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

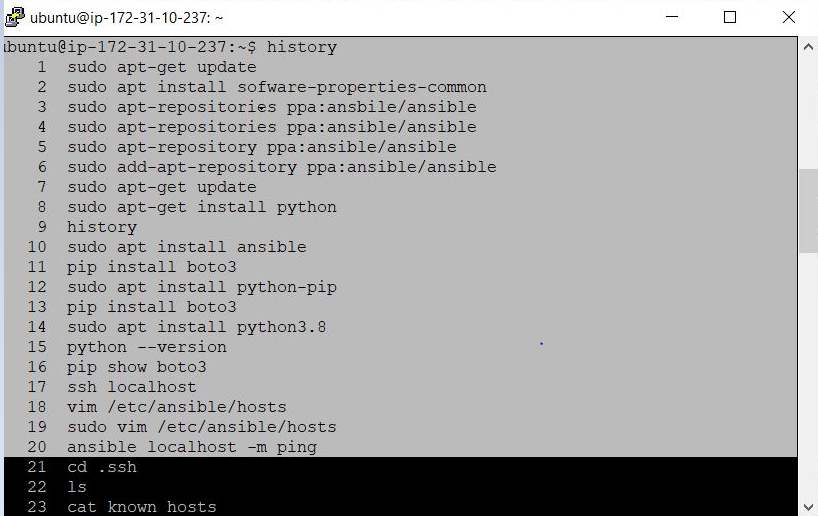
**The following things to be kept in check:**

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:

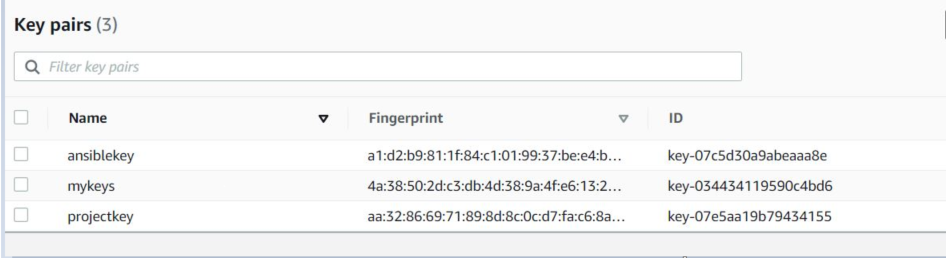
* Project and tester details
* Concepts used in the project
* Links to the GitHub repository to verify the project completion
* Your conclusion on enhancing the application and defining the USPs (Unique Selling Points)

