

Setting up of Kubernetes Cluster

- [Setup master and worker node VM's](#)
- [Create service account](#)
- [Accessing Kubernetes Dashboard](#)
- [Setting up persistent storage using OpenEBS](#)
- [Deploy private docker registry in kubernetes](#)
- [Installation of Ingress nginx](#)
- [Host Docker private registry using Ingress](#)

Setup master and worker node VM's

Setup VPS and install docker (all nodes)

Refer: <https://projects-srv2.kondgekar.com/projects/cbofferwall/wiki/0000-preliminary-setup>

Fix swap (all nodes)

Installing kubernetes needs swap should be disabled. Check if swap is enabled and then disable if swap is enabled.

Disable swap

```
sudo swapoff -a
```

Remove / comment out respective swap entry from fstab file

```
sudo nano /etc/fstab
```

```
GNU nano 4.8 /etc/fstab
# /etc/fstab: static file system information.
#
# Use 'blkid' to print the universally unique identifier for a
# device; this may be used with UUID= as a more robust way to name devices
# that works even if disks are added and removed. See fstab(5).
#
# <file system> <mount point> <type> <options> <dump> <pass>
# / was on /dev/sda2 during installation
UUID=f1b6db43-01e1-4851-a53e-b0c4eee7d6a7 / ext4 errors=remount-ro 0 0
# /boot was on /dev/sda1 during installation
UUID=12f6445c-9ef7-440f-bccc-600023baf92b /boot ext4 defaults noatime 0 0
#/swapfile none swap sw 0 0
```

Install kubeadm, Kubelet And Kubectl (all nodes)

Refer: <https://kubernetes.io/docs/setup/production-environment/tools/kubeadm/install-kubeadm/>

```
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
```

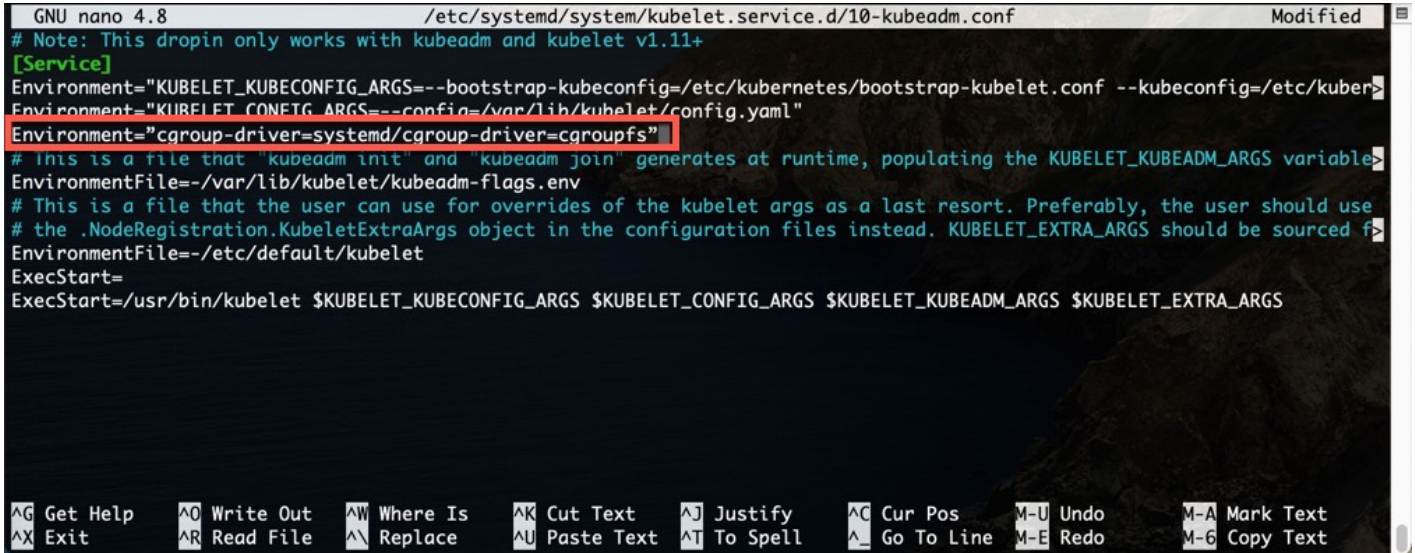
Update Kubernetes Configuration

(all nodes)

```
sudo nano /etc/systemd/system/kubelet.service.d/10-kubeadm.conf
```

