Air-Gapped Kubernetes and Docker Installation on an Offline Server

Overview

This guide outlines the process to install Kubernetes and Docker on an offline machine. It includes downloading the required Docker and Kubernetes packages, transferring them to the offline server, and performing the installation. Additionally, it covers pulling and loading Kubernetes images and setting up networking components like Flannel and Nginx Ingress Controller.

Prerequisites

Online machine with internet access.

Offline machine with SSH access.

Private SSH key for connecting to the offline server.

SCP enabled for file transfers between machines.

Steps

1. Prepare the Online Machine

Install Required Packages

On the online machine, update the package list and install the necessary packages for setting up Docker and SSHFS:

# Copy code

```
sudo apt-get update
sudo apt-get -y install \
  apt-transport-https \
  ca-certificates \
  curl \
  gnupg \
  lsb-release \
  sshfs
```

# Download Docker Packages

# Create a directory for Docker packages, add the Docker repository, and download the necessary packages:

# Copy code

```
mkdir -p ~/lr-airgap/docker-ce
cd ~/lr-airgap/docker-ce
```

# Add Docker's official GPG key and repository

```
sudo mkdir -p /etc/apt/keyrings
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o
/etc/apt/keyrings/docker.gpg
sudo apt-key adv --keyserver keyserver.ubuntu.com --recv-keys 7EA0A9C3F273FCD8

# set up the repository:

echo \
"deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker.gpg]
https://download.docker.com/linux/ubuntu \
$(lsb_release -cs) stable" | sudo tee /etc/apt/sources.list.d/docker.list >
/dev/null
```

# Update the package list and download Docker packages

```
sudo apt-get update
# if a GPG error when
sudo chmod a+r /etc/apt/keyrings/docker.gpg

sudo apt-get download docker-ce docker-ce-cli containerd.io docker-compose-plugin
# Ignore Warning like: "W: ... _apt ..."

#Docker Engine does not implement the CRI, it was removed from the kubelet in
version 1.24.
```

```
#https://github.com/Mirantis/cri-dockerd/releases
wget https://github.com/Mirantis/cri-dockerd/releases/download/v0.3.14/cri-
dockerd_0.3.14.3-0.ubuntu-jammy_amd64.deb
```

# Download Kubernetes Packages

# Create a directory for Kubernetes packages and download the required versions:

# Copy code

```
mkdir -p ~/lr-airgap/kube

cd ~/lr-airgap/kube
```

# Add the Kubernetes repository and update the package list

```
curl -fsSL https://pkgs.k8s.io/core:/stable:/v1.29/deb/Release.key | sudo gpg --
dearmor -o /etc/apt/keyrings/kubernetes-apt-keyring.gpg
echo 'deb [signed-by=/etc/apt/keyrings/kubernetes-apt-keyring.gpg]
https://pkgs.k8s.io/core:/stable:/v1.29/deb/ /' | sudo tee
/etc/apt/sources.list.d/kubernetes.list
sudo apt-get update
# To check latest available version
kubeadm config images list
```

# Download specific versions of Kubernetes packages

```
sudo apt-get download kubelet=1.29.5-1.1 kubeadm=1.29.5-1.1 kubectl=1.29.5-1.1
cri-tools=1.29.0-1.1 conntrack ebtables kubernetes-cni socat selinux-utils
```

# Package and Transfer the Files

# After downloading the Docker and Kubernetes packages, compress them into tar files for easier transfer:

# Copy code

```
cd
tar -czvf lr-d4r-k8s.tar.gz lr-airgap
```

2. Download Kubernetes Images

Pull the required Kubernetes images using Docker, then save them as tar files:

# Copy code

```
mkdir k8s-images
cd k8s-images
```

# Pull Kubernetes images

```
sudo docker pull registry.k8s.io/kube-apiserver:v1.29.5
sudo docker pull registry.k8s.io/kube-controller-manager:v1.29.5
sudo docker pull registry.k8s.io/kube-scheduler:v1.29.5
sudo docker pull registry.k8s.io/kube-proxy:v1.29.5
sudo docker pull registry.k8s.io/coredns/coredns:v1.11.1
sudo docker pull registry.k8s.io/pause:3.9
sudo docker pull registry.k8s.io/etcd:3.5.12-0
```

