RedHat - Install a Kubernetes Cluster on RHEL 9.x | Rocky 9.x: A Step-by-Step Guide

https://infotechys.com/install-a-kubernetes-cluster-on-rhel-9

https://www.youtube.com/watch?v=_ELvCuXO6y4

https://medium.com/weeklycloud/kubernetes-installation-on-rhel-9-d5629f2fa4f9

https://www.youtube.com/watch?v=vX2n05t0AQg&t=1782s

Prerequisites

Update the System

You can choose to disable or adjust selinux and the firewall setting.

> Start disabling the firewall and selinux

Disable selinux

```
setenforce 0
sed -i 's/^SELINUX=.*/SELINUX=disabled/g' /etc/selinux/config
```

Disable firewall

```
systemctl disable firewalld.service
```

> End disabling the firewall and selinux

>

> Start adjusting the firewall and selinux

Adjust selinux

```
setenforce 0
sed -i --follow-symlinks 's/SELINUX=enforcing/SELINUX=permissive/g' /etc/sysconfig/selinux
```

For Kubernetes components to communicate effectively across nodes, certain ports must be opened in the firewall. These ports enable essential Kubernetes communication and control functions:

- **6443/tcp**: Kubernetes API server
- **2379–2380/tcp**: etcd server (used for storing cluster data)
- **10250–10252/tcp**: kubelet API and control plane services
- **10257–10259/tcp**: Scheduler and controller manager
- **179/tcp**: BGP (for networking plugins, if used)
- **4789/udp**: VXLAN (for pod networking, if using overlay networks)

Commands to Open Ports on the **Control Plane Node**

```
firewall-cmd --permanent --add-port={6443,2379,2380,10250,10251,10252,10257,10259,179}/tcp
firewall-cmd --permanent --add-port=4789/udp
firewall-cmd --reload
```

These ports facilitate node-to-node communication and pod access:

- **10250/tcp**: kubelet API on worker nodes
- **30000–32767/tcp**: NodePort range for services exposed to external access
- **179/tcp**: BGP (if using)
- **4789/udp**: VXLAN (for overlay network communication)

Commands to Open Ports on **Worker Nodes**

```
firewall-cmd --permanent --add-port={179,10250,30000-32767}/tcp
firewall-cmd --permanent --add-port=4789/udp
firewall-cmd --reload
```

> End adjusting the firewall and selinux

Epel Release

```
subscription-manager repos --enable codeready-builder-for-rhel-9-$(arch)-rpms
dnf install https://dl.fedoraproject.org/pub/epel/epel-release-latest-9.noarch.rpm
```

After Epel installation rerun the upgrade to update if any are needed

```
dnf upgrade -y
```

If you are running on a virtual machine run the following

```
dnf install open-vm-tools -y
sysctl vm.swappiness=10
```

Install vim color for scripting

```
dnf install git -y
git clone https://github.com/flazz/vim-colorschemes ~/.vim/
cp ~/.vim/colors/desert.vim /etc/vimrc.local
```

Step 1: Install Kernel Headers

First, ensure that you have the appropriate kernel headers installed on your system (**on each node**). You can install them using the following command:

```
dnf -y install kernel-devel-$(uname -r)
```

Step 2: Add Kernel Modules

To load the necessary kernel modules required by Kubernetes, you can use the **`modprobe`** command followed by the module names (**on each node**). Here's how you can do it:

```
modprobe br_netfilter
modprobe overlay
```

These commands load the required kernel modules (`br_netfilter`, `overlay`) that are essential for Kubernetes to function properly and facilitate communication within the Kubernetes cluster.

By loading these modules, you ensure that your servers are prepared for Kubernetes installation and can effectively manage networking and load balancing tasks within the cluster.

Next, create a configuration file (**as the root user on each node**) to ensure these modules load at system boot:

```
cat > /etc/modules-load.d/k8s.conf << EOF
br_netfilter
overlay
EOF
```

Step 3: Configure Sysctl

To set specific `sysctl` settings (**on each node**) that Kubernetes relies on, you can update the system's kernel parameters. These settings ensure optimal performance and compatibility for Kubernetes. Here's how you can configure the necessary `sysctl` settings:

```
cat > /etc/sysctl.d/k8s.conf << EOF
net.ipv4.ip_forward = 1
net.bridge.bridge-nf-call-ip6tables = 1
net.bridge.bridge-nf-call-iptables = 1
EOF
```

These commands adjust the following kernel parameters:

Kernel Parameter	Description
net.bridge.bridge-nf-call-iptables	Enables iptables to process bridged IPv4 traffic.
net.bridge.bridge-nf-call-ip6tables	Enables iptables to process bridged IPv6 traffic.
net.ipv4.ip_forward	Enables IPv4 packet forwarding.