This will open a text editor, enter the following line after the last “Environment Variable”:

```
Environment="cgroup-driver=systemd/cgroup-driver=cgroupfs"
```



```
GNU nano 4.8 /etc/systemd/system/kubelet.service.d/10-kubeadm.conf Modified
# Note: This dropin only works with kubeadm and kubelet v1.11+
[Service]
Environment="KUBELET_KUBECONFIG_ARGS=--bootstrap-kubeconfig=/etc/kubernetes/bootstrap-kubelet.conf --kubeconfig=/etc/kubernetes/kubelet.conf"
Environment="KUBELET_CONFIG_ARGS=--config=/var/lib/kubelet/config.yaml"
Environment="cgroup-driver=systemd/cgroup-driver=cgroupfs"
# This is a file that "kubeadm init" and "kubeadm join" generates at runtime, populating the KUBELET_KUBEADM_ARGS variable
EnvironmentFile=/var/lib/kubelet/kubeadm-flags.env
# This is a file that the user can use for overrides of the kubelet args as a last resort. Preferably, the user should use
# the .NodeRegistration.KubeletExtraArgs object in the configuration files instead. KUBELET_EXTRA_ARGS should be sourced from this file.
EnvironmentFile=/etc/default/kubelet
ExecStart=
ExecStart=/usr/bin/kubelet $KUBELET_KUBECONFIG_ARGS $KUBELET_CONFIG_ARGS $KUBELET_KUBEADM_ARGS $KUBELET_EXTRA_ARGS
```

Start kubernetes cluster (on master)

We are going to use [Flannel](#) as a networking for pods

```
sudo kubeadm init --apiserver-advertise-address=<ip-address-of-kmaster-vm> --pod-network-cidr=10.244.0.0/16
```

1. You will get the below output. The commands marked as (1), execute them as a non-root user. This will enable you to use kubectl from the CLI
2. The command marked as (2) should also be saved for future. This will be used to join nodes to your cluster

```
[addons] Applied essential addon: kube-proxy

Your Kubernetes control-plane has initialized successfully!

To start using your cluster, you need to run the following as a regular user:

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

Alternatively, if you are the root user, you can run:

export KUBECONFIG=/etc/kubernetes/admin.conf

You should now deploy a pod network to the cluster.
Run "kubectl apply -f [podnetwork].yaml" with one of the options listed at:
https://kubernetes.io/docs/concepts/cluster-administration/addons/

Then you can join any number of worker nodes by running the following on each as root:

kubeadm join 75.119.139.251:6443 --token techiterian@v \
--discovery-token-ca-cert-hash sha256:8271c12e00000000000000000000000000000000000000000000000000000000
techiterian@v
```

Execute commands as mentioned above.

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

Verify if cluster is running (on master)

```
kubectl get pods -o wide --all-namespaces
```

```
techiterian@vmi593089:~$ kubectl get pods -o wide --all-namespaces
NAMESPACE   NAME                                READY   STATUS    RESTARTS   AGE   IP              NODE                                READINESS GATES
kube-system  coredns-558bd4d5db-6mb2j           0/1     Pending   0           8m41s  <none>          <none>                             <none>
kube-system  coredns-558bd4d5db-kbx4n           0/1     Pending   0           8m41s  <none>          <none>                             <none>
kube-system  etcd-vmi593089.contaboserver.net    1/1     Running   0           8m55s  75.119.139.251 vmi593089.contaboserver.net        <none>
kube-system  kube-apiserver-vmi593089.contaboserver.net 1/1     Running   0           8m46s  75.119.139.251 vmi593089.contaboserver.net        <none>
kube-system  kube-controller-manager-vmi593089.contaboserver.net 1/1     Running   0           8m46s  75.119.139.251 vmi593089.contaboserver.net        <none>
kube-system  kube-proxy-f7kz9                    1/1     Running   0           8m40s  75.119.139.251 vmi593089.contaboserver.net        <none>
kube-system  kube-scheduler-vmi593089.contaboserver.net 1/1     Running   0           8m46s  75.119.139.251 vmi593089.contaboserver.net        <none>
```

Notice that all pods are running except coredns. It will be running once we setup pod network in the next step

