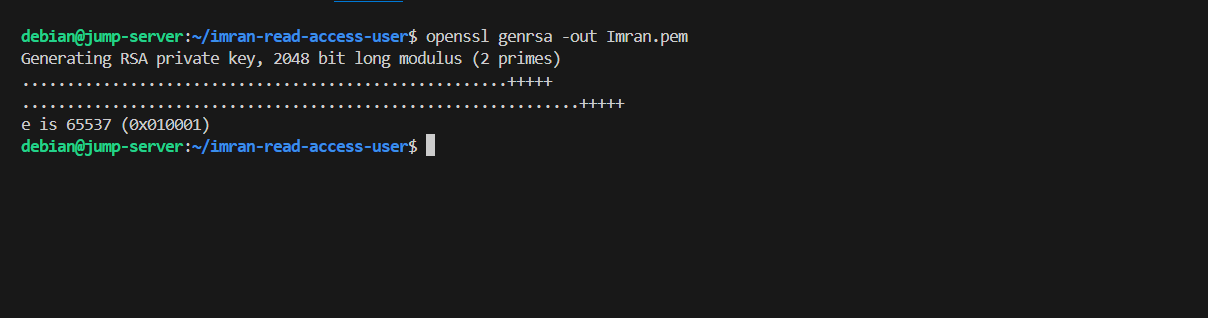
# **Create a new user in GKE Autopilot Private Cluster**

To test RBAC we need to a user, so first create a new user. Below is the step-by-step guide to creating a new user named **Imran** in a GKE Private cluster.

**Step 1: Generate Certificates for the User**

Generate a private key for Imran using RSA algorithm (4096 bits):

**openssl genrsa -out Imran.pem**

This command will generate an RSA private key.

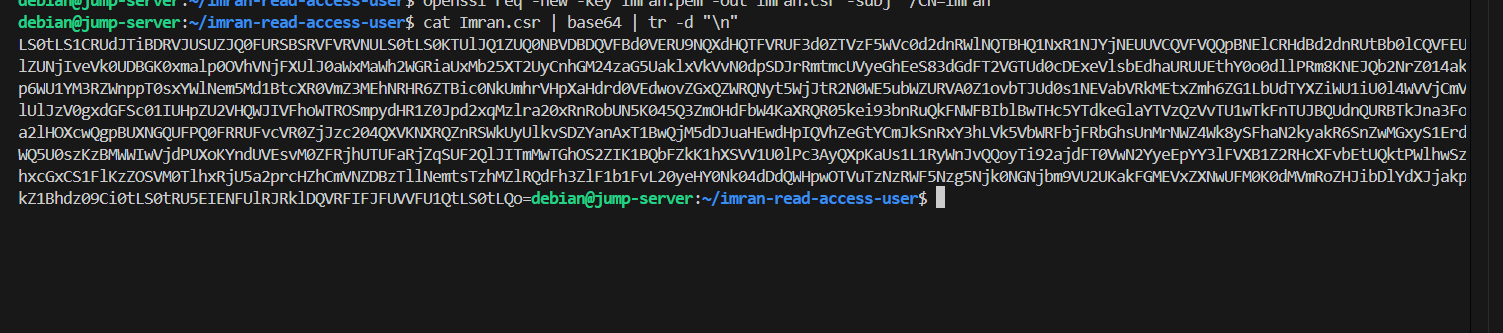
**Step 2: Create a Certificate Signing Request (CSR) for Imran:**

**openssl req -new -key Imran.pem -out Imran.csr -subj "/CN=Imran"**

## **Step 3: Create a Certificate Signing Request (CSR)**

1. Obtain the base64-encoded representation of the CSR:

**cat Imran.csr | base64 | tr -d "\n"**

Here we encode the CSR to be used in the **CertificateSigningRequest**.

2. Use the output to create a CertificateSigningRequest resource:

cat <<EOF | kubectl apply -f -

apiVersion: certificates.k8s.io/v1

kind: CertificateSigningRequest

metadata:

name: Imran

spec:

request: 

