K8s lab part 3– Adding a worker node

This guide sets up an additional K8s worker on VMware workstation that you can add to your k8s cluster.

**Remember**: Text in red is supposed to be executed, text in blue needs to be pasted in or added. Text in green is output from the system.

Steps:

1. Install Centos 7.8 on one or more VMs, depending on how many worker nodes you’ll be building. Configure the VM(s) exactly like the k8s-controller VM in lab 1 but name the worker(s) “k8s-workerx” (where “x” is a logical number 1,2,3 etc) and give them only 768MB of RAM.
2. Use MobaXterm to connect to the VM(s) through SSH. Use “MultiExec” if you need to send commands to more than one VM at the same time!
3. Install nano or your favorite editor in the workers:

yum install -y nano

1. Configure the hostname if you didn’t already during setup of the VM. Don’t forget to add the k8s-controller (and any other workers) to the host file as well:

hostnamectl set-hostname k8s-worker01

nano /etc/hosts

192.168.x.y k8s-worker1  
 192.168.x.z k8s-controller

1. Install docker:

yum check-update

yum install -y yum-utils device-mapper-persistent-data lvm2

yum-config-manager --add-repo <https://download.docker.com/linux/centos/docker-ce.repo>

yum install -y docker

docker --version

systemctl start docker

systemctl enable docker

systemctl status docker

1. Install Kubernetes:

sudo nano /etc/yum.repos.d/kubernetes.repo

[kubernetes]

name=Kubernetes

baseurl=https://packages.cloud.google.com/yum/repos/kubernetes-el7-x86\_64

enabled=1

gpgcheck=1

repo\_gpgcheck=1

gpgkey=https://packages.cloud.google.com/yum/doc/yum-key.gpg https://packages.cloud.google.com/yum/doc/rpm-package-key.gpg

yum install -y kubelet-1.19.2

yum install -y kubeadm-1.19.2

systemctl start kubelet

systemctl enable kubelet

systemctl status kubelet (Check “active” and “enabled” to make sure the kubelet is started)

1. Shut down firewalld:

systemctl disable firewalld

systemctl stop firewalld

1. Update/Create Iptables settings:

nano /etc/sysctl.d/k8s.conf

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

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

sysctl --system

1. Disable SELinux:

setenforce 0

sed -i 's/^SELINUX=enforcing$/SELINUX=permissive/' /etc/selinux/config

1. Disable swap:

sed -i '/swap/d' /etc/fstab

swapoff -a

1. Remove the CSI driver from the controller node. Make sure there are no PVCs left before you do!

**On the controller node:**

cd ~/csi-unity/dell-csi-helm-installer/

./csi-uninstall.sh --namespace unity

kubectl -n unity get pods ***(Make sure to wait until all pods are terminated)***

1. From UnityVSA, remove the controller host from the hosts list?
2. Now login to your **CONTROLLER NODE**. Execute the following command to retrieve the “join string”. Copy everything from “kubeadm join” all the way to the end:

kubeadm token create --print-join-command

1. Go back to your new **WORKER NODE(s)** and execute the join string you captured on each of the workers. You should see something like:

This node has joined the cluster:

\* Certificate signing request was sent to apiserver and a response was received.

\* The Kubelet was informed of the new secure connection details.

1. From your **CONTROLLER NODE** check if the worker node is visible. Note the “-o wide” for more details:

kubectl get nodes -o wide

NAME STATUS ROLES AGE VERSION INTERNAL-IP EXTERNAL-IP OS-IMAGE KERNEL-VERSION CONTAINER-RUNTIME

k8s-controller Ready master 4h13m v1.19.2 192.168.0.10 <none> CentOS Linux 7 (Core) 3.10.0-1127.el7.x86\_64 docker://1.13.1

k8s-worker01 Ready <none> 2m41s v1.19.2 192.168.0.11 <none> CentOS Linux 7 (Core) 3.10.0-1127.el7.x86\_64 docker://1.13.1

1. Finally we need to make sure the new worker node is ready to work with the CSI driver. For that to work we need to setup some things like the iscsi driver and rpcbind (for NFS) on the **WORKER NODE(s)**:

yum install -y iscsi-initiator-utils

yum install -y device-mapper-multipath

systemctl enable iscsi

yum install -y rpcbind

systemctl start rpcbind

systemctl enable rpcbind

1. Taint the controller node back so that pods no longer get scheduled on the controller (but just the worker(s) ):

kubectl taint node k8s-controller node-role.kubernetes.io/master:NoSchedule

1. To make sure the snapshot pod runs on one of the workers (and not the controller anymore), just delete the pod. It will automatically be rescheduled on a worker node:

kubectl delete pod snapshot-controller-0

1. In order to get the worker node supported in the CSI driver, you need to reinstall the CSI driver from the **CONTROLLER NODE**:

./csi-install.sh --namespace unity --values ./myvalues.yaml

kubectl -n unity get pods **(Should show TWO or MORE unity-node-xxxx ‘s running)**

All done! You should now have your Kubernetes single-node setup converted into an actual cluster with separated controller and worker functionality!