University of New Haven

DATA SCIENCE CAPSTONE PROJECT

MULTITENANT GPU CLUSTERING

BY

RAHUL AKKINENI

YASASWINI SURYADEVARA

UNDER THE GUIDANCE OF DR. VAHID BEHZADAN

DOCUMENTATION

OBJECTIVE

The main objective of the project is to setup a GPU Cluster that can provide multitenancy. Clustering workstations with GPU and providing GPU to the users as per their requirement. To obtain this objective we have set a list of goals to accomplish step by step. The goals are:

1)Setting up kubernetes on bare metal

2)Setting up jupyter hub on bare metal

3)Providing user authentication and resource management

5)Providing GPU to the users as per their requirement

KUBERNETES

Kubernetes (also known as k8s or "kube") is an open-source container orchestration platform that automates many of the manual processes involved in deploying, managing, and scaling containerized applications.

In other words, you can cluster together groups of hosts running Linux containers, and Kubernetes helps you easily and efficiently manage those clusters.

Kubernetes clusters can span hosts across on-premises, public, private, or hybrid clouds. For this reason, Kubernetes is an ideal platform for hosting cloud-native applications that require rapid scaling, like real-time data streaming through Apache Kafka. Modern applications are increasingly built using containers, which are microservices packaged with their dependencies and configurations. Kubernetes is open-source software for deploying and managing those containers at scale—and it’s also the Greek word for helmsmen of a ship or pilot. Build, deliver, and scale containerized apps faster with Kubernetes, sometimes referred to as “k8s” or “k-eights.”

Kebernetes Terminology:

As is the case with most technologies, language specific to Kubernetes can act as a barrier to entry. Let's break down some of the more common terms to help you better understand Kubernetes.

Control plane: The collection of processes that control Kubernetes nodes. This is where all task assignments originate.

Nodes: These machines perform the requested tasks assigned by the control plane.

Pod: A group of one or more containers deployed to a single node. All containers in a pod share an IP address, IPC, hostname, and other resources. Pods abstract network and storage from the underlying container. This lets you move containers around the cluster more easily.

Replication controller: This controls how many identical copies of a pod should be running somewhere on the cluster.

Service: This decouples work definitions from the pods. Kubernetes service proxies automatically get service requests to the right pod—no matter where it moves in the cluster or even if it's been replaced.

Kubelet: This service runs on nodes, reads the container manifests, and ensures the defined containers are started and running.

kubectl: The command line configuration tool for Kubernetes.

Storage:

Kubernetes storage is based on the concepts of volumes: there are ephemeral volumes that are often simply called "volumes" and there are "Persistent Volumes" that are meant for long-term storage.

Volumes (Ephemeral):

Volumes are the basic storage unit in kubernetes. Volumes decouple the storage from the container and tie it to the pod.

Persistent Volumes:

Kubernetes persistent volumes or PVs used to be pieces of network attached storage but can be local storage in the worker node now where the pod is running as well with the addition of local persistent volumes in kubernetes 1.10.

There are two kinds of PV provisioning:

1. Manual Provisioning

2. Dynamic Provisioning

Manual Provisioning:

