LEAP CLOUD 2025

LC2025 - Kube Installation

# 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:

sudo dnf 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:

sudo modprobe br\_netfilter

sudo modprobe ip\_vs

sudo modprobe ip\_vs\_rr

sudo modprobe ip\_vs\_wrr

sudo modprobe ip\_vs\_sh

sudo modprobe overlay

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/kubernetes.conf << EOF

br\_netfilter

ip\_vs

ip\_vs\_rr

ip\_vs\_wrr

ip\_vs\_sh

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/kubernetes.conf << EOF

net.ipv4.ip\_forward = 1

net.bridge.bridge-nf-call-ip6tables = 1

net.bridge.bridge-nf-call-iptables = 1

EOF

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:

sudo swapoff -a

This command turns off all swap devices.

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

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.

sudo dnf config-manager --add-repo https://download.docker.com/linux/centos/docker-ce.repo

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

sudo dnf makecache

Now, install the containerd.io package:

sudo dnf -y install containerd.io

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:

sudo sh -c "containerd config default > /etc/containerd/config.toml" ; cat /etc/containerd/config.toml

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

sudo vim /etc/containerd/config.toml

SystemdCgroup = true

sudo systemctl enable --now containerd.service

Reboot your machine.

sudo systemctl reboot

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

sudo systemctl status containerd.service

# Step 6: Set Firewall Rules

To allow specific ports used by Kubernetes components through the firewall, you can execute the following commands (on each node):

sudo firewall-cmd --zone=public --permanent --add-port=6443/tcp

sudo firewall-cmd --zone=public --permanent --add-port=2379-2380/tcp

sudo firewall-cmd --zone=public --permanent --add-port=10250/tcp

sudo firewall-cmd --zone=public --permanent --add-port=10251/tcp

sudo firewall-cmd --zone=public --permanent --add-port=10252/tcp

sudo firewall-cmd --zone=public --permanent --add-port=10255/tcp

sudo firewall-cmd --zone=public --permanent --add-port=5473/tcp

sudo firewall-cmd --reload

# 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:/v1.29/rpm/

enabled=1

gpgcheck=1

gpgkey=https://pkgs.k8s.io/core:/stable:/v1.29/rpm/repodata/repomd.xml.key

exclude=kubelet kubeadm kubectl cri-tools kubernetes-cni

EOF

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.

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

systemctl enable --now kubelet.service

# 

# 

**Master Node only**

# **Step 8: Initializing Kubernetes Control Plane**

Great! Let’s proceed with initializing the Kubernetes control plane on the master node. Here’s how we can do it:

sudo kubeadm config images pull

[config/images] Pulled registry.k8s.io/kube-apiserver:v1.29.3

[config/images] Pulled registry.k8s.io/kube-controller-manager:v1.29.3

[config/images] Pulled registry.k8s.io/kube-scheduler:v1.29.3

[config/images] Pulled registry.k8s.io/kube-proxy:v1.29.3

[config/images] Pulled registry.k8s.io/coredns/coredns:v1.11.1

[config/images] Pulled registry.k8s.io/pause:3.9

[config/images] Pulled registry.k8s.io/etcd:3.5.12-0

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.

sudo kubeadm init --pod-network-cidr=10.244.0.0/16

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

mkdir -p $HOME/.kube

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

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

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/v3.26.1/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/v3.26.1/manifests/custom-resources.yaml

Using wget:

wget https://raw.githubusercontent.com/projectcalico/calico/v3.26.1/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:

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

sudo kubeadm token create --print-join-command

kubeadm join 192.168.1.26:6443 --token kyy7v6.h26obyvthe08ohsn --discovery-token-ca-cert-hash sha256:cb67fddec41469cf1f495db34008ae1a41d3f24ce418b46d5aefb262a1721f43

**Worker Node**

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

sudo kubeadm join <MASTER\_IP>:<MASTER\_PORT> --token <TOKEN> --discovery-token-ca-cert-hash <DISCOVERY\_TOKEN\_CA\_CERT\_HASH>

Replace <MASTER\_IP>, <MASTER\_PORT>, <TOKEN>, and <DISCOVERY\_TOKEN\_CA\_CERT\_HASH> with the appropriate values from the join command generated on the master node.

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

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

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

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.