**CAPESTONE PROJECT-1**

**POST GRADUATE PROGRAM IN DEVOPS**

**NAME: TATA SAIKRISHNA**

**MAIL:dcgm.crmr@gmail.com**

**Mobile:7382801892**

**Registration ID:731272**

**https://github.com/tatasaikrishna/devops-capstone-project.git**

**Infra Optimization.**

Course-end Project 1

DESCRIPTION

Create a DevOps infrastructure for an e-commerce application to run on high-availability mode.

**Background of the problem statement:**  
A popular payment application, **EasyPay**where users add money to their wallet accounts, faces an issue in its payment success rate. The timeout that occurs with  
the connectivity of the database has been the reason for the issue.  
While troubleshooting, it is found that the database server has several downtime instances at irregular intervals. This situation compels the company to create their own infrastructure that runs in high-availability mode.  
Given that online shopping experiences continue to evolve as per customer expectations, the developers are driven to make their app more reliable, fast, and secure for improving the performance of the current system.

**Implementation requirements:**

1. Create the cluster (EC2 instances with load balancer and elastic IP in case of AWS)
2. Automate the provisioning of an EC2 instance using Ansible or Chef Puppet
3. Install Docker and Kubernetes on the cluster
4. Implement the network policies at the database pod to allow ingress traffic from the front-end application pod
5. Create a new user with permissions to create, list, get, update, and delete pods
6. Configure application on the pod
7. Take snapshot of ETCD database
8. Set criteria such that if the memory of CPU goes beyond 50%, environments automatically get scaled up and configured

**The following tools must be used:**

1. EC2
2. Kubernetes
3. Docker
4. Ansible or Chef or Puppet

**The following things to be kept in check:**

1. You need to document the steps and write the algorithms in them.
2. The submission of your GitHub repository link is mandatory. In order to track your tasks, you need to share the link of the repository.
3. Document the step-by-step process starting from creating test cases, then executing them, and recording the results.
4. You need to submit the final specification document, which includes:

* Project and tester details
* Concepts used in the project
* Links to the GitHub repository to verify the project completion
* Your conclusion on enhancing the application and defining the USPs (Unique Selling Points)

**4.Implement the network policies at the database pod to allow ingress traffic from the front-end application pod**

Create a Kubernetes Network Policy that allows ingress traffic from the front-end application pod to the database pod. You can define the policy in YAML format and specify the pod selector, the ingress rules, and the protocol and port to allow.

**$vim ingress.yaml**

apiVersion: networking.k8s.io/v1

kind: NetworkPolicy

metadata:

name: db-allow-front-end

spec:

podSelector:

matchLabels:

app: database

policyTypes:

- Ingress

ingress:

- from:

- podSelector:

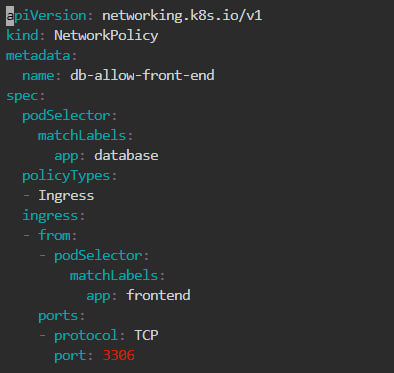
matchLabels:

app: frontend

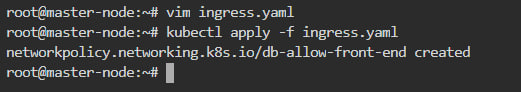
ports:

- protocol: TCP

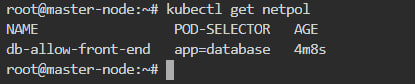
port: 3306



# kubectl apply –f ingress.yaml



#kubectl get netpol



1. **To create a new user with permissions to create, list, get, update, and delete pods in Kubernetes**

**Create a new user account with a unique username and password.**

#kubectl create serviceaccount myuser

#kubectl create role myrole --verb=create,list,get,update,delete --resource=pods

#kubectl create rolebinding myuser-binding --role=myrole --serviceaccount=default:myuser

#kubectl run mypod --image=nginx --serviceaccount=myuser

**$vim pod-admin.yaml**

**Create a new role with the required permissions. For example, you can create a role named "pod-admin" with the following permissions:**

apiVersion: rbac.authorization.k8s.io/v1

kind: Role

metadata:

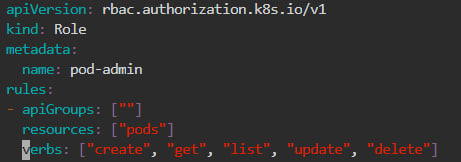
name: pod-admin

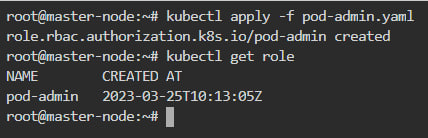
rules:

- apiGroups: [""]

resources: ["pods"]

verbs: ["create", "get", "list", "update", "delete"]





**Bind the role to the user account using a RoleBinding. For example, you can create a RoleBinding named "pod-admin-binding" to associate the "pod-admin" role with the user "newuser".**

**#vim pod-admin-binding**

apiVersion: rbac.authorization.k8s.io/v1

kind: RoleBinding

metadata:

name: pod-admin-binding

subjects:

- kind: User

name: newuser

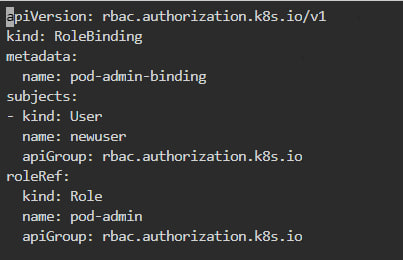
apiGroup: rbac.authorization.k8s.io

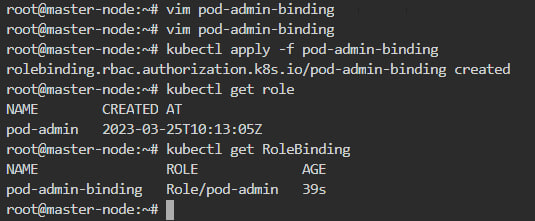
roleRef:

kind: Role

name: pod-admin

apiGroup: rbac.authorization.k8s.io





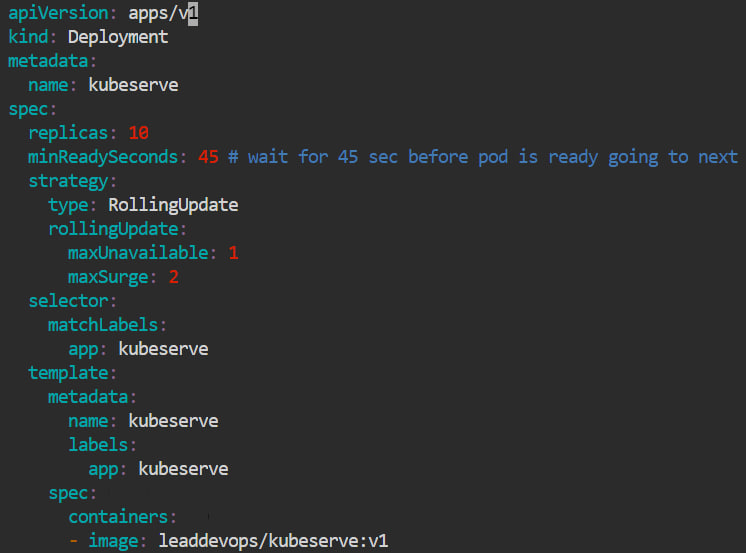
**Once you have created the user, role, and role binding, the user will have the necessary permissions to create, list, get, update, and delete pods in the Kubernetes cluster.**

1. **Configure application on the pod**

**To create the requested pods, we scale up/down**

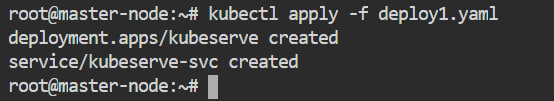
**In Rolling update/roll back , no down time will be there.**

