# Kubernetes Cluster Setup on Multiple Virtual Machines

This document will cover how to setup a multi-node Kubernetes cluster with persistent storage. In this guide we will use 3 VMs with Ubuntu 16.04 server and NFS as a persistent storage.

### Requirements

**Virtual Machines** – Three VMs with Ubuntu 16.04 Server pre-installed. VMs should be in the same network and communicate each other. See Ubuntu 16.04 KVM setup guide to setup a one-box with 3 KVMs.

**NFS Server -** See Setup NFS-server-kernel on the host machine to setup a NFS-Server\_Kernel on Ubuntu 16.04 in the one-box host machine.

#### Step 1. Install package on each vm

Run following commands in each VM to install necessary packages to setup Kubernetes.

|  |
| --- |
| sudo apt-get update && apt-get install -y docker.io apt-transport-https ebtables ethtool   curl -s <https://packages.cloud.google.com/apt/doc/apt-key.gpg> | sudo apt-key add -   sudo cat <<EOF >/etc/apt/sources.list.d/kubernetes.list  deb <http://apt.kubernetes.io/> kubernetes-xenial main  EOF   sudo apt-get update && apt-get install -y kubelet kubeadm kubectl nfs-common    sudo sysctl net.bridge.bridge-nf-call-iptables=1  sudo usermod -aG docker $USER |

#### Step 2: Start Master node using kubeadm

Run following commands on a VM as a Kubernetes master node.

|  |
| --- |
| sudo kubeadm init --pod-network-cidr=10.244.0.0/16 \  --apiserver-advertise-address=<IP address of VM>  mkdir -p $HOME/.kube   sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config   sudo chown $(id -u):$(id -g) $HOME/.kube/config |

Save the output of kubeadm init command. This will contain the kubeadm join command that will be used on the worked nodes later.

#### Step 3: POD network setup

In this guide, we will use weave-network for POD network.

|  |
| --- |
| export kubever=$(kubectl version | base64 | tr -d '\n')  kubectl apply -f "<https://cloud.weave.works/k8s/net?k8s-version=$kubever>" |

#### Step 4: Wait until all pods in Master nodes are up

Wait until all pods including kube-dns pods are all up and running.

|  |
| --- |
| kubectl get pods --all-namespaces -w |

#### Step 5: Join kubernetes nodes

After master node is setup on a VM, run following commands to join other VMs to the Kubernetes cluster.

|  |
| --- |
| sudo sysctl net.bridge.bridge-nf-call-iptables=1  kubeadm --join command you have saved from master's kubeadm --init command execution at Step 2. |

#### Step 6: Check the result

Run following command and verify all pods’ state is ‘RUNNING’

|  |
| --- |
| kubectl get nodes #all nodes should be READY  kubectl get pods --all-namespaces #all pods should be healthy and running |

### Setup a persistent volume provisioner (NFS based)

In this guide, we will demonstrate Persistent Storage using NFS and NFS volume provision service based on <https://github.com/kubernetes-incubator/external-storage/tree/master/nfs-client>

#### Step 1: configure nfs-client-provisioner yaml file

Download deployment.yml file by running following command

|  |
| --- |
| wget <https://raw.githubusercontent.com/erickangMSFT/sqldevops/master/docker_cluster/kubernetes/nfs-provisioner/deployment.yml> |

Open the downloaded deployment.yml file and update IP address for your NFS server.

|  |
| --- |
| kind: Deployment  apiVersion: extensions/v1beta1  metadata:  name: nfs-client-provisioner  spec:  replicas: 1  strategy:  type: Recreate  template:  metadata:  labels:  app: nfs-client-provisioner  spec:  containers:  - name: nfs-client-provisioner  image: quay.io/external\_storage/nfs-client-provisioner:latest  volumeMounts:  - name: nfs-client-root  mountPath: /persistentvolumes  env:  - name: PROVISIONER\_NAME  value: fuseim.pri/ifs  - name: NFS\_SERVER  value: <IP address of NFS server>  - name: NFS\_PATH  value: /k8svolumes  volumes:  - name: nfs-client-root  nfs: server: <IP address of NFS server>  path: /k8svolumes |

#### Step 2. Deploy nfs client provisioner

|  |
| --- |
| kubectl apply -f deployment.yml    kubectl apply -f <https://raw.githubusercontent.com/erickangMSFT/sqldevops/master/docker_cluster/kubernetes/nfs-provisioner/class.yml>    kubectl apply -f <https://raw.githubusercontent.com/erickangMSFT/sqldevops/master/docker_cluster/kubernetes/nfs-provisioner/auth/serviceaccount.yaml>    kubectl apply -f <https://raw.githubusercontent.com/erickangMSFT/sqldevops/master/docker_cluster/kubernetes/nfs-provisioner/auth/clusterrole.yaml>    kubectl apply -f   <https://github.com/erickangMSFT/sqldevops/blob/master/docker_cluster/kubernetes/nfs-provisioner/auth/clusterrolebinding.yaml>    kubectl patch deployment nfs-client-provisioner -p '{"spec":{"template":{"spec":{"serviceAccount":"nfs-client-provisioner"}}}}' |

# Appendix

## Ubuntu 16.04 KVM setup guide

### Section 1: Setting up host machine (one-time setup)

First step involves configuring host with kvm and virtinst packages. We will then setup a network bridge for VMs to communicate.

#### Step 1: Install Packages in the host machine

|  |
| --- |
| sudo su  apt-get install qemu-kvm libvirt-bin virtinst bridge-utils cpu-checker  modprobe vhost\_net   lsmod | grep vhost   echo vhost\_net >> /etc/modules |

#### Step 2: Get Network Device Name

Run 'ifconfig' or 'ls /sys/class/net' command to get the network device name of your host machine.

