Lesson-End Project

Managing Multiple Namespaces with Respective Roles

Project agenda: To manage and set permissions for multiple namespaces and roles within a Kubernetes cluster, ensuring structured access control and operational security across different environments

Description: As Kubernetes is adopted within larger organizations, it is crucial to manage access for various teams effectively. In this project, you will simulate an environment where there are two namespaces: Simplilearn and CKA. These namespaces represent different teams or projects. You will set up service accounts, roles, and role bindings to regulate access within these namespaces.

Tools required: kubeadm, kubectl, kubelet, and containerd

Prerequisites: A Kubernetes cluster (refer to Demo 01 from Lesson 01 for setting up a cluster)

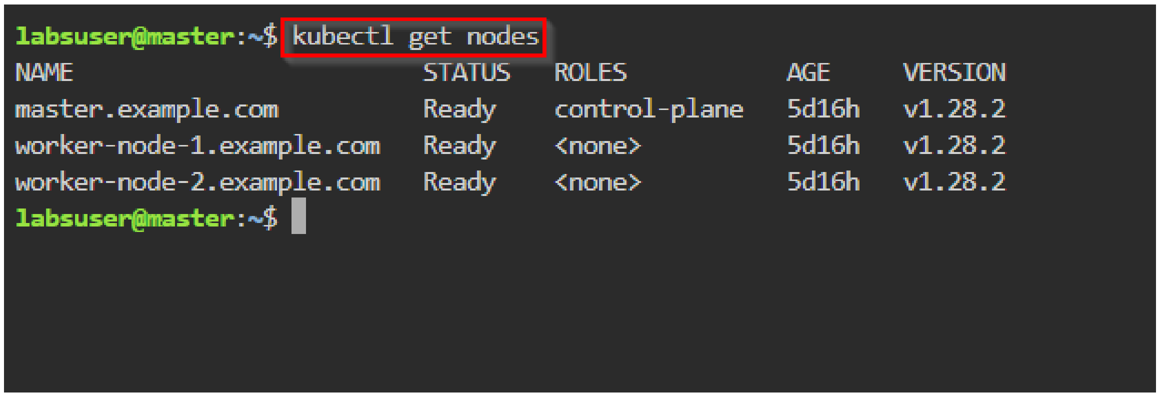
Expected deliverables: A Kubernetes cluster with the Simplilearn and CKA namespaces, along with the respective service accounts, roles, and role bindings

Steps to be followed:

1. Validate the Kubernetes cluster
2. Set up a namespace and its users
3. Check API access for the service account in the created namespace
4. Set up another namespace and its users
5. Check API access for the service account in created namespace

Step 1: Validate the Kubernetes cluster

1.1 To check the status of the nodes and ensure the cluster is running, run the following command:
kubectl get nodes

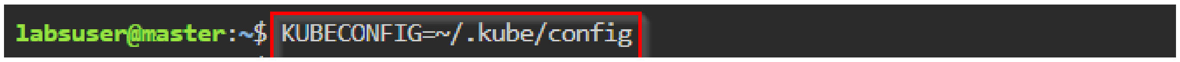


```
labsuser@master:~$ kubectl get nodes
NAME                         STATUS   ROLES           AGE     VERSION
master.example.com           Ready    control-plane   5d16h   v1.28.2
worker-node-1.example.com    Ready    <none>          5d16h   v1.28.2
worker-node-2.example.com    Ready    <none>          5d16h   v1.28.2
labsuser@master:~$
```

Note: Refer to Demo 01 of Lesson 01 for guidance on creating a Kubernetes cluster

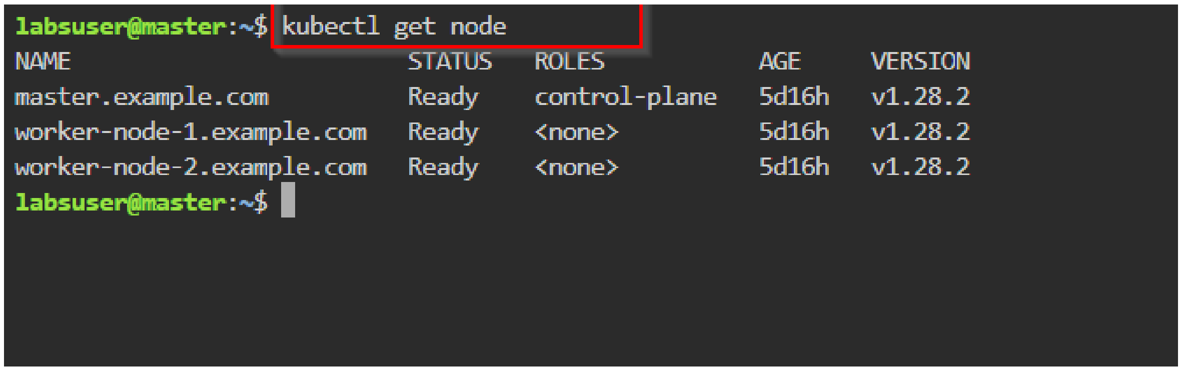
Step 2: Set up a namespace and its users

2.1 Use the following **KUBECONFIG** command for storage settings in the home directory:
KUBECONFIG=~/.kube/config



```
labsuser@master:~$ KUBECONFIG=~/.kube/config
```

2.2 Run the following command to display a list of all nodes in a Kubernetes cluster:
kubectl get nod

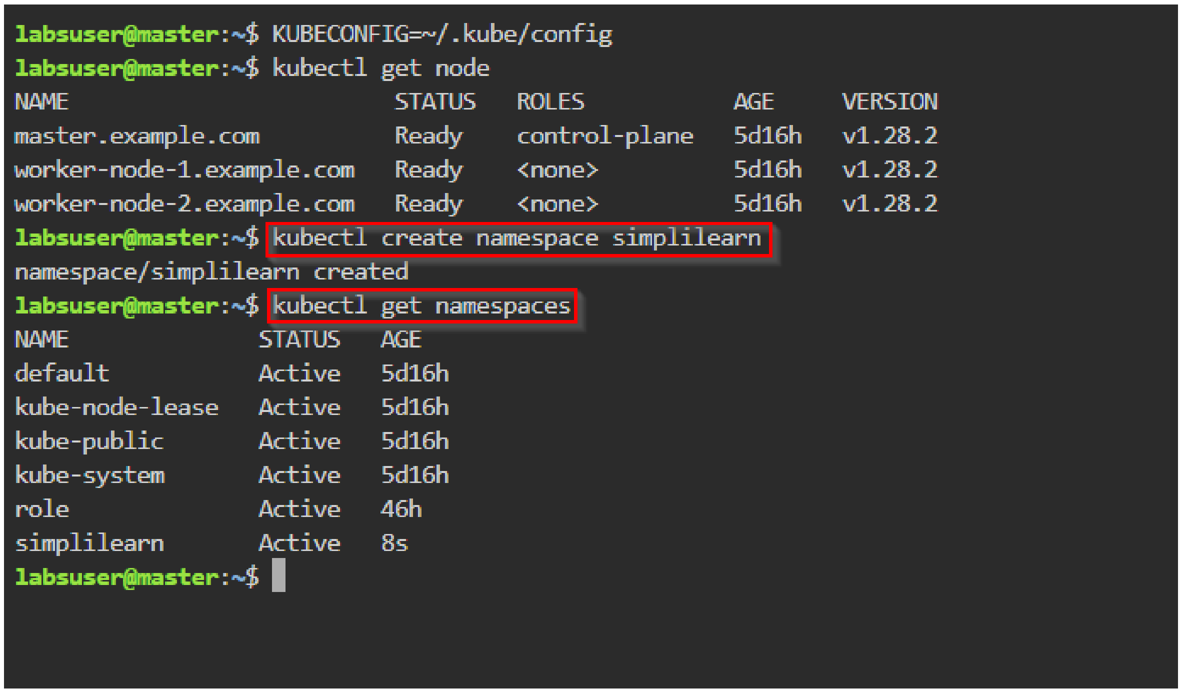


```
labsuser@master:~$ kubectl get node
NAME                         STATUS   ROLES           AGE     VERSION
master.example.com           Ready    control-plane   5d16h   v1.28.2
worker-node-1.example.com    Ready    <none>          5d16h   v1.28.2
worker-node-2.example.com    Ready    <none>          5d16h   v1.28.2
labsuser@master:~$
```