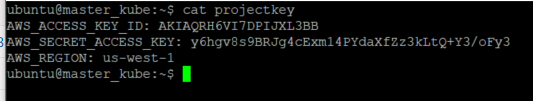
Project Implementation Requirements

**Envriomental Installation Ansible:**

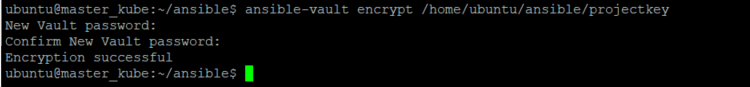


**Key Pairs: projectkey (project\_user)and vault encrypt**

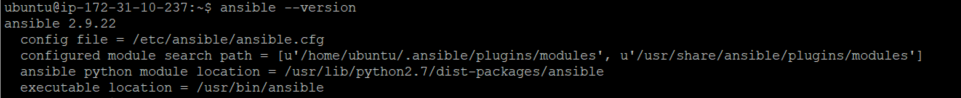


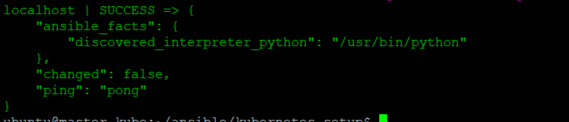


**Encrypt keypair projectkeys. Password: master\_kube1**



**Ansible Version Installed:**





**Ansible Playbook:**

**Ansible Modules used from resource:**

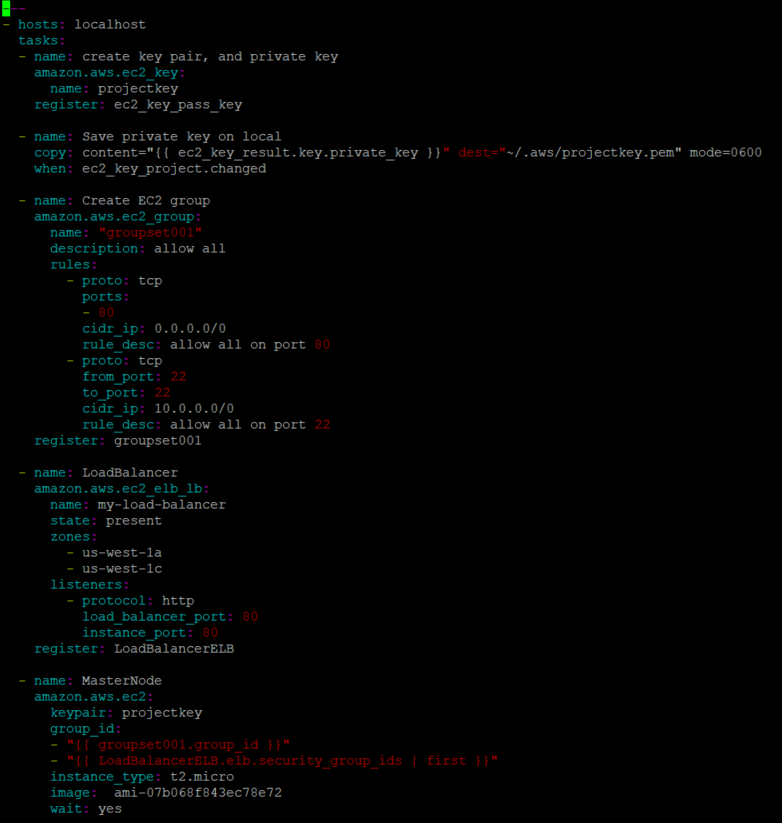
<https://docs.ansible.com/ansible/latest/collections/amazon/aws/ec2_module.html>

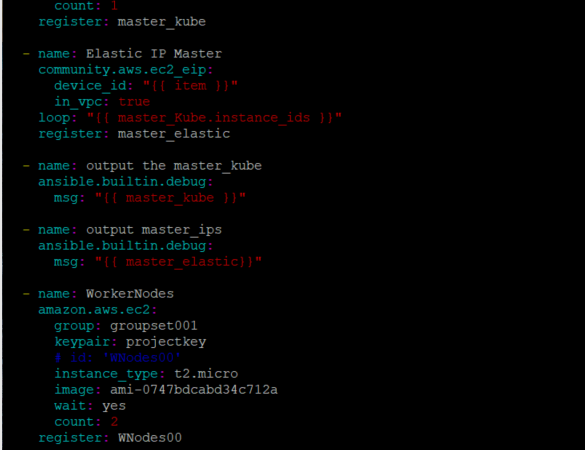
<https://docs.ansible.com/ansible/latest/collections/community/aws/ec2_eip_module.html>

<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.





**Install Docker and Kubernetes:**

**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/)

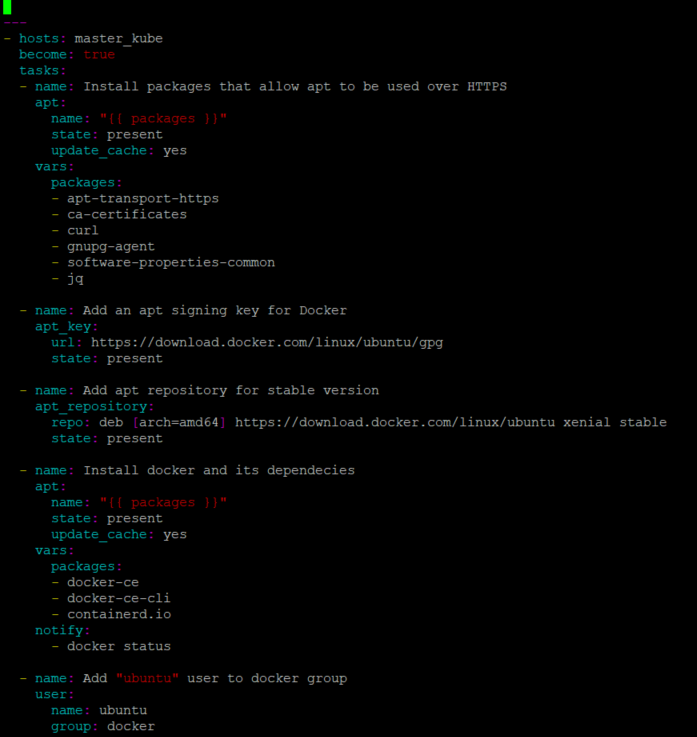
Create vagrantfile with master\_kube(master) **vagrantfile**. Denote 2 count number of nodes. We will create two playbooks.

