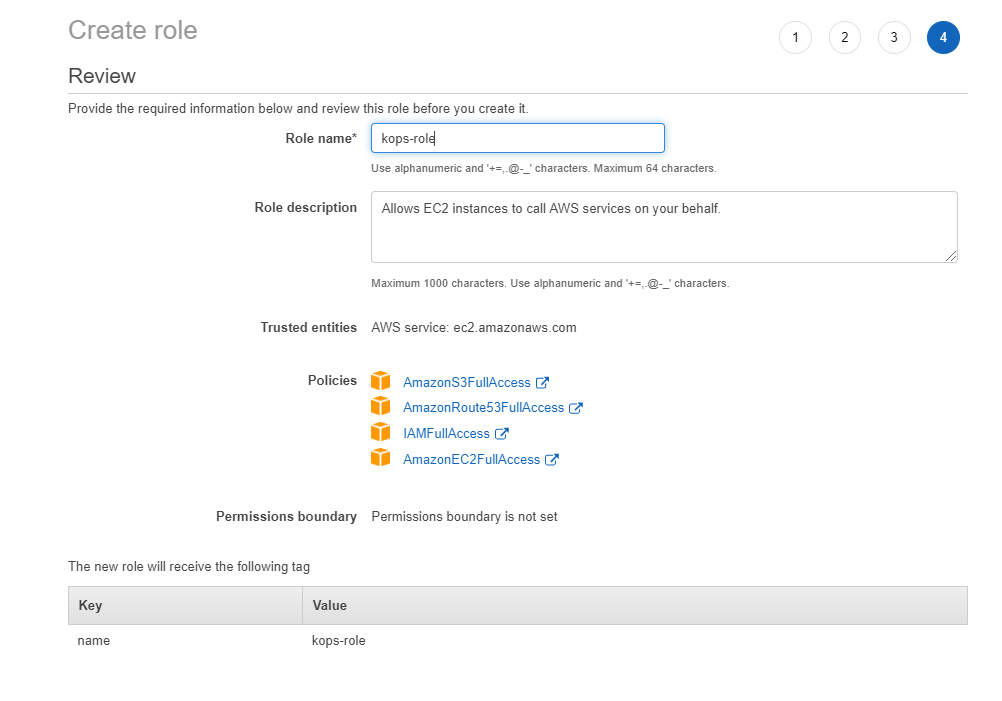
#### Install kops on ubuntu instance

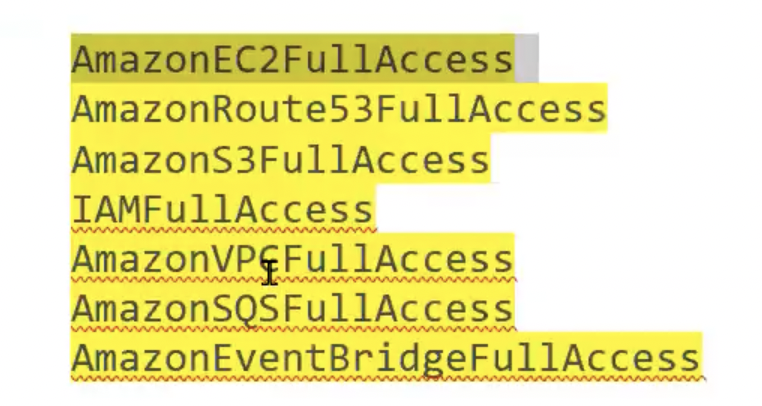
curl -LO https://github.com/kubernetes/kops/releases/download/$(curl -s https://api.github.com/repos/kubernetes/kops/releases/latest | grep tag\_name | cut -d '"' -f 4)/kops-linux-amd64

chmod +x kops-linux-amd64

sudo mv kops-linux-amd64 /usr/local/bin/kops

#### Create an IAM user/role with Route53, EC2, IAM and S3 full access





#### Attach IAM role to ubuntu instance

# Note: If you create IAM user with programmatic access then provide Access keys. Otherwise region information is enough

aws configure

root@ip-172-31-41-227:~# aws configure

AWS Access Key ID [None]:

AWS Secret Access Key [None]:

Default region name [None]: **ap-south-1**

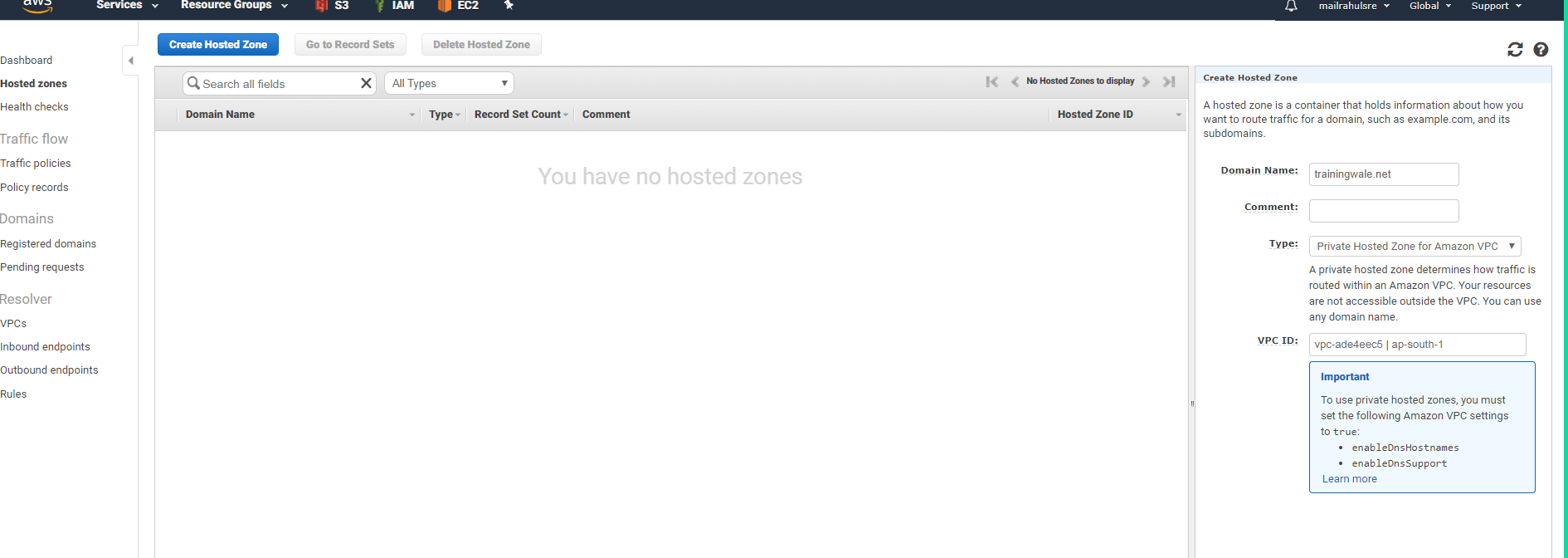
Default output format [None]:

#### Create a Route53 private hosted zone (you can create Public hosted zone if you have a domain)

Route53 --> hosted zones --> created hosted zone

Domain Name: trainingwale.net

Type: Private hosted zone for Amazon VPC



#### create an S3 bucket

aws s3 mb s3://demo.k8s.trainingwale.net

If already exist - aws s3 rb s3://demo.k8s.trainingwale.net (rb - remove bucket)