2.3 Create the **simplilearn** namespace and verify its creation using the following commands:

kubectl create namespace simplilearn

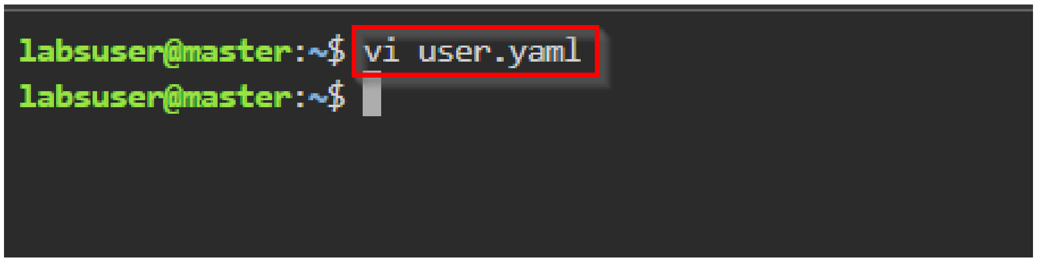
kubectl get namespaces



```
labsuser@master:~$ KUBECONFIG=~/.kube/config
labsuser@master:~$ kubectl get node
NAME                          STATUS   ROLES           AGE     VERSION
master.example.com            Ready    control-plane   5d16h   v1.28.2
worker-node-1.example.com     Ready    <none>          5d16h   v1.28.2
worker-node-2.example.com     Ready    <none>          5d16h   v1.28.2
labsuser@master:~$ kubectl create namespace simplilearn
namespace/simplilearn created
labsuser@master:~$ kubectl get namespaces
NAME              STATUS   AGE
default           Active   5d16h
kube-node-lease   Active   5d16h
kube-public       Active   5d16h
kube-system       Active   5d16h
role              Active   46h
simplilearn       Active   8s
labsuser@master:~$
```

2.4 Create a file called user.yaml using the command below:

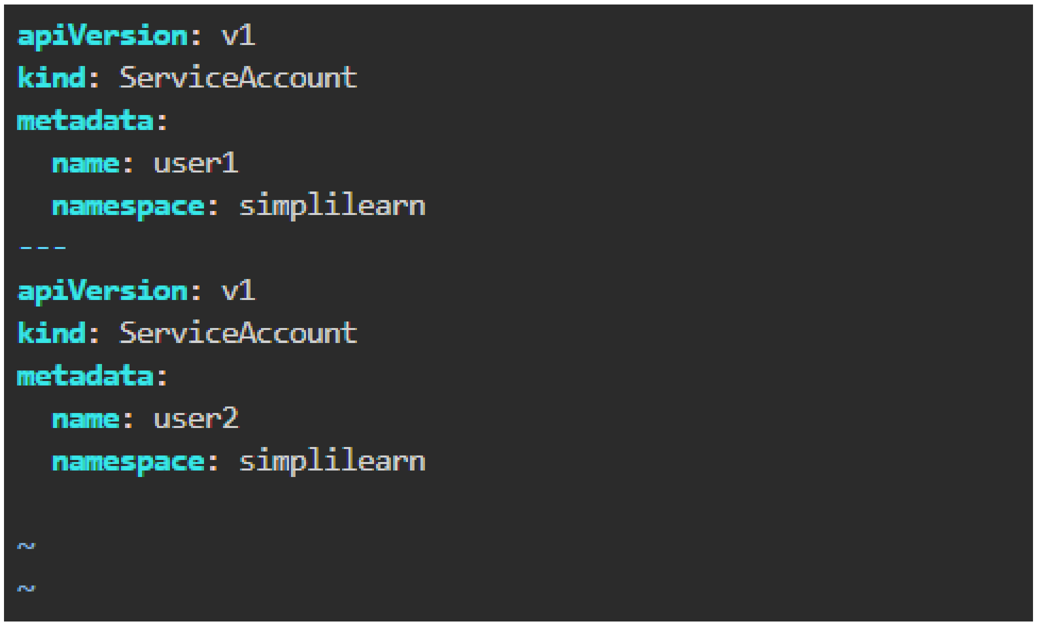
vi user.yaml



```
labsuser@master:~$ vi user.yaml
labsuser@master:~$
```

2.5 In the **user.yaml** file, define two service accounts (**user1** and **user2**) for the **simplilearn** namespace using the following configurations:

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: user1
  namespace: simplilearn
---
apiVersion: v1
kind: ServiceAccount
metadata:
  name: user2
  namespace: simplilearn
```



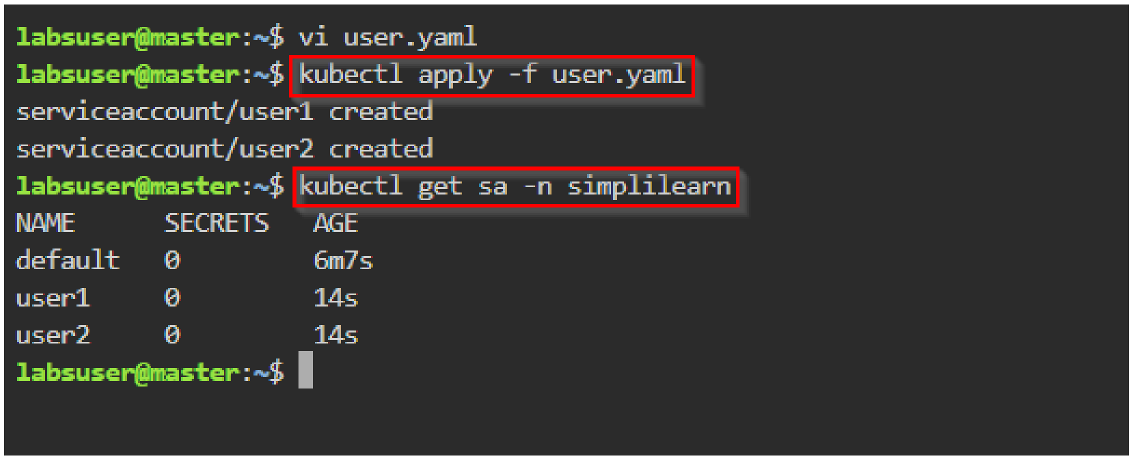
```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: user1
  namespace: simplilearn
---
apiVersion: v1
kind: ServiceAccount
metadata:
  name: user2
  namespace: simplilearn

~
~
```

2.6 Create the service accounts and validate their creation using the following commands:

kubectl apply -f user.yaml

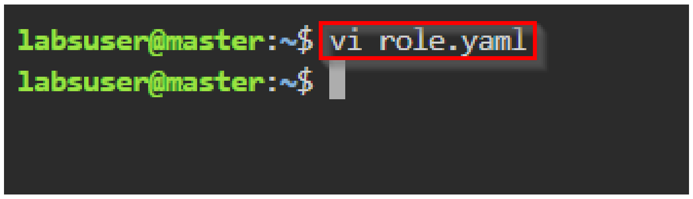
kubectl get sa -n simplilearn