Install POD network (Flannel) (on master)

```
kubectl apply -f https://raw.githubusercontent.com/coreos/flannel/master/Documentation/kube-flannel.yml
```

```
techiterian@vmi593089:~$ kubectl apply -f https://raw.githubusercontent.com/coreos/flannel/master/Documentation/kube-flannel.yml
Warning: policy/v1beta1 PodSecurityPolicy is deprecated in v1.21+, unavailable in v1.25+
podsecuritypolicy.policy/psp.flannel.unprivileged created
clusterrole.rbac.authorization.k8s.io/flannel created
clusterrolebinding.rbac.authorization.k8s.io/flannel created
serviceaccount/flannel created
configmap/kube-flannel-cfg created
daemonset.apps/kube-flannel-ds created
techiterian@vmi593089:~$
```

```
kubectl get pods -o wide --all-namespaces
```

```
techiterian@vmi593089:~$ kubectl get pods --all-namespaces -o wide
NAMESPACE   NAME                                     READY   STATUS    RESTARTS   AGE   IP              NODE                                     NOMINATED NODE   READINESS GATES
kube-system  coredns-558bd4d5db-6mb2j               1/1     Running   0           20m   10.244.0.2      vmi593089.contaboserver.net            <none>            <none>
kube-system  coredns-558bd4d5db-kbx4n               1/1     Running   0           20m   10.244.0.3      vmi593089.contaboserver.net            <none>            <none>
kube-system  etcd-vmi593089.contaboserver.net        1/1     Running   0           20m   75.119.139.251  vmi593089.contaboserver.net            <none>            <none>
kube-system  kube-apiserver-vmi593089.contaboserver.net 1/1     Running   0           20m   75.119.139.251  vmi593089.contaboserver.net            <none>            <none>
kube-system  kube-controller-manager-vmi593089.contaboserver.net 1/1     Running   0           20m   75.119.139.251  vmi593089.contaboserver.net            <none>            <none>
kube-system  kube-flannel-ds-5wvrr                  1/1     Running   0           108s  75.119.139.251  vmi593089.contaboserver.net            <none>            <none>
kube-system  kube-proxy-f7kz9                       1/1     Running   0           20m   75.119.139.251  vmi593089.contaboserver.net            <none>            <none>
kube-system  kube-scheduler-vmi593089.contaboserver.net 1/1     Running   0           20m   75.119.139.251  vmi593089.contaboserver.net            <none>            <none>
```

Notice that all pods are now running

Install Kubernetes Dashboard (on master)

Refer: <https://kubernetes.io/docs/tasks/access-application-cluster/web-ui-dashboard/>

Check available nodes (on master)

```
kubectl get nodes
```



```
Last login: Wed May 19 13:30:41 2021 from 49.33.209.127
techiterian@vmi593089:~$ kubectl get nodes
NAME                                STATUS    ROLES    AGE   VERSION
vmi593089.contaboserver.net        Ready    control-plane,master   29m   v1.21.1
techiterian@vmi593089:~$
```

Notice that only one node is available which is a master node.

Add worker node (on worker node)

Use below command to join Kubernetes cluster from worker node

```
kubeadm join <master-node-ip-address>:6443 --token <generated-token> \
--discovery-token-ca-cert-hash <generated-hash>
```

```
techiterian@vmi593088:~$ sudo kubeadm join 75.119.139.251:6443 --token <generated-token> \
--discovery-token-ca-cert-hash <generated-hash>
[sudo] password for techiterian:
[preflight] Running pre-flight checks
[WARNING IsDockerSystemdCheck]: detected "cgroupfs" as the Docker cgroup driver. The recommended driver is "systemd". Please follow the guide at https://kubernetes.io/docs/setup/cri/
[preflight] Reading configuration from the cluster...
[preflight] FYI: You can look at this config file with 'kubectl -n kube-system get cm kubeadm-config -o yaml'
[kubelet-start] Writing kubelet configuration to file "/var/lib/kubelet/config.yaml"
[kubelet-start] Writing kubelet environment file with flags to file "/var/lib/kubelet/kubeadm-flags.env"
[kubelet-start] Starting the kubelet
[kubelet-start] Waiting for the kubelet to perform the TLS Bootstrap...

This node has joined the cluster:
* Certificate signing request was sent to apiserer and a response was received.
* The Kubelet was informed of the new secure connection details.

Run 'kubectl get nodes' on the control-plane to see this node join the cluster.
techiterian@vmi593088:~$
```

Check if node is added in cluster (on master)

