Eric Gosiengfiao

Final Project: DevOps Capstone Project

Gitnub: <https://github.com/ericredlinesolutions/CapStoneProject>

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

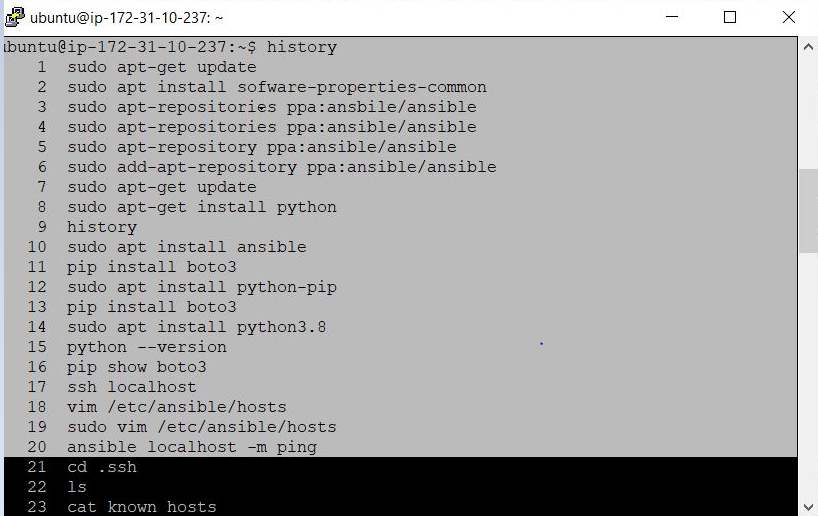
**The following things to be kept in check:**

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:

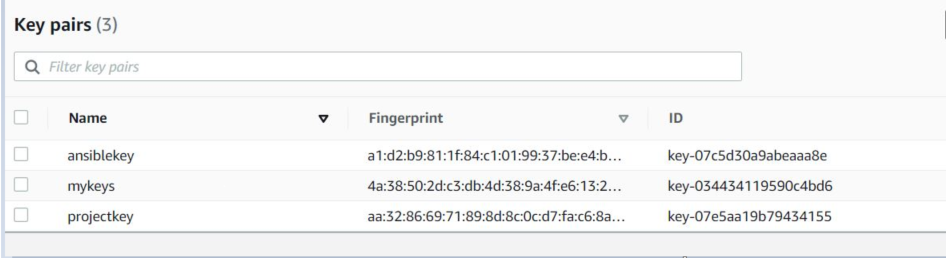
* Project and tester details
* Concepts used in the project
* Links to the GitHub repository to verify the project completion
* Your conclusion on enhancing the application and defining the USPs (Unique Selling Points)

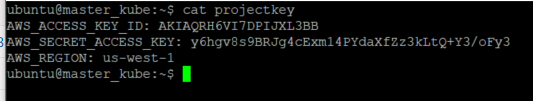
**Project Implementation Requirements:**

**Enviromental Installation Ansible:**

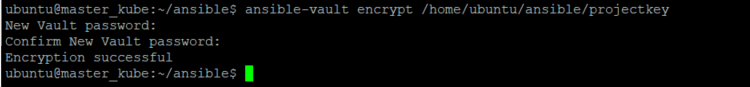


**Key Pairs: projectkey (project\_user)and vault encrypt**

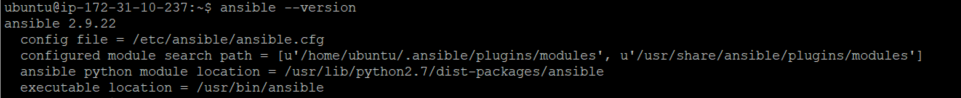




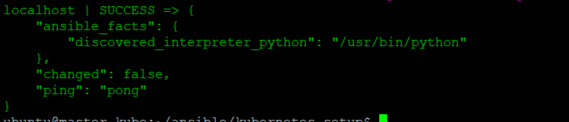
**Encrypt keypair projectkeys. Password: master\_kube1**