# Save images as tar files

```
sudo docker save registry.k8s.io/kube-apiserver:v1.29.5 > kube-
apiserver_v1.29.5.tar
sudo docker save registry.k8s.io/kube-controller-manager:v1.29.5 > kube-
controller-manager_v1.29.5.tar
sudo docker save registry.k8s.io/kube-scheduler:v1.29.5 > kube-
scheduler_v1.29.5.tar
sudo docker save registry.k8s.io/kube-proxy:v1.29.5 > kube-proxy_v1.29.5.tar
sudo docker save registry.k8s.io/coredns/coredns:v1.11.1 > coredns_v1.11.1.tar
sudo docker save registry.k8s.io/pause:3.9 > pause_3.9.tar
sudo docker save registry.k8s.io/etcd:3.5.12-0 > etcd_3.5.12-0.tar
```

3. Transfer Files to the Offline Machine

Using SCP, transfer all tar files from the online machine to the offline server:

# Copy code

```
scp -i "$KEY_FILE" lr-d4r-k8s.tar.gz k8s-images.tar.gz flannel.tar.gz
nginx.tar.gz ubuntu@"$OFFLINE_SERVER_IP":/home/ubuntu/
```

# Copy Files on Offline Node Server

```
scp -i "$KEY_FILE" lr-d4r-k8s.tar.gz k8s-images.tar.gz
ubuntu@"$OFFLINE_SERVER_IP":/home/ubuntu/
```

4. Install Docker and Kubernetes on the Offline Machine

SSH into the offline machine and execute the following steps to install Docker, Kubernetes, and load the images:

# On Master Server

# Copy code

```
ssh -i "$KEY_FILE" ubuntu@"$OFFLINE_SERVER_IP"
```

# Unpack the tar files

```
cd /home/ubuntu
```

```
tar -xzvf lr-d4r-k8s.tar.gz
tar -xzvf k8s-images.tar.gz
tar -xzvf flannel.tar.gz
tar -xzvf nginx.tar.gz
```

# Install Docker

```
cd lr-airgap/docker-ce
sudo dpkg -i *
```

```
sudo systemctl enable docker
sudo systemctl start docker
sudo systemctl enable cri-docker.service
sudo systemctl enable --now cri-docker.socket
sudo systemctl daemon-reload
sudo docker version
#sudo systemctl status docker
#sudo systemctl status cri-dockerd
sudo systemctl enable cri-docker.service
sudo systemctl enable --now cri-docker.socket
#sudo systemctl status cri-docker.service
sudo systemctl start cri-docker.service
#sudo systemctl status cri-docker.service
```

# Install Kubernetes

```
cd ../kube
sudo dpkg -i *
sudo apt-mark hold kubelet kubeadm kubectl
```

# Load Kubernetes images

```
cd ../../k8s-images
for x in *.tar; do
 sudo docker load < "$x" && echo "Loaded from file $x"
done
```

# Load Flannel and Nginx images

```
cd ~/flannel

sudo docker load < flannel-cni-plugin-v1.4.1-flannel1.tar
sudo docker load < flannel-v0.25.2.tar
```

```
cd ~/nginx
# unpack and load images
for x in *.tar; do
    sudo docker load < "$x" && echo "Loaded from file $x"
done
```

# On Worker Node

# Copy code

```
ssh -i "$KEY_FILE" ubuntu@"$Worker_OFFLINE_SERVER_IP
```

# Unpack the tar files

```
cd /home/ubuntu
tar -xzvf lr-d4r-k8s.tar.gz
tar -xzvf k8s-images.tar.gz
```

# Install Docker

```
cd lr-airgap/docker-ce
```

```
sudo dpkg -i *
```

```
sudo systemctl enable docker
sudo systemctl start docker
sudo systemctl enable cri-docker.service
sudo systemctl enable --now cri-docker.socket
sudo systemctl daemon-reload
sudo docker version
#sudo systemctl status docker
#sudo systemctl status cri-dockerd
sudo systemctl enable cri-docker.service
sudo systemctl enable --now cri-docker.socket
#sudo systemctl status cri-docker.service
sudo systemctl start cri-docker.service
#sudo systemctl status cri-docker.service
```