```
labsuser@master:~$ vi user.yaml
labsuser@master:~$ kubectl apply -f user.yaml
serviceaccount/user1 created
serviceaccount/user2 created
labsuser@master:~$ kubectl get sa -n simplilearn
NAME      SECRETS   AGE
default   0         6m7s
user1     0         14s
user2     0         14s
labsuser@master:~$
```

2.7 Run the following command to create a file named **role.yaml**:

vi role.yaml

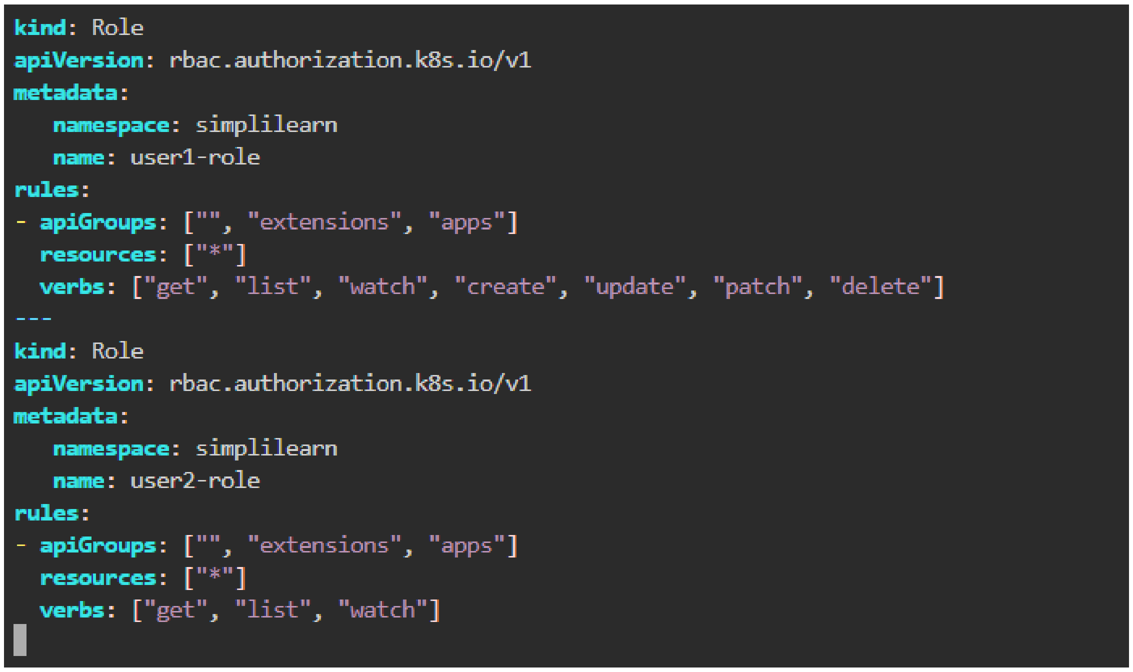


```
labsuser@master:~$ vi role.yaml
labsuser@master:~$
```

2.8 In the **user.yaml** file, define two service accounts (**user1** and **user2**) for the **simplilearn** namespace using the following configurations:

```
kind: Role
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  namespace: simplilearn
  name: user1-role
rules:
- apiGroups: ["", "extensions", "apps"]
  resources: ["*"]
  verbs: ["get", "list", "watch", "create", "update", "patch", "delete"]
---
kind: Role
apiVersion: rbac.authorization.k8s.io/v1
```

```
metadata:
   namespace: simplilearn
   name: user2-role
rules:
- apiGroups: ["", "extensions", "apps"]
  resources: ["*"]
  verbs: ["get", "list", "watch"]
```

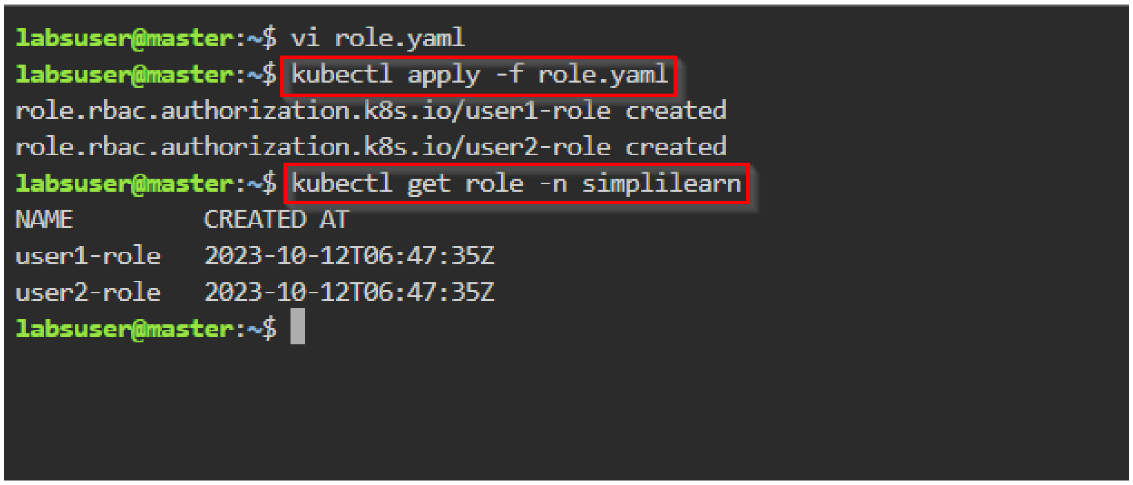


```
kind: Role
apiVersion: rbac.authorization.k8s.io/v1
metadata:
   namespace: simplilearn
   name: user1-role
rules:
- apiGroups: ["", "extensions", "apps"]
  resources: ["*"]
  verbs: ["get", "list", "watch", "create", "update", "patch", "delete"]
---
kind: Role
apiVersion: rbac.authorization.k8s.io/v1
metadata:
   namespace: simplilearn
   name: user2-role
rules:
- apiGroups: ["", "extensions", "apps"]
  resources: ["*"]
  verbs: ["get", "list", "watch"]
```

2.9 Apply the roles and verify their status using the following commands:

kubectl apply -f role.yaml

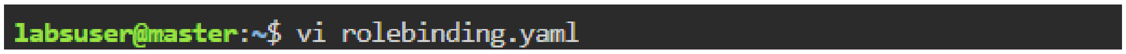
kubectl get role -n simplilearn



```
labsuser@master:~$ vi role.yaml
labsuser@master:~$ kubectl apply -f role.yaml
role.rbac.authorization.k8s.io/user1-role created
role.rbac.authorization.k8s.io/user2-role created
labsuser@master:~$ kubectl get role -n simplilearn
NAME         CREATED AT
user1-role   2023-10-12T06:47:35Z
user2-role   2023-10-12T06:47:35Z
labsuser@master:~$
```

2.10 Run the following command to create a file named **rolebinding.yaml**:

vi rolebinding.yaml

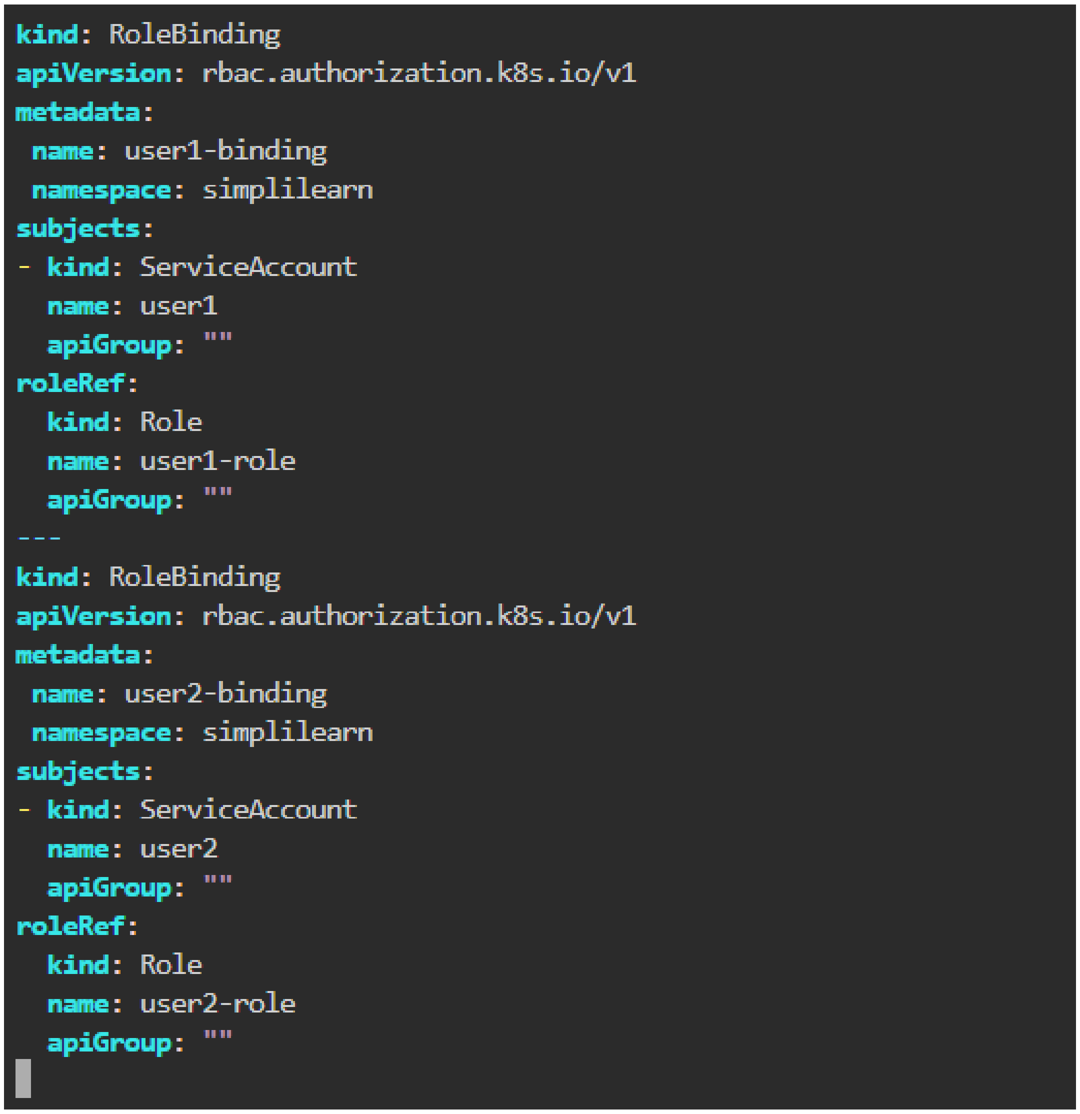