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

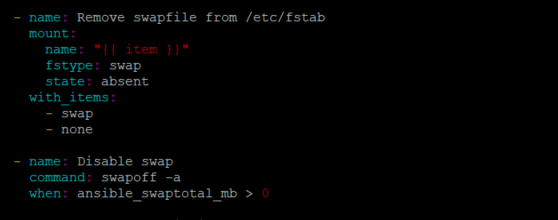
**node-playbook.yml will be executed for the workernodes.**

Description Steps

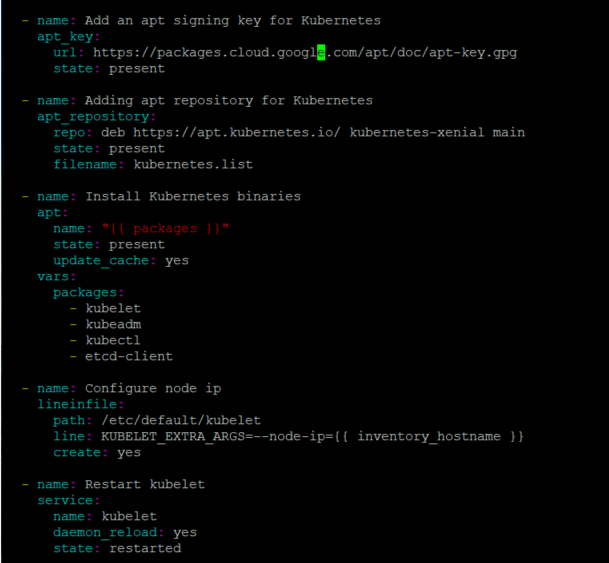
1. Install Docker and its dependent components



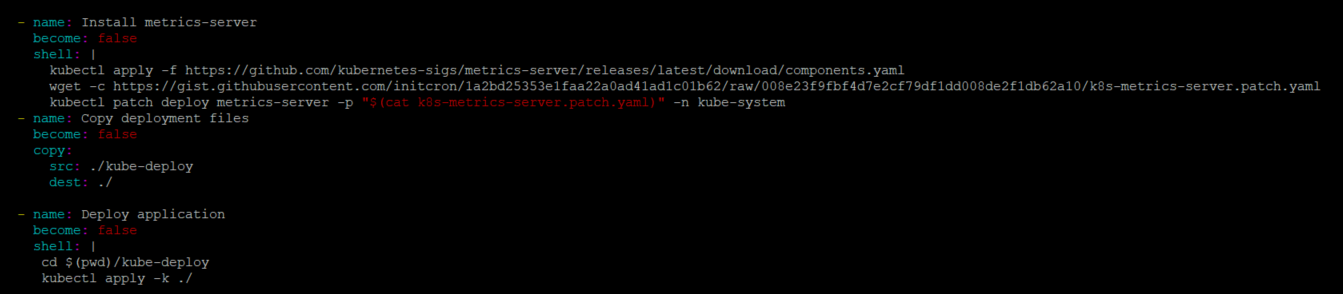
1. Disabling Swap for Kubelet.