Initially worker node will show status as not ready

```
techiterian@vmi593089:~$ kubectl get nodes
NAME                                STATUS    ROLES    AGE   VERSION
vmi593088.contaboserver.net        NotReady    <none>    10s   v1.21.1
vmi593089.contaboserver.net        Ready    control-plane,master   40m   v1.21.1
techiterian@vmi593089:~$
```

Wait for some time and it will be shown as Ready

```
techiterian@vmi593089:~$ kubectl get nodes
NAME                                STATUS    ROLES    AGE   VERSION
vmi593088.contaboserver.net        Ready    <none>    3m25s v1.21.1
vmi593089.contaboserver.net        Ready    control-plane,master 43m    v1.21.1
techiterian@vmi593089:~$
```

“Kubernetes cluster is now running. You can now run containerized applications and make it available over web using specific setup

Create service account

Ref: <https://computingforgeeks.com/create-admin-user-to-access-kubernetes-dashboard/>

admin-sa.yml

```
---
apiVersion: v1
kind: ServiceAccount
metadata:
  name: techiterian-cluster-admin
  namespace: kube-system
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: techiterian-cluster-admin
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: cluster-admin
subjects:
- kind: ServiceAccount
  name: techiterian-cluster-admin
  namespace: kube-system
```

Run following command

```
kubectl apply -f admin-sa.yml
```

To obtain admin user token

Set a variable to store the name of the service account.

```
SA_NAME="techiterian-cluster-admin"
```

To view token

```
kubectl -n kube-system describe secret $(kubectl -n kube-system get secret | grep ${SA_NAME})  
| awk '{print $1}'
```

Token obtained from above command can be used to access Kubernetes dashboard

Accessing Kubernetes Dashboard

To install kubernetes dashboard on master node make sure to install it before worker nodes join k8s cluster.

Obtain login token for service account (on master)

```
SA_NAME=" <service- account-name>"  
# eg SA_NAME=" techiterian-cluster-admin"
```

to print token

```
kubectl -n kube-system describe secret $(kubectl -n kube-system get secret | grep ${SA_NAME}  
| awk '{print $1}')
```

Service account along with a token will be printed

```
techiterian@prd-master1: ~/Documents/k8s-rapidoreach-prod$ kubectl -n kube-system describe  
secret $(kubectl -n kube-system get secret | grep ${SA_NAME} | awk '{print $1}')
```

Name:	techiterian-cluster-admin-token-cvmgg
Namespace:	kube-system
Labels:	<none>
Annotations:	kubernetes.io/service-account.name: techiterian-cluster-admin kubernetes.io/service-account.uid: e5538ac5-41c1-44aa-9f4e-b06888cc86f0
Type:	kubernetes.io/service-account-token
Data	

```
====  
namespace: 11 bytes  
token:      <token here>  
ca.crt:     1066 bytes
```

Start proxy to access kubernetes dashboard (on master)

```
kubectl proxy
```

Open ssh tunnel (on local machine)

Kubernetes dashboard will currently be accessible only from master node where kubectl proxy is executed. To access it from remote machine use ssh tunnel

```
ssh -L 8001:localhost:8001 techiterian@prd-master1.rapidoreach.com
```

Above command will open ssh tunnel which will forward all traffic from localhost port 8001 to kubernetes proxy port 8001

Access Kubernetes dashboard using url

<http://localhost:8001/api/v1/namespaces/kubernetes-dashboard/services/https:kubernetes-dashboard:/proxy/>

“Kubernetes dashboard is not accessible from local machine

Setting up persistent storage using OpenEBS

Install iSCSI on all nodes

Refer: <https://docs.openebs.io/docs/next/prerequisites.html>

Install iSCSI tools if not installed

```
sudo apt-get update
sudo apt-get install open-iscsi
sudo systemctl enable --now iscsid
```