signerName: kubernetes.io/kube-apiserver-client

expirationSeconds: 86400 # one day

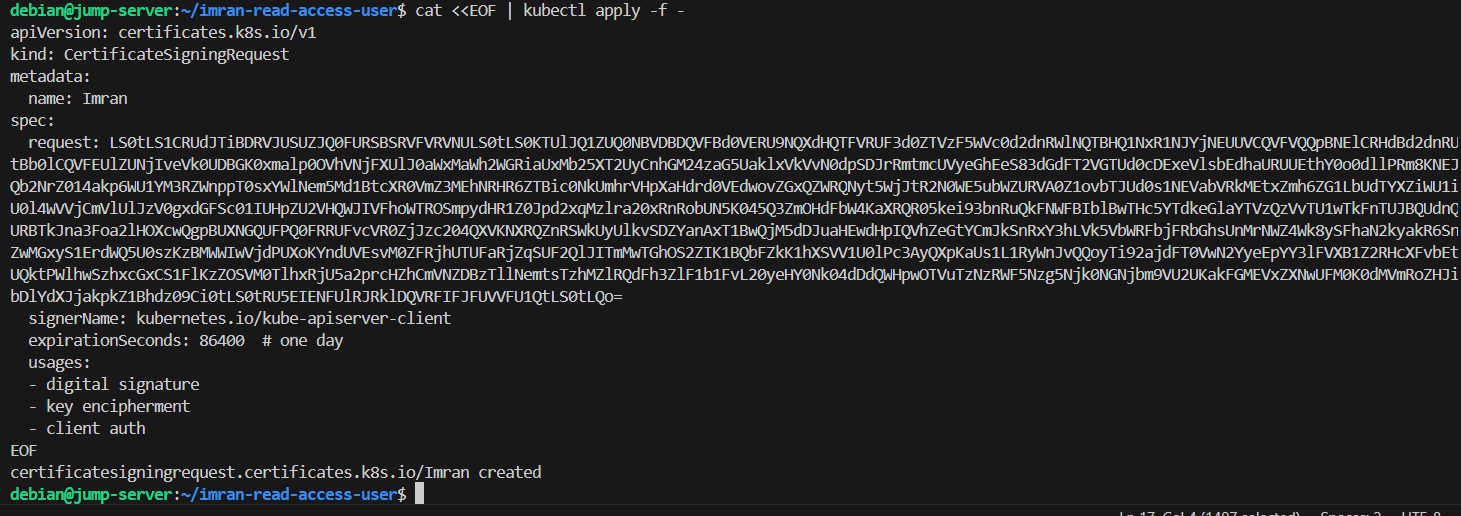
usages:

- digital signature

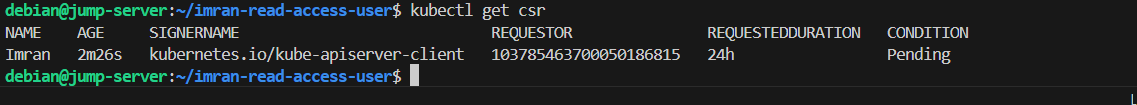
- key encipherment

- client auth

EOF

**CSR status**

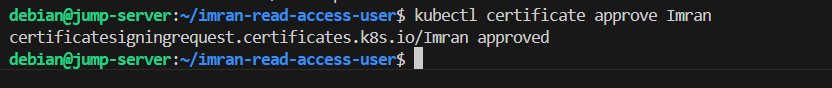
**kubectl get csr**



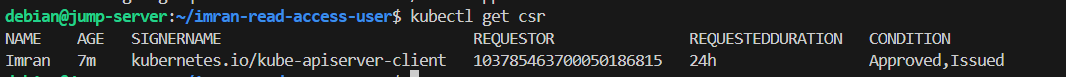
**Step 4: Sign the Certificate Using the Cluster Certificate Authority**

1. Approve the CertificateSigningRequest for Imran:

**kubectl certificate approve Imran**

2. Check the CSR status:

**kubectl get csr**

**Step 5: Create a Configuration Specific to the User**

1. Extract the signed certificate from the CertificateSigningRequest:

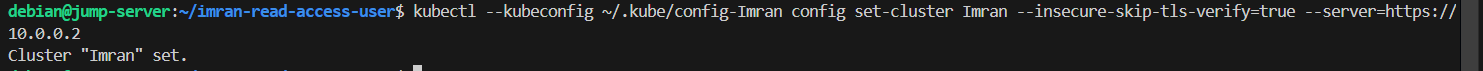
**kubectl get csr/Imran -o jsonpath="{.status.certificate}" | base64 -d > Imran.crt**

2. Create new user config file:

Use the kubectl config set-cluster command to set up the cluster information:

**kubectl --kubeconfig ~/.kube/config-Imran config set-cluster Imran --insecure-skip-tls-verify=true --server=https://10.0.0.2**

