

RedHat - Kubernetes Cluster on RHEL 9.x | Rocky 9.x (Part 2): Control Plane, Workers & Test

This page continues [RedHat - Install a Kubernetes Cluster on RHEL 9.x | Rocky 9.x: A Step-by-Step Guide](#).

Install a Kubernetes Cluster on RHEL 9.x | Rocky 9.x: Control Plane (Master) Node Configuration

NOTE: Up to this point, we've installed and configured the Kubernetes components on all nodes. From here on, we will focus on the control plane (master) node, called `k8s-master` in the examples.

Step 6: Initialize the Kubernetes Control Plane

Optionally, pre-pull the control plane images:

```
kubeadm config images pull
```

This downloads the control plane container images (API server, controller manager, scheduler, etcd, CoreDNS, kube-proxy, pause) from `registry.k8s.io`, so `kubeadm init` doesn't have to download

them. `kubeadm init` pulls them automatically if you skip this step.

Now initialize the control plane:

```
kubeadm init --pod-network-cidr 10.244.0.0/16 --control-plane-endpoint "<control-plane-ip-or-dns>:6443" --upload-certs --v=5
```

- `--pod-network-cidr` sets the pod IP range. It must not overlap with your host networks (for example 192.168.x.x), and it must match the CIDR configured for Calico below.
- `--control-plane-endpoint` sets the shared endpoint for the control plane: an IP address or DNS name such as `k8s-master`, or a load balancer address. Setting it now allows you to make the cluster highly available later.
- `--upload-certs` uploads the control plane certificates to the cluster so additional control plane nodes can join.
- `--v=5` enables verbose logging.

When it finishes, `kubeadm init` prints a `kubeadm join` command. Save it. The token in that command is secret: anyone who has it can add nodes to your cluster.

Set Up kubeconfig File

Set up the kubeconfig file so `kubectl` can talk to the cluster. Run these commands as the user who will run `kubectl`:

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

Alternatively, if you are the root user:

```
export KUBECONFIG=/etc/kubernetes/admin.conf
```

`admin.conf` grants cluster-admin rights, and `super-admin.conf` bypasses RBAC entirely. Do not share either file; the official docs recommend moving `super-admin.conf` to a safe location.

Deploy Pod Network (Calico)

A CNI network add-on is required. CoreDNS does not start until one is installed. The steps below follow the current official Calico operator install for **Calico v3.33.0**, which is tested against Kubernetes 1.35, 1.36 and 1.37.

Install the Calico custom resource definitions (this manifest is for Kubernetes 1.36 or later):

```
kubectl create -f
https://raw.githubusercontent.com/projectcalico/calico/v3.33.0/manifests/v3_projectcalico_org.yaml
```

Install the Tigera Operator:

```
kubectl create -f https://raw.githubusercontent.com/projectcalico/calico/v3.33.0/manifests/tigera-operator.yaml
```

Download the custom resources manifest (iptables data plane):

```
curl -O https://raw.githubusercontent.com/projectcalico/calico/v3.33.0/manifests/custom-resources.yaml
```

Or using `wget`:

```
wget https://raw.githubusercontent.com/projectcalico/calico/v3.33.0/manifests/custom-resources.yaml
```

The manifest defaults to a `192.168.0.0/16` pod pool. Change the `cidr` setting to match the `--pod-network-cidr` you passed to `kubeadm init`:

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

Finally, create the Calico custom resources:

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

Watch the rollout until every component shows `True` in the `AVAILABLE` column (this takes a few minutes):

```
watch kubectl get tigerastatus
```

Step 7: Join Worker Nodes

After the control plane is initialized, join the worker nodes to the cluster. Kubernetes provides a join command that includes a token and the control plane's address.