Check if iSCSI is running

```
sudo systemctl status iscsid
```

```
● iscsid.service - iSCSI initiator daemon (iscsid)
   Loaded: loaded (/lib/systemd/system/iscsid.service; enabled; vendor preset: enabled)
   Active: active (running) since Wed 2021-05-19 14:32:56 CEST; 17s ago
 TriggeredBy: ● iscsid.socket
    Docs: man:iscsid(8)
   Process: 48196 ExecStartPre=/lib/open-iscsi/startup-checks.sh (code=exited, status=0/SUCCESS)
   Process: 48203 ExecStart=/sbin/iscsid (code=exited, status=0/SUCCESS)
  Main PID: 48207 (iscsid)
    Tasks: 2 (limit: 19148)
   Memory: 2.4M
    CGroup: /system.slice/iscsid.service
            └─48206 /sbin/iscsid
              └─48207 /sbin/iscsid

May 19 14:32:56 vmi593089.contaboserver.net systemd[1]: Starting iSCSI initiator daemon (iscsid)...
May 19 14:32:56 vmi593089.contaboserver.net iscsid[48203]: iSCSI logger with pid=48206 started!
May 19 14:32:56 vmi593089.contaboserver.net systemd[1]: iscsid.service: Failed to parse PID from file /run/iscsid.pid: In>
May 19 14:32:56 vmi593089.contaboserver.net systemd[1]: Started iSCSI initiator daemon (iscsid).
May 19 14:32:57 vmi593089.contaboserver.net iscsid[48206]: iSCSI daemon with pid=48207 started!
~
techiterian@vmi593089:~$
```


Make sure you have admin context

```
kubectl config use-context kubernetes-admin@kubernetes
```

Remove taint from master node

We need to use master node as a storage node where all database files and persistent volumes will be stored. Remove taint from master node so that master node will be available to schedule pods

```
kubectl taint node <masternode_name> node-role.kubernetes.io/master:NoSchedule-
```

Install using kubectl

Download file <https://openebs.github.io/charts/openebs-operator.yaml>

to install OpenEBS to use volumes from master node we need to label master node with appropriate label

Use following command to label master node

```
kubectl label nodes <node-name> node=openebs
```

Next, in the downloaded openebs-operator.yaml, find the PodSpec for `openebs-provisioner`, `maya-apiserver`, `openebs-snapshot-operator`, `openebs-admission-server` and `openebs-ndm` pods and add the following key-value pair under `nodeSelector` field

nodeSelector field will be available in pod > specs > template > specs > nodeSelector

```
nodeSelector:
  node: openebs
```

Install OpenEBS with updated command

```
kubectl apply -f openebs-operator.yml
```

Check if OpenEBS has been installed

```
kubectl get pods -n openebs -o wide
```

Output should show running pods

```
techitarian@prd-master1:~/Documents/k8s-rapidoreach-prod$ kubectl get pods -n openebs -o wide
NAME                                READY   STATUS    RESTARTS   AGE   IP              NODE                                NOMINATED NODE   READINESS GATES
maya-apiserver-6fb6d8d5ff-2npgc     1/1     Running   0           15m   10.244.0.8      prd-master1.rapidoreach.com        <none>           <none>
openebs-admission-server-79c84d84d6-7622v  1/1     Running   0           15m   10.244.0.6      prd-master1.rapidoreach.com        <none>           <none>
openebs-localpv-provisioner-75cc7566c6-4spz5  1/1     Running   0           15m   10.244.1.8      prd-worker1.rapidoreach.com        <none>           <none>
openebs-ndm-gdw4s                   1/1     Running   0           13m   75.119.139.251  prd-master1.rapidoreach.com        <none>           <none>
openebs-ndm-operator-79d5597f77-7hp77       1/1     Running   0           15m   10.244.1.9      prd-worker1.rapidoreach.com        <none>           <none>
openebs-provisioner-8d6cf9796-7sc5p         1/1     Running   0           15m   10.244.0.9      prd-master1.rapidoreach.com        <none>           <none>
openebs-snapshot-operator-7c476dcc86-mxjh2   2/2     Running   0           15m   10.244.0.7      prd-master1.rapidoreach.com        <none>           <none>
techitarian@prd-master1:~/Documents/k8s-rapidoreach-prod$
```

Deploy private docker registry in kubernetes

Step 1: Creating files for authentication

Make a folder and open terminal within a folder created

```
mkdir registry  
cd registry
```

Create tls certificate and a key

```
openssl req -x509 -newkey rsa:4096 -days 3650 -nodes -sha256 -keyout certs/tls.key -out  
certs/tls.crt -subj "/CN=<docker-registry.mydomain.com>"
```

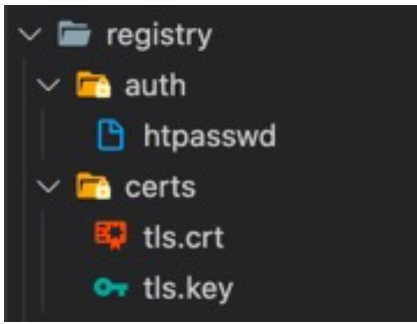
Use htpasswd to add user authentication for registry access. If htpasswd is not installed then install using below command.

```
sudo apt-get install apache2-utils
```

Create authentication file using htpasswd

```
htpasswd -Bbn <your_username> <your_password> > auth/htpasswd
```

At the end of this you will have folders as follows



Step 2: Create kubernetes secrets

Kubernetes secrets is a way of storing secrets / keys in kubernetes master storage.