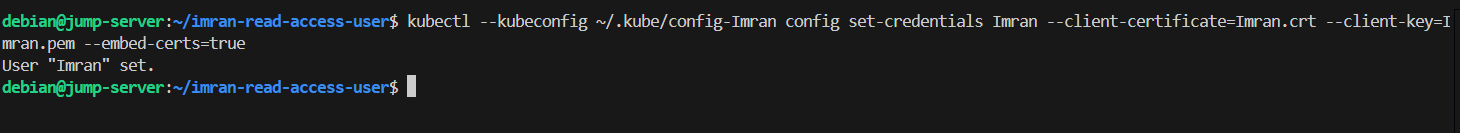
Note: Server= Cluster private endpoint

3. Set user credentials:

Use the kubectl config set-credentials command to set up the user credentials:

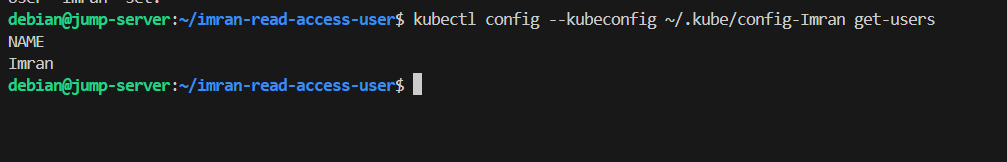
**kubectl --kubeconfig ~/.kube/config-Imran config set-credentials Imran --client-certificate=Imran.crt --client-key=Imran.pem --embed-certs=true**

Note: Replace Imran.crt and Imran.pem with the paths to your user certificate and private key files respectively.



4. Check the user status

**kubectl config --kubeconfig ~/.kube/config-Imran get-users**



5. Set context information:

Use the kubectl config set-context command to set up the context information:

**kubectl --kubeconfig ~/.kube/config-Imran config set-context Imran --cluster=Imran --user=Imran**

6. Use the context:

Finally, use the kubectl config use-context command to use the newly created context:

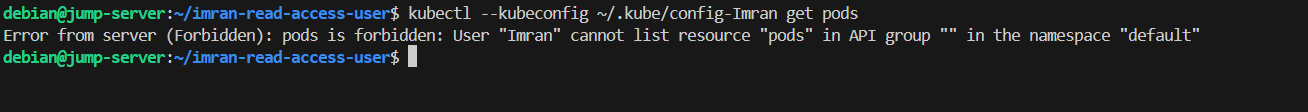


Now, your config-Imran file is configured with the necessary cluster, user, and context information. You can use it with kubectl commands by passing --kubeconfig ~/.kube/config-Imran. Make sure to replace placeholder values with your actual configuration details.

Example:

**kubectl --kubeconfig ~/.kube/config-Imran get pods**

We’ve successfully created a new user named “Imran”. However, when we try to access the pods using this user, we encounter a **Forbidden** error:



This error occurs because we haven’t assigned any permissions to the “**Imran**” user yet; we’ve only created the user.

Now, let’s proceed to create new Cluster roles and Cluster role bindings and associate them with the “**Imran**” user using the following steps.

# **Writing RBAC Rules**

Let’s create RBAC rules and apply to the above user:

**Example 1: Allow read-only access to all resources in the Cluster**

**1. Cluster Role YAML:**

Create a file named **clusterrole.yaml** with the following content:

apiVersion: rbac.authorization.k8s.io/v1

kind: ClusterRole

metadata:

name: node-viewer

rules:

- apiGroups: [""]

resources: ["\*"]

verbs: ["get", "list"]

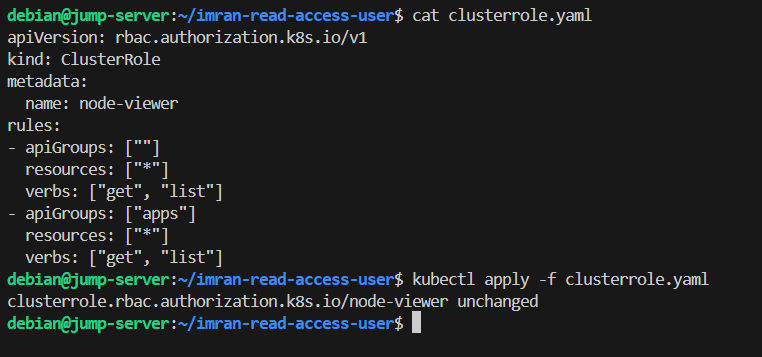
- apiGroups: ["apps"]

resources: ["\*"]

verbs: ["get", "list"]

Apply the Role using:

**kubectl apply -f clusterrole.yaml**



**2. Cluster Role Binding .yaml**

Create a file named **clusterbind.yaml** with the following content:

apiVersion: rbac.authorization.k8s.io/v1

kind: ClusterRoleBinding

metadata:

name: node-viewer-binding

subjects:

- kind: User

name: Imran # username

apiGroup: rbac.authorization.k8s.io

roleRef:

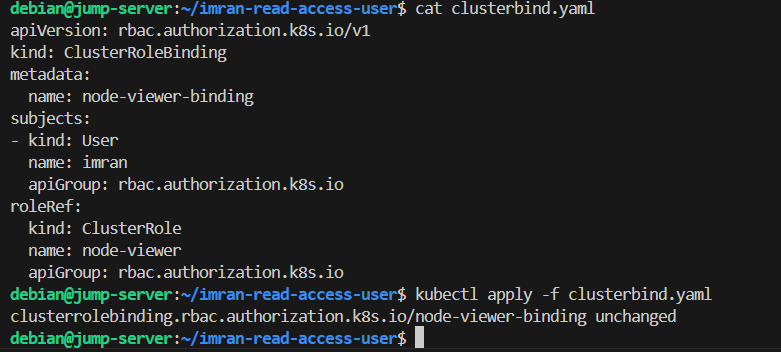
kind: ClusterRole

name: node-viewer

apiGroup: rbac.authorization.k8s.io

Apply the ClusterRoleBinding using:

**kubectl apply -f clusterbind.yaml**



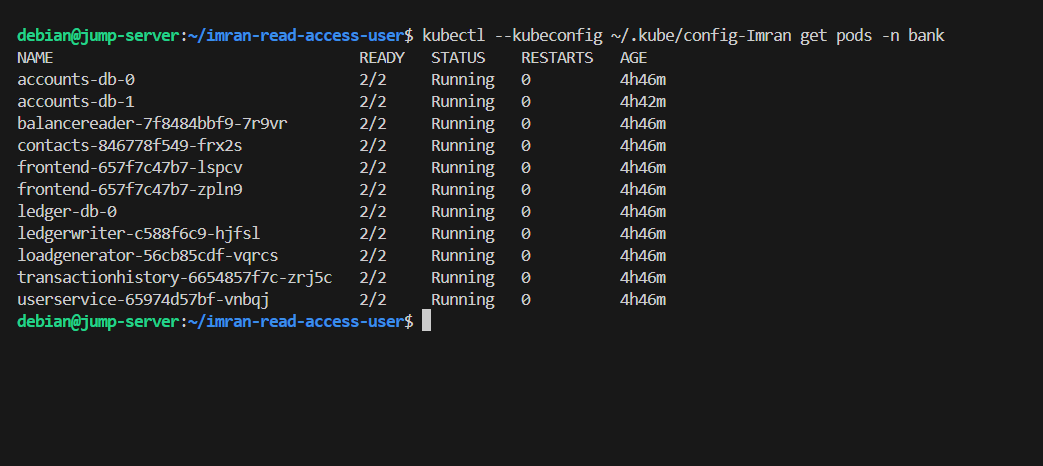
This ClusterRoleBinding binds the “I**mran**” user to the “**node-reader**” Role in the Cluster.

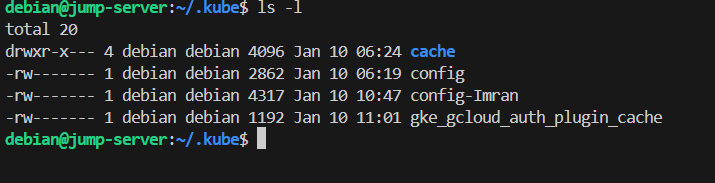
Now, the “I**mran**” user should be able to list and get pods in the Cluster using the provided kubeconfig i.e., **~/.kube/config-Imran.**

We’ve set up a new Cluster role and Cluster role binding for the user “**Imran**”. Now, let’s try accessing the pods in the Cluster with the following command.

Output:

**Kubectl –kubeconfig ~/.kube/config-Imran get pods –n bank**

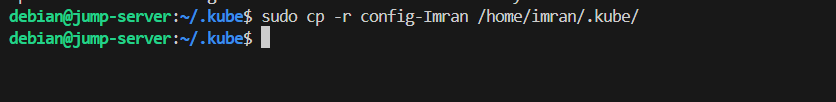
**Step 6: Verify if the .kube folder contains a file named config-Imran.**



The config-Imran configuration file was successfully created, as demonstrated above.