**Ansible Version Installed:**



ansible localhost -m ping



**Ansible Playbook: AWS Deploy EC2 Instance:**

**Ansible Modules used from resource:**

<https://docs.ansible.com/ansible/latest/collections/amazon/aws/ec2_module.html>

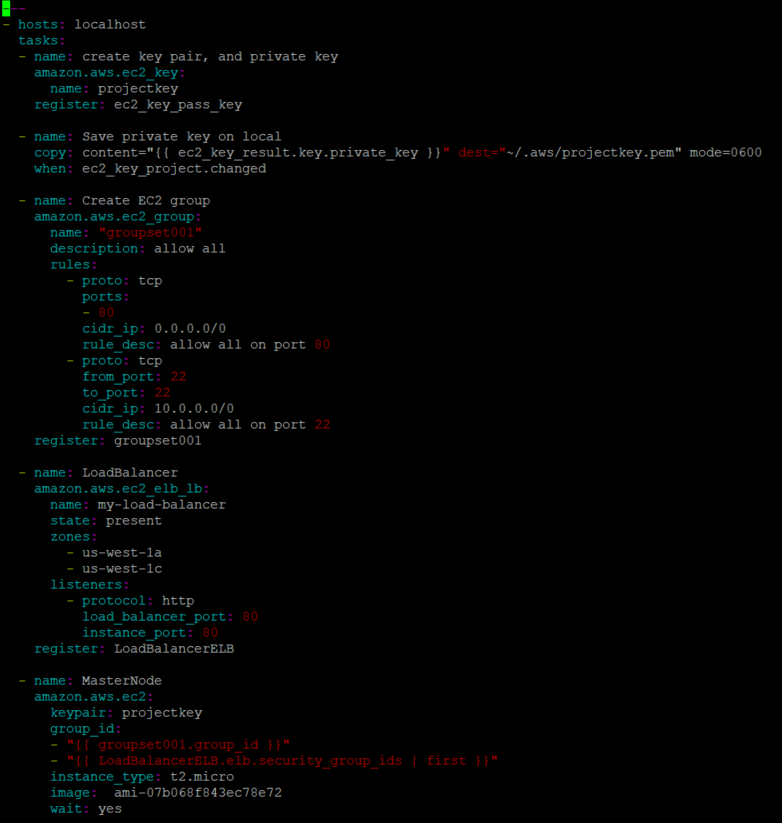
<https://docs.ansible.com/ansible/latest/collections/community/aws/ec2_eip_module.html>

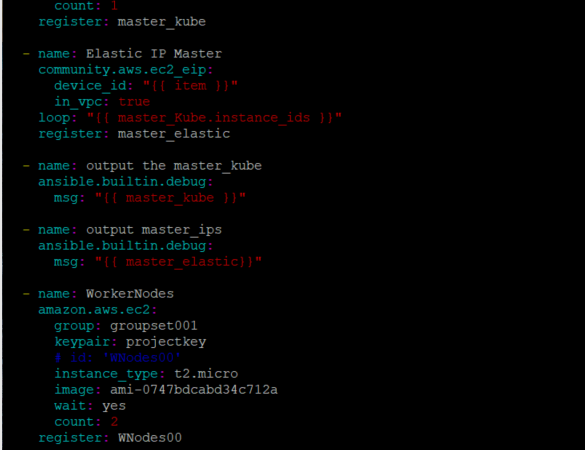
<https://docs.ansible.com/ansible/latest/collections/amazon/aws/ec2_elb_lb_module.html>

<https://docs.ansible.com/ansible/latest/collections/amazon/aws/ec2_key_module.html>

**EC2playbook.yaml** will automate the provisioning of an EC2 instances. Create EC2 group, LoadBalancer, keypair, Elastic IP and two WorkerNodes.

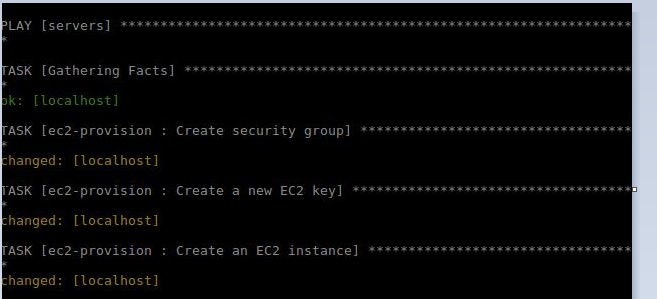
vi EC2playbook.yaml



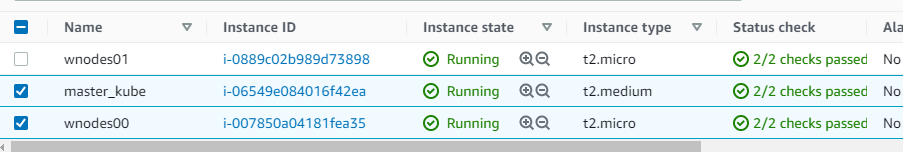


**ansible**

ansible-playbook EC2playbook.yaml



**AWS EC2 Instace Dashboard**



**Install Docker and Kubernetes on a cluster:**

**Resource:**

[**https://kubernetes.io/blog/2019/03/15/kubernetes-setup-using-ansible-and-vagrant/**](https://kubernetes.io/blog/2019/03/15/kubernetes-setup-using-ansible-and-vagrant/)

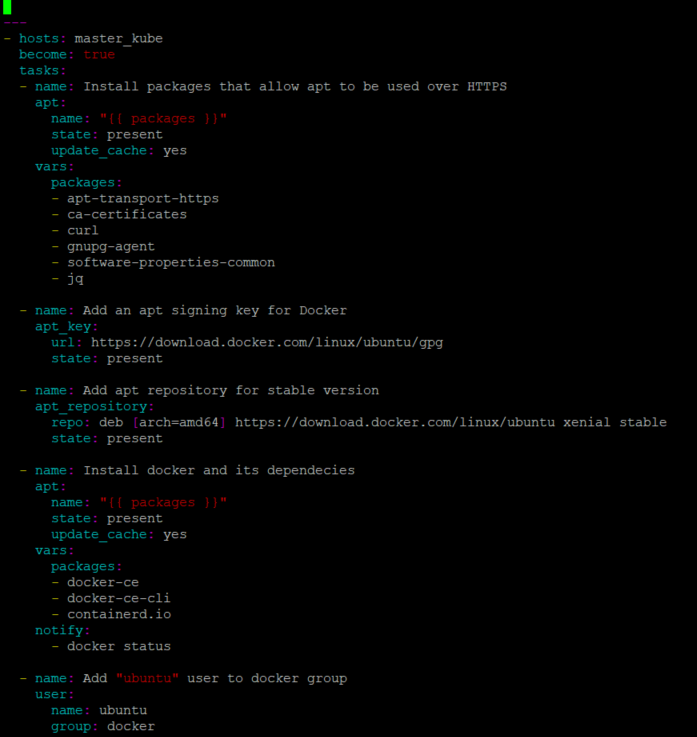
Create vagrantfile with master\_kube(master) **vagrantfile**. Denote 2 count number of nodes. We will create two playbooks.

