Setting up Kubernetes cluster on local servers

This document is divided into six sections: setting up the OS, setting up cluster, creating default storage class, accessing azure private repository, configure load balancer, terraform changes. All commands are written for the ubuntu operating system.

Section 1: setting up OS

The following step should be done on all servers used to create this cluster:

- ❖ Create a new partition (sda4) 800Gb (approximately) with cfdisk

```
sudo cfdisk /dev/sda
```

The following steps should be done on network block device servers:

- ❖ Set up nbd-servers
- Install nbd-server

```
sudo apt install nbd-server
```

- Paste this following configuration in /etc/nbd-server/config

```
[generic]
# If you want to run everything as root rather than the
nbd user, you
# may either say "root" in the two following lines, or
remove them
# altogether. Do not remove the [generic] section,
however.
#       user = nbd
#       group = nbd
        includedir = /etc/nbd-server/conf.d
        allowlist = true
```

```
# What follows are export definitions. You may create
as much of them as
# you want, but the section header has to be unique.
[partition1]
  exportname = /dev/sda4
```

- Run the following command to restart nbd-server

```
sudo systemctl restart nbd-server.service
```

The following steps should be done on the network block device client:

❖ Set up nbd-client:

- Install nbd-client

```
sudo apt install nbd-client
sudo modprobe nbd
```

- Load module on boot by pasting the following configuration into the file /etc/modules-load.d/modules.conf:

```
# /etc/modules: kernel modules to load at boot time.
#
# This file contains the names of kernel modules that
should be loaded
# at boot time, one per line. Lines beginning with "#"
are ignored.
nbd
```

- Create nbd partition by running the following command on the nbd-client for each nbd-server:

```
sudo nbd-client <IP address of server> -N <partition1>
```

The following steps should be done on network block device client:

- ❖ Create a filesystem that combines all created partitions
- Create physical volumes out of all partitions created in the above step

```
sudo pvcreate /dev/nbd0 /dev/nbd1 /dev/sda4
```

- Create volume group consisting of all the physical volumes

```
sudo vgcreate shareddisk /dev/nbd0 /dev/nbd1 /dev/sda4
```

- Create logical volume

```
sudo lvcreate -L <size> /dev/shareddisk --name
"sharedVolume"
```

- Format volume to ext4 filesystem and mount it to virtual directory

```
sudo mkfs.ext4 /dev/shareddisk/sharedvolume
sudo mount /dev/shareddisk/sharedvolume
/sharedfilesystem
```

Section 2: setting up Cluster

The following steps should be done on all servers used to create this cluster:

- ❖ Install containerd container runtime:
- Copy this code block to a file called install-containerd.sh

```
cat <<EOF | sudo tee /etc/modules-load.d/k8s.conf
overlay
br_netfilter
EOF

sudo modprobe overlay
sudo modprobe br_netfilter
```

```
# sysctl params required by setup, params persist across reboots
cat <<EOF | sudo tee /etc/sysctl.d/k8s.conf
net.bridge.bridge-nf-call-iptables  = 1
net.bridge.bridge-nf-call-ip6tables = 1
net.ipv4.ip_forward                 = 1
EOF

# Apply sysctl params without reboot
sudo sysctl --system

sudo apt-get update
sudo apt-get -y install containerd

sudo mkdir -p /etc/containerd
containerd config default | sudo tee /etc/containerd/config.toml
sudo sed -i 's/SystemdCgroup \= false/SystemdCgroup \= true/g'
/etc/containerd/config.toml
sudo systemctl restart containerd
```

- Use this command to set permissions to the file:

```
chmod +rwx install-containerd.sh
```

- Use this command to run the bash script:

```
./install-containerd.sh
```

❖ Install k8s components:

- Copy this code block to a file called install-kube-components.sh

```
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

curl -fsSL https://pkgs.k8s.io/core:/stable:/v1.29/deb/Release.key | sudo gpg --
dearmor -o /etc/apt/keyrings/kubernetes-apt-keyring.gpg

sudo apt-get update

sudo apt-get install --allow-change-held-packages -qy kubelet=1.29.3-1.1
kubectl=1.29.3-1.1 kubeadm=1.29.3-1.1
```

- Use this command to set permission to the file:

```
chmod +rwx install-kube-components.sh
```

- Use this command to run the bash script:

```
./install-kube-components.sh
```

The following steps should only be run on the master server:

- ❖ Disable swap memory:

```
sudo swapoff -a
```

- ❖ Set up the master processes:

```
sudo kubeadm init
```

- ❖ Move the config file to ~/.kube/config:

```
cp /etc/kubernetes/admin.conf ~/.kube/config
```

- ❖ Change the name of the context used in the config file. In my case, it is called demo-server:

```
contexts:
- context:
    cluster: kubernetes
    user: kubernetes-admin
  name: demo-server
current-context: demo-server
```

