**Prerequisites:**

**1. Docker**

**2. MongoDB**

**3. NVIDIA Docker**

**4. CUDA**

**5. NVIDIA Drivers**

**6. Docker Private Repository**

**7. Kubernetes**

**8. NFS (Server & client)**

**"Installation for DOCKER "**

"Removing old docker packages..."

$ sudo apt purge docker\* containerd\* -y #Remove all docker-related

packages

$ sudo apt-get remove docker docker-engine docker.io containerd runc -y

$ sudo rm -rf /var/lib/docker /etc/docker

$ sudo groupdel docker

$ sudo rm -rf /var/run/docker.sock

$ sudo rm -rf /usr/bin/docker-compose

$ sudo apt autoremove -y

$ sudo apt autoclean -y

$ sudo apt-get update -y

$ sudo update-ca-certificates

$ sudo apt-get install apt-transport-https ca-certificates curl gnupg lsb-release -y

$ curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o /usr/share/keyrings/docker-archive-keyring.gpg

$ echo "deb [arch=amd64 signed-by=/usr/share/keyrings/docker-archive-keyring.gpg] https://download.docker.com/linux/ubuntu $(lsb\_release -cs) stable" | sudo tee /etc/apt/sources.list.d/docker.list > /dev/null

$ sudo apt-get update -y

$ sudo apt-get install docker-ce=5:20.10.7~3-0~ubuntu-$(lsb\_release -cs) docker-ce-cli=5:20.10.7~3-0~ubuntu-$(lsb\_release -cs) containerd.io -y

$ sudo dpkg --configure -a

# sudo systemctl restart docker.service

"Running $ sudo docker run hello-world> command to verify docker is installed"

$ sudo docker run hello-world

"Docker is installed"

"Adding docker in >> sudoers list to run docker commands without >> sudo"

$ sudo groupadd docker

$ sudo usermod -aG docker $USER

newgrp docker

$ sudo systemctl show -p ActiveState --value docker.service

**#“Done...”**

**“Installing MongoDB.....”**

$ sudo apt-get update

$ wget -qO - https://www.mongodb.org/static/pgp/server-5.0.asc | sudo apt-key add -

$ echo "deb [ arch=amd64,arm64 ] https://repo.mongodb.org/apt/ubuntu(lsb\_release -sc)/mongodb-org/5.0 multiverse" | sudo tee /etc/apt/sources.list.d/mongodb-org-5.0.list

$ sudo apt-get update

$ sudo apt-get install -y mongodb-org

$ echo "mongodb-org hold" | sudo dpkg --set-selections

$ echo "mongodb-org-server hold" | sudo dpkg --set-selections

$ echo "mongodb-org-shell hold" | sudo dpkg --set-selections

$ echo "mongodb-org-mongos hold" | sudo dpkg --set-selections

$ echo"mongodb-org-tools hold" | sudo dpkg --set-selections

"MongoDB is installed on your system..."

$ sudo systemctl start mongod

$ sudo systemctl enable mongod

$ sudo systemctl show -p ActiveState --value mongod

"Check status of MongoDB service by using following command:"

$ sudo systemctl status mongod

**“MongoDB installed successfully.”**

**"Installing NVIDIA docker..."**

$ sudo apt update

$ curl -s -L https://nvidia.github.io/nvidia-docker/gpgkey | apt-key add distribution=$(. /etc/os-release;echo $ID$VERSION\_ID)

$ curl -s -L https://nvidia.github.io/nvidia-docker/$distribution/nvidia-docker.list | \

sudo tee /etc/apt/sources.list.d/nvidia-docker.list

$ sudo apt-get update

$ sudo apt-get install -y nvidia-docker2

$ sudo pkill -SIGHUP dockerd

**"NVIDIA Docker installed successfully"**

**"Installing CUDA toolkit..."**

$ sudo apt update -y

$ sudo apt install nvidia-cuda-toolkit -y

export PATH=/usr/local/cuda/bin${PATH:+:${PATH}} >> ~/.bashrc

**"CUDA toolkit installed successfully"**

**"Installing NVIDIA Drivers....."**

$ sudo apt-get install linux-headers-$(uname -r)

$ distribution=$(. /etc/os-release;echo $ID$VERSION\_ID | sed -e 's/\.//g') && wget <https://developer.download.nvidia.com/compute/cuda/repos/$distribution/x86_64/cuda-$distribution.pin> && sudo mv cuda-$distribution.pin /etc/apt/preferences.d/cuda-repository-pin-600