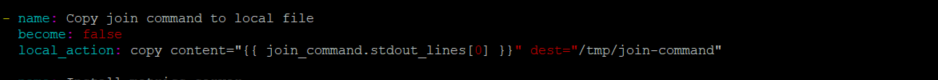
1. Add packages Kubelet, kubeadm and kubectl



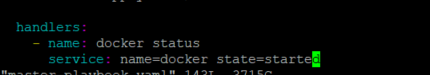
1. Implement Weavenet and Metric Server



1. Join the node to the cluster



1. Check status docker.



**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**

**https://8gwifi.org/docs/kube-rbac.jsp**

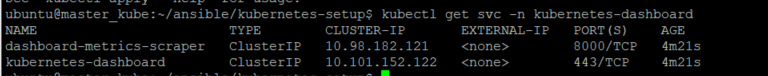
**Deployment:**

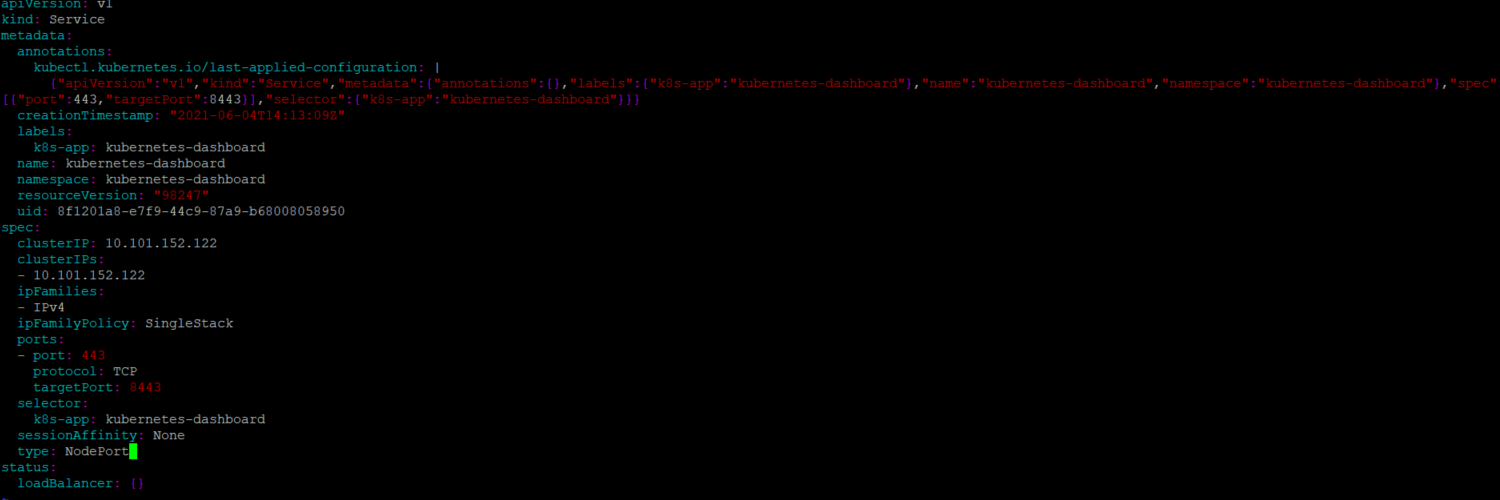
1. **Deploy Kubernetes Dashboard**

kubectl apply -f

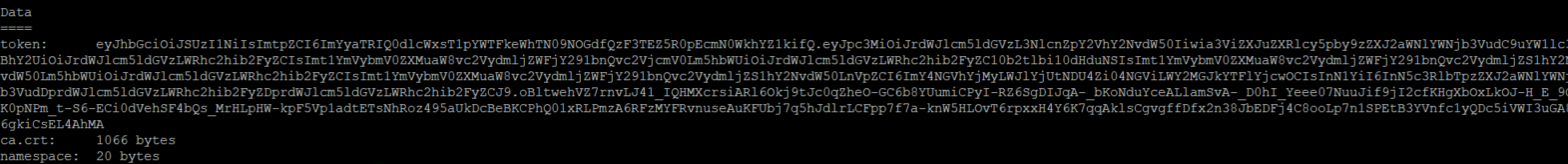
https://raw.githubusercontent.com/kubernetes/dashboard/v2.0.0/aio/deploy/recommended.yaml