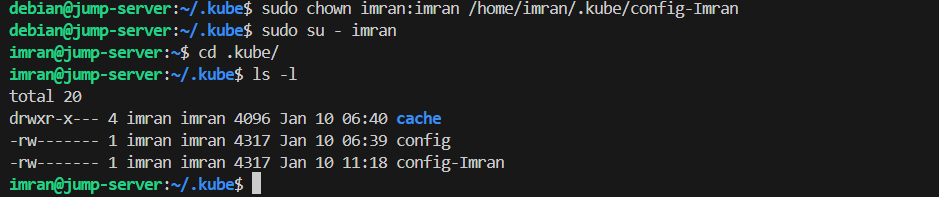
**Step 7: Copy the config-Imran file to the .kube folder of the Imran user on the Jump server using the following command:**

sudo cp -r config-Imran /home/imran/.kube/



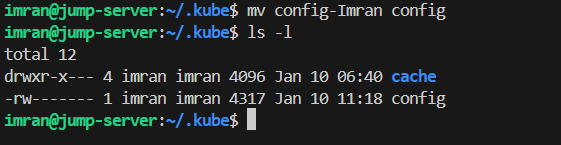
**Step 8: Change the ownership of the config-Imran file to the Imran user using the following command:**

sudo chown imran:imran /home/imran/.kube/config-Imran



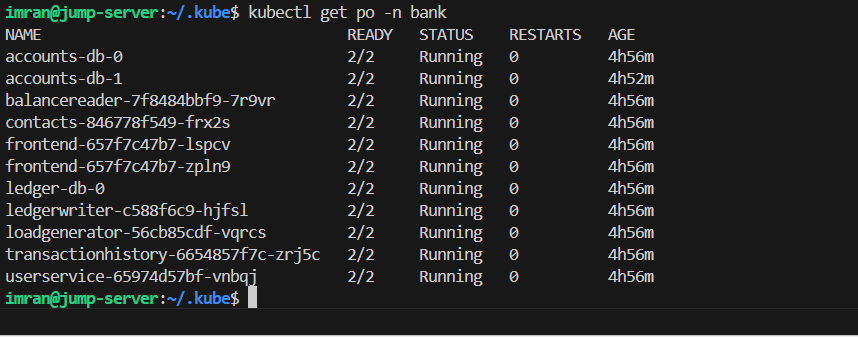
**Step 9: Rename the config-Imran file to config using the following command:**

mv config-Imran config



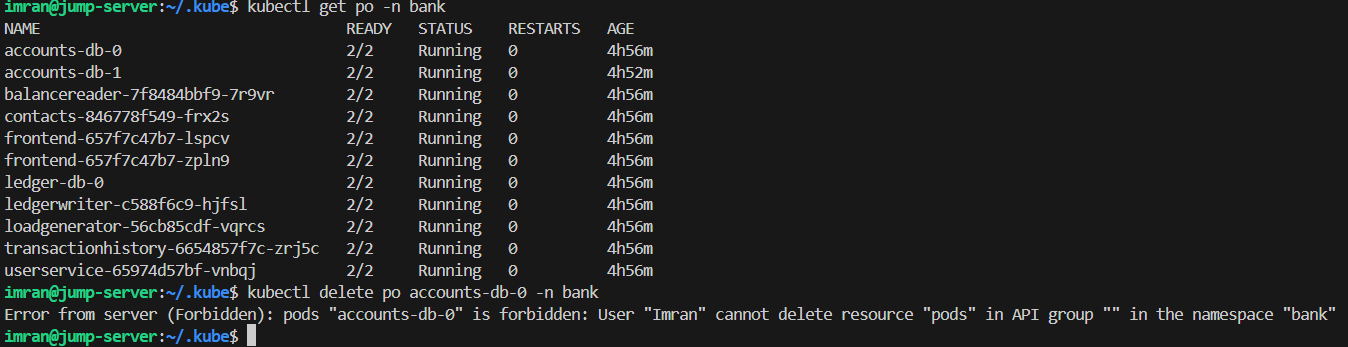
**Step 10: check the status of the pods in the bank namespace, you can use the following kubectl command:**

Kubectl get po –n bank



**Step 11: Imran user doesn't have the necessary permissions to delete pods in the bank namespace due to the read-only access. To delete a pod, you'll need to grant the Imran user the appropriate permissions.**

kubectl delete po accounts-db-0 -n bank



We have successfully created a user in the GKE Private cluster and granted read access at cluster levels using cluster roles, and cluster role bindings.