$ sudo apt-key adv --fetch-keys https://developer.download.nvidia.com/compute/cuda/repos/$distribution/x86\_64/3bf863cc.pub && "deb http://developer.download.nvidia.com/compute/cuda/repos/$distribution/x86\_64 /" | sudo tee /etc/apt/sources.list.d/cuda.list

$ sudo apt-get update

$ sudo apt-get -y install cuda-drivers

**"NVIDIA Drivers installed successfully."**

**“Creating Docker Private Repository..... ”**

**Requirements:**

a) Docker installed

b) At least two machines for server-client setup

- server1

- client1

**1) On all machines**

1. Add hostname and ip address of every host in all hosts list

$ sudo nano /etc/hosts

2. Save the file.

**2) On server :**

1. Login as root

$ sudo su

2. Make a folder where docker files will be stored

$ mkdir -p /docker\_data/certs

3. Check whether OPENSSL is installed or not

$ openssl

If not installed then install it by following command

$ apt install openssl

4. Generate a certificate which will check authentication

$ openssl req -newkey rsa:4096 -nodes -sha256 –keyout /docker\_data/certs/domain.key -x509 -days 365 -out /docker\_data/certs/domain.crt

5. While generating certificate we need to fill up the information. Its

not necessary to fill up all fields. We can skip by hitting just ‘Enter’.

But the main field we have to fill up is of ‘ Common Name ’ through which

communication takes place.

- So enter the name exactly same as your server hostname (e.g. dockerrepo).

6. Check whether the files are generated

$ ll /docker\_data/certs/

7. Create a folder to save docker images

$ mkdir -p /docker\_data/images

8. Now create a private docker registry

$ docker run -d -p 5000:5000 -v /docker\_data/images:/var/lib/registry -v /docker\_data/certs:/certs -e REGISTRY\_HTTP\_TLS\_CERTIFICATE=/certs/domain.crt -e REGISTRY\_HTTP\_TLS\_KEY=/certs/domain.key --restart=on-failure --name myregistry docker.io/registry

ex. For Retinanet

$ docker run -d -p 5000:5000 \

-v /docker\_data/images:/var/lib/registry \

-v /docker\_data/certs:/certs \

-e REGISTRY\_HTTP\_TLS\_CERTIFICATE=/certs/domain.crt \

-e REGISTRY\_HTTP\_TLS\_KEY=/certs/domain.key \

--restart on-failure \

--name retinanet \

docker.io/registry

#### retinanet is your docker repository name where you are storing your retinanet

9. Check if container is running or not

$ docker ps -a

10. Also verify images

$ docker images

11. Lets pull new image

$ docker pull nginx

$ docker images ... verify newly downloaded image

12. Rename the docker images before pushing them to private

docker registry

$ docker tag docker.io/nginx localhost:5000/my-nginx

$ docker images ... check newly renamed image in list

13. Now push newly created image to private registry

$ docker push localhost:5000/my-nginx

14. You can view pushed images in directory

$ ll /docker\_data/images/docker/registry/v2/repositories/

15. Now copy the domain.crt and domain.key files to client side

$ scp -r /docker\_data/certs/domain.crt client1:/home/client1/Desktop

16. Configure client1 node

$ ssh client1@ip-address

**3) On client :**

1. Login as root

$ sudo su

2. Move to certs.d folder

$ cd /etc/docker/certs.d

Create the certs.d folder if not present.

3. Create directory and name it as we specified in certificate common

name

$ mkdir -p dockerrepo:5000

4. Copy the domain.crt to dockerrepo folder

$ cp -rf /home/client1/Desktop/domain.crt dockerrepo\:5000/

5. Now pull the required image form server1

$ docker pull dockerrepo:5000/my-nginx

6. Verify the images pulled

$ docker images

7. Run the image

$ docker run -d -it - -name server1 dockerrepo:5000/my-nginx /bin/bash

8. Confirm if its running

$ docker ps -a

9. Now login to server1 to make changes and

push it back to private registry

$ docker exec -it server1 /bin/bash

10. As a change create a directory and exit

$ mkdir test\_dir

$ exit

11. We need to commit before push

$ docker commit server1 dockerrepo:5000/new-nginx:v1

12. Check if new image is created

$ docker images

13. Now push the newly created image

$ docker push dockerrepo:5000/new-nginx:v1

ex. docker push dockerrepo:5000/retinanet:v1

14. Stop the running container

$ docker stop server1

15. Remove the container

$ docker rm server1

16. Now to check if push operation was successfull,

pull the image we pushed to server

$ docker pull dockerrepo:5000/new-nginx:v1

If pull operation starts then its correct.