Create a secret to store tls certificates

The below command creates a **Secret of type** **tls** named **certs-secret** in the **default namespace** from the pair of public/private keys we just created.

```
kubectl create secret tls registry-certs-secret --cert=<path-to-registry-  
folder>/certs/tls.crt --key=<path-to-registry-folder>/certs/tls.key
```

The Secret **auth-secret** that we create from the **htpasswd** file is of type generic which means the Secret was created from a local file.

```
kubectl create secret generic registry-auth-secret --from-file=<path-to-registry-  
folder>/auth/htpasswd
```

Step 3: Create storage class and persistent volume claim for repository storage

We are using OpenEBS to configure and manage persistent volumes

Create storage class

docker-registry-sc.yaml

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: docker-registry-sc
  annotations:
    openebs.io/cas-type: local
    cas.openebs.io/config: |
      - name: StorageType
        value: hostpath
      - name: BasePath
        value: <path-where-registry-should-be-stored>
    # eg value: /home/techiterian/Documents/volumes/docker_registry
provisioner: openebs.io/local
reclaimPolicy: Retain
volumeBindingMode: WaitForFirstConsumer
```

Create persistent volume claim

docker-registry-pvc.yaml

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: docker-registry-pvc # Specify name for pvc
spec:
  storageClassName: docker-registry-sc # Make sure storage class name is correctly spelled
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 2G # Specify appropriate storage
```

Execute below commands to create StorageClass and Persistent Volume Claim

```
kubectl apply -f <path-to-docker-registry-sc.yaml>
```

```
kubectl apply -f <path-to-docker-registry-pvc.yaml>
```

Step 4: Create a deployment for docker registry

docker-registry-deployment.yaml

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: docker-registry-deployment
spec:
  selector:
    matchLabels:
      app: docker-registry-deployment
  template:
    metadata:
      labels:
        app: docker-registry-deployment
    spec:
      nodeSelector: # Specify node selector to specify on which node docker registry should
be running. Persistent volumes will be created on same node
        kubernetes.io/hostname: prd-master1.rapidoreach.com
      containers:
        - name: docker-registry-deployment
          image: registry:2.7.1
          resources:
            limits:
              memory: "256Mi"
              cpu: "500m"
          ports:
            - containerPort: 5000
          volumeMounts:
            - mountPath: "/var/lib/registry" # /var/lib/registry is a common location within a
pod where all registries will be stored
              name: registry-database-volume
```



```

- mountPath: "/certs"
  name: certs-volume
- mountPath: "/auth"
  name: auth-volume
env: # Environment variables being set which will be used by registry pod
- name: REGISTRY_AUTH
  value: "htpasswd"
- name: REGISTRY_AUTH_HTPASSWD_REALM
  value: "Registry Realm"
- name: REGISTRY_AUTH_HTPASSWD_PATH
  value: "/auth/htpasswd"
- name: REGISTRY_HTTP_TLS_CERTIFICATE
  value: "/certs/tls.crt"
- name: REGISTRY_HTTP_TLS_KEY
  value: "/certs/tls.key"
volumes:
- name: registry-database-volume # Persistent volume claim is used here
  persistentVolumeClaim:
    claimName: docker-registry-pvc
- name: certs-volume # Secret used here to make certificates available within created
pod
  secret:
    secretName: registry-certs-secret
- name: auth-volume # Secret used here to make htpasswd auth available within created
pod
  secret:
    secretName: registry-auth-secret

```

Use below command to create a deployment

```
kubectl apply -f <path-to-docker-registry-deployment.yaml>
```

Step 5: Create node port service

Node port service will be used to expose created docker registry over internet

```
docker-registry-node-port.yaml
```

```
apiVersion: v1
kind: Service
metadata:
  name: docker-registry-node-port
spec:
  type: NodePort
  ports:
    - port: 5000 # Port which will be used internally by all pods
      targetPort: 5000 # Target port of docker registry pod where registry is listening
      nodePort: 30001 # Public port where docker registry will be available
  selector:
    app: docker-registry-deployment # Make sure selector should match with the label of
    deployment
```

Use below command to create a node port service