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



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

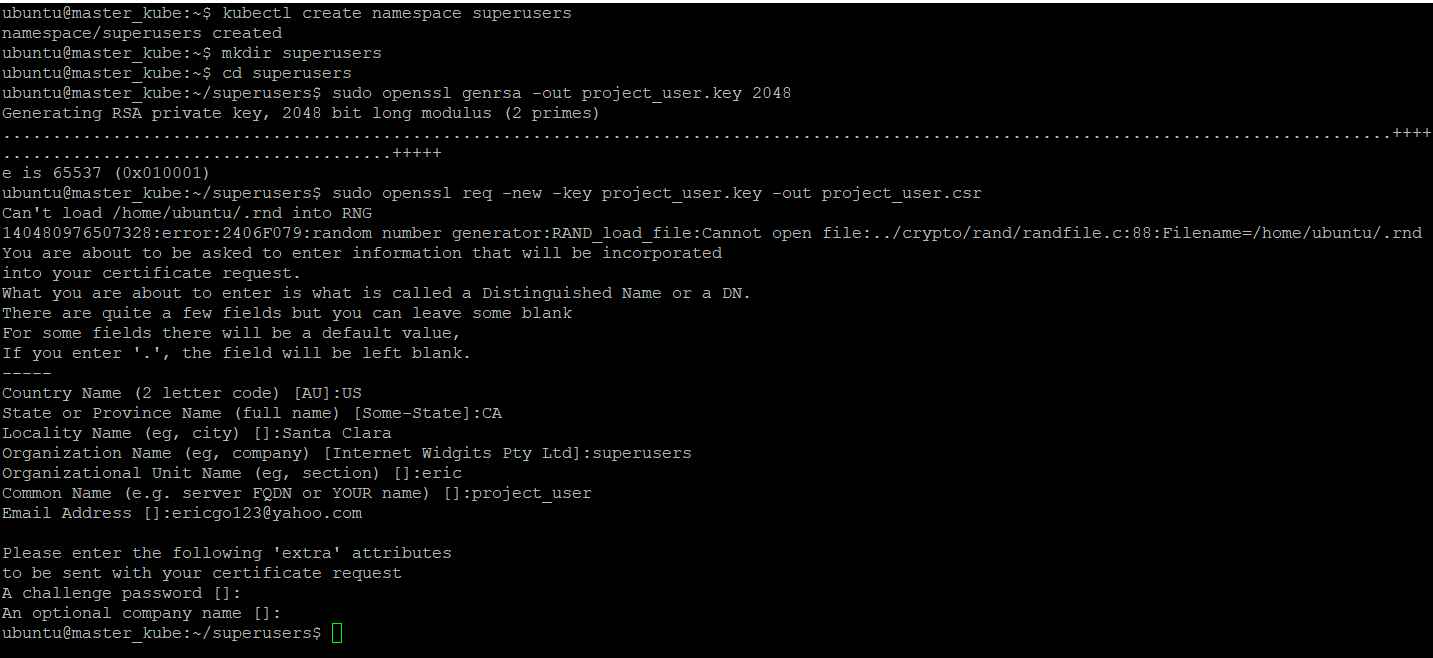
kubectl -n kubernetes-dashboard describe secret $(kubectl -n kubernetes-dashboard get secret | grep kubernetes-dashboard | awk '{print $1}') >> grab token