17. Now run the image

$ docker run -d -it - -name server1 dockerrepo:5000/new-nginx:v1

/bin/bash

18. Login to the server

$ docker exec -it server1 /bin/bash

19. Now check the ‘test\_dir’ is present or not as a change we pushed

before

$ ll

20. Go to Step 10 if want to make changes any.

**“Kubernetes Setup ....”**

**Prerequisites (On all machines) :**

**Install NVIDIA drivers on all GPU systems.**

**Install Docker .**

**After Docker installation on GPU system, we have to allow docker to use GPU.**

**So on GPU systems, edit the file as below and save it.**

## sudo nano /*etc*/*docker*/*daemon*.*json*

{

"default-runtime": "nvidia",

"runtimes": {

"nvidia": {

"path": "nvidia-container-runtime",

"runtimeArgs": []

}

},

"exec-opts": ["native.cgroupdriver=systemd"],

"log-driver": "json-file",

"log-opts": {

"max-size": "100m"

},

"storage-driver": "overlay2"

}

**On all systems:**

$ sudo swapoff -a

$ sudo apt-get update

$ sudo apt-get install -y apt-transport-https ca-certificates curl

$ sudo curl -fsSLo /usr/share/keyrings/kubernetes-archive-keyring.gpg https://packages.cloud.google.com/apt/doc/apt-key.gpg

$ echo "deb [signed-by=/usr/share/keyrings/kubernetes-archive-keyring.gpg] https://apt.kubernetes.io/ kubernetes-xenial main" | sudo tee /etc/apt/sources.list.d/kubernetes.list

$ sudo apt-get update

$ sudo apt-get install -y kubelet kubeadm kubectl

$ sudo apt-mark hold kubelet kubeadm kubectl

**On Master:**

$ sudo kubeadm init –apiserver-advertise-address=192.168.0.0

**Save the output in a text file.**

**Follow the instructions as suggested on terminal at this stage.**

$ mkdir -p $HOME/.kube

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

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

$ kubectl apply -f "https://cloud.weave.works/k8s/net?k8s-version=$(kubectl version | base64 | tr -d '\n')"

**On Worker nodes :**

$ sudo kubeadm join <join command saved on master node>

**On Master :**

$ kubectl get nodes

Wait until all nodes are in ready state.

**Run below command on Master to install nvidia-plugin on all GPU Worker nodes :**

$ **sudo kubectl create -f https://raw.githubusercontent.com/NVIDIA/k8s-device-plugin/1.0.0-beta4/nvidia-device-plugin.yml**

**YOUR CLUSTER IS READY ...!!!**

**"Installing NFS SERVER on your server..."**

$ sudo apt update

$ sudo apt install nfs-kernel-server

$ sudo systemctl restart nfs-kernel-server

Run the command below by specifying the NFS mount directory name.

$ sudo mkdir -p /mnt/nfs\_share

$ sudo chown -R nobody:nogroup /mnt/nfs\_share/

$ sudo chmod 777 /mnt/nfs\_share/

$ sudo nano /etc/exports

/mnt/nfs\_share client\_IP\_1 (re,sync,no\_subtree\_check)

/mnt/nfs\_share client\_IP\_2 (re,sync,no\_subtree\_check)

**Save the file.**

**"Install NFS CLIENT on system..."**

$ sudo apt update

$ sudo apt install nfs-common

Create an NFS Mount Point on Client

$ sudo mkdir -p /mnt/nfs\_clientshare

$ sudo mount server-ip:/mnt/nfs\_share /mnt/nfs\_clientshare

Now, test the NFS share by adding files in the NFS server

$ cd /mnt/nfs\_share/

$ touch file1.txt file2.txt file3.txt

**“Done”**

**Wget https://github.com/etcd-io/etcd/releases/download/${RELEASES}/etcd- ${RELEASES}-- https://github.com/etcd-io/etcd/releases/download/v3.5.5/etcd-v3.5.5-linux-amd64.tar.gz**