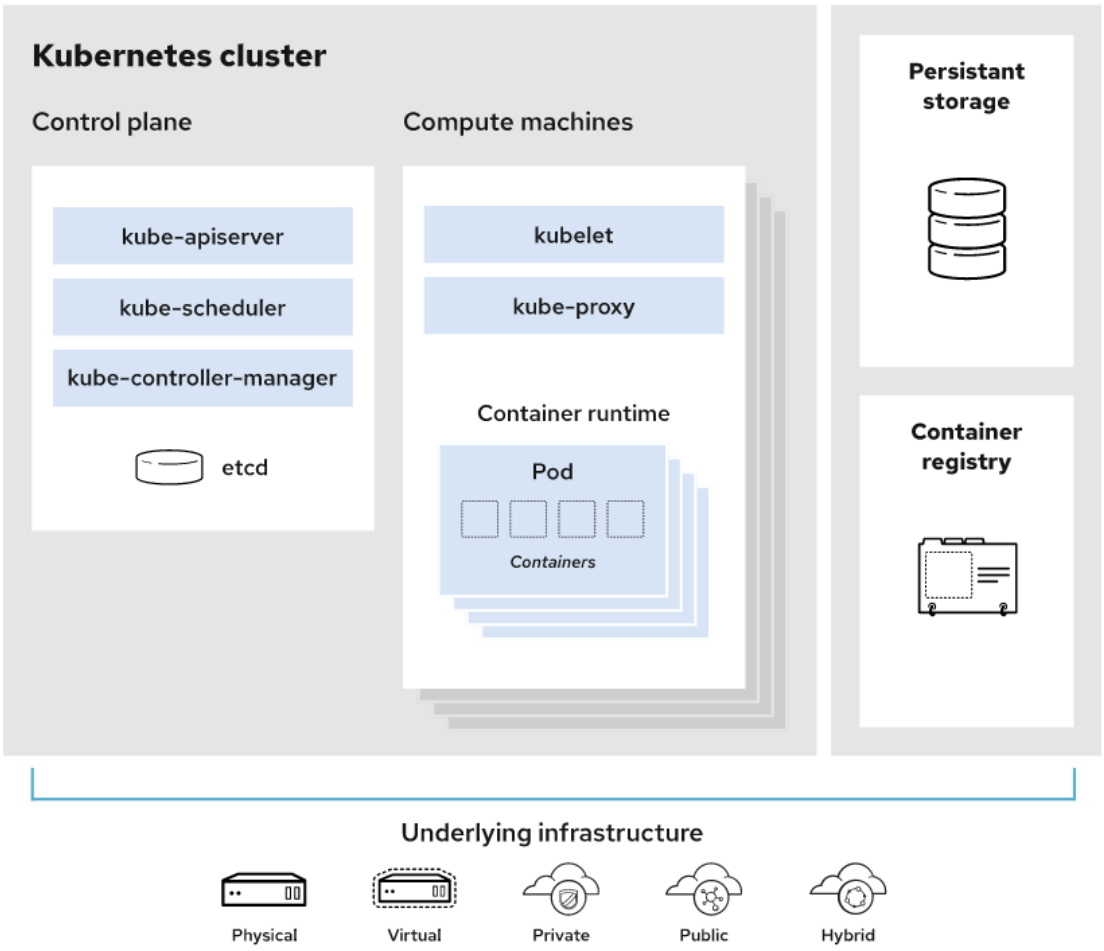
Manual provisioning consists of manually creating a persistent volume. This step is usually done by an administrator who has access to the underlying storage provider and have knowledge of the disk configuration.

Dynamic Provisioning:

Dynamic provisioning uses an API object called storage class ("StorageClass") where you configure a provisioner to dynamically create persistent volumes when requested using a PVC.

Storage classes are created as part of the bootstrapping of the kubernetes cluster by the administrator. The main goal of storage classes is to eliminate the need for cluster administrators to pre-provision storage and allow them to be created on-demand. Users can then request storage by specifying the storage class and the size of the volume needed in their claims.