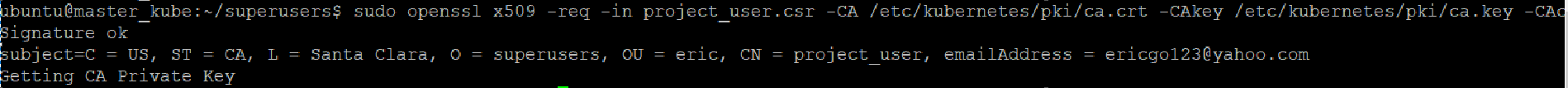
Create a namespace: superusers

kubectl create namespace superusers

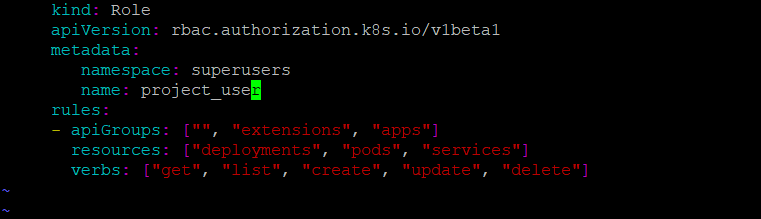
Create directory and private key



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



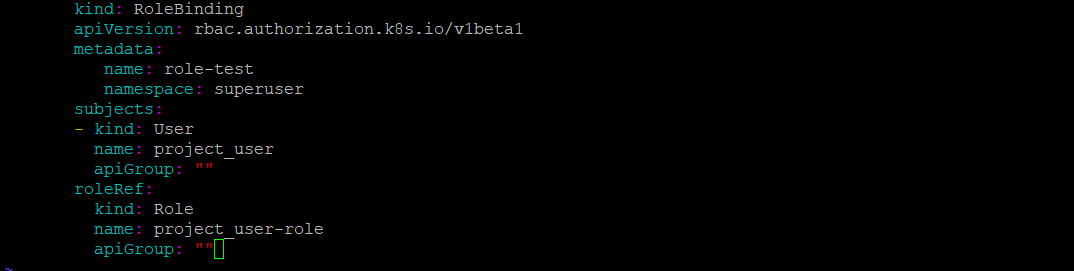
**vi projectrole.yaml**



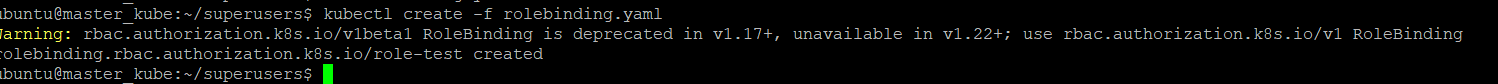
kubectl create -f projectrole.yaml



vi rolebinding.yaml

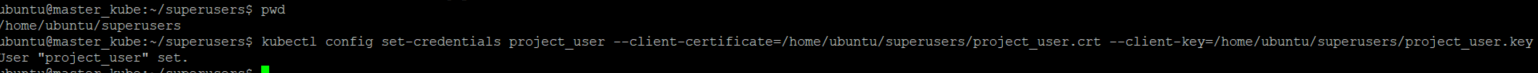


kubectl create -f rolebinding.yaml



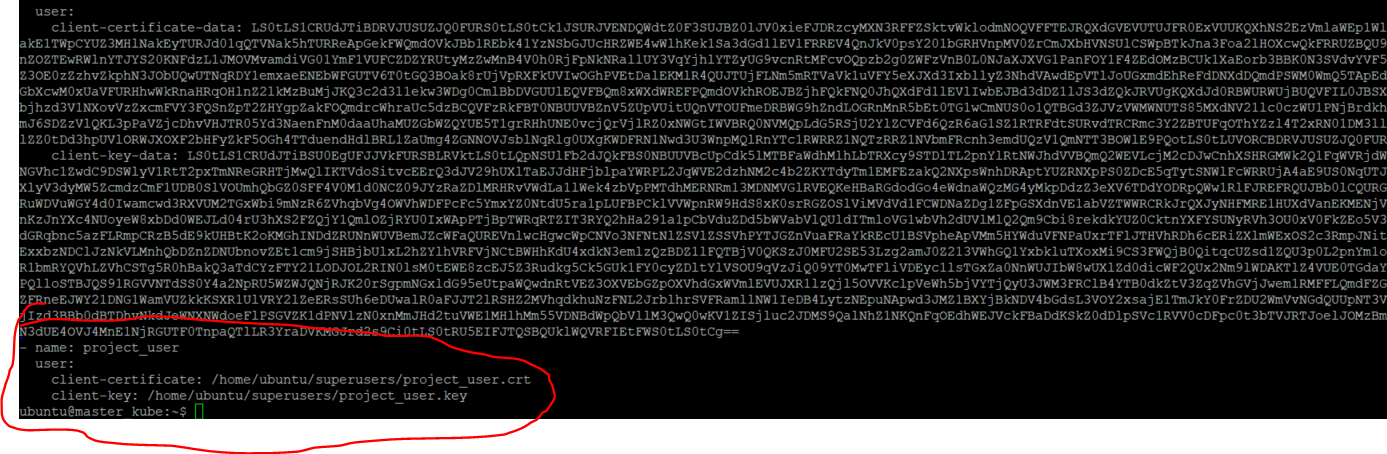
**Set Credentials**

**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

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