**master-playbook.yml will be executed for the master node**. - ansible-playbook master-playbook.yaml

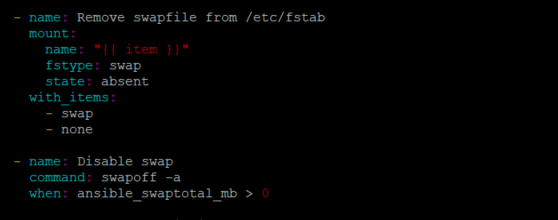
**node-playbook.yml will be executed for the workernodes**. -ansible-plabook node-playbook.yml

Description Steps

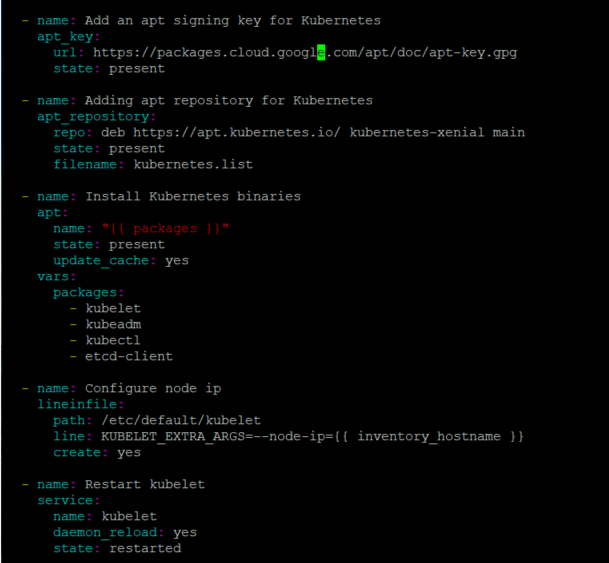
1. Install Docker and its dependent components



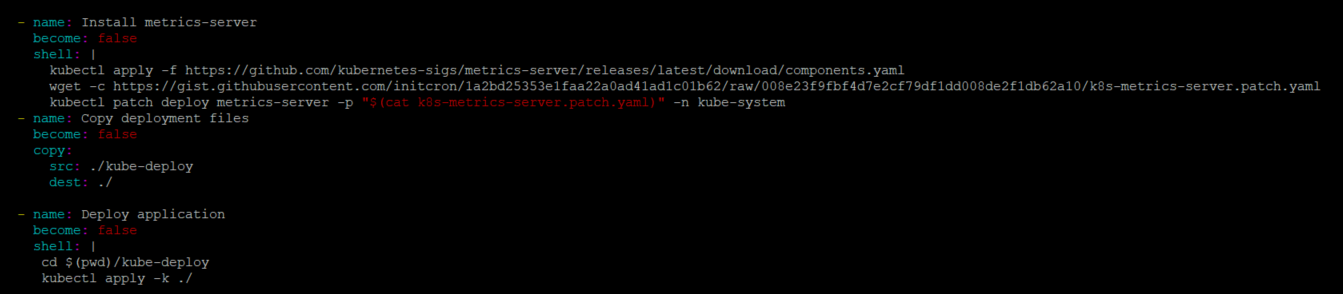
1. Disabling Swap for Kubelet.



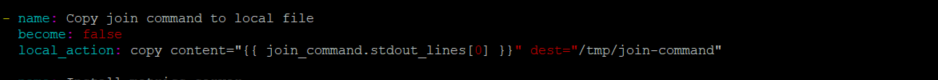
1. Add packages Kubelet, kubeadm and kubectl



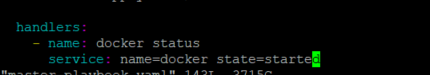
1. Implement Weavenet and Metric Server



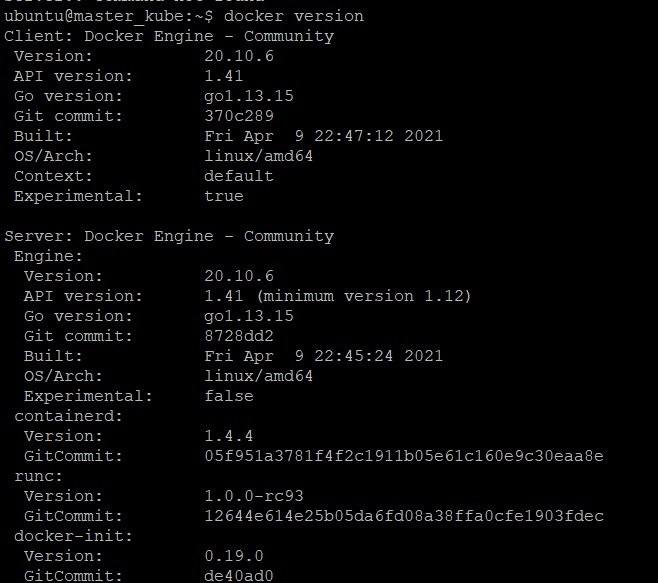
1. Join the node to the cluster

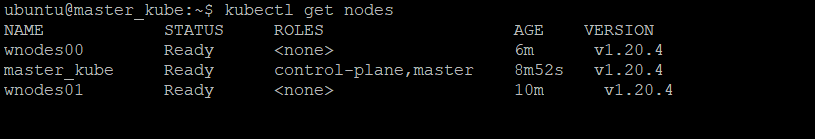


1. Check status docker.



Docker version:





**Create a new user with permissions to create, list, get update and delete pods:**

* 1. **First deploy the Kubernetes Dashboard**
  2. **Create a user**
  3. **Apply the permission**

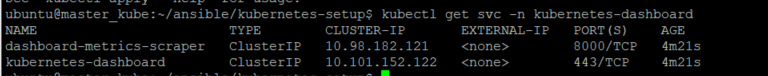
**Deployment:**

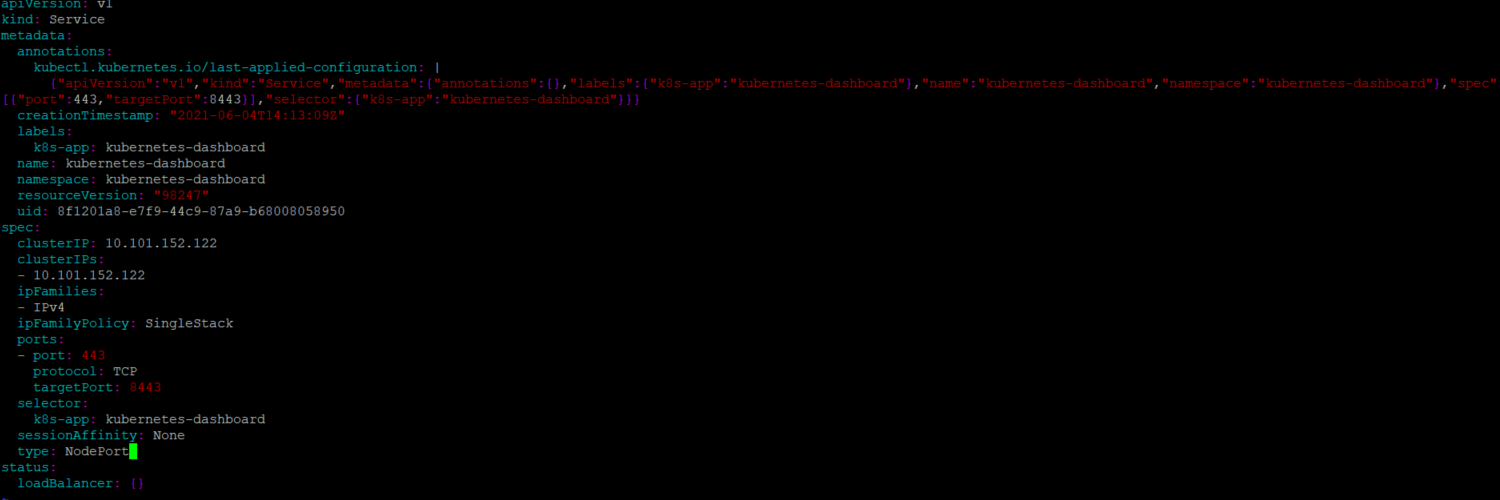
* 1. **Deploy Kubernetes Dashboard**

kubectl apply -f

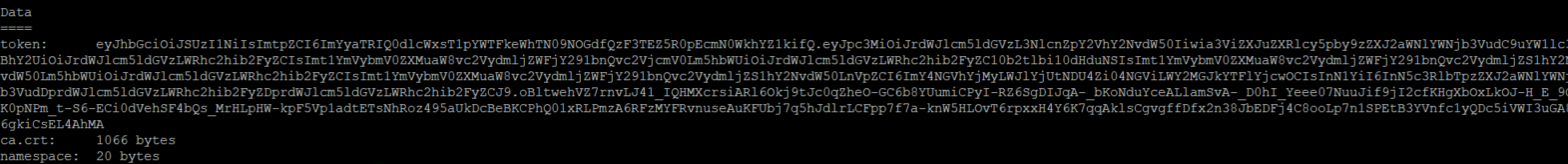
https://raw.githubusercontent.com/kubernetes/dashboard/v2.0.0/aio/deploy/recommended.yaml

kubectl get svc -n kubernetes-dashboard >>> check service dashboard



kubectl edit svc kubernetes-dashboard -n kubernetes-dashboard >NodePort 

kubectl -n kubernetes-dashboard describe secret $(kubectl -n kubernetes-dashboard get secret | grep kubernetes-dashboard | awk '{print $1}') >> grab token