```
kubectl apply -f <path-to-docker-registry-node-port.yaml>
```

Step 6: Configure docker to trust self signed tls certificate

We must copy the `tls.crt` that we created earlier as `"ca.crt"` into a custom `/etc/docker/certs.d/<subdomain-where-registry-is-accessible>` directory in all the nodes in our cluster to make sure that our self-signed certificate is trusted by Docker. Note that the directory that is created inside `/etc/docker/certs.d` should have the name of the

format `<registry_name>:<registry_port>` for `<subdomain-where-registry-is-accessible>`

```
sudo mkdir -p /etc/docker/certs.d/<subdomain-where-registry-is-accessible>
```

Copy tls certificate

```
cp <path-to-registry-folder>/certs/tls.crt /etc/docker/certs.d/<subdomain-where-registry-is-accessible>/ca.crt
```

Step 7: Testing of private docker registry

Lets try to connect with private docker registry. Perfereably use master node

```
docker login <docker-registry-fully-qualified-domain-name>:<port> -u myuser -p mypasswd
```

It will output

```
techiterian@prd-master1:~/Documents/registry$ docker login docker-registry.kondgekar.com:30001 -u techiterian -p mypasswd
WARNING! Using --password via the CLI is insecure. Use --password-stdin.
WARNING! Your password will be stored unencrypted in /home/techiterian/.docker/config.json.
Configure a credential helper to remove this warning. See
https://docs.docker.com/engine/reference/commandline/login/#credentials-store

Login Succeeded
techiterian@prd-master1:~/Documents/registry$
```

“ If Login is not successfull then make sure you follow step 6 and copied **tls.crt** as **ca.crt** in appriprate folder of whichever machine you are running above command.

Additionally check if username and password is correctly put same as that of Step 1

Step 8: Use secret to store image pull login details

To authorize a pod to pull image from private registry we need to provide image pull credentials to pod. Lets create a secret whcih will store these credentials

```
kubectl create secret docker-registry regcred --docker-server=<your-registry-server> --docker-username=<your-name> --docker-password=<your-password>
```

Step 9: Test if custom image can be pushed to out private docker registry

```
# Pull standard image available on docker hub
docker pull nginx

# Tag the image for private registry
docker tag nginx:latest <your-registry-server>/mynginx:v1

# Push docker image to private repository
docker push <your-registry-server>/mynginx:v1
```

Command output will look like

```
techiterian@prd-master1:~/Documents/registry$ docker pull nginx
Using default tag: latest
latest: Pulling from library/nginx
Digest: sha256:df13abe416e37eb3db4722840dd479b00ba193ac6606e7902331dcea50f4f1f2
Status: Image is up to date for nginx:latest
docker.io/library/nginx:latest
techiterian@prd-master1:~/Documents/registry$ docker tag nginx:latest docker-registry.kongdekar.com:30001/mynginx:v1
techiterian@prd-master1:~/Documents/registry$ docker push docker-registry.kongdekar.com:30001/mynginx:v1
The push refers to repository [docker-registry.kongdekar.com:30001/mynginx]
f0f30197ccf9: Layer already exists
eeb14ff930d4: Layer already exists
c9732df61184: Layer already exists
4b8db2d7f35a: Layer already exists
431f409d4c5a: Layer already exists
02c055ef67f5: Layer already exists
v1: digest: sha256:eba373a0620f68ffdc3f217041ad25ef084475b8feb35b992574cd83698e9e3c size: 1570
techiterian@prd-master1:~/Documents/registry$
```

Step 10: Deploy a pod and use private registry to pull image from

Refer: <https://kubernetes.io/docs/tasks/configure-pod-container/pull-image-private-registry/>

Installation of Ingress nginx

Refer: <https://github.com/kubernetes/ingress-nginx>

Install using baremetal

Refer: <https://kubernetes.github.io/ingress-nginx/deploy/#bare-metal>

Download script to specific folder

```
cd <folder-path>
wget https://raw.githubusercontent.com/kubernetes/ingress-nginx/controller-
v0.46.0/deploy/static/provider/baremetal/deploy.yaml
```

Rename download .yaml file

```
mv ./deploy.yaml ingress-nginx-controller.yaml
```

in order to use ingress and assign an external ip address we need to use Load Balancer instead of NodePort of ingress nginx controller. Update ingress-nginx-controller.yaml with following lines