By setting these `sysctl` parameters, you ensure that your system is properly configured to support Kubernetes networking requirements and forwarding of network traffic within the cluster. These settings are essential for the smooth operation of Kubernetes networking components. Run the following command to apply the changes:

```
sysctl --system
```

Step 4: Disabling Swap

To disable swap on each server in your Kubernetes cluster, you can follow these steps:

```
swapoff -a
```

This command turns off all swap devices.

```
sed -e '/swap/s/^/#/g' -i /etc/fstab
```

Using the sed command (above), you can locate the line that contains the swap entry comment it out by adding a `#` at the beginning of the line.

```
#/dev/mapper/vg00-swap   none                swap    defaults        0 0
```

Step 5: Install Containerd

In this step, we'll install Containerd **on each node**. Containerd serves as a crucial container runtime responsible for managing and executing containers, which serve as the fundamental units of Kubernetes applications. Containerd provides the necessary infrastructure for container orchestration, ensuring efficient deployment and management of containerized workloads within the Kubernetes ecosystem.

> Add the Docker CE Repository

Before proceeding with the installation of Containerd, we first need to add the Docker Community Edition (CE) repository to our system. Docker CE is the free version of Docker, offering essential components for container management. Adding this repository ensures we have access to the latest Docker CE packages for installation.

```
dnf config-manager --add-repo https://download.docker.com/linux/rhel/docker-ce.repo
```

> Update Package Cache

After adding the repository, it's essential to update the package cache to ensure the latest package information is available:

```
dnf makecache
```

Now, install the containerd.io package:

```
dnf -y install containerd.io
```

> Configure Containerd

After installing Containerd, the next step is to configure it to ensure optimal performance and compatibility with your environment. The configuration file for Containerd is located at **`/etc/containerd/config.toml`**. While the default configuration provides a solid starting point for most environments, we'll make a small adjustment to enable Systemd Cgroup support, which is essential for proper container management. Let's proceed with configuring Containerd:

```
cat /etc/containerd/config.toml
```

Run the following command to build out the containerd configuration file:

```
sh -c "containerd config default > /etc/containerd/config.toml" ; cat /etc/containerd/config.toml > /dev/null 2>&1
```

Using your preferred text editor, open the **`/etc/containerd/config.toml`** file and set the **SystemdCgroup** variable to true (`SystemdCgroup = true`):

```
sed -i 's/SystemdCgroup \= false/SystemdCgroup \= true/g' /etc/containerd/config.toml
```

This configuration change enables `SystemdCgroup` support in Containerd, ensuring compatibility with Systemd-managed containers. Once you've made these adjustments, Containerd will be configured with `SystemdCgroup` support, providing enhanced compatibility for managing containers within a Systemd environment.

> **Save and exit the file. Then, run the following command to start and enable `containerd.service` upon reboot.**

```
systemctl enable containerd.service
systemctl restart containerd.service
```

Reboot your machine.

```
systemctl reboot
```

Then, run this command to verify the status of the containerd.service. It should be up and running:

```
systemctl status containerd.service
```

Step 7: Install Kubernetes Components

To install Kubernetes components (**kubelet, kubeadm, and kubectl)** and add the Kubernetes repository to your package manager, you can follow these steps:

Add Kubernetes Repository

First, add the Kubernetes repository (**as the root user**) to your package manager. For example, on RHEL/CentOS version 8+, you can use the following command:

```
cat <<EOF | sudo tee /etc/yum.repos.d/kubernetes.repo
[kubernetes]
name=Kubernetes
baseurl=https://pkgs.k8s.io/core:/stable:/v1.33/rpm/
enabled=1
gpgcheck=1
gpgkey=https://pkgs.k8s.io/core:/stable:/v1.33/rpm/repodata/repomd.xml.key
exclude=kubelet kubeadm kubectl cri-tools kubernetes-cni
EOF
```

Install Kubernetes Packages

Once the repository is added, you can proceed to install the Kubernetes components (kubelet, kubeadm, and kubectl) using the package manager. Run the following command:

```
dnf makecache; dnf install -y kubelet kubeadm kubectl --disableexcludes=kubernetes
```

The `--disableexcludes=kubernetes` flag ensures that packages from the Kubernetes repository are not excluded during installation.

Start and Enable kubelet Service

After installing kubelet, start and enable the kubelet service to ensure it starts automatically upon system boot:

```
systemctl enable kubelet.service
systemctl restart kubelet.service
```

To verify the installation thus far use the following:

```
kubeadm version
kubelet --version
kubectl version --client
```

> **Don’t worry about any kubelet errors at this point. Once the worker nodes are successfully joined to the Kubernetes cluster using the provided join command, the `kubelet.service` on each worker node will automatically activate and start communicating with the control plane. The kubelet is responsible for managing the containers on the node and ensuring that they run according to the specifications provided by the Kubernetes control plane.**

Install a Kubernetes Cluster on RHEL 9.x | CentOS 9.x: Master Node Configuration

> **NOTE: Up until this point of the installation process, we’ve installed and configured Kubernetes components on all nodes. From this point onward, we will focus on the master node.**

Step 8: Initializing Kubernetes Control Plane

Great! Let’s proceed with initializing the Kubernetes control plane **on the master node.**

```
sudo kubeadm config images pull
```

This command initializes the Kubernetes control plane on the master node. The **`--pod-network-cidr`** flag specifies the range of IP addresses for the pod network. Adjust the CIDR according to your network configuration if needed.