**# vim delploy1.yaml**

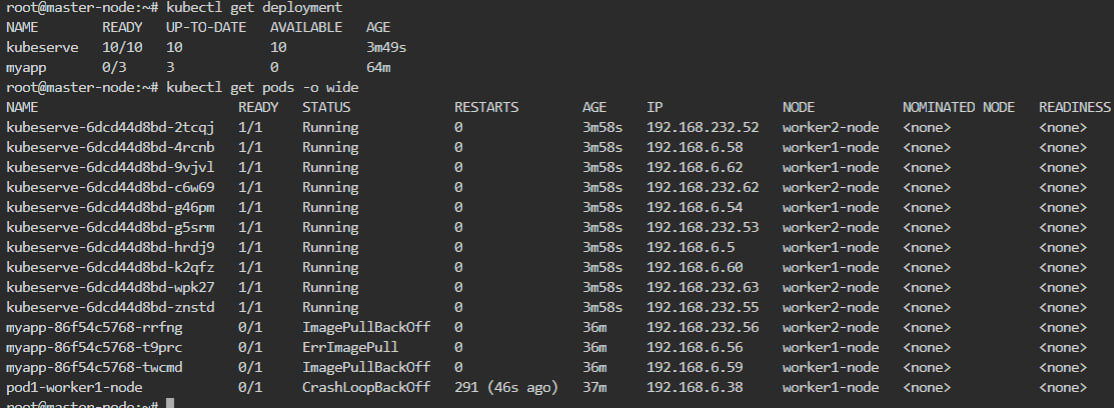
****

****

**#kubectl apply –f deploy1.yaml**

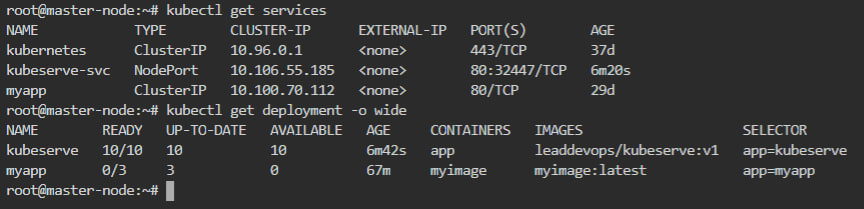
****

**#kubectl get pods –o wide**

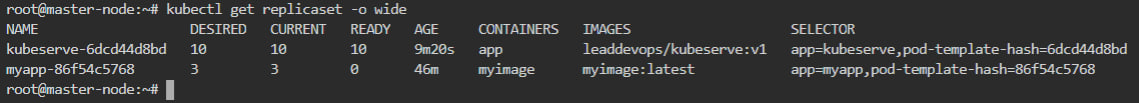
****

**#kubectl get services**

**#kubectl deployment –o wide**

****

**#kubectl get replicaset –o wide**

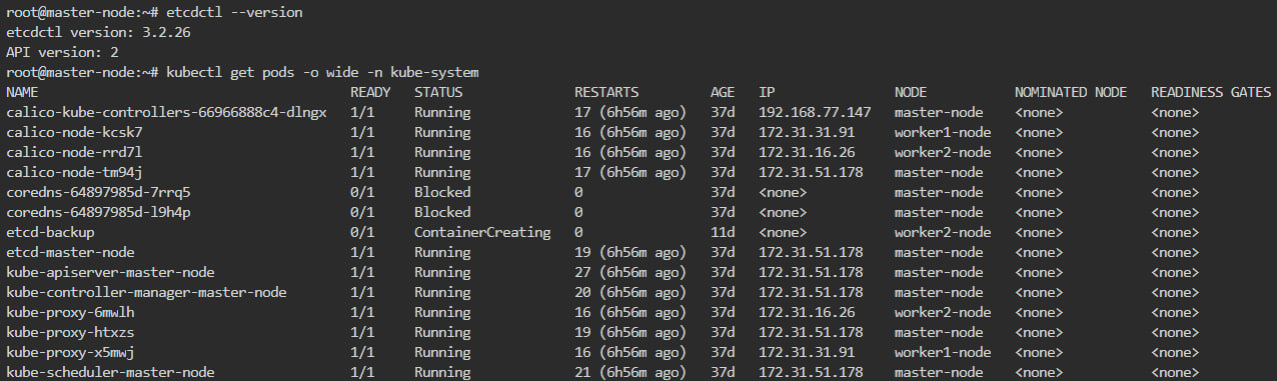
****

1. **Take snapshot of ETCD database**

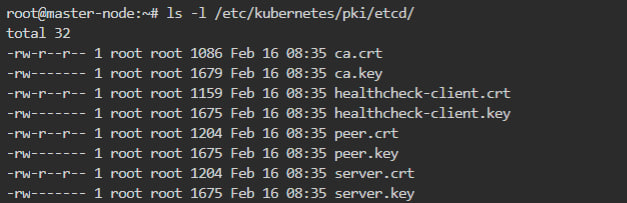
etcdctl --version

kubectl get pods -o wide

kubectl get pods -o wide -n kube-system



#ls -l /etc/kubernetes/pki/etcd/



#mkdir /etcd-backup

#ETCDCTL\_API=3 etcdctl --endpoints=172.31.51.178:2379 --cacert /etc/kubernetes/pki/etcd/ca.crt --cert /etc/kubernetes/pki/etcd/server.crt --key /etc/kubernetes/pki/etcd/server.key snapshot save /tmp/bakup