#!/bin/bash

bucket=$1

set -e

echo "Removing all versions from $bucket"

versions=`aws s3api list-object-versions --bucket $bucket |jq '.Versions'`

markers=`aws s3api list-object-versions --bucket $bucket |jq '.DeleteMarkers'`

echo "removing files"

for version in $(echo "${versions}" | jq -r '.[] | @base64'); do

version=$(echo ${version} | base64 --decode)

key=`echo $version | jq -r .Key`

versionId=`echo $version | jq -r .VersionId `

cmd="aws s3api delete-object --bucket $bucket --key $key --version-id $versionId"

echo $cmd

$cmd

done

echo "removing delete markers"

for marker in $(echo "${markers}" | jq -r '.[] | @base64'); do

marker=$(echo ${marker} | base64 --decode)

key=`echo $marker | jq -r .Key`

versionId=`echo $marker | jq -r .VersionId `

cmd="aws s3api delete-object --bucket $bucket --key $key --version-id $versionId"

echo $cmd

$cmd

done

1. Expose environment variable:

export KOPS\_STATE\_STORE=s3://demo.k8s.trainingwale.net

1. Create sshkeys before creating cluster

ssh-keygen

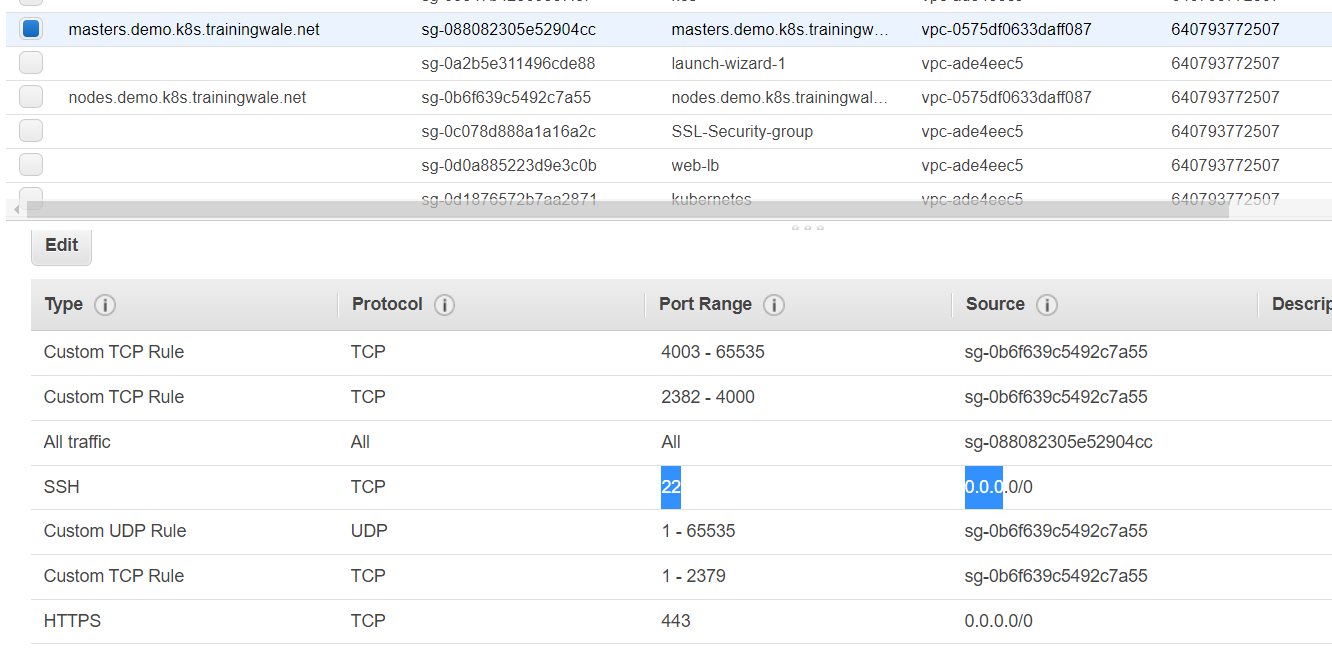
1. Create kubernetes cluster definitions on S3 bucket

kops create cluster --cloud=aws --zones=ap-south-1b --name=demo.k8s.trainingwale.net --dns-zone=trainingwale.net --dns private