```
labsuser@master:~$ vi rolebinding.yaml
```

2.11 Bind the service account to the role using the following code:

```
kind: RoleBinding
apiVersion: rbac.authorization.k8s.io/v1
metadata:
 name: user1-binding
 namespace: simplilearn
subjects:
- kind: ServiceAccount
  name: user1
  apiGroup: ""
roleRef:
  kind: Role
  name: user1-role
  apiGroup: ""
---
kind: RoleBinding
```

```
apiVersion: rbac.authorization.k8s.io/v1
metadata:
 name: user2-binding
 namespace: simplilearn
subjects:
- kind: ServiceAccount
  name: user2
  apiGroup: ""
roleRef:
  kind: Role
  name: user2-role
  apiGroup: ""
```

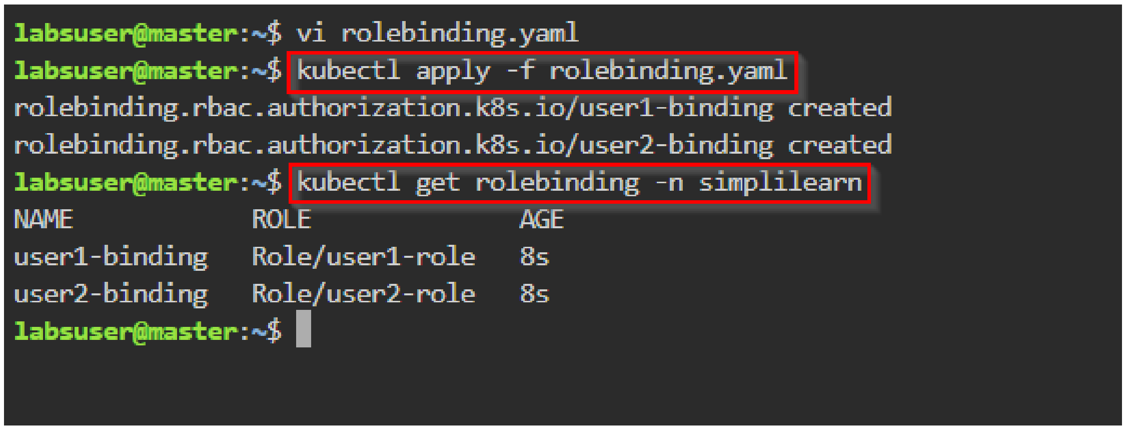


```yaml
kind: RoleBinding
apiVersion: rbac.authorization.k8s.io/v1
metadata:
 name: user1-binding
 namespace: simplilearn
subjects:
- kind: ServiceAccount
  name: user1
  apiGroup: ""
roleRef:
  kind: Role
  name: user1-role
  apiGroup: ""
---
kind: RoleBinding
apiVersion: rbac.authorization.k8s.io/v1
metadata:
 name: user2-binding
 namespace: simplilearn
subjects:
- kind: ServiceAccount
  name: user2
  apiGroup: ""
roleRef:
  kind: Role
  name: user2-role
  apiGroup: ""
```

2.12 Apply the role bindings and verify their status using the following commands:

kubectl apply -f rolebinding.yaml

kubectl get rolebinding -n simplilearn



```
labsuser@master:~$ vi rolebinding.yaml
labsuser@master:~$ kubectl apply -f rolebinding.yaml
rolebinding.rbac.authorization.k8s.io/user1-binding created
rolebinding.rbac.authorization.k8s.io/user2-binding created
labsuser@master:~$ kubectl get rolebinding -n simplilearn
NAME            ROLE              AGE
user1-binding   Role/user1-role   8s
user2-binding   Role/user2-role   8s
labsuser@master:~$
```

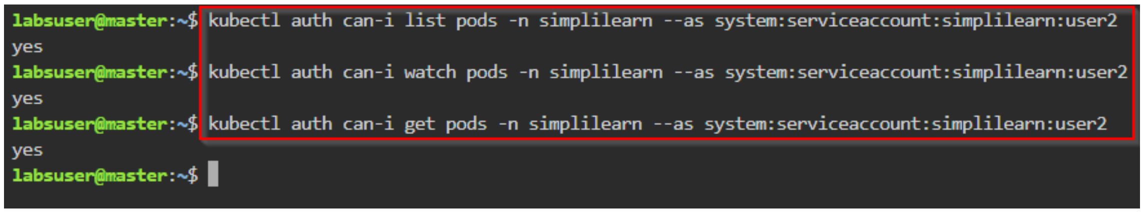
Step 3: Check API access for the service account in the created namespace

3.1 To check API access (list, get, watch) for the service account provided in the role binding, use the following commands:

kubectl auth can-i list pods -n simplilearn --as system:serviceaccount:simplilearn:user2

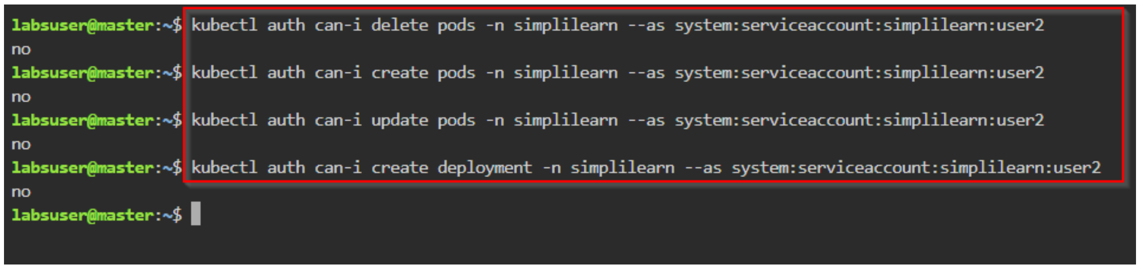
kubectl auth can-i watch pods -n simplilearn --as system:serviceaccount:simplilearn:user2

kubectl auth can-i get pods -n simplilearn --as system:serviceaccount:simplilearn:user2



```
labsuser@master:~$ kubectl auth can-i list pods -n simplilearn --as system:serviceaccount:simplilearn:user2
yes
labsuser@master:~$ kubectl auth can-i watch pods -n simplilearn --as system:serviceaccount:simplilearn:user2
yes
labsuser@master:~$ kubectl auth can-i get pods -n simplilearn --as system:serviceaccount:simplilearn:user2
yes
labsuser@master:~$
```

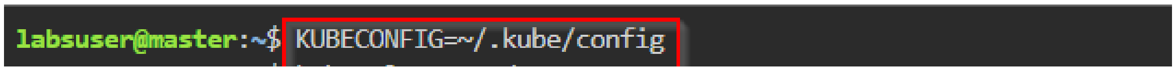
3.2 To check API access (other than list, get, watch) for the service account which is not provided in the role binding, use the following commands:
kubectl auth can-i delete pods -n simplilearn --as system:serviceaccount:simplilearn:user2
kubectl auth can-i create pods -n simplilearn --as system:serviceaccount:simplilearn:user2
kubectl auth can-i update pods -n simplilearn --as system:serviceaccount:simplilearn:user2
kubectl auth can-i create deployment -n simplilearn --as system:serviceaccount:simplilearn:user2



