Kubernetes Installation and Setup

VATSAL MORADIYA

SOLUTION ARCHITECT

Contents

1 Pre-requisites

This setup shows how to install the Kubernetes(v1.28) on Ubuntu 22.04 or Ubuntu 20.04. Following are system requirements on each node:

1. Minimum **2GB RAM** or more
2. Minimum **2 CPU cores** / or 2 vCPU
3. **20 GB** free disk space on /var or more
4. **sudo user** with admin rights
5. Internet connectivity on each node

2 System Configuration Settings

For latest Kubernetes version, the supported container runtime is *containerd*.

```
sudo swapoff -a
## Comment the /swap line in /etc/fstab
```

Load the following kernel modules on all the nodes,

```
sudo tee /etc/modules-load.d/containerd.conf <<EOF
overlay
br_netfilter
EOF

sudo modprobe overlay
sudo modprobe br_netfilter
```

Set the following Kernel parameters for Kubernetes, and restart the services.

```
sudo tee /etc/sysctl.d/kubernetes.conf <<EOT
net.bridge.bridge-nf-call-ip6tables = 1
net.bridge.bridge-nf-call-iptables = 1
net.ipv4.ip_forward = 1
EOT

sudo sysctl --system
```

3 Installing Container Runtime

Install dependencies for containerd.

```
sudo apt update
sudo apt install -y curl gnupg2 software-properties-common \
apt-transport-https ca-certificates
```

Download and enable Docker Repository

```
sudo curl -fsSL https://download.docker.com/linux/ubuntu/gpg \
| sudo gpg --dearmour -o /etc/apt/trusted.gpg.d/docker.gpg

sudo add-apt-repository "deb [arch=amd64] \
https://download.docker.com/linux/ubuntu $(lsb_release -cs) stable"
```

Install containerd and create configuration file.

```
sudo apt update
sudo apt install -y containerd.io

containerd config default | sudo tee \
/etc/containerd/config.toml >/dev/null 2>&1

## Find the SystemdCgroup in the /etc/containerd/config.toml
## and update it to true (if false).
```

Download the NVIDIA Container Toolkit from the following link: nvidia-container-toolkit

```
## Do this on the master node and all the worker nodes with GPU.

sudo nvidia-ctk runtime configure --runtime=containerd

## Find the default_runtime_name in the /etc/containerd/config.toml
## and update it to "nvidia" (if "runc").

sudo systemctl restart containerd
```

4 Install kubectl, kubeadm, kubelet

Install dependencies for Kubernetes.

```
sudo apt-get update
sudo apt-get install -y apt-transport-https ca-certificates curl
```

Download public signing key and add kubernetes to apt repository.

```
## In systems with Ubuntu 20.04 or older, create the keyrings folder.
sudo mkdir -m 755 -p /etc/apt/keyrings

curl -fsSL https://pkgs.k8s.io/core:/stable:/v1.28/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:/v1.28/deb/ /" | \
sudo tee /etc/apt/sources.list.d/kubernetes.list
```

Install Kuberenetes components

```
sudo apt update

## To install the latest version,
sudo apt install kubectl kubeadm kubelet

## Check the version
sudo kubeadm version
```

Make containerd as runtime endpoint in the service.

```
systemctl status kubelet
## Find the path 10-kubeadm.conf from the output of the last command.

sudo vi <path/to/10-kubeadm.conf>
## Add the following line to the configuration file (This is a single line)
Environment="KUBELET_EXTRA_ARGS=--container-runtime-endpoint=unix://
/var/run/containerd/containerd.sock"

sudo systemctl daemon-reload
sudo systemctl restart kubelet.service
```

Reboot all the system after completing the installation process.