```
spec:
  type: LoadBalancer
  externalIPs:
    - <valid-external-ip>
```


263	apiVersion: v1	263	apiVersion: v1
264	kind: Service	264	kind: Service
265	metadata:	265	metadata:
266	annotations:	266	annotations:
267	labels:	267	labels:
268	helm.sh/chart: ingress-nginx-3.30.0	268	helm.sh/chart: ingress-nginx-3.30.0
269	app.kubernetes.io/name: ingress-nginx	269	app.kubernetes.io/name: ingress-nginx
270	app.kubernetes.io/instance: ingress-nginx	270	app.kubernetes.io/instance: ingress-nginx
271	app.kubernetes.io/version: 0.46.0	271	app.kubernetes.io/version: 0.46.0
272	app.kubernetes.io/managed-by: Helm	272	app.kubernetes.io/managed-by: Helm
273	app.kubernetes.io/component: controller	273	app.kubernetes.io/component: controller
274	name: ingress-nginx-controller	274	name: ingress-nginx-controller
275	namespace: ingress-nginx	275	namespace: ingress-nginx
276	spec:	276	spec:
277-	type: LoadBalancer	277+	type: NodePort
278-	externalIPs:		
279-	- 75.119.139.251		
280	ports:	278	ports:
281	- name: http	279	- name: http
282	port: 80	280	port: 80
283	protocol: TCP	281	protocol: TCP

Install ingress controller using following command

```
kubectl apply -f ingress-nginx-controller.yaml
```

Above command will create following pods and services

```
techiterian@prd-master1:~/Documents/k8s-rapidoreach-prod/0070-create-ingress-service$ kubectl get all -n ingress-nginx
```

NAME	READY	STATUS	RESTARTS	AGE
pod/ingress-nginx-admission-create-bcndf	0/1	Completed	0	7h4m
pod/ingress-nginx-admission-patch-x4p4l	0/1	Completed	0	7h4m
pod/ingress-nginx-controller-55bc4f5576-72mp9	1/1	Running	0	7h4m

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
service/ingress-nginx-controller	LoadBalancer	10.97.133.19	75.119.139.251	80:31168/TCP,443:31267/TCP	7h4m
service/ingress-nginx-controller-admission	ClusterIP	10.107.101.183	<none>	443/TCP	7h4m

NAME	READY	UP-TO-DATE	AVAILABLE	AGE
deployment.apps/ingress-nginx-controller	1/1	1	1	7h4m

NAME	DESIRED	CURRENT	READY	AGE
replicaset.apps/ingress-nginx-controller-55bc4f5576	1	1	1	7h4m

NAME	COMPLETIONS	DURATION	AGE
job.batch/ingress-nginx-admission-create	1/1	11s	7h4m
job.batch/ingress-nginx-admission-patch	1/1	12s	7h4m

```
techiterian@prd-master1:~/Documents/k8s-rapidoreach-prod/0070-create-ingress-service$
```

“ ingress nginx controller is now installed. We can now create ingress object to correctly route traffix to NodePort or ClusterIP services

Host Docker private registry using Ingress

Make sure ingress nginx is installed and following command shows external Ip assigned

```
kubectl get services -o wide -n ingress-nginx
```

```
techiterian@prd-master1:~/Documents/k8s-rapidoreach-prod/0070-create-ingress-service$ kubectl get services -o wide -n ingress-nginx
NAME                                TYPE                CLUSTER-IP      EXTERNAL-IP      PORT(S)                                AGE      SELECTOR
ingress-nginx-controller            LoadBalancer        10.97.133.19     75.119.139.251   80:31168/TCP,443:31267/TCP            7h28m    app.kubernetes.io/name=ingress-nginx
ingress-nginx-controller-admission ClusterIP             10.107.101.183  <none>           443/TCP                                7h28m    app.kubernetes.io/name=ingress-nginx
```

if external ip is not assigned then nginx controller needs modifications. Refer: [Ingress installation](#)

Create ingress

prd-ingress-service.yaml

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: prd-ingress-service
  annotations:
    # use the shared ingress-nginx
    kubernetes.io/ingress.class: "nginx"
  labels:
    name: prd-ingress-service
spec:
  tls:
    - hosts:
        - <your-subdomain-domain>
      secretName: <tls-certificate secret>
  rules:
    - host: <your-subdomain-domain>
      http:
        paths:
          - pathType: Prefix
```

```
path: "/"
backend:
  service:
    name: <service-name>
    port:
      number: <service-local-port>
```

“ Docker private registry will now be available over internet