```
labsuser@master:~$ kubectl auth can-i delete pods -n simplilearn --as system:serviceaccount:simplilearn:user2
no
labsuser@master:~$ kubectl auth can-i create pods -n simplilearn --as system:serviceaccount:simplilearn:user2
no
labsuser@master:~$ kubectl auth can-i update pods -n simplilearn --as system:serviceaccount:simplilearn:user2
no
labsuser@master:~$ kubectl auth can-i create deployment -n simplilearn --as system:serviceaccount:simplilearn:user2
no
labsuser@master:~$
```

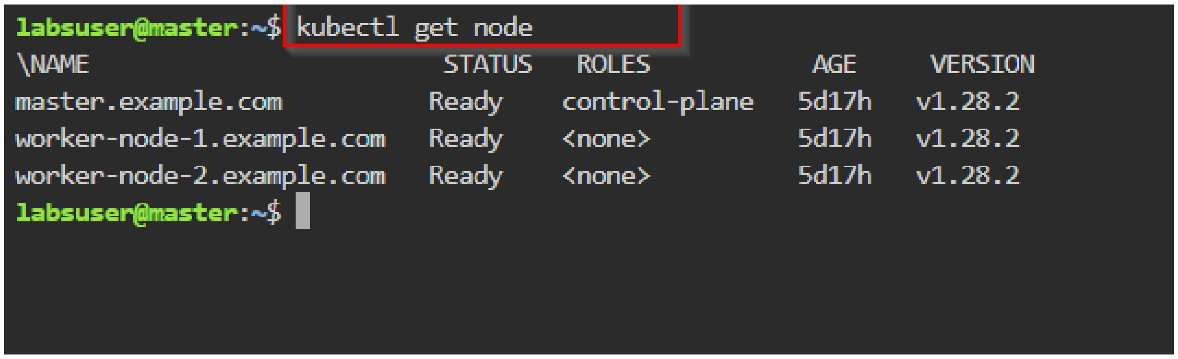
Step 4: Set up another namespace and its users

4.1 Use the following **KUBECONFIG** command for storage settings in the home directory:
KUBECONFIG=~/.kube/config



```
labsuser@master:~$ KUBECONFIG=~/.kube/config
```

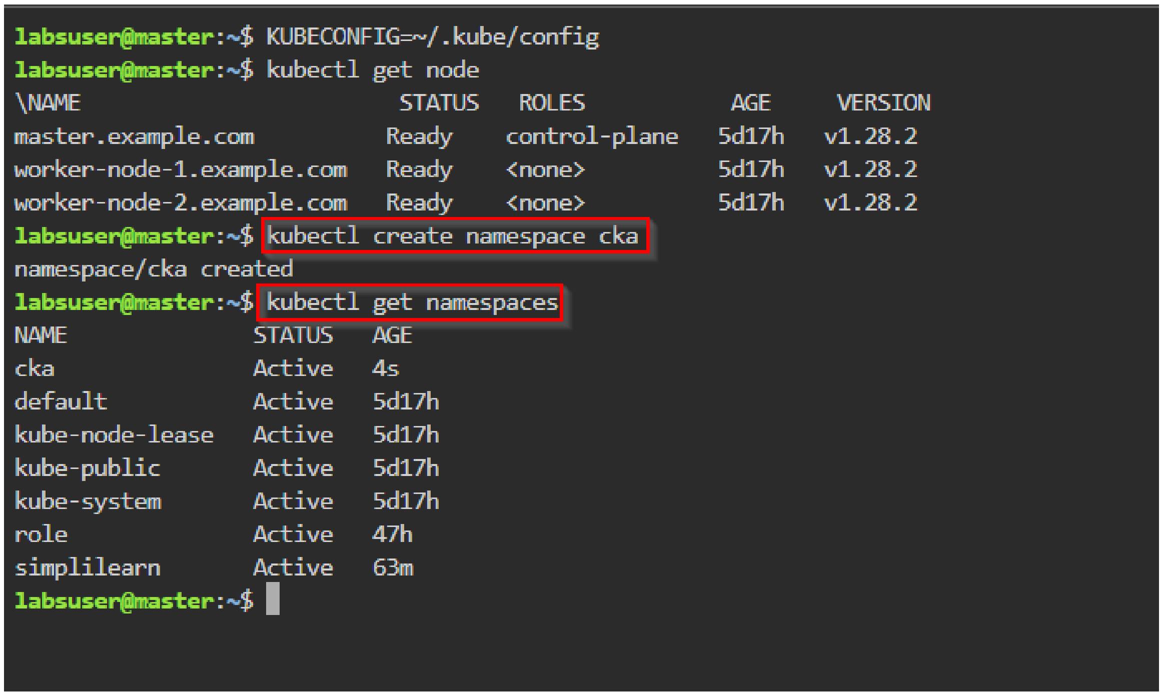
4.2 Run the following command to display a list of all nodes in a Kubernetes cluster:
kubectl get nod



```
labsuser@master:~$ kubectl get node
\NAME                          STATUS   ROLES           AGE     VERSION
master.example.com            Ready    control-plane   5d17h   v1.28.2
worker-node-1.example.com     Ready    <none>          5d17h   v1.28.2
worker-node-2.example.com     Ready    <none>          5d17h   v1.28.2
labsuser@master:~$
```

4.3 Create the **cka** namespace and verify its creation using the following commands:

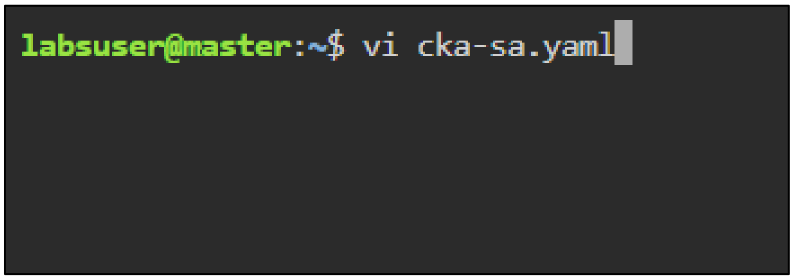
kubectl create namespace cka
kubectl get namespaces



```
labsuser@master:~$ KUBECONFIG=~/.kube/config
labsuser@master:~$ kubectl get node
\NAME                        STATUS   ROLES           AGE     VERSION
master.example.com          Ready    control-plane   5d17h   v1.28.2
worker-node-1.example.com   Ready    <none>          5d17h   v1.28.2
worker-node-2.example.com   Ready    <none>          5d17h   v1.28.2
labsuser@master:~$ kubectl create namespace cka
namespace/cka created
labsuser@master:~$ kubectl get namespaces
NAME              STATUS   AGE
cka               Active   4s
default           Active   5d17h
kube-node-lease   Active   5d17h
kube-public       Active   5d17h
kube-system       Active   5d17h
role              Active   47h
simplilearn       Active   63m
labsuser@master:~$
```

4.4 Run the following command to create a **cka-sa.yaml** file:

vi cka-sa.yaml

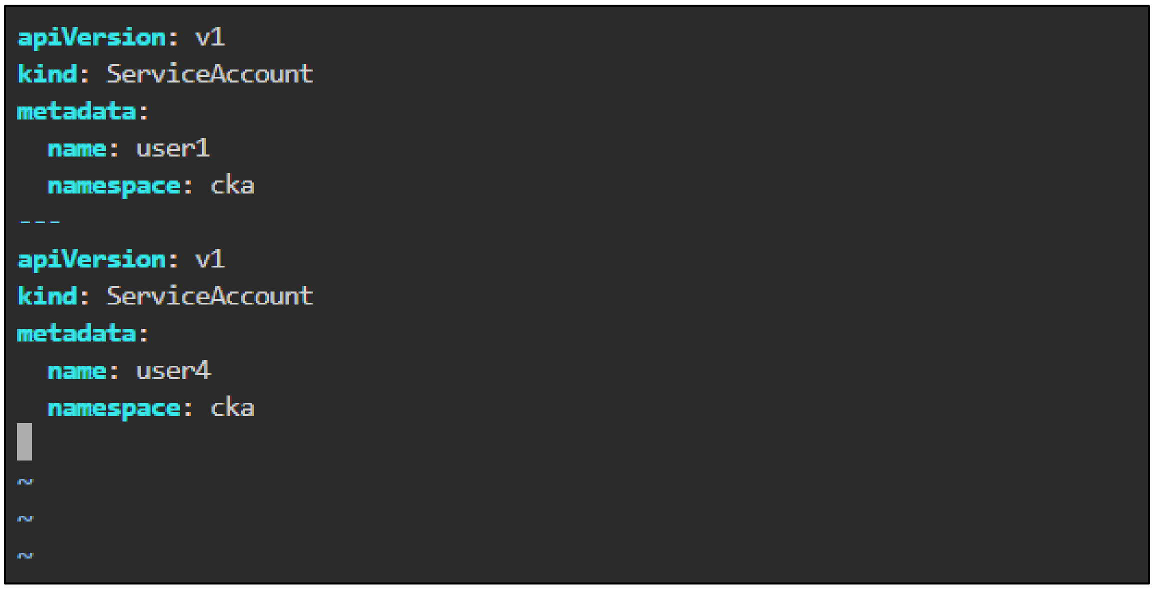