5 Create a Cluster

To initialize the Kubernetes Cluster Components,

```
sudo su -

## Initialize the cluster
kubeadm init --pod-network-cidr=10.244.0.0/16

## Export the config file as environment variable
echo "export KUBECONFIG=/etc/kubernetes/admin.conf" >> ~/.bashrc
source ~/.bashrc

## Create a networking pod using Flannel
kubectl apply -f https://raw.githubusercontent.com/\
coreos/flannel/master/Documentation/kube-flannel.yml

## Check the status of all pods
watch kubectl get pods -A
```

Generate the token to join worker nodes to the cluster

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

Copy the command printed on the terminal and paste it on the worker nodes to join.

```
## Example (To be done on the worker nodes)
kubeadm join 192.168.1.155:6443 --token 0c51pa.vdyedw37xdxxnwg8 \
--discovery-token-ca-cert-hash \
56:484497f05be227cc83fe961fc3f7bde887e199cb7bf1a643c04c7034987c9a03
```

Verify all the nodes are joined and ready

```
kubectl get nodes
```

6 Enable and Detect GPU on GPU nodes

Install Helm Packages

```
curl https://raw.githubusercontent.com/helm/helm/main/scripts/get-helm-3 \
| bash
```

Enable GPU Support for Kubernetes using Helm

```
helm repo add nvdp https://nvidia.github.io/k8s-device-plugin
helm repo add nvgfd https://nvidia.github.io/gpu-feature-discovery
helm repo update
```

1. When MIG is disabled.

```
helm install \
   --version=0.14.3 \
   --generate-name \
   --namespace gpu-operator \
   --create-namespace \
   --set migStrategy=none \
   nvdp/nvidia-device-plugin

helm install \
   --version=0.8.2 \
   --generate-name \
   --namespace gpu-operator \
   --create-namespace \
   --set migStrategy=none \
   nvgfd/gpu-feature-discovery
```

2. When MIG is enabled and all the MIG instances are of same configuration

```
helm install \
   --version=0.14.3 \
   --generate-name \
   --namespace gpu-operator \
   --create-namespace \
   --set migStrategy=single \
   nvdp/nvidia-device-plugin
```

```
helm install \
   --version=0.8.2 \
   --generate-name \
   --namespace gpu-operator \
   --create-namespace \
   --set migStrategy=single \
   nvgfd/gpu-feature-discovery
```

3. When MIG is enabled and the MIG instances are of multiple configurations

```
helm install \
   --version=0.14.3 \
   --generate-name \
   --namespace gpu-operator \
   --create-namespace \
   --set migStrategy=mixed \
   nvdp/nvidia-device-plugin

helm install \
   --version=0.8.2 \
   --generate-name \
   --namespace gpu-operator \
   --create-namespace \
   --set migStrategy=mixed \
   nvgfd/gpu-feature-discovery
```

7 Adding User with RBAC Authentication

Copy the k8s folder to your system in the zip file along with.

```
cp -r <path_to_k8s> ./
cd k8s

chmod +x authentication.sh
./authentication.sh
## Enter the username and validity for the user in the cluster.
## Make sure the user exists on all the nodes with same uid and gid.
```

Create a ClusterRole, RoleBinding and ResourceQuota for the user

```
## Modify the userClusterRole.yaml file for vairous user permissions.
kubectl apply -f userClusterRole.yaml

## Fill the <username> in the userRoleBinding.yaml
kubectl apply -f userRoleBinding.yaml

## Modify the userQuota.yaml file with the restrictions on the user.
## Fille the <username> in the userQuota.yaml file.
kubectl apply -f userResourceQuota.yaml
```

8 Deploying a Pod in the cluster from a User

Copy the files created for RBAC Authentication to the user's home folder

```
mkdir -p <user_home_dir>/.kube
cp UserAuthentication/<username>/* <user_home_dir>/.kube/
cp example.yaml <user_home_dir>/
chown -R <username>:<username> <user_home_dir>

su - <username>
## Update the KUBECONFIG environment variable for the user
echo "export KUBECONFIG=\$HOME/.kube/\$(whoami).config" >> ~/.bashrc

source ~/.bashrc
```

Run the `example.yaml` to create a Pod with PyTorch container from NGC

```
## Modify the quotas section and paths if needed as per the requirement.
kubectl apply -f example.yaml
```

Launch a JupyterLab to access the container.

```
kubectl get pods
kubectl exec -it torch-example -- /bin/bash

jupyter-lab --ip=0.0.0.0 --allow-root --NotebookApp.token="123" --port=8888
```

Access the Jupyter Lab from the web browser using the link `http://<system-ip>:8888` and the token "123".