- ❖ Install weave net daemon set:

```
$ kubectl apply -f https://github.com/weaveworks/weave/releases/download/v2.8.1/weave-daemonset-k8s.yaml
```

- ❖ Run this command to print the join command that is going to be used on the worker nodes:

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

The following steps should only be run on the worker nodes:

- ❖ Run the join command command attained from the previous step it should look something like this:

```
$ sudo swapoff -a
$ kubeadm join <token>
```

Section 3: Create a default storage class

The following commands should be run on the master server:

- Install the nfs-kernel-server package, which allows you to share directories:

```
sudo apt update
sudo apt install nfs-kernel-server
```

- Create a directory that stores all of our persistent volumes and change permissions:

```
sudo chown nobody:nogroup /sharedfilesystem
```

- Edit the /etc/exports file and add this line to the file:

```
/sharedfilesystem *(rw,sync,no_subtree_check,no_root_squash,no_all_squash,insecure)
```

- Start the nfs server:

```
sudo systemctl restart nfs-kernel-server
```

- Export /srv/nfs/kubedata directory:

```
sudo exportfs -rav
```

The following commands should be run on each of the worker servers:

- Install the nfs-common package:

```
sudo apt update
sudo apt install nfs-common
```

The following steps should be run on the master server:

- Create the nfs-subdir-external provisioner through helm. In order to install helm visit this website: Helm | Installing Helm

```
$ helm repo add nfs-subdir-external-provisioner https://kubernetes-sigs.github.io/nfs-subdir-external-provisioner/
$ helm install nfs-subdir-external-provisioner nfs-subdir-external-provisioner/nfs-subdir-external-provisioner \
```

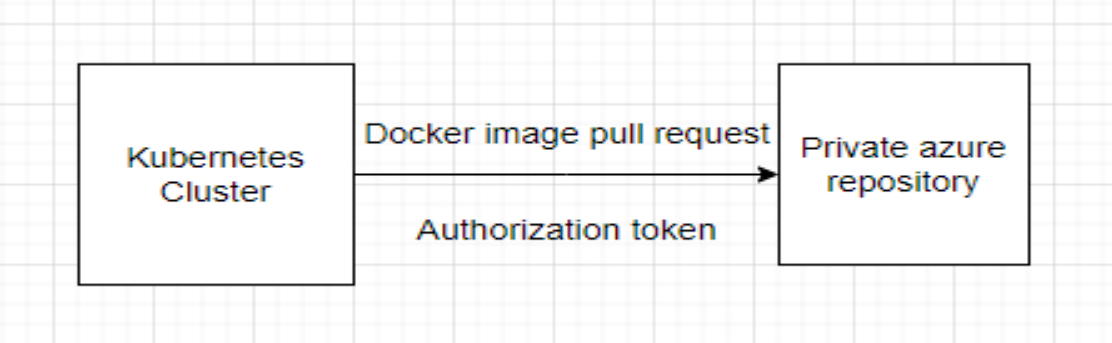
```
    --set nfs.server=192.168.66.128 \
    --set nfs.path=/sharedfilesystem
```

- ❖ Make storage class default by using this command:

```
kubectl patch storageclass nfs-client -p '{"metadata":
{"annotations":{"storageclass.kubernetes.io/is-default-class":"true"}}}'
```

Section 4: Accessing private azure repository from cluster

Since the docker images we want to access are in a private azure repository, we will be passing an authorization token along with the image pull requests to the repository.



To accomplish this, in this section, we will be storing the authorization token as a secret in the cluster. In the next section, we will use this secret to pull images from the repository.

Run the following commands on the master node:

```
$ kubectl create secret -n demo-server docker-registry acr-auth --docker-
server=xvisorwestus2.azurecr.io --docker-username=xvisorwestus2 --docker-
password=<password>
```