```
labsuser@master:~$ vi cka-sa.yaml
```

4.5 Add the following code to the **cka-sa.yaml** file to define service accounts **user1** and **user4** for the **cka** namespace:

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: user3
  namespace: cka
---
```

apiVersion: v1
kind: ServiceAccount
metadata:
  name: user4
  namespace: cka

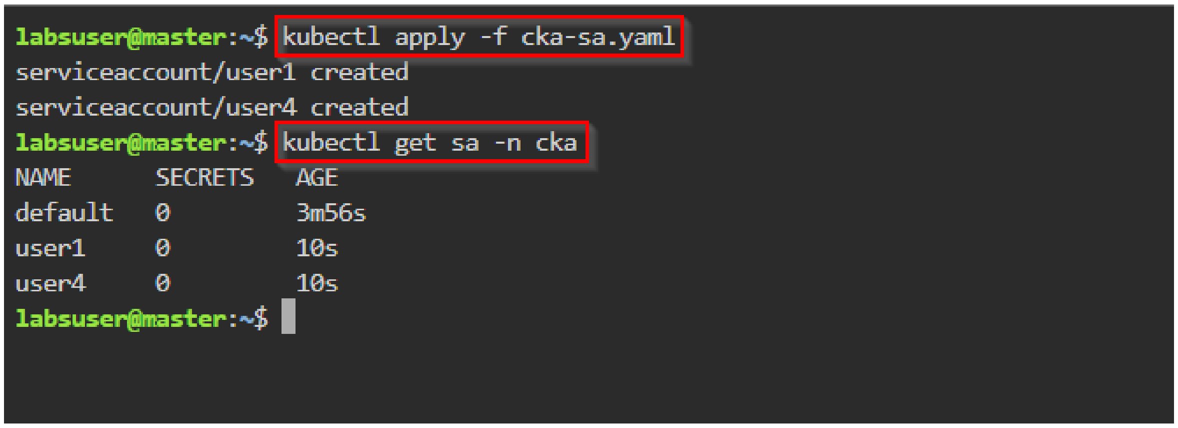


```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: user1
  namespace: cka
---
apiVersion: v1
kind: ServiceAccount
metadata:
  name: user4
  namespace: cka

~
~
~
```

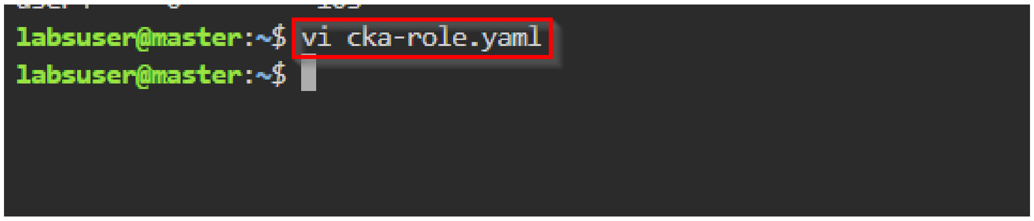
Note: You can choose any name for the users based on your preference.

4.6 Apply the service accounts and verify their status using the following commands:
kubectl apply -f cka-sa.yaml
kubectl get sa -n cka



```
labsuser@master:~$ kubectl apply -f cka-sa.yaml
serviceaccount/user1 created
serviceaccount/user4 created
labsuser@master:~$ kubectl get sa -n cka
NAME      SECRETS   AGE
default   0         3m56s
user1     0         10s
user4     0         10s
labsuser@master:~$
```

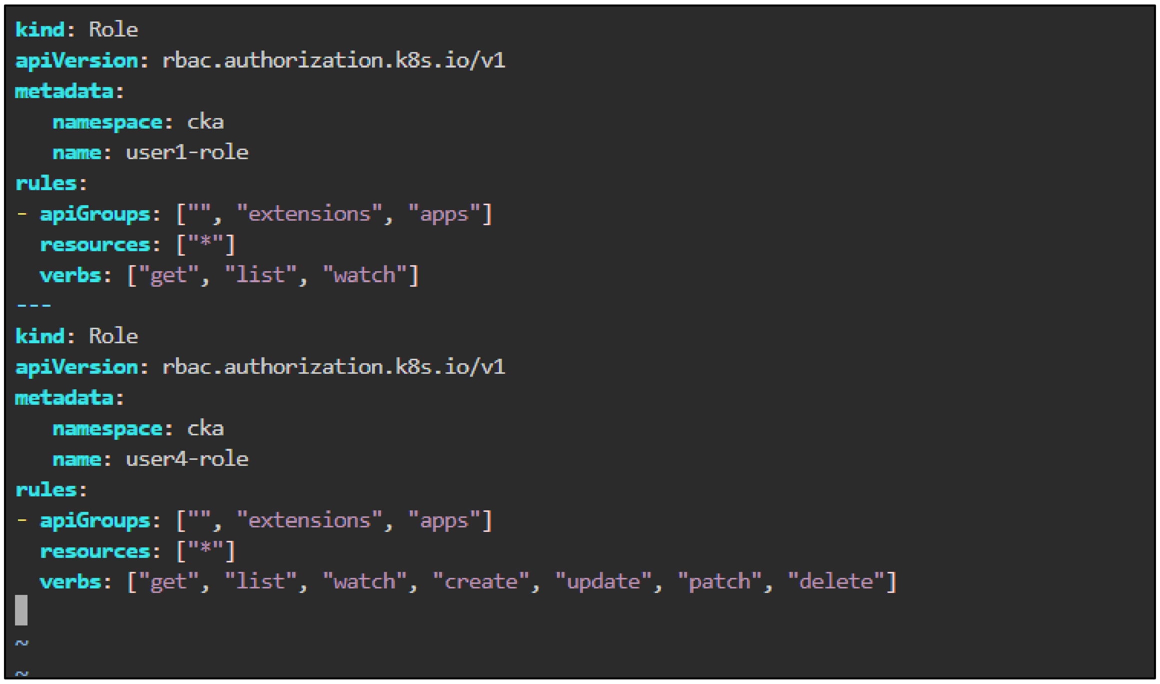
4.7 Run the following command to create a file named **cka-role.yaml**:



```
labsuser@master:~$ vi cka-role.yaml
labsuser@master:~$
```

4.8 Add the following code to the file **cka-role.yaml** to create roles for **user1** and **user4**:

```
kind: Role
apiVersion: rbac.authorization.k8s.io/v1
metadata:
   namespace: cka
   name: user1-role
rules:
- apiGroups: ["", "extensions", "apps"]
  resources: ["*"]
  verbs: ["get", "list", "watch"]
---
kind: Role
apiVersion: rbac.authorization.k8s.io/v1
metadata:
   namespace: cka
   name: user4-role
rules:
- apiGroups: ["", "extensions", "apps"]
  resources: ["*"]
  verbs: ["get", "list", "watch", "create", "update", "patch", "delete"]
```