* 1. **Create a user**

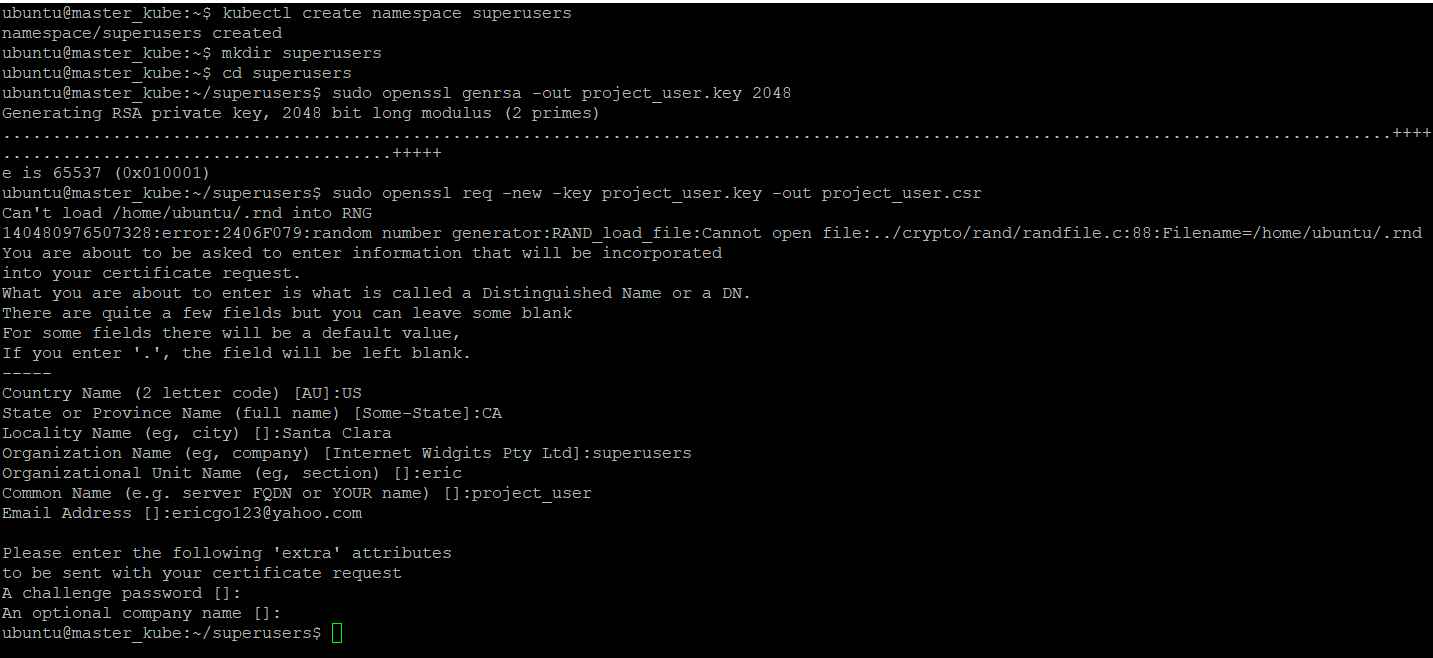
namespace: superusers

username: project\_user

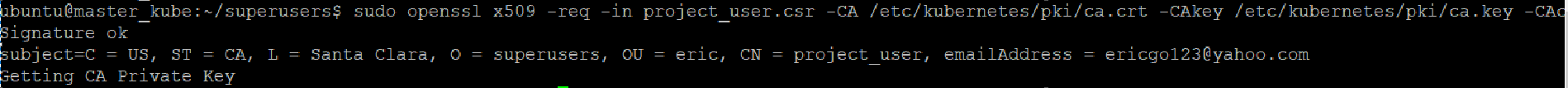
kubectl create namespace superusers

sudo openssl genrsa -out project\_user.key 2048

sudo openssl req -new -key project\_user.key -out project\_user.csr

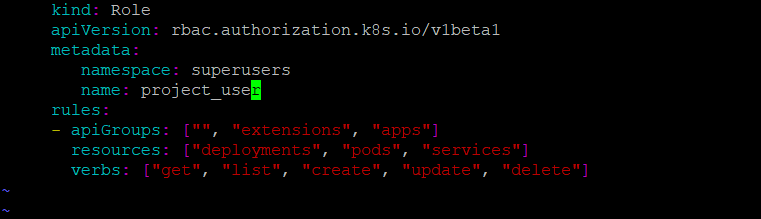


sudo openssl x509 -req -in project\_user.csr -CA /etc/kubernetes/pki/ca.crt -CAkey /etc/kubernetes/pki/ca.key -CAcreateserial -out project\_user.crt -days 1000