# Install Kubernetes

```
cd ../kube
sudo dpkg -i *
sudo apt-mark hold kubelet kubeadm kubectl
```

# Load Kubernetes images

```
cd ../../k8s-images
for x in *.tar; do
 sudo docker load < "$x" && echo "Loaded from file $x"
done
```

# Steps to Initialize the Kubernetes Cluster with kubeadm

Once you have installed Kubernetes components and prepared your system, follow these steps to initialize the Kubernetes control plane using kubeadm

1. Prepare the System

Ensure your system is configured correctly by disabling swap and configuring necessary kernel modules and sysctl parameters:

# Copy code

# Disable swap

```
sudo swapoff -a
```

# Ensure swap remains off after reboot

```
sudo sed -i '/ swap / s/^\(.*\)$/#\1/g' /etc/fstab
setenforce 0
sudo touch /etc/selinux/config

sudo bash -c 'cat <<EOF >  /etc/selinux/config
SELINUX=permissive
EOF'
```

# Set sysctl parameters required by Kubernetes

sudo modprobe overlay

sudo modprobe br_netfilter

# Apply sysctl settings for Kubernetes networking

```
sudo bash -c 'cat <<EOF >  /etc/sysctl.d/k8s.conf
net.bridge.bridge-nf-call-ip6tables = 1
net.bridge.bridge-nf-call-iptables = 1
EOF'
```

# Apply sysctl parameters

```
sudo sysctl --system
sudo systemctl enable kubelet.service
```

# Run Code From 1. Prepare the System on Node server also

# Run On Master

2. Create the kubeadm-config.yaml

You can customize your cluster setup by creating a kubeadm configuration file. Here's a basic example for initializing a control plane:

# Copy code

```
cat <<EOF >  kubeadm-config.yaml
# kubeadm-config.yaml
kind: ClusterConfiguration
apiVersion: kubeadm.k8s.io/v1beta3
kubernetesVersion: v1.29.5
---
kind: KubeletConfiguration
apiVersion: kubelet.config.k8s.io/v1beta1
cgroupDriver: systemd   # <--- driver
EOF
```

pod Subnet: This must match the pod network configuration you plan to use, such as Flannel or Calico. In the example above, it's set for Flannel (10.244.0.0/16).

3. Initialize the Kubernetes Control Plane

Run the kubeadm init command to initialize the control plane using your configuration file:

# Copy code

```
sudo kubeadm init --control-plane-endpoint="10.0.2.229:6443" --apiserver-
advertise-address=10.0.2.229 --pod-network-cidr=10.244.0.0/16  --cri-
socket=unix:///var/run/cri-dockerd.sock --skip-phases=preflight
```

If everything runs successfully, you should see output similar to this:

# Copy code

Your Kubernetes control-plane has been initiated 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
```

4. Set Up the kubectl Configuration

Set up your kubectl configuration to allow you to interact with the cluster:

# Copy code

# Set up kubeconfig for kubectl

```
 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 be able to run kubectl commands to interact with your cluster:

# Copy code

```
kubectl get nodes
```

5. Apply the Pod Network (e.g., Flannel)

After initializing the control plane, apply the pod network add-on (such as Flannel) to enable communication between nodes and pods:

# Copy code

```
kubectl apply -f
https://raw.githubusercontent.com/coreos/flannel/master/Documentation/kube-
flannel.yml
```

Once the network plugin is installed, the kube-dns pod (or CoreDNS in newer versions) should become available, and your cluster should be fully functional.

6. Join Worker Nodes (Optional)

To add worker nodes to the cluster, you will need the join command generated during the kubeadm init step. It looks something like this:

# Copy code

```
kubeadm join <master-ip>:6443 --token <token> --discovery-token-ca-cert-hash
sha256:<hash>
```

You can retrieve this join command later by running:

# Copy code

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

7. Verify the Cluster Setup

Check the cluster and pod network status:

# Copy code

```
kubectl get nodes

kubectl get pods -n kube-system
```

Control plane nodes should show a Ready status. System pods such as kube-apiserver, kube-controller-manager, etcd, and CoreDNS should be running.

Conclusion

Following these steps will help you set up a fully functional Kubernetes control plane using kubeadm. The next steps would involve adding worker nodes, deploying applications, and managing your cluster based on your requirements.