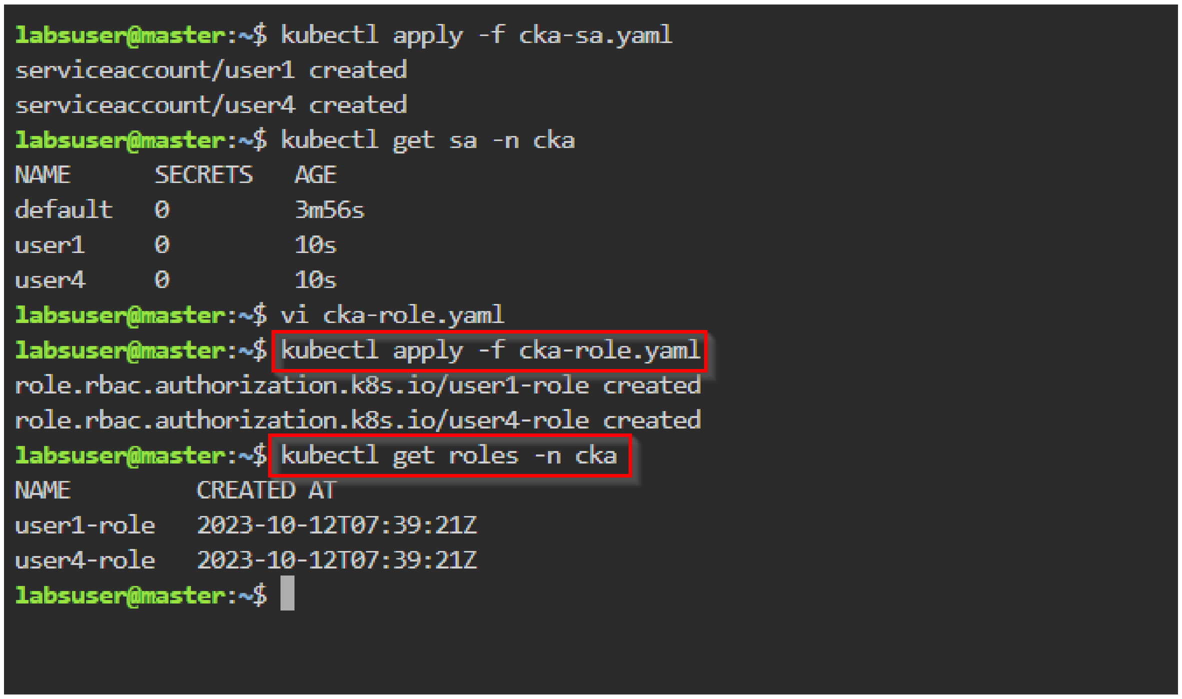

```
kind: Role
apiVersion: rbac.authorization.k8s.io/v1
metadata:
   namespace: cka
   name: user1-role
rules:
- apiGroups: ["", "extensions", "apps"]
  resources: ["*"]
  verbs: ["get", "list", "watch"]
---
kind: Role
apiVersion: rbac.authorization.k8s.io/v1
metadata:
   namespace: cka
   name: user4-role
rules:
- apiGroups: ["", "extensions", "apps"]
  resources: ["*"]
  verbs: ["get", "list", "watch", "create", "update", "patch", "delete"]

~
~
```

4.9 Apply the roles and verify their status using the following commands:

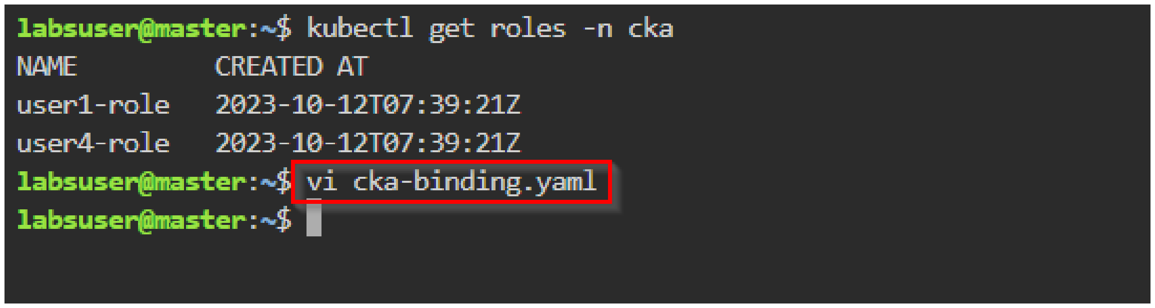
kubectl apply -f cka-role.yaml

kubectl get roles -n cka


```
labsuser@master:~$ kubectl apply -f cka-sa.yaml
serviceaccount/user1 created
serviceaccount/user4 created
labsuser@master:~$ kubectl get sa -n cka
NAME      SECRETS   AGE
default   0         3m56s
user1     0         10s
user4     0         10s
labsuser@master:~$ vi cka-role.yaml
labsuser@master:~$ kubectl apply -f cka-role.yaml
role.rbac.authorization.k8s.io/user1-role created
role.rbac.authorization.k8s.io/user4-role created
labsuser@master:~$ kubectl get roles -n cka
NAME         CREATED AT
user1-role   2023-10-12T07:39:21Z
user4-role   2023-10-12T07:39:21Z
labsuser@master:~$
```

4.10 Run the following command to create a file named **cka-binding.yaml**:

vi cka-binding.yaml

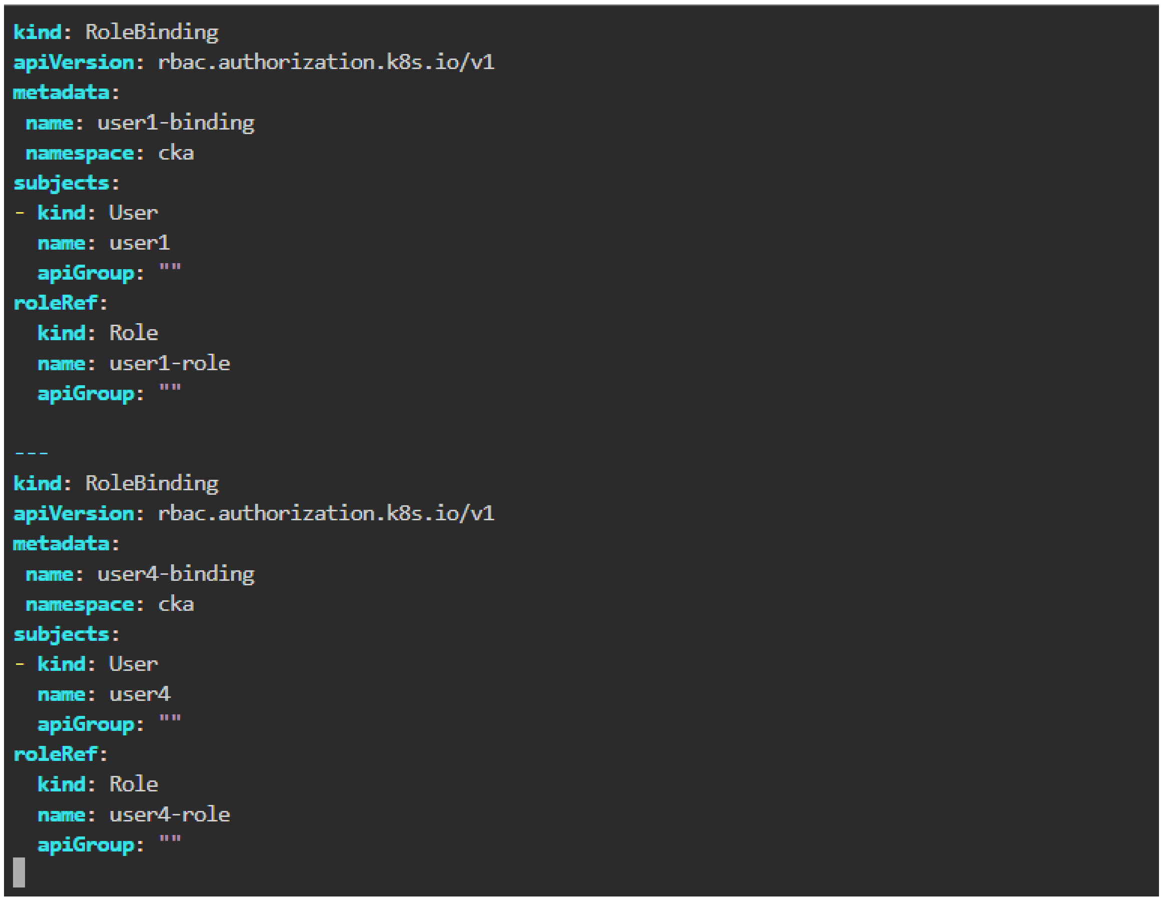


```
labsuser@master:~$ kubectl get roles -n cka
NAME         CREATED AT
user1-role   2023-10-12T07:39:21Z
user4-role   2023-10-12T07:39:21Z
labsuser@master:~$ vi cka-binding.yaml
labsuser@master:~$
```

