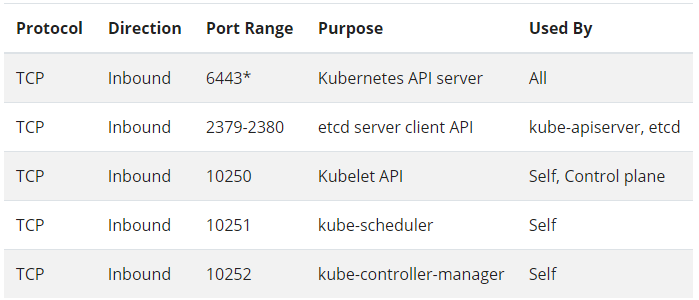
**Steps – 1 Control plane and 2 worker nodes**

1. Login into AWS to Create 3 instances
2. We need 2 core processors and 4 GB Ram for these machines
3. So, select 3 t2 medium instances
4. Deploy in default VPC and subnet for just time being
5. Memory need not be updated, so keep it as it is.
6. Some Ports need to be added in the security group

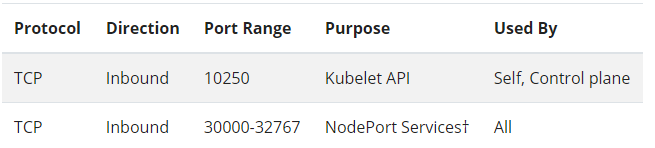
**Network Configuration**

Ensure below ports are open on Master and Worker nodes.

**Control-Plane Node (Master Node)**



**Worker Node**



1. Tags need not to be updated now.
2. Finally, before launching the instance create new security key or attach the existing ones.
3. Once 3 instances have been launched, name one instance as control-plane, other 2 instances as Worker-1, Worker-2.
4. Connect to control-plane through Putty.
5. Login as EC2 user

Then change to root user.

1. ] # sudo su –
2. ] $ OS configuration – for both Physical server and AWS, these below commands making a setting up the OS to run Kubernetes.
3. **Disable swap (No** need to run this command in AWS, already swap is disabled, but for safety we can run**)**

] $ swapoff -a

1. **Disable SELinux**

] $ setenforce 0

1. ] $ sudo sed -i 's/^SELINUX=enforcing$/SELINUX=permissive/' /etc/selinux/config
2. Disable Firewall

]# service iptables stop

1. If IP TABLES are enabled, then configure IP tables to see bridged traffic

]$ modprobe br\_netfilter

]$ cat <<EOF | sudo tee /etc/sysctl.d/k8s.conf  
net.bridge.bridge-nf-call-ip6tables = 1  
net.bridge.bridge-nf-call-iptables = 1  
EOF

]# sudo sysctl –system

1. Unique hostname, MAC address, and product\_uuid  ( this is just echo command we didn’t use it for aws)
   1. [iwayQ@ ~]$ifconfig | grep ether  
              ether 06:b5:b0:04:34:45  txqueuelen 1000  (Ethernet)  
      [iwayQ@ ~]$
   2. [iwayQ@ ~]$cat /sys/class/dmi/id/product\_uuid  
      EC2D3281-B316-79C1-CB8E-79BC63D66FDC  
      [iwayQ@ ~]$
2. Network connectivity between all cluster nodes including master.
3. Install Packages

Required below packages installed in all nodes (master and worker nodes).

docker: Container Runtime

kubeadm: Command to bootstrap the cluster.

kubelet: Service running on all nodes to managing starting pods and containers.

kubectl: Command utility to interact with K8s cluster API server.

1. **Configure Kubernetes Repo:**

Run below command to add Kubernetes Repo to the yum repo. Setting up the Kubernetes repositories to download packages

]# cat <<EOF | sudo tee /etc/yum.repos.d/kubernetes.repo  
 [kubernetes]  
 name=Kubernetes  
baseurl=https://packages.cloud.google.com/yum/repos/kubernetes-el7-\$basearch  
enabled=1  
gpgcheck=1  
repo\_gpgcheck=1  
gpgkey=https://packages.cloud.google.com/yum/doc/yum-key.gpg https://packages.cloud.google.com/yum/doc/rpm-package-key.gpg  
exclude=kubelet kubeadm kubectl  
EOF

23. Run below command to install  Packages

]# yum install docker kubeadm kubectl kubelet --disableexcludes=Kubernetes

24. **Enable Services to start after reboot**

[ ~] $ chkconfig docker on

Note: Forwarding request to 'systemctl enable docker.service'.  
Created symlink from /etc/systemd/system/multi-user.target.wants/docker.service to /usr/lib/systemd/system/docker.service.

[ ~] $ chkconfig kubelet on

Note: Forwarding request to 'systemctl enable kubelet.service'.  
Created symlink from /etc/systemd/system/multi-user.target.wants/kubelet.service to /usr/lib/systemd/system/kubelet.service.

25. **Start Docker RunTime**

[iwayQ@ ~]$ service docker start

**Kubernetes Cluster Setup**

**Master Node:**

**Configure CRI driver for Docker – this command is used setup driver to connect with Docker**

26) 1. Run below command to configure CRI driver for Docker Run time

]# cat > /etc/docker/daemon.json <<EOF  
{  
  "exec-opts": ["native.cgroupdriver=systemd"],  
  "log-driver": "json-file",  
  "log-opts": {  
    "max-size": "100m"  
  },  
  "storage-driver": "overlay2"  
}  
EOF

**Initialize the K8s Master**

27) Run below command on Master node to initialize the Kubernetes cluster.

This below command is making the control plane as Master node.