Get Join Command on the Control Plane Node

If you didn't save the join command from `kubeadm init`, or its token has expired (the token created by `kubeadm init` is valid for 24 hours), generate a new one on the control plane node:

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

Run Join Command on Worker Nodes

Run the join command as root on each worker node. It looks like this:

```
kubeadm join <control-plane-host>:6443 --token <token> --discovery-token-ca-cert-hash sha256:<hash>
```

Verify Worker Node Join

Back on the control plane node, check that the workers have joined:

```
kubectl get nodes
```

This should list all nodes in the cluster: the control plane node and the newly joined workers. When Calico is running on a node, its status becomes **Ready**, which means it can accept workloads.

NGINX Test Deployment

To test your cluster, deploy a simple application such as an NGINX web server. Here's a sample YAML manifest that deploys 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 YAML above to a file named `nginx-deployment.yaml`, then apply it with `kubectl apply`:

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

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

This creates three NGINX pod replicas, each running a container that exposes port 80. To check the status of the deployment:

```
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:

```
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 running, expose NGINX outside the cluster with 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 YAML above to a file named `nginx-service.yaml`, then apply it:

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

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

This creates a Service of type LoadBalancer. To see its external IP address and ports:

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

Once you have the external IP address, open it in a web browser. You should see the default NGINX welcome page, which shows the cluster is serving web traffic.

On bare-metal or VM clusters (like this one), nothing provisions LoadBalancer services out of the box, so `EXTERNAL-IP` stays `<pending>` unless you install a load-balancer implementation. A LoadBalancer Service still gets a NodePort by default, though: take the port after `80:` in the `PORT(S)` column and browse to `http://<any-node-ip>:<node-port>`.

Sources

- Kubernetes: Installing kubeadm: <https://kubernetes.io/docs/setup/production-environment/tools/kubeadm/install-kubeadm/>
- Kubernetes: Container Runtimes: <https://kubernetes.io/docs/setup/production-environment/container-runtimes/>
- Kubernetes: Creating a cluster with kubeadm: <https://kubernetes.io/docs/setup/production-environment/tools/kubeadm/create-cluster-kubeadm/>
- Kubernetes: Ports and Protocols: <https://kubernetes.io/docs/reference/networking/ports-and-protocols/>
- Kubernetes: Releases and support periods: <https://kubernetes.io/releases/> (current stable: <https://dl.k8s.io/release/stable.txt>)
- Kubernetes: Swap memory management: <https://kubernetes.io/docs/concepts/cluster-administration/swap-memory-management/>

- Kubernetes: kubeadm token: <https://kubernetes.io/docs/reference/setup-tools/kubeadm/kubeadm-token/>
- Kubernetes: Service type LoadBalancer: <https://kubernetes.io/docs/concepts/services-networking/service/#loadbalancer>
- containerd: Getting started: <https://github.com/containerd/containerd/blob/main/docs/getting-started.md>
- Docker: Install Docker Engine on RHEL (repository and containerd.io): <https://docs.docker.com/engine/install/rhel/>
- CRI-O packaging: <https://github.com/cri-o/packaging/blob/main/README.md#usage>
- Calico: Install on on-premises deployments: <https://docs.tigera.io/calico/latest/getting-started/kubernetes/self-managed-onprem/onpremises>
- Calico: System requirements: <https://docs.tigera.io/calico/latest/getting-started/kubernetes/requirements>
- Calico: Configure NetworkManager: <https://docs.tigera.io/calico/latest/operations/troubleshoot/troubleshooting#configure-networkmanager>
- Calico: Typha overview: <https://docs.tigera.io/calico/latest/reference/typha/overview>
- Calico releases: <https://github.com/projectcalico/calico/releases>
- Fedora EPEL: Getting started: <https://docs.fedoraproject.org/en-US/epel/getting-started/>
- Red Hat Enterprise Linux release dates: <https://access.redhat.com/articles/3078>
- Rocky Linux news (9.8 release): <https://rockylinux.org/news>

Revision #1

Created 24 October 2026 14:10:46 by Admin

Updated 24 October 2026 14:10:46 by Admin