4.11 Add the following code to the file **cka-binding.yaml** to bind the service account and role:

```
kind: RoleBinding
apiVersion: rbac.authorization.k8s.io/v1
metadata:
 name: user1-binding
 namespace: cka
subjects:
- kind: ServiceAccount
  name: user3
  apiGroup: ""
roleRef:
  kind: Role
  name: user1-role
  apiGroup: ""
---
kind: RoleBinding
apiVersion: rbac.authorization.k8s.io/v1
metadata:
 name: user4-binding
 namespace: cka
subjects:
- kind: ServiceAccount
  name: user4
  apiGroup: ""
roleRef:
  kind: Role
```

name: user4-role
apiGroup: ""

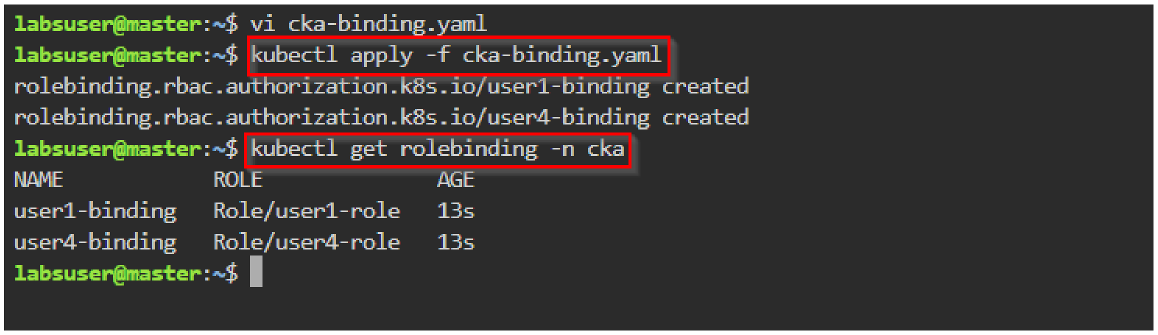


```
kind: RoleBinding
apiVersion: rbac.authorization.k8s.io/v1
metadata:
 name: user1-binding
 namespace: cka
subjects:
- kind: User
  name: user1
  apiGroup: ""
roleRef:
  kind: Role
  name: user1-role
  apiGroup: ""

---
kind: RoleBinding
apiVersion: rbac.authorization.k8s.io/v1
metadata:
 name: user4-binding
 namespace: cka
subjects:
- kind: User
  name: user4
  apiGroup: ""
roleRef:
  kind: Role
  name: user4-role
  apiGroup: ""
```

4.12 Apply the role bindings and verify their status using the following commands:

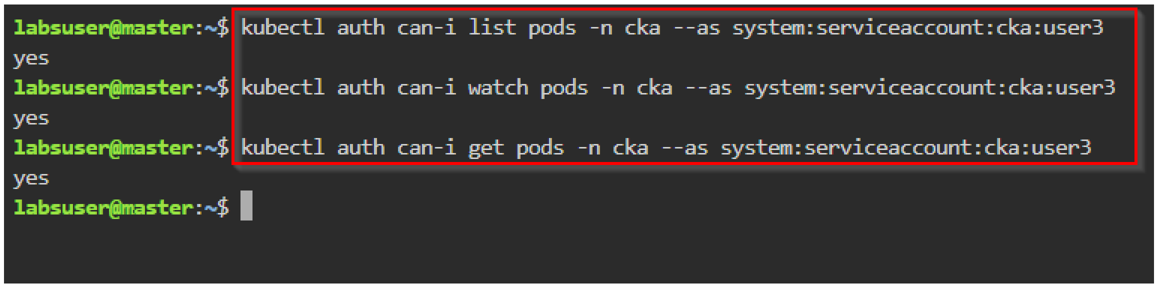
kubectl apply -f cka-binding.yaml
kubectl get rolebinding -n cka



```
labsuser@master:~$ vi cka-binding.yaml
labsuser@master:~$ kubectl apply -f cka-binding.yaml
rolebinding.rbac.authorization.k8s.io/user1-binding created
rolebinding.rbac.authorization.k8s.io/user4-binding created
labsuser@master:~$ kubectl get rolebinding -n cka
NAME            ROLE              AGE
user1-binding   Role/user1-role   13s
user4-binding   Role/user4-role   13s
labsuser@master:~$
```

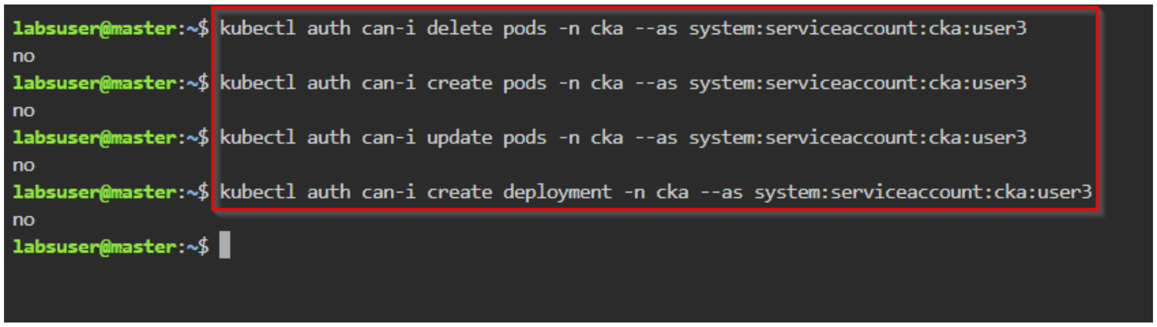
Step 5: Check API access for the service account in the created namespace

5.1 Check API access (list, get, watch) for the service account provided in the role binding using the following commands:
kubectl auth can-i list pods -n cka --as system:serviceaccount:cka:user3
kubectl auth can-i watch pods -n cka --as system:serviceaccount:cka:user3
kubectl auth can-i get pods -n cka --as system:serviceaccount:cka:user3



```
labsuser@master:~$ kubectl auth can-i list pods -n cka --as system:serviceaccount:cka:user3
yes
labsuser@master:~$ kubectl auth can-i watch pods -n cka --as system:serviceaccount:cka:user3
yes
labsuser@master:~$ kubectl auth can-i get pods -n cka --as system:serviceaccount:cka:user3
yes
labsuser@master:~$
```

5.2 Check API access (other than list, get, watch) for the service account which is not provided in the role binding using the following commands:
kubectl auth can-i delete pods -n cka --as system:serviceaccount:cka:user3
kubectl auth can-i create pods -n cka --as system:serviceaccount:cka:user3
kubectl auth can-i update pods -n cka --as system:serviceaccount:cka:user3
kubectl auth can-i create deployment -n cka --as system:serviceaccount:cka:user3



```
labsuser@master:~$ kubectl auth can-i delete pods -n cka --as system:serviceaccount:cka:user3
no
labsuser@master:~$ kubectl auth can-i create pods -n cka --as system:serviceaccount:cka:user3
no
labsuser@master:~$ kubectl auth can-i update pods -n cka --as system:serviceaccount:cka:user3
no
labsuser@master:~$ kubectl auth can-i create deployment -n cka --as system:serviceaccount:cka:user3
no
labsuser@master:~$
```

By following the above steps, you have successfully managed and set permissions for multiple namespaces and roles within your Kubernetes cluster, ensuring structured access control and operational security across different environments.