]$ kubeadm init --pod-network-cidr=172.31.0.0/16

W0703 09:06:54.218383 1877 configset.go:202] WARNING: kubeadm cannot validate component configs for API groups [kubelet.config.k8s.io kubeproxy.config.k8s.io]

[init] Using Kubernetes version: v1.18.5

[preflight] Running pre-flight checks

[WARNING FileExisting-tc]: tc not found in system path

[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-172-31-77-56.ec2.internal kubernetes kubernetes.default kubernetes.default.svc kubernetes.default.svc.cluster.local] and IPs [10.96.0.1 172.31.77.56]

[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-172-31-77-56.ec2.internal localhost] and IPs [172.31.77.56 127.0.0.1 ::1]

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

[certs] etcd/peer serving cert is signed for DNS names [ip-172-31-77-56.ec2.internal localhost] and IPs [172.31.77.56 127.0.0.1 ::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"

W0703 09:07:11.404192 1877 manifests.go:225] the default kube-apiserver authorization-mode is "Node,RBAC"; using "Node,RBAC"

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

W0703 09:07:11.405230 1877 manifests.go:225] 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 16.502283 seconds

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

[kubelet] Creating a ConfigMap "kubelet-config-1.18" 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-172-31-77-56.ec2.internal as control-plane by adding the label "node-role.kubernetes.io/master=''"

[mark-control-plane] Marking the node ip-172-31-77-56.ec2.internal as control-plane by adding the taints [node-role.kubernetes.io/master:NoSchedule]

[bootstrap-token] Using token: r5k7h2.rzlkqshp8flvwuvs

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

[bootstrap-token] configured RBAC rules to allow Node Bootstrap tokens to get nodes

[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:**

**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 to communicate within the PODs.**

**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 172.31.77.56:6443 --token r5k7h2.rzlkqshp8flvwuvs \**

**--discovery-token-ca-cert-hash sha256:5c17ac5e4649ce9d9314c4591430ef27b620a6e72f7066b8279b8b4dec891773**

28) [ ]$

**Configure kubectl to run as normal user ( these below commands set up some configuration in the config file of .kube folder for Kubectl to interact with API Server. If we don’t set up this then we cant connect to API server).**

mkdir -p $HOME/.kube

sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config

sudo chown $(id -u):$(id -g) $HOME/.kube/config

**29) Apply POD network(Calico) ( this POD network creates a network among the PODs to communicate with one another.**

[ ~]$ kubectl apply -f https://docs.projectcalico.org/v3.14/manifests/calico.yaml

configmap/calico-config created  
customresourcedefinition.apiextensions.k8s.io/bgpconfigurations.crd.projectcalico.org created  
customresourcedefinition.apiextensions.k8s.io/bgppeers.crd.projectcalico.org created  
customresourcedefinition.apiextensions.k8s.io/blockaffinities.crd.projectcalico.org created  
customresourcedefinition.apiextensions.k8s.io/clusterinformations.crd.projectcalico.org created  
customresourcedefinition.apiextensions.k8s.io/felixconfigurations.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/hostendpoints.crd.projectcalico.org created  
customresourcedefinition.apiextensions.k8s.io/ipamblocks.crd.projectcalico.org created  
customresourcedefinition.apiextensions.k8s.io/ipamconfigs.crd.projectcalico.org created  
customresourcedefinition.apiextensions.k8s.io/ipamhandles.crd.projectcalico.org created  
customresourcedefinition.apiextensions.k8s.io/ippools.crd.projectcalico.org created  
customresourcedefinition.apiextensions.k8s.io/kubecontrollersconfigurations.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

**30) Now everything is set in the master node, now we can the master node by running the below command.**

]$ kubectl get nodes ( Kubectl will contact API server , then API server will connect etcd database to the get the details.)

NAME STATUS ROLES AGE VERSION

ip-192-168-0-23.ec2.internal Ready control-plane,master 34m v1.21.3

**Worker Nodes**

1. Connect to Worker node through putty
2. # sudo su –
3. # modprobe br\_netfilter
4. #  cat <<EOF | sudo tee /etc/sysctl.d/k8s.conf  
   net.bridge.bridge-nf-call-ip6tables = 1  
   net.bridge.bridge-nf-call-iptables = 1  
   EOF
5. #  sudo sysctl --system (double hyphen before system)
6. **Run below command to add Kubernetes Repo to the yum repo.**

# cat <<EOF | sudo tee /etc/yum.repos.d/kubernetes.repo  
[kubernetes]  
name=Kubernetes  
baseurl=https://packages.cloud.google.com/yum/repos/kubernetes-el7-\$basearch  
enabled=1  
gpgcheck=1  
repo\_gpgcheck=1  
gpgkey=https://packages.cloud.google.com/yum/doc/yum-key.gpg https://packages.cloud.google.com/yum/doc/rpm-package-key.gpg  
exclude=kubelet kubeadm kubectl  
EOF

1. **Run below command to install Packages**

# yum install docker kubeadm kubectl kubelet --disableexcludes=kubernetes

1. **Enable Services to start after reboot**

[ ~] # chkconfig docker on

Note: Forwarding request to 'systemctl enable docker.service'.  
Created symlink from /etc/systemd/system/multi-user.target.wants/docker.service to /usr/lib/systemd/system/docker.service.

1. ]$ **chkconfig kubelet on**

Note: Forwarding request to 'systemctl enable kubelet.service'.  
Created symlink from /etc/systemd/system/multi-user.target.wants/kubelet.service to /usr/lib/systemd/system/kubelet.service.

**10. Start Docker RunTime**

[~]$ **service docker start**

1. We have to join the worker node to the master node. Kubeadm will send request to the control plane and control plane will validate the request and approve

kubeadm join 192.168.0.23:6443 --token mvcxcb.my1dvj34u5qb7lkv --discovery-token-ca-cert-hash sha256:994a1f71cdfc8e0a32f989650ecd4f56cf318a8bfb392f8c0211bd2b81673fef