**c. Apply the permission verbs: get, list, create, update, delete**

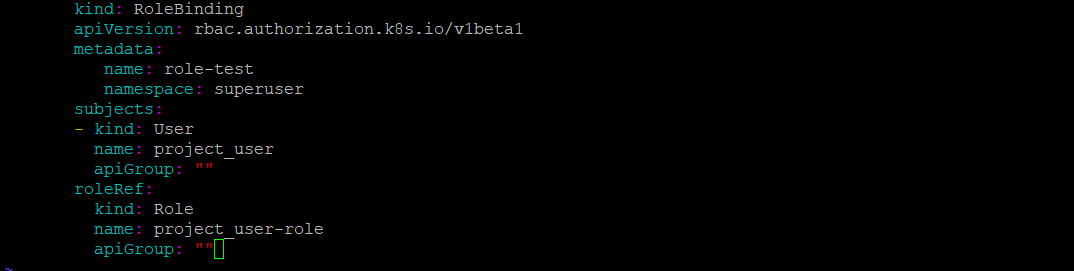
Create a role vi projectrole.yaml



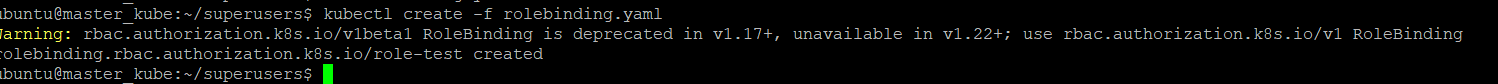
kubectl create -f projectrole.yaml



Create a binding vi rolebinding.yaml

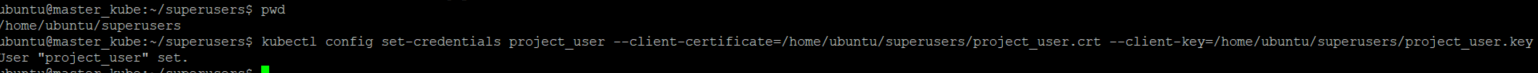


kubectl create -f rolebinding.yaml



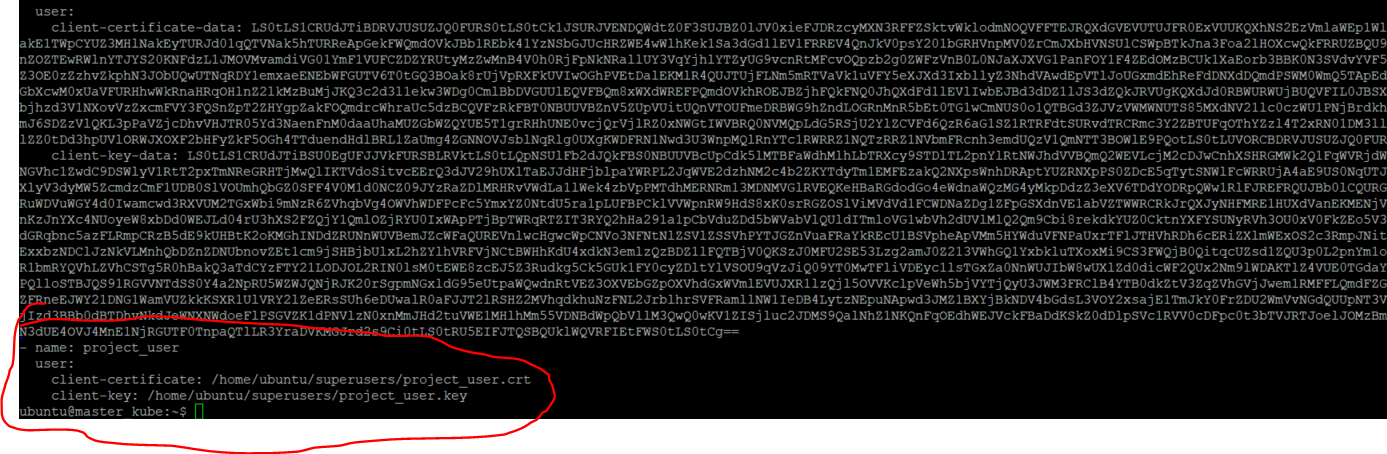
**Set Credentials**

**kubectl config set-credentials project\_user --client-certificate=/home/ubuntu/superusers/project\_user.crt --client-key=/home/ubuntu/superusers/project\_user.key**



**Confirmation role binding with the user**

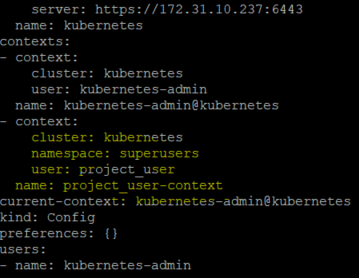
cat .kube/config



Set the context and edited the .kube/config to remove the admin credentials

kubectl config set-context project\_user-context --cluster=kubernetes --namespace=superusers --user=project\_user