#### Step 3: Modify /etc/network/interfaces file

NOTE: This step is not required if the VMs can communicate over the network with each other. For example, if you have a hyper-visor switch that is used by all the VMs then you don’t need to setup a bridge.

Edit '/etc/network/interface' file. Make sure to change the network device name to a matching name in your /sys/class/net. In this sample, in this guide we use eno1.

/etc/network/interface file

|  |
| --- |
| # This file describes the network interfaces available on your system  # and how to activate them. For more information, see interfaces(5).   source /etc/network/interfaces.d/\*   # The loopback network interface  auto lo  iface lo inet loopback   auto br0  iface br0 inet dhcp  bridge\_ports <eno1: specify your network device name>  bridge\_stp off  bridge\_fd 0  bridge\_maxwait 0 |

#### Step 4: Restart networking service

Run following command:

|  |
| --- |
| sudo systemctl restart networking |

#### Step 5: install additional package to operate KVM environment

We will create VM images in /var/kvm/images folder.

|  |
| --- |
| sudo apt-get -y install libosinfo-bin libguestfs-tools virt-top  sudo mkdir -p /var/kvm/images |

### Section 2. Prepare a KVM image template for Kubernetes (one-time setup)

### Step 1: Create new KVM image from Ubuntu 16.04

|  |
| --- |
| sudo virt-install \  --name k8snode \  --ram 12288 \  --disk path=/var/kvm/images/k8snode.img,size=50 \  --vcpus 2 \  --os-type linux \  --os-variant ubuntu16.04 \  --network bridge=br0 \  --graphics none \  --console pty,target\_type=serial \  --location '<http://us.archive.ubuntu.com/ubuntu/dists/xenial/main/installer-amd64/>' \  --extra-args 'console=ttyS0,115200n8 serial' |

#### Step 2: Install Ubuntu

* Step 1 will enter the console interface of the newly created image. Continue the setup flow of ubuntu.
* For Disk partitioning, select 'Manual' and set the entire disk as '/' root directory and do not create a SWAP partition.
* For the feature selection, select OpenSSH server and Basic Ubuntu Server as minimum features.

#### Step 3: Exit from the virsh console

* After Step 2. the VM will restart and console will be empty since the terminal output is not correctly configured yet.
* Press CTRL+] to exit from the console.

#### Step 4: Shutdown KVM instance and configure the terminal

|  |
| --- |
| virsh shutdown k8snode    sudo guestmount -d k8snode -i /mnt    sudo ln -s /mnt/lib/systemd/system/getty@.service \ /mnt/etc/systemd/system/getty.target.wants/getty@ttyS0.service    sudo umount /mnt |

#### Step 5: Install docker, k8s required packages

|  |
| --- |
| virsh    start k8snode    console k8snode |

Run the following commands in k8snode console

k8snode console

|  |
| --- |
| sudo apt-get update && apt-get install -y docker.io apt-transport-https ebtables ethtool   curl -s <https://packages.cloud.google.com/apt/doc/apt-key.gpg> | sudo apt-key add -   sudo cat <<EOF >/etc/apt/sources.list.d/kubernetes.list  deb <http://apt.kubernetes.io/> kubernetes-xenial main  EOF   sudo apt-get update && apt-get install -y kubelet kubeadm kubectl nfs-common    sudo sysctl net.bridge.bridge-nf-call-iptables=1  sudo usermod -aG docker $USER |

You have finished the one-time setup to configure host machine and create KVM image template to provision Kubernetes cluster.

### Section 3. Clone VM images and run

Modify <yourname> in below script to an appropriate unique value.

|  |
| --- |
| sudo su    virt-clone --original k8snode --name master --file /var/kvm/images/master.img --replace  guest mount -d master -i /mnt  sed -i '2s/.\*/127.0.1.1 k8s-master-<yourname>/' /mnt//etc/hosts  sed -i '1s/.\*/k8s-master-<yourname>/' /mnt/etc/hostname  umount /mnt   virt-clone --original k8snode --name agent1 --file /var/kvm/images/agent1.img --replace  guestmount -d agent1 -i /mnt  sed -i '2s/.\*/127.0.1.1 k8s-agent1-<yourname>/' /mnt/etc/hosts  sed -i '1s/.\*/k8s-agent1-<yourname>/' /mnt/etc/hostname  umount /mnt  virt-clone --original k8snode --name agent2 --file /var/kvm/images/agent2.img --replace  guestmount -d agent2 -i /mnt  sed -i '2s/.\*/127.0.1.1 k8s-agent2-<yourname>/' /mnt/etc/hosts  sed -i '1s/.\*/k8s-agent2-<yourname>/' /mnt/etc/hostname  umount /mnt |

Run following command to start VMs:

|  |
| --- |
| virsh start master  virsh start agent1  virsh start agent2 |

## Setup NFS-server-kernel on the host machine

In this section, we will explore how to setup a persistent volume provisioner using NFS as an example. Persistent volume will be used later for provisioning of SQL Server vNext CTP1.

|  |
| --- |
| sudo apt-get install nfs-kernel-server    sudo mkdir -p /k8svolumes    sudo chown -R nobody:nogroup /k8svolumes    echo "/k8svolumes  <VM IP range xxx.xxx.0.0>/16(rw,async,no\_subtree\_check,no\_root\_squash)" | sudo tee -a /etc/exportfs    sudo exportfs -a |