**To create a backup of etcd in a file /tmp/bakup with yaml file**

apiVersion: v1

kind: Pod

metadata:

name: etcd-backup

namespace: kube-system

spec:

containers:

- name: etcd-backup

image: gcr.io/etcd-development/etcdctl:v3.4.15

command:

- "/bin/sh"

- "-c"

- "ETCDCTL\_API=3 etcdctl --endpoints=https:// 172.31.51.178:2379 --cacert=/etc/kubernetes/pki/etcd/ca.crt --cert=/etc/kubernetes/pki/etcd/server.crt --key=/etc/kubernetes/pki/etcd/server.key snapshot save /tmp/bakup"

volumeMounts:

- name: etcd-certs

mountPath: /etc/kubernetes/pki/etcd

readOnly: true

- name: backup-dir

mountPath: /tmp

volumes:

- name: etcd-certs

secret:

secretName: etcd-certs

- name: backup-dir

hostPath:

path: /tmp

type: DirectoryOrCreate

* 1. **Set criteria such that if the memory of CPU goes beyond 50%, environments automatically get scaled up and configured**

To set up automatic scaling of your Kubernetes environment based on CPU and memory usage, you can use Kubernetes Horizontal Pod Autoscaler (HPA). HPA can automatically adjust the number of replicas of a Deployment, ReplicaSet or StatefulSet based on the observed CPU or memory utilization of the pods.

Here are the steps to set up an HPA with a threshold of 50% CPU and memory usage:

Add resource limits to your container spec in your pod configuration file to define the maximum amount of CPU and memory that the container can use. For example:

**#vim cupapp.yaml**

apiVersion: v1

kind: Pod

metadata:

name: myapp

spec:

containers:

- name: myapp-container

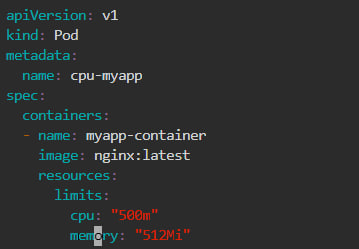
image: nginx:latest

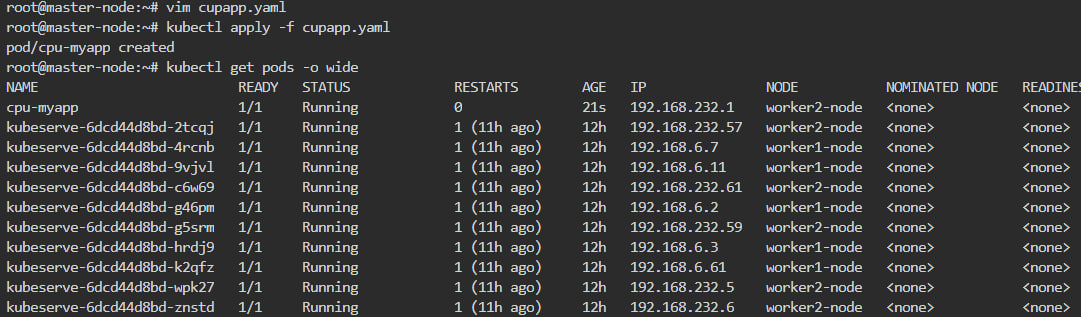
resources:

limits:

cpu: "500m"

memory: "512Mi"





This will limit the CPU usage to 50% (500m out of 1000m) and the memory usage to 512Mi.

Create a Horizontal Pod Autoscaler (HPA) for your Deployment, ReplicaSet or StatefulSet using the kubectl autoscale command. For example:

kubectl autoscale deployment myapp-deployment --cpu-percent=50 --min=1 --max=10

This will create an HPA for the myapp-deployment with a CPU usage target of 50%, a minimum of 1 replica and a maximum of 10 replicas.

The HPA controller will monitor the CPU and memory usage of the pods and adjust the number of replicas based on the configured CPU and memory thresholds. If the CPU or memory usage exceeds 50%, the HPA will increase the number of replicas up to the maximum configured number of replicas. If the CPU or memory usage drops below 50%, the HPA will decrease the number of replicas down to the minimum number of replicas.

Note that you can also use the kubectl edit hpa command to edit the HPA configuration or manually update the number of replicas of your deployment using the kubectl scale command.

With this setup, your Kubernetes environment will automatically scale up and down based on the CPU and memory usage of your pods, ensuring that your application always has enough resources to handle the workload.

Thank you.