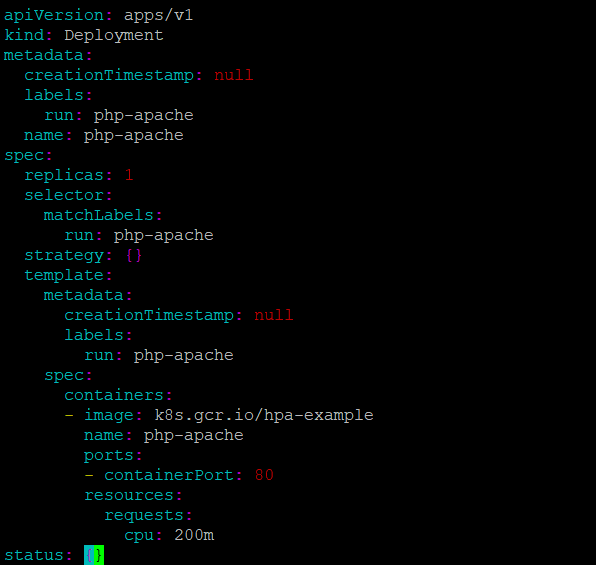


**Configure application on the pod**

The hpa.yml will include the configuration of the pod

vi hpa.yaml

kubectl create -f hpa.yml

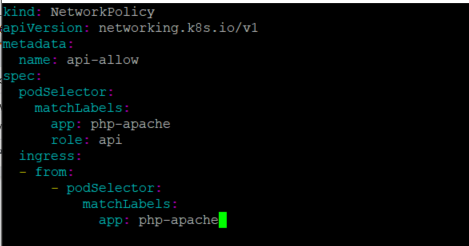


**Implement the network policies at the database pod to allow ingress traffic from the front-end application pod**

**We will create ingress allow traffic using podselector that has an app: php-apache pod**

**vi ingress.yaml**

**kubectl create -f ingress.yaml**



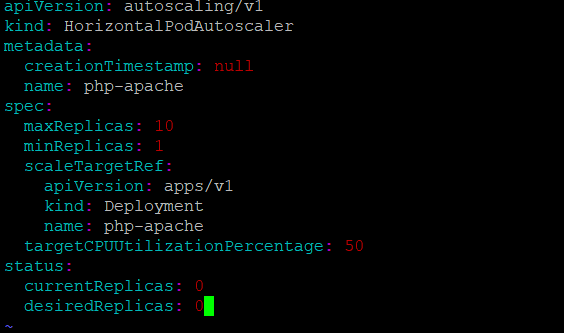
**Set criteria such that if the memory of CPU goes beyond 50% environements automatically get scaled up and configured**

**We will use two yaml file:**

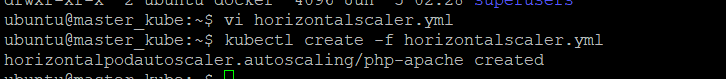
**horizantalscaler.yaml** will scale up on targeted CPU utilization exceeding 50% threshold