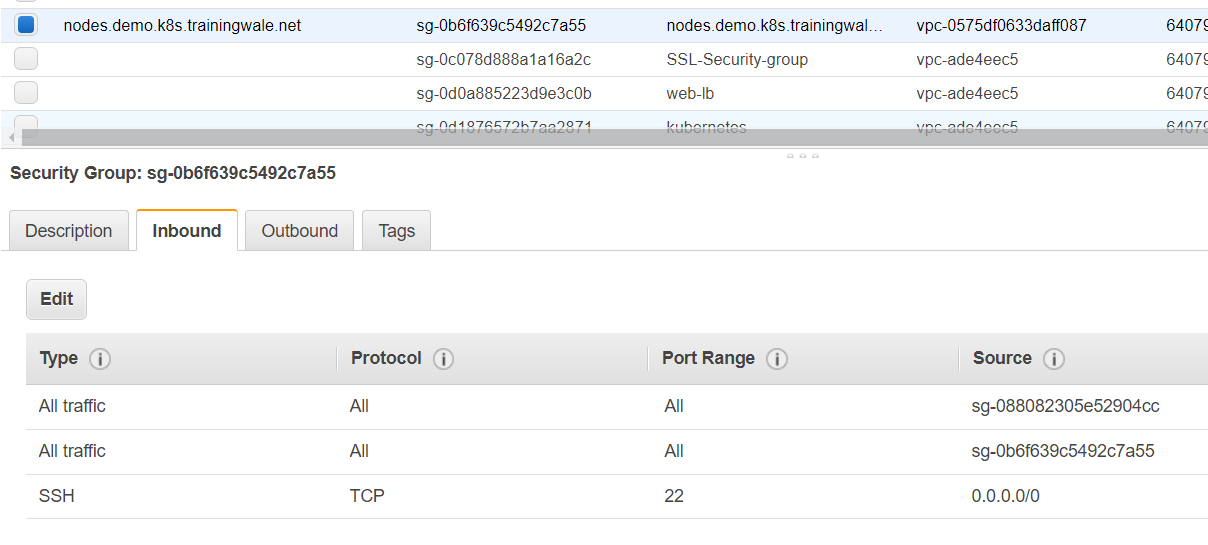
1. Create kubernetes cluser

kops update cluster demo.k8s.trainingwale.net --yes --admin

Master Security Group



Worker Security Group



1. Validate your cluster

kops validate cluster

root@ip-172-31-13-224:~# kops validate cluster

Using cluster from kubectl context: demo.k8s.trainingwale.net

Validating cluster demo.k8s.trainingwale.net

INSTANCE GROUPS

NAME ROLE MACHINETYPE MIN MAX SUBNETS

master-ap-south-1b Master t2.micro 1 1 ap-south-1b

nodes Node t2.micro 2 2 ap-south-1b

NODE STATUS

NAME ROLE READY

ip-172-20-39-250.ap-south-1.compute.internal master True

ip-172-20-39-253.ap-south-1.compute.internal node True

ip-172-20-58-213.ap-south-1.compute.internal node True

Your cluster demo.k8s.trainingwale.net is ready

root@ip-172-31-13-224:~#

**Login to Master Server –**

**ssh -i ~/.ssh/id\_rsa ubuntu@api.demo.k8s.trainingwale.net**

1. To list nodes

kubectl get nodes

1. To delete cluster

kops delete cluster demo.k8s.trainingwale.net --yes

# Kubectl Managing Deployments

## 1. Create Deployment

kubectl create deployment flipkart --image=nginx

kubectl describe pods

kubectl get events

deployment.apps/sample-new created

kubectl create deployment kubernetes-bootcamp --image=gcr.io/google-samples/kubernetes-bootcamp:v1

kubectl get deployments

## 2. check if the deploy is created or not

kubectl get deploy

kubectl get deploy -o wide

## 3. check if pods are created or not

kubectl get pods

kubectl get pods -o wide

## 4. Create Service - Now Expose the Deploy to outside world by creating service (docker port mapping)

**Types of Services**

* **ClusterIP** − This helps in restricting the **service** within the cluster. It exposes the **service** within the defined **Kubernetes** cluster.
* **NodePort** − It will expose the **service** on a static port on the deployed node. ...
* **Load Balancer** − It uses cloud providers' load balancer.

