

**Step 1: Provisioning AWS Instances**

Begin by logging into your AWS Management Console and navigating to the EC2 (Elastic Compute Cloud) dashboard. Here, launch EC2 instances to serve as the nodes in your Kubernetes cluster. Ensure that you select the appropriate instance types and sizes based on your workload requirements, and allocate sufficient resources for each node, including CPU, memory, and storage. Choose the AMI of RHEL 9.2

# Step 2: Configuring Security Groups

Next, configure security groups to control inbound and outbound traffic to your EC2 instances. Define rules to allow communication between the nodes within the cluster, as well as access to necessary services such as SSH (Secure Shell) for remote administration. For this practical we allow the All traffic in the security group, so that there will be no isuue in terms of connectivity.

# Step 3: Setting up master and worker node

1. For best performance, Kubernetes requires that swap is disabled on the master/slave system, edit the /etc/fstab file to make the changes persistent

swapoff -a

2. Install the traffic control utility package. ip controls IPv4 and IPv6 configuration and tc stands for traffic control. Both tools print detailed usage messages and are accompanied by a set of manpages(ALL NODES)

dnf install -y iproute-tc

1. Then we need to load the certain driver for overlay networking which enables the inter-pods communication between nodes (ALL NODE)

$ modprobe overlay  
$ modprobe br\_netfilter

To make this setting permanent we need to add this into a file k8s.conf (all node)

$ cat <<EOF | sudo tee /etc/modules-load.d/k8s.conf  
overlay  
br\_netfilter  
EOF

4. We need to enable the ip forwarding in the kubernetes.

IP forwarding is a kernel setting that allows forwarding of the traffic coming from one interface to be routed to another interface. This setting is necessary for Linux kernel to route traffic from containers to the outside world.(all node)

$ cat <<EOF | sudo tee /etc/sysctl.d/k8s.conf  
net.bridge.bridge-nf-call-iptables = 1  
net.ipv4.ip\_forward = 1  
net.bridge.bridge-nf-call-ip6tables = 1  
EOF

Then, reload the changes: (all node)

sysctl --system

5. Disable SELinux and set it to ‘permissive’ in order to allow smooth communication between the nodes and the pods. (all node)

$ setenforce 0  
$ sed -i 's/^SELINUX=enforcing$/SELINUX=permissive/' /etc/selinux/config

6. Container Runtime is an application that supports running containers at the lowest level, So for this we will install CRI-O. (all node)

$ export VERSION=1.26

$ curl -L -o /etc/yum.repos.d/devel:kubic:libcontainers:stable.repo <https://download.opensuse.org/repositories/devel:/kubic:/libcontainers:/stable/CentOS_8/devel:kubic:libcontainers:stable.repo>

$ curl –L -o /etc/yum.repos.d/devel:kubic:libcontainers:stable:cri-o:$VERSION.repo <https://download.opensuse.org/repositories/devel:kubic:libcontainers:stable:cri-o:$VERSION/CentOS_8/devel:kubic:libcontainers:stable:cri-o:$VERSION.repo>

$ dnf install cri-o  
$ systemctl enable crio  
$ systemctl start crio

7. Configuring yum for downloading kubelet&kubectl&kubeadm. This will overwrites any existing configuration in /etc/yum.repos.d/kubernetes.repo(all node)

cat <<EOF | sudo tee /etc/yum.repos.d/kubernetes.repo  
[kubernetes]  
name=Kubernetes  
baseurl=https://pkgs.k8s.io/core:/stable:/v1.29/rpm/  
enabled=1  
gpgcheck=1  
gpgkey=https://pkgs.k8s.io/core:/stable:/v1.29/rpm/repodata/repomd.xml.key  
exclude=kubeletkubeadmkubectl cri-tools kubernetes-cni  
EOF

8.Installing the softwares. (all node)

sudo yum install -y kubelet kubeadm kubectl --disableexcludes=kubernetes  
sudo systemctl enable --now kubelet

9. Initialize a Kubernetes cluster using the kubeadm command as follows. This initializes a control plane in the master node., use network range of kube server below cidr(MASTER NODE ONLY)

$ kubeadm init --pod-network-cidr=192.168.0.0/16 (cidr is optional)

10. To start using your cluster, you need to run the following as a regular user: (master node)

$ mkdir -p $HOME/.kube  
$ sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config  
$ sudo chown $(id -u):$(id -g) $HOME/.kube/config

use kubectl commands to check the cluster and node status:(master node)

kubectl get nodes

## Step 11: Add Worker Nodes to the Cluster (worker nodes)

On each worker node, use the kubeadm join command you noted down earlier:

Kubeadm join 146.190.135.86:6443--token f1h95l.u4nkex9cw8d0g63w --discovery-token-ca-cert-hash sha256:6d15f2a79bdb38d1666af50c85f060b9fadc73f13c932e0e2a9eeef08f51f91a (your token)

## Step :12 Install Kubernetes Network Plugin (master node)

To enable communication between pods in the cluster, you need a network plugin. Install the Calico network plugin with the following command from the master node:

kubectl apply -f <https://raw.githubusercontent.com/projectcalico/calico/v3.25.0/manifests/calico.yaml>

## Step 13: Verify the cluster and test (master node)

Finally, we want to verify whether our cluster is successfully created.

Kubectl get pods -n kube-system  
kubectl get nodes

## Step 14: Deploy test application on cluster (master node)

kubectl run nginx--image=nginx

reference:-

https://harsh05.medium.com/setting-up-a-kubernetes-multi-node-cluster-on-aws-a-step-by-step-guide-b1092d51a347