Document creating of cluster and all needed access rights

1. Set up three virtual machines for the master and 2 worker nodes on Flexible Engine.
2. Install a Linux distribution of CentOs on all the virtual machines.
3. Install Docker on all the virtual machines. Kubernetes uses Docker to manage containers.

    a. Check for the availability of the port and turn off the swap:

```
swapoff -a
nc 127.0.0.1 6443
```

    b. Install Containerd for the Master and the worker:

        i. **Forwarding IPv4 and letting iptables see bridged traffic**

```
cat <<EOF | sudo tee /etc/modules-load.d/k8s.conf
overlay
br_netfilter
EOF

 sudo modprobe overlay
sudo modprobe br_netfilter

# sysctl params required by setup, params persist across
reboots
cat <<EOF | sudo tee /etc/sysctl.d/k8s.conf
net.bridge.bridge-nf-call-iptables  = 1
net.bridge.bridge-nf-call-ip6tables = 1
net.ipv4.ip_forward                 = 1
EOF

# Apply sysctl params without reboot
sudo sysctl –system
#Verify that the br_netfilter, overlay modules are loaded by
running below instructions:
lsmod | grep br_netfilter
lsmod | grep overlay
#Verify that the net.bridge.bridge-nf-call-iptables, net.bridge.bridge-
nf-call-ip6tables, net.ipv4.ip_forward system variables are set to
1 in your sysctl config by running below instruction:
sysctl net.bridge.bridge-nf-call-iptables
net.bridge.bridge-nf-call-ip6tables net.ipv4.ip_forward
```

        ii. Install Containerd

```
yum remove docker \
                  docker-client \
                  docker-client-latest \
                  docker-common \
                  docker-latest \
                  docker-latest-logrotate \
                  docker-logrotate \
                  docker-engine
```

yum install -y yum-utils

yum-config-manager \

--add-repo \

https://download.docker.com/linux/centos/docker-ce.repo

sudo yum install docker-ce docker-ce-cli containerd.io docker-buildx-plugin docker-compose-plugin

systemctl start docker

#then vim this file

vim /etc/containerd/config.toml

#disabled_plugins = ["cri"]

4. Install Kubernetes on the master node using the following command:

```
cat <<EOF | sudo tee /etc/yum.repos.d/kubernetes.repo
[kubernetes]
name=Kubernetes
baseurl=https://packages.cloud.google.com/yum/repos/kubernetes-el7-
\$basearch
enabled=1
gpgcheck=1
gpgkey=https://packages.cloud.google.com/yum/doc/rpm-package-key.gpg
exclude=kubelet kubeadm kubectl
EOF

# Set SELinux in permissive mode (effectively disabling it)
sudo setenforce 0
```

```
sudo sed -i 's/^SELINUX=enforcing$/SELINUX=permissive/'
/etc/selinux/config

sudo yum install -y kubelet kubeadm kubectl --
disableexcludes=kubernetes

sudo systemctl enable --now kubelet
```

5. Initialize the Kubernetes cluster on the master node using the following command:

```
sudo kubeadm init --pod-network-cidr=192.168.0.0/16
```

6. After the initialization is complete, copy the join command to join the worker nodes to the cluster. The join command will be displayed in the output of the previous command.

7. Log in to each worker node and run the join command to join the node to the cluster. The join command will be different for each node.

8. Install a network plugin to enable communication between the pods on the worker nodes. You can use Calico, Flannel, or any other network plugin that supports Kubernetes. For example, to install flannel, run the following command on the master node:

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

9. Verify that the cluster is working by running the following command on the master node:

```
kubectl get nodes
```

A network policy in Kubernetes is used to control the traffic flow between pods in a cluster. It can also be used to restrict access to the Kubernetes API to a specific set of IP addresses.

Here's an example of how to create a network policy that allows access to the Kubernetes API from a specific set of IP addresses:

1. First, create a file called **kube-api-access.yaml** with the following content:

```
kind: NetworkPolicy

apiVersion: networking.k8s.io/v1
```

```
metadata:
  name: kube-api-access
spec:
  podSelector:
    matchLabels:
      component: kube-apiserver
  policyTypes:
  - Ingress
  ingress:
  - from:
    - ipBlock:
        cidr: 192.168.0.0/24
    - ipBlock:
        cidr: 10.244.0.0/24
    ports:
    - protocol: TCP
      port: 6443
```

The IP address range 10.244.0.0/24 and 192.168.0.0/24 is allowed to access the Kubernetes API on port **6443**.

2. Apply the network policy :

```
kubectl apply -f kube-api-access.yaml
```

This will create a network policy called **kube-api-access** that restricts access to the Kubernetes API to the specified IP address range.

Second task:

1- Install ingress controller by:

```
kubectl apply -f https://raw.githubusercontent.com/kubernetes/ingress-nginx/master/deploy/static/provider/cloud/deploy.yaml
```

2- Deploy "Juice Shop" application, you can use official docker image from ‘bkimminich/juice-shop’.

```
---
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    app: juice-shop
  name: juice-shop
  namespace: juice-shop
spec:
  replicas: 1
  selector:
    matchLabels:
      app: juice-shop
  template:
    metadata:
      labels:
        app: juice-shop
```

```
  spec:
    containers:
    - image: bkimminich/juice-shop
      imagePullPolicy: Always
      name: juice-shop-container
      ports:
      - containerPort: 3000
```

3- Expose “Juice Shop” application inside the cluster using the a service.

```
---
apiVersion: v1
kind: Service
metadata:
  name: juice-shop-entrypoint
  namespace: juice-shop
spec:
  type: ClusterIP
  #clusterIP: 10.107.57.210
  selector:
    app: juice-shop
  ports:
  - port: 80
    name: http
    targetPort: 3000
```

4- Expose “Juice Shop” application to outside the cluster using the nginx ingress.

```
---
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: juice-shop-ingress
  annotations:
    nginx.ingress.kubernetes.io/rewrite-target: /
spec:
  rules:
    - host: juice-shop.com
      http:
        paths:
          - path: /juice-shop
            pathType: Prefix
            backend:
              service:
                name: juice-shop-entrypoint
                port:
                  name: http
```