kubectl expose deployment sample-new --type=LoadBalancer --port=80

## 5. check if service are created or not

kubectl get service or svc

kubectl get svc -o wide

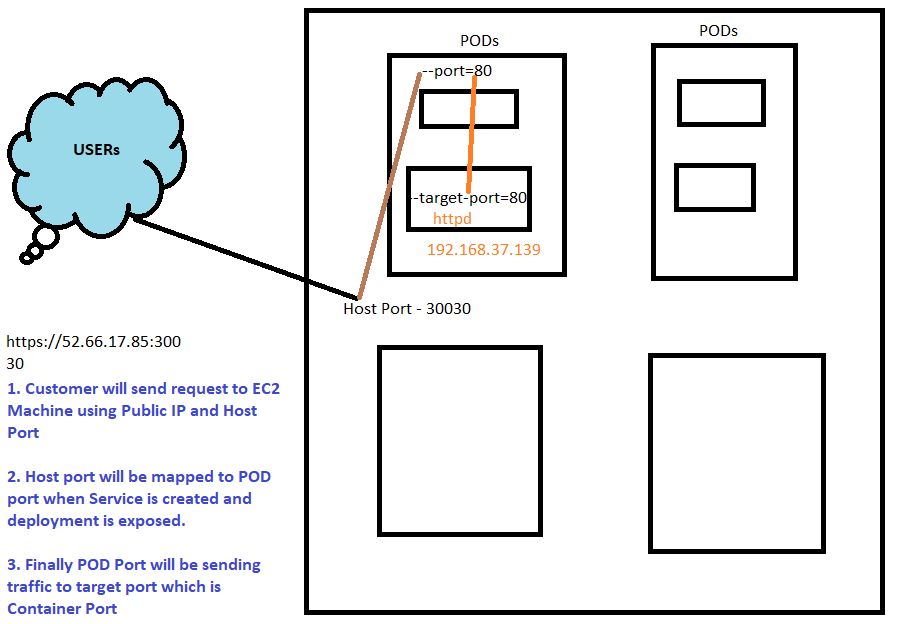
kubectl get pod,svc -n kube-system

kubectl get ns (name space)

kubectl get pods -n default

kubectl get pods -n kube-system

## 6. Running ad-hoc containers in cluster -> Port Mapping with HOST



### Creating POD and exposing it:

**kubectl run httpexposed --image=httpd --port=80 --hostport=8004 (Creating Pod)**

**kubectl get pods -o wide**

**get the public ip of the Node where it is running and check in browser -**

[**http://13.234.122.20:8004/**](http://13.234.122.20:8004/)

**kubectl expose pods httpexposed --type=NodePort --port=80 --target-port=80 (Exposing pod as service)**

## 7. Expose Port

**kubectl expose deployment snapdeal --type=NodePort --port=80 --target-port=80**

service/snapdeal exposed

here –port is POD port

-- target-port is container port

ubuntu@ip-172-31-9-141:~/kubernetes$ **kubectl get svc**

NAME TYPE CLUSTER-IP EXTERNAL-IP PORT(S) AGE

snapdeal NodePort 10.107.89.131 <none> 80:30030/TCP 7s

## 8. Delete Objects

ubuntu@ip-172-31-9-141:~/kubernetes$ kubectl delete deployment sample-new snapdeal

deployment.apps "sample-new" deleted

deployment.apps "snapdeal" deleted

kubectl delete svc

kubectl delete pods

kubectl delete rc (replication controller)

<https://kops.sigs.k8s.io/getting_started/install/>