Kubernetes Architecture

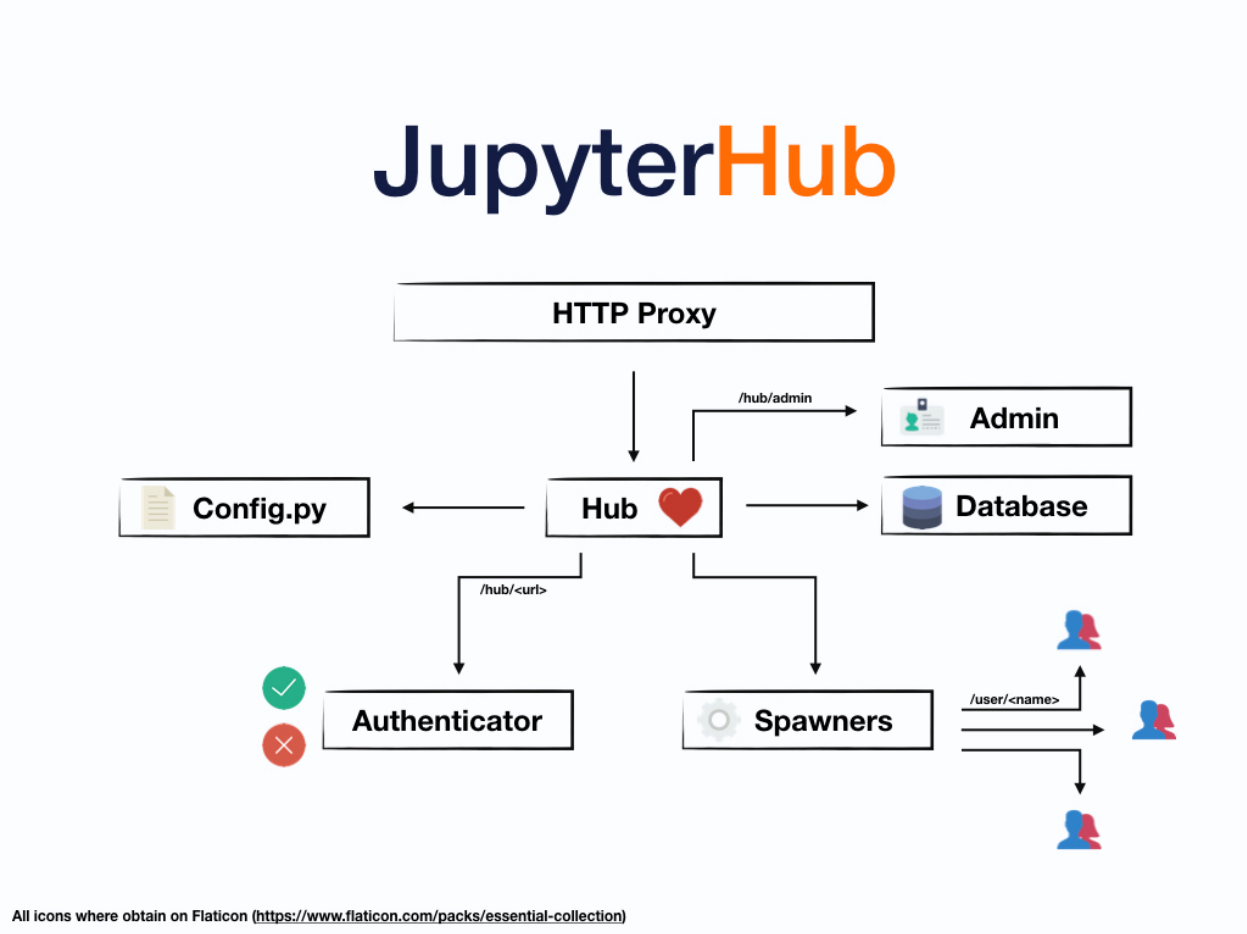


JUPUTERHUB

JupyterHub is the best way to serve Jupyter notebook for multiple users. It can be used in a class of students, a corporate data science group or scientific research group. It is a multi-user Hub that spawns, manages, and proxies multiple instances of the single-user Jupyter notebook server.

JupyterHub allows users to interact with a computing environment through a webpage. As most devices have access to a web browser, JupyterHub makes it is easy to provide and standardize the computing environment for a group of people (e.g., for a class of students or an analytics team).

This project will help you set up your own JupyterHub on a cloud/on-prem k8s environment and leverage its scalable nature to support a large group of users. Thanks to Kubernetes, we are not tied to a specific cloud provider.



IMPLEMANTATION

Setting up Kubernetes cluster using kubeadm

Follow the instructions to setup Kubernetes cluster on Bare metal(Ubuntu 20.04LTS)

Requirements

1. Master node with at least 2G RAM and 2CPU's
2. Worker node with at least 1G RAM AND 1CPU (Can add more worker nodes based on the requirement)

Run the following commands on both master and worker

All the commands below are to be performed as root user unless otherwise specified

sudo su

STEP 1

Disabling the firewall

ufw disable

STEP 2

Disable 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

Kubernetes Setup

STEP 5

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=1.20.5-00 kubelet=1.20.5-00 kubectl=1.20.5-00

Only on Master node

STEP 7

Update the below command with the ip address of master

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

STEP 8

Run the following commands exit the root to run kubectl commands as non-root user

mkdir -p $HOME/.kube

sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config

sudo chown $(id -u):$(id -g) $HOME/.kube/config

STEP 9

Deploy Calico network as root user and exit

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

STEP 10

On worker node join the cluster

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

STEP 11

Verify the cluster in master node if the node is joined

kubectl get nodes

Your Kubernetes cluster is ready!

Deploy MetalLB on Bare metal

After setting up kubernetes cluster we need to deploy metallb. MetalLB provides a network load-balancer implementation for Kubernetes clusters that do not run on a supported cloud provider, effectively allowing the usage of LoadBalancer Services within any cluster.

