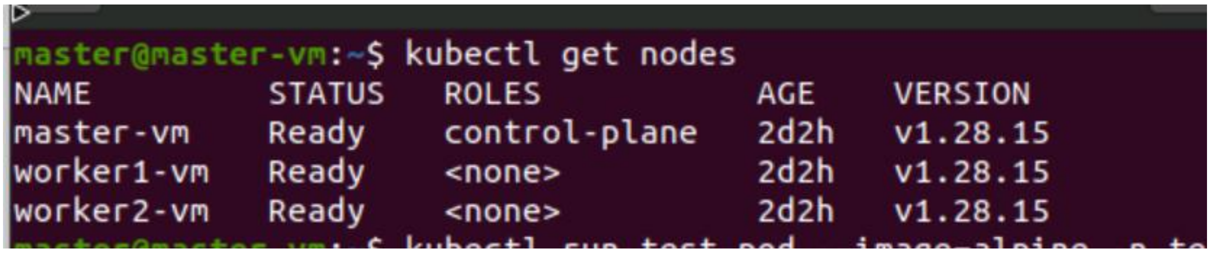
Kubernetes Multi-Tenant Project

Step 1: Check if Any Worker Node is Ready

Run the following command to check the status of worker nodes:

kubectl get nodes



```
master@master-vm:~$ kubectl get nodes
NAME         STATUS   ROLES           AGE    VERSION
master-vm    Ready    control-plane   2d2h   v1.28.15
worker1-vm   Ready    <none>          2d2h   v1.28.15
worker2-vm   Ready    <none>          2d2h   v1.28.15
```

Step 2: Install Calico for Networking

Apply the Calico manifest to enable networking:

kubectl apply -f https://docs.projectcalico.org/manifests/calico.yaml

Step 3: Create Namespaces for Tenants

To isolate tenants, create separate namespaces:

kubectl create namespace tenant-a

kubectl create namespace tenant-b

Step 4: Create Folder Structure for YAML Files

Create the folder structure to organize YAML files for each tenant:

mkdir -p ~/k8s-multi-tenant/tenant-a

mkdir -p ~/k8s-multi-tenant/tenant-b

cd ~/k8s-multi-tenant

Step 5: Create Deployment and Service for Tenant A

Create tenant-a-app.yaml in the tenant-a/ directory with the following contents:

apiVersion: apps/v1

```
kind: Deployment
metadata:
  name: tenant-a-app
  namespace: tenant-a
spec:
  replicas: 2
  selector:
    matchLabels:
      app: tenant-a-app
  template:
    metadata:
      labels:
        app: tenant-a-app
    spec:
      containers:
      - name: tenant-a-app
        image: nginx
---
apiVersion: v1
kind: Service
metadata:
  name: tenant-a-service
  namespace: tenant-a
spec:
  selector:
    app: tenant-a-app
  ports:
    - protocol: TCP
      port: 80
      targetPort: 80
```

Apply the configuration:

```
kubectl apply -f tenant-a/tenant-a-app.yaml
```

Step 6: Restrict Network Access for Tenant A

Create tenant-a-restrict.yaml in the tenant-a/ directory with the following contents:

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: tenant-a-restrict
  namespace: tenant-a
spec:
  podSelector:
    matchLabels:
      app: tenant-a-app
  policyTypes:
  - Ingress
  ingress:
  - from:
    - podSelector:
        matchLabels:
          app: tenant-a-app
```

Apply the network policy:

```
kubectl apply -f tenant-a/tenant-a-restrict.yaml
```

Step 7: Create Deployment and Service for Tenant B

Create tenant-b-app.yaml in the tenant-b/ directory with the following contents:

```
apiVersion: apps/v1
```

```
kind: Deployment
metadata:
 name: tenant-b-app
 namespace: tenant-b
spec:
 replicas: 2
 selector:
  matchLabels:
   app: tenant-b-app
 template:
  metadata:
   labels:
    app: tenant-b-app
  spec:
   containers:
   - name: tenant-b-app
    image: nginx
---
apiVersion: v1
kind: Service
metadata:
 name: tenant-b-service
 namespace: tenant-b
spec:
 selector:
  app: tenant-b-app
 ports:
  - protocol: TCP
   port: 80
```

```
    targetPort: 80
```

Apply the deployment:

```
kubectl apply -f tenant-b/tenant-b-app.yaml
```

Verify the deployment:

```
kubectl get pods -n tenant-b
kubectl get svc -n tenant-b
```

Step 8: Restrict Network Access for Tenant B

Create tenant-b-restrict.yaml in the tenant-b/ directory with the following contents:

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: tenant-b-restrict
  namespace: tenant-b
spec:
  podSelector:
    matchLabels:
      app: tenant-b-app
  policyTypes:
  - Ingress
  ingress:
  - from:
    - podSelector:
        matchLabels:
          app: tenant-b-app
```

Apply the network policy:

kubectl apply -f tenant-b/tenant-b-restrict.yaml

Step 9: Verify Network Policy

To verify the network policy for Tenant B, run the following commands:

kubectl get networkpolicy -n tenant-b

kubectl describe networkpolicy tenant-b-restrict -n tenant-b

Step 10: Final Folder Structure

The final folder structure should look like this:

```
k8s-multi-tenant/
|— tenant-a/
│   ├— tenant-a-app.yaml
│   ├— tenant-a-restrict.yaml
|— tenant-b/
│   ├— tenant-b-app.yaml
│   ├— tenant-b-restrict.yaml
```

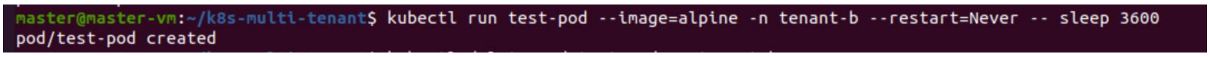
Step 11: Test Tenant Isolation

Create a test pod in tenant-b and check access to tenant-a:

In worker docker run

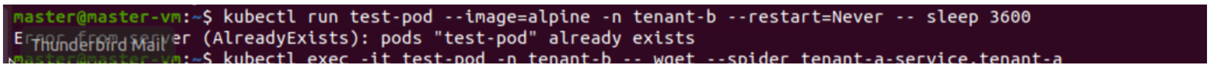
docker pull alpine

kubectl run test-pod --image=alpine -n tenant-b --restart=Never -- sleep 3600

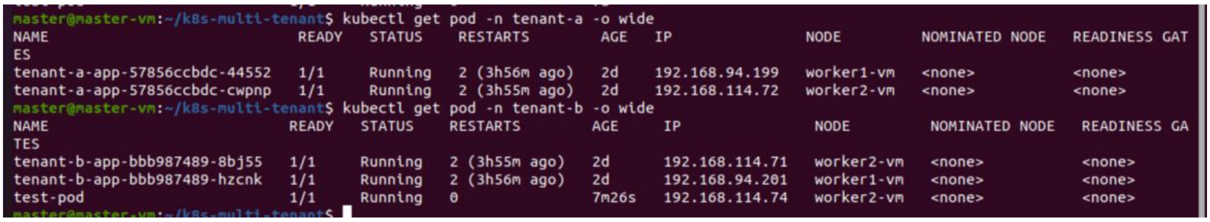


```
master@master-vm:~/k8s-multi-tenant$ kubectl run test-pod --image=alpine -n tenant-b --restart=Never -- sleep 3600
pod/test-pod created
```

kubectl exec -it test-pod -n tenant-b -- wget --spider tenant-a-service.tenant-a



```
master@master-vm:~$ kubectl run test-pod --image=alpine -n tenant-b --restart=Never -- sleep 3600
Error from server (AlreadyExists): pods "test-pod" already exists
master@master-vm:~$ kubectl exec -it test-pod -n tenant-b -- wget --spider tenant-a-service.tenant-a
```



```
master@master-vm:~/k8s-multi-tenant$ kubectl get pod -n tenant-a -o wide
NAME                            READY   STATUS    RESTARTS        AGE   IP               NODE         NOMINATED NODE   READINESS GAT
ES
tenant-a-app-57856ccbdc-44552   1/1     Running   2 (3h56m ago)   2d    192.168.94.199   worker1-vm   <none>           <none>
tenant-a-app-57856ccbdc-cwpnp   1/1     Running   2 (3h55m ago)   2d    192.168.114.72   worker2-vm   <none>           <none>
master@master-vm:~/k8s-multi-tenant$ kubectl get pod -n tenant-b -o wide
NAME                           READY   STATUS    RESTARTS        AGE     IP               NODE         NOMINATED NODE   READINESS GA
TES
tenant-b-app-bbb987489-8bj55   1/1     Running   2 (3h55m ago)   2d      192.168.114.71   worker2-vm   <none>           <none>
tenant-b-app-bbb987489-hzcnk   1/1     Running   2 (3h56m ago)   2d      192.168.94.201   worker1-vm   <none>           <none>
test-pod                       1/1     Running   0               7m26s   192.168.114.74   worker2-vm   <none>           <none>
master@master-vm:~/k8s-multi-tenant$
```