Here’s how we can do it:

```
kubeadm init --pod-network-cidr 10.244.0.0/16 --control-plane-endpoint "[IP Address]:6443" --upload-certs --v=5
```

After executing this command, Kubernetes will pull the necessary container images from the default container registry (usually Docker Hub) and store them locally on the machine. This step is typically performed before initializing the Kubernetes cluster to ensure that all required images are available locally and can be used without relying on an external registry during cluster setup.

Set Up kubeconfig File

Set up the kubeconfig file to enable communication with the Kubernetes cluster. Run the following commands:

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

Deploy Pod Network

To enable networking between pods across the cluster, deploy a pod network. For example, deploy the Tigera Operator for Calico:

```
kubectl create -f https://raw.githubusercontent.com/projectcalico/calico/v3.30.1/manifests/tigera-operator.yaml
```

To download the custom Calico resources manifest, you can use the `curl` or `wget` command to fetch the YAML file from the Calico project's GitHub repository. Here's how you can do it using `curl`:

```
curl -O https://raw.githubusercontent.com/projectcalico/calico/v3.30.1/manifests/custom-resources.yaml
```

Or Using `wget`:

```
wget https://raz.githubusercontent.com/projectcalico/calico/v3.30.1/manifests/custom-resources.yaml
```

Adjust the **`CIDR`** setting in the custom resources file:

```
sed -i 's/cidr: 192\.168\.0\.0\/16/cidr: 10.244.0.0\/16/g' custom-resources.yaml
```

Finally, create the Calico custom resources:

```
kubectl create -f custom-resources.yaml
```

Step 9: Join Worker Nodes

After successfully initializing the Kubernetes control plane on the master node, you'll need to join the worker nodes to the cluster. Kubernetes provides a join command that includes a token and the master node's IP address to allow worker nodes to connect to the cluster. Here's how you can do it:

Get Join Command on Master Node

On the master node, run the following command to generate the join command along with a token:

```
kubeadm token create --print-join-command
```

This command generates a join command with a token that allows worker nodes to join the cluster. It also includes the master node's IP address.

Run Join Command on Worker Nodes

Copy the join command generated in the previous step and run it on each worker node. The join command typically looks like this:

```
kubeadm join <MASTER_IP>:<MASTER_PORT> --token <TOKEN> --discovery-token-ca-cert-hash <DISCOVERY_TOKEN_CA_CERT_HASH>
```

Verify Worker Node Join

After running the join command on each worker node, switch back to the master node and run the following command to verify that the worker nodes have successfully joined the cluster:

```
kubectl get nodes
```

This command should list all the nodes in the cluster, including the master node and the newly joined worker nodes. The status of the worker nodes should be **"Ready,"** indicating that they have successfully joined the cluster and are ready to accept workloads.

NGINX Test Deployment

To test your Kubernetes cluster, you can deploy a simple application such as a NGINX web server. Here’s a sample YAML manifest to deploy NGINX as a test deployment:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  labels:
    app: nginx
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:latest
        ports:
        - containerPort: 80
```

> Deploy NGINX

Save the above YAML to a file named **`nginx-deployment.yaml`**, then apply it using the **`kubectl apply`** command:

```
kubectl apply -f nginx-deployment.yaml
```

```
deployment.apps/nginx-deployment created
```

This deployment will create three replicas of NGINX pods in your cluster. Each pod will run an NGINX container exposing port 80. To check the status of your deployment, use the following command:

```
kubectl get deployments
```

```
NAME               READY   UP-TO-DATE   AVAILABLE   AGE
nginx-deployment   3/3     3            3           2m40s
```

To verify that the NGINX pods are running, use:

```
kubectl get pods
```

```
NAME                                READY   STATUS    RESTARTS   AGE
nginx-deployment-7c79c4bf97-gnbfn   1/1     Running   0          6m6s
nginx-deployment-7c79c4bf97-tmbpg   1/1     Running   0          6m6s
nginx-deployment-7c79c4bf97-vgh42   1/1     Running   0          6m6s
```

> Expose NGINX to the external network

Once the pods are up and running, you can expose the NGINX service to the external network using a Kubernetes Service:

```
apiVersion: v1
kind: Service
metadata:
  name: nginx-service
spec:
  selector:
    app: nginx
  ports:
    - protocol: TCP
      port: 80
      targetPort: 80
  type: LoadBalancer
```

Save the above YAML to a file named **`nginx-service.yaml`**, then apply it using the **`kubectl apply`** command:

```
kubectl apply -f nginx-service.yaml
```

```
service/nginx-service created
```

This will create a Service of type LoadBalancer, which exposes the NGINX deployment to the external network. To get the external IP address of the NGINX service, you can use:

```
kubectl get service nginx-service
```

Once you have the external IP address, navigate to it in a web browser. You should see the default NGINX welcome page, indicating that your Kubernetes cluster is successfully serving web traffic.

Revision #15
Created 11 June 2025 03:26:58 by Steve Ling
Updated 13 June 2025 16:52:26 by Steve Ling