DATA SCIENCE CAPSTONE PROJECT

MACHINE LEARNING GPU CLUSTERING

Implementation

Setting up Kubernetes

STEP 1

Disabling the firewall

ufw disable

STEP 2

Disabling Swap

```
swapoff -a
sed -i '/swap/d' /etc/fstab
```

STEP 3

Update sysctl settings for Kubernetes networking

```
cat >>/etc/sysctl.d/kubernetes.conf<<EOF
net.bridge.bridge-nf-call-ip6tables = 1
net.bridge.bridge-nf-call-iptables = 1
EOF
sysctl --system
```

STEP 4

Install docker engine

```
apt install -y apt-transport-https ca-certificates curl gnupg-agent software-properties-common
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | apt-key add -
add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu $(lsb_release -cs) stable"
apt update
```

apt install -y docker-ce=5:19.03.10~3-0~ubuntu-focal containerd.io

STEP 5

Kubernetes Setup

Add Apt repository

curl -s https://packages.cloud.google.com/apt/doc/apt-key.gpg | apt-key add -
echo"deb https://apt.kubernetes.io/ kubernetes-xenial main" > /etc/apt/sources.list.d/kubernetes.list

STEP 6

Install Kubernetes components

apt update
apt install -y kubeadm=1.18.5-00 kubelet=1.18.5-00 kubectl=1.18.5-00

ONLY ON K-MASTER

Initialize Kubernetes Cluster

Update the below command with the ip address of kmaster

kubeadm init --apiserver-advertise-address=172.16.16.100 --pod-network-cidr=192.168.0.0/16 --ignore-preflight-errors=all

Deploy Calico network

kubectl --kubeconfig=/etc/kubernetes/admin.conf create -f https://docs.projectcalico.org/v3.14/manifests/calico.yaml

Cluster join command

kubeadm token create --print-join-command

To be able to run kubectl commands as non-root user

If you want to be able to run kubectl commands as non-root user, then as a non-root user perform these

```
mkdir -p $HOME/.kube
sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
sudo chown $(id -u):$(id -g) $HOME/.kube/config
```

On Kworker

Join the cluster

Use the output from **kubeadm token create** command in previous step from the master server and run here.

Use kubectl get nodes command to check if the nodes are ready.

Installing helm

Run the helm installer.

curl https://raw.githubusercontent.com/helm/helm/master/scripts/get-helm-3 | bash

Verify helm.

helm list

Setting up NFS Subdir External Provisioner

Install NFS Kernel Server in Ubuntu

sudo apt update

sudo apt install nfs-kernel-server

Create NFS Export Directory

sudo mkdir -p /srv/nfs/kubedata

sudo chown -R nobody:nogroup /srv/nfs/kubedata

Edit the etc exports file

sudo vi /etc/exports

add the following line

/srv/nfs/kubedata *(rw,sunc,no_subtree_check,no_root_squash,no_all_sqaush,insecure)

Save the file

Enable and start the nfs server

sudo systemctl enable --now nfs-server

sudo exportfs -rav

Run sudo showmount -e localhost it tells you what directories are being exported through the nfs

To verify login to your Kubernetes nodes check if the nodes can mount nfs volumes

apt install -y nfs-common

mount -t nfs ip_address:/srv/nfs/kubedata

check the mount and unmount it

umount /mnt

Check if the system already have nfs server if not configure nfs server

Helm install NFS subdir external provisioner

First add helm repo

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

Install NFS subdir external provisioner

helm install nfs-subdir-external-provisioner nfs-subdir-external-provisioner/nfs-subdir-external-provisioner \ --set nfs.server=x.x.x.x \ --set nfs.path=/exported/path

Run kubectl get pods to check nfs pod is running

Set up Metallb

We have to enable strict ARP mode

kubectl edit configmap -n kube-system kube-proxy

set strictARP as true

apiVersion: kubeproxy.config.k8s.io/v1alpha1

kind: KubeProxyConfiguration

mode: "ipvs"

ipvs:

  strictARP: true

Run the following commands

kubecltl apply -f https://raw.githubusercontent.com/metallb/metallb/v0.9.6/manifests/namespace.yaml
kubecltl apply -f https://raw.githubusercontent.com/metallb/metallb/v0.9.6/manifests/metallb.yaml

Create metal_config.yaml using the following file

vi metal_config.yaml

Replace PUBLIC_IP_ADDRESS with the actual public ip address of your instance

apiVersion: v1

kind: ConfigMap

metadata:

  namespace: metallb-system

  name: config

data:

  config: |

    address-pools:

    - name: default

      protocol: layer2

      addresses:

      - PUBLIC_IP_ADDRESS-PUBLIC_IP_ADDRESS

Apply custom metal load balancer configuration — metal_config.yaml

kubectl apply -f metal_config.yaml

Check the metal load balancer status

kubectl --namespace=metallb-system get pods

Setting up Jupyterhub

Prepare configuration file

1. Generate a random hex string representing 32 bytes to use as a security token. Run this command in a terminal and copy the output:

openssl rand -hex 32

2. Create and start editing a file called config.yaml. In the code snippet below we start the widely available nano editor

nano config.yaml

3. Write the following into the config.yaml file but instead of writing <RANDOM-HEX> paste the generated hex string you copied in step 1.

```
proxy:
  secretToken: "<RANDOM_HEX>"
hub:
  db:
    type: sqlite-memory
singleuser:
  storage:
    dynamic:
      storageClass: nfs-client
```

1. Save the config.yaml file. In the nano editor this is done by pressing **CTRL+X** or **CMD+X** followed by a confirmation to save the changes.

Install JupyterHub

1. Make Helm aware of the JupyterHub Helm chart repository so you can install the JupyterHub chart from it without having to use a long URL name.

helm repo add jupyterhub https://jupyterhub.github.io/helm-chart/helm repo update

2. Now install the chart configured by your config.yaml by running this command from the directory that contains your config.yaml:

RELEASE=jhub

NAMESPACE=jhub

helm upgrade --cleanup-on-fail \ --install $RELEASE jupyterhub/jupyterhub \ --namespace $NAMESPACE \ --create-namespace \ --version=0.10.6 \ --values config.yaml

3. While Step 2 is running, you can see the pods being created by entering in a different terminal:

kubectl get pod --namespace jhub

4. To remain sane we recommend that you enable autocompletion for kubectl and set a default value for the --namespace flag:

kubectl config set-context $(kubectl config current-context) --namespace ${NAMESPACE:-jhub}

5. Find the IP we can use to access the JupyterHub. Run the following command until the EXTERNAL-IP of the proxy-public service is available like in the example output.

Kubectl get service --namespace jhub

Your jupyterhub setup is ready

References

https://github.com/justmeandopensource/kubernetes/blob/master/docs/install-cluster-ubuntu-20.md

https://github.com/kubernetes-sigs/nfs-subdir-external-provisioner

https://zero-to-jupyterhub.readthedocs.io/en/stable/jupyterhub/installation.html

https://georgepaw.medium.com/jupyterhub-with-kubernetes-on-single-bare-metal-instance-tutorial-67cbd5ec0b00

https://metallb.universe.tf/installation/

https://dev.to/upindersujlana/upgrade-kubernetes-cluster-to-1-19-4-using-kubeadm-3ien

https://www.tecmint.com/install-nfs-server-on-ubuntu/

https://www.youtube.com/watch?v=DF3v2P8ENEg&t=188s