**hpa.yaml** will deploy the application PHP Apache and Services.

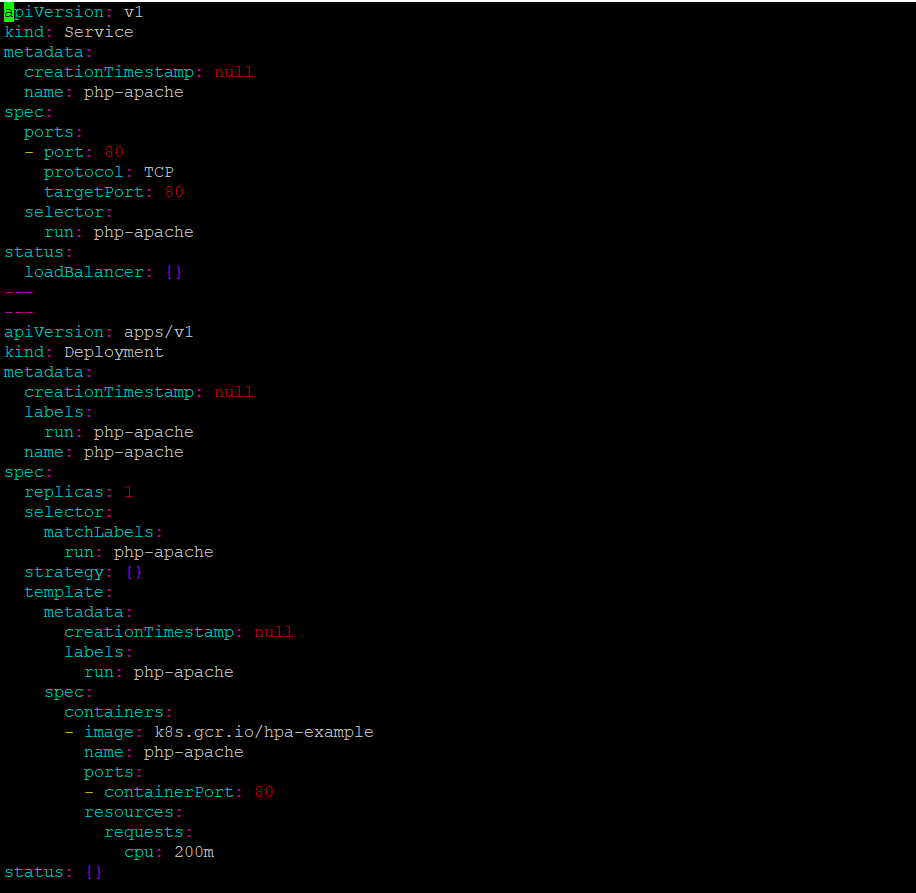
vi horizontalscaler.yml

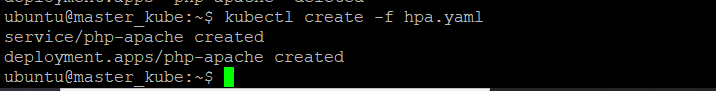


kubectl create -f horizontalscaler.yml



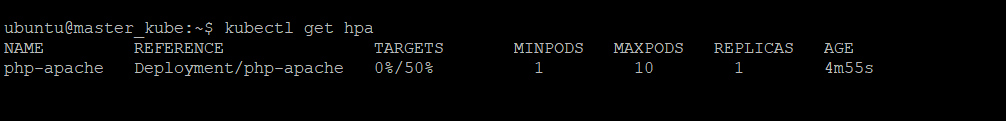
vi hpa.yaml





Health-Check the horizonalscaler

* 1. kubectl get nodes hpa = show the value 1 minpods, 10 maxpods 1 replica

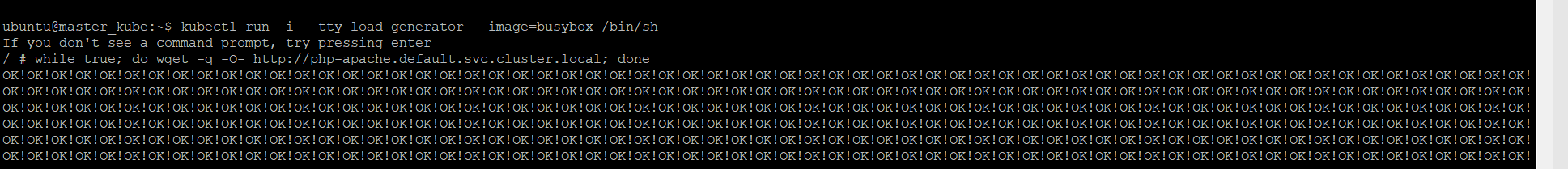


* 1. This command will start the load generator:

**kubectl run -i --tty load-generator --image=busybox /bin/sh**

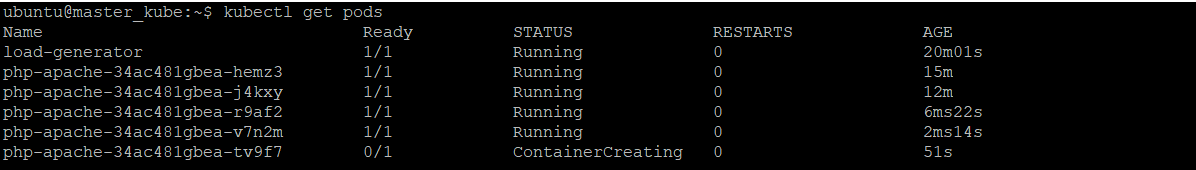
This commad will perform a loop

**while true; do wget -q -O- http://php-apache.default.svc.cluster.local;**



* 1. Scaling when it reaches 50% CPU utilization

kubectl get pods

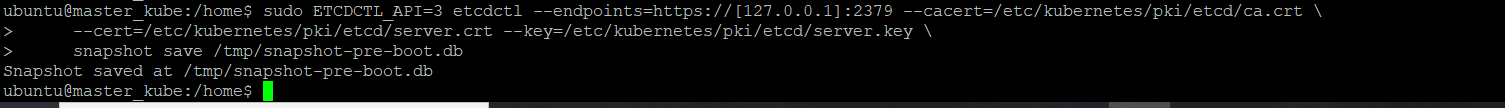


**Take snapshots of ETCD database**

sudo ETCDCTL\_API=3 etcdctl --endpoints=https://[127.0.0.1]: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/snapshot-pre-boot.db



To restore ETCD snapshot to a repository folder:

sudo ETCDCTL\_API=3 etcdctl --endpoints=https://[127.0.0.1]:2379 --cacert=/etc/kubernetes/pki/etcd/ca.crt \

--name=master \

--cert=/etc/kubernetes/pki/etcd/server.crt --key=/etc/kubernetes/pki/etcd/server.key \

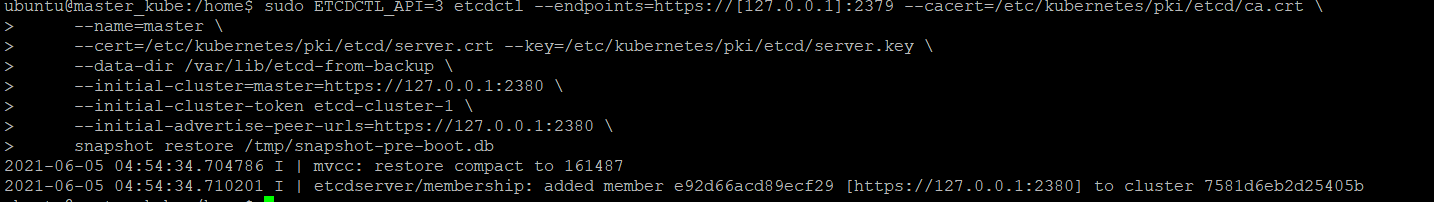
--data-dir /var/lib/etcd-from-backup \

--initial-cluster=master=https://127.0.0.1:2380 \

--initial-cluster-token etcd-cluster-1 \

--initial-advertise-peer-urls=https://127.0.0.1:2380 \

snapshot restore /tmp/snapshot-pre-boot.db



**Conclusion:**

Implementing Ansible automation provision in AWS EC2 instances will prevent costly expense and downtime versus manual provisioning.

Docker containers coupled with Kubenetes management tools will be the infrastructure solution for scalability and will prevent Easy Pay from timing out due to increased demand from its customer base.

New user permission will add security to Easy Pay access to its Kubernetes management tool. Since, ETCD is used to store all Kubernetes (management tool) data, it is critical that we create a backup/snaphost and a restore protocol in case of a emergency.