Check the ip address of the node

ip a s

Install sipcalc tool and analyse the network that is shown by running above command

sudo apt install sipcalc

which sipcalc

sipcalc x.x.x.x/20

Take atleast 10 ip address in the usable range and assign it to metallb

Installation By Manifest

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

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

On first install only

kubectl create secret generic -n metallb-system memberlist --from-literal=secretkey="$(openssl rand -base64 128)"

Verify if all the components are running

kubectl -n metallb-system get all

Deploy a config map

create a metallb.yaml file

nano /tmp/metallb.yaml

paste the following code in the file

```
apiVersion: v1
kind: ConfigMap
metadata:
  namespace: metallb-system
  name: config
```

```
data:
  config: |
    address-pools:
    - name: default
      protocol: layer2
      addresses:
      - x.x.x.x-x.x.x.x
```

then run

kubectl apply -f /tmp/metallb.yaml

Now you will be able to see external ip for the deployments

NFS Subdirectory External Provisioner

The NFS subdir external provisioner is an automatic provisioner for Kubernetes that uses your already configured NFS server, automatically creating Persistent Volumes.

Prerequisites

Kubernetes >=1.9 Existing NFS Share

First we need to check if we have nfs server if not we need to setup nfs-server

Install NFS-Kernel Server in Ubuntu

The first step is to install the nfs-kernel-server package on the server.

sudo apt update

sudo apt install nfs-kernel-server

Create an NFS Export Directory

sudo mkdir -p /srv/nfs/kubedata

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

Grant NFS Share Access to Client Systems

Permissions for accessing the NFS server are defined in the /etc/exports file.

sudo nano /etc/exports

add the following line

/srv/nfs/kubedata   *(rw,sync,no_subtree_check,no_root_squash,no_all_squash,insecure)

save it and exit Enable and start the nfs server

sudo systemctl enable --now nfs-server

sudo exportfs -rav

sudo showmount -e localhost

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

apt install -y nfs-common

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

check the mount and unmount it

umount /mnt

Install helm

The simplest way to install Helm is to run Helm's installer script in a terminal

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

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

change the ip address and give path as /srv/nfs/kubedata

helm install nfs-subdir-external-provisioner nfs-subdir-external-provisioner/nfs-subdir-external-provisioner --set nfs.server=x.x.x.x --set nfs.path=/srv/nfs/kubedata

Run **helm list** to check if nfs-provisioner is deployed

Run **kubectl get pods** to check if nfs pod is running

Now your NFS dynamic provisioner is set!

Set the storage class as default

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

Setting up jupyterhub on bare metal

Now that we have a Kubernetes cluster and Helm setup, we can proceed by using Helm to install JupyterHub and related Kubernetes resources using a Helm chart. We will be setting jupyterhub version 0.11.1

Install JupyterHub

First we need to add helm repository

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

helm repo update

Pull the values file before installing jupyterhub and redirect it to temporary yaml file

helm show values jupyterhub/jupyterhub > /tmp/jupyterhub.yaml

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

paste the output in secretToken in your yaml file **nano /tmp/jupyterhub.yaml** file

Now helm install jupyterhub

helm install jupyterhub jupyterhub/jupyterhub --values /tmp/jupyterhub.yaml

Run **helm list** and check for jupyterhub

Run **kubectl get all** to check all the pods and deployments are running.

Copy and paste the external ip next to **service/proxy-public** and paste it in your browser to run jupyterhub.

Now you have basic jupyterhub running on bare metal.

Setting up jupyterhub on GKE

Login to your free tier google cloud platform

In the upper left hand corner you can see the options button, in that you can see all the different things google cloud can do.

STEP 1

Select the GKE(Google Kubernetes Engine) from the options

STEP 2

Click on create button and configure standard kubernetes cluster

STEP 3

Leave the default values or change nodes, security, metadata as per your requirement by clicking on default pool

STEP 4

Click on create and wait for the cluster to be created

STEP 5

Connect to the cloud shell

STEP 6

Run **kubectl get nodes** to see if nodes are ready

STEP 7

Set up helm

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

STEP 8

Generate a random hex string representing 32 bytes to use as a security token.

openssl rand -hex 32

STEP 9

Create and start editing a file called config.yaml.

nano config.yaml

STEP 10

Write the following into the config.yaml file but instead of writing paste the generated hex string you copied in step 1

```
proxy:
  secretToken: "<RANDOM_HEX>"
```

STEP 11

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

STEP 12

```
RELEASE=jhub
NAMESPACE=jhub


helm upgrade --cleanup-on-fail \
  --install $RELEASE jupyterhub/jupyterhub \
  --namespace $NAMESPACE \
  --create-namespace \
```

--version=0.10.6 \

--values config.yaml

STEP 13

Run **kubectl get pod --namespace jhub** to check all pods are running

STEP 14

Run **kubectl get service --namespace jhub** to get the external ip and paste it in your web browser

Your jupyterhub is ready!

References

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

https://metallb.universe.tf/

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://docs.giantswarm.io/advanced/gpu/