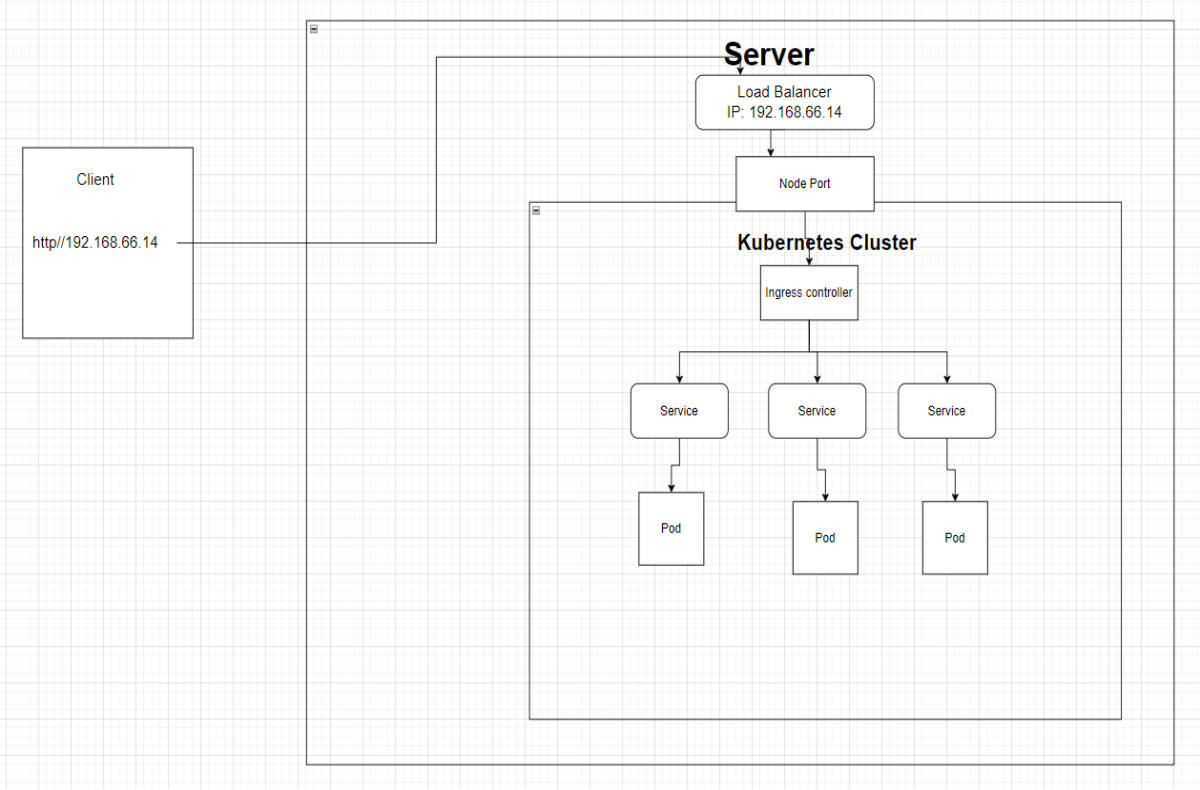
In the above command, replace <namespace> with the namespace you would like to create the secret in. In my case, I have created the

secret in ‘demo’ namespace. After completion, it should look something like this:

```
tnagi@marble:~$ kubectl -n demo get secrets
NAME                                  TYPE                                  DATA   AGE
acr-auth                              kubernetes.io/dockerconfigjson        1      20h
```

Section 5: Configure load balancer

Down below is the high level ingress route structure. In the steps, we will go through advertising ip addresses for load balancers:



Run the following commands on the master node:

- ❖ Install metallb by applying this manifest:

```
kubectl apply -f
https://raw.githubusercontent.com/metallb/metallb/v0.14.4/config/manifests/
metallb-native.yaml
```

- Create manifest file called ipaddressPool.yaml with the following configuration (change the address pool depending on the load balancer IP address you wish to use).

```
apiVersion: metallb.io/v1beta1
kind: IPAddressPool
metadata:
    name: first-pool
    namespace: metallb-system
spec:
  addresses:
  - 192.168.66.14-192.168.66.15
```

- Create manifest file called ipaddressAdvertisement.yaml

```
apiVersion: metallb.io/v1beta1
kind: L2Advertisement
metadata:
  name: example
  namespace: metallb-system
spec:
  ipAddressPools:
    - first-pool
```

- Run the following commands to create and then advertise the ip address pool:

```
kubectl apply -f ipaddressPool.yaml
kubectl apply -f ipaddressAdvertisement.yaml
```

- Change the loadBalancerIp value under shared/values/traefik-values.yaml to match your ip address pool.

```
service:
  spec:
    loadBalancerIP: 192.168.66.14
```

Section 6: Terraform code changes

- In the providers.tf file, change the backend to local, since we will not be using the cloud to store the state.

```
terraform {
  backend "local" {
    path = "/tmp/terraform.tfstate"
  }
```

- In the providers.tf file, change the 'config_context' for both helm and kubernetes to the name of your new context. In my case, it is called 'demo-server'.

```
provider "helm" {
  kubernetes {
    config_path    = "~/.kube/config"
    config_context = "demo-server"
  }
}


provider "kubernetes" {
  config_path    = "~/.kube/config"
  config_context = "demo-server"
  experiments {
    manifest_resource = true
  }
}
```

- **Remove** the 'node_selector' fields that point to the nodes on the cloud. The kubernetes scheduler master process will run its scheduler algorithm to pick the node to schedule the pod. However, if you want to strictly schedule a pod on a certain node in the cluster, use this format:

```
"nodeSelector" = {
  "kubernetes.io/hostname" = "worker-node-1"
}
```

- Delete all 'tolerations' from all files

- In all resources of type ‘kubernetes_deployment’, create an ‘image_pull_secrets’ field which uses the secret that we have created in the last section to authorize pulling docker images from the private azure repository.