Kubernetes Security Hands-On Labs — User Access and Cluster Permissions

Author : Sandeep Kumar Sharma

Introduction:

Security is one of the most critical aspects of Kubernetes. Controlling who can access what in a Kubernetes cluster is handled through **RBAC (Role-Based Access Control)**. This lab series will help you understand how to create users, assign roles, and apply permissions in a simple, beginner-friendly way.

Lab 1 — Create a New User and Provide Cluster Access

Objective:

Create a new user named **Sandeep** and grant limited access to the Kubernetes cluster.

Step 1 — Create a Key and Certificate for the New User

We'll use OpenSSL to generate a private key and certificate for user authentication.

```
# Generate private key for user
openssl genrsa -out sandeep.key 2048

# Create a certificate signing request (CSR)
openssl req -new -key sandeep.key -out sandeep.csr -subj "/CN=sandeep/O=dev-
team"

# Sign the CSR using the Kubernetes CA (certificate authority)
sudo openssl x509 -req -in sandeep.csr
  -CA /etc/kubernetes/pki/ca.crt
  -CAkey /etc/kubernetes/pki/ca.key
  -CAcreateserial -out sandeep.crt -days 365
```

Explanation: - `/CN=sandeep` — Common Name is the username. - `/O=dev-team` — The group the user belongs to.

Step 2 — Configure kubeconfig for the User

Create a new kubeconfig file for the user.

```
kubectl config set-cluster kubernetes-cluster
  --server=https://<MASTER_NODE_IP>:6443
  --certificate-authority=/etc/kubernetes/pki/ca.crt
  --embed-certs=true
  --kubeconfig=sandeep-config

kubectl config set-credentials sandeep
  --client-certificate=sandeep.crt
  --client-key=sandeep.key
  --embed-certs=true
  --kubeconfig=sandeep-config

kubectl config set-context sandeep-context
  --cluster=kubernetes-cluster
  --user=sandeep
  --kubeconfig=sandeep-config

kubectl config use-context sandeep-context --kubeconfig=sandeep-config
```

Now the user **Sandeep** has a separate kubeconfig file named `sandeep-config`.

Step 3 — Test Access (Without Permission)

Try to list pods:

```
kubectl --kubeconfig=sandeep-config get pods
```

You'll see a **"forbidden"** message because the user has no permissions yet.

Step 4 — Create a Role and RoleBinding

We'll now give **Sandeep** permission to view resources in the `dev` namespace.

Step 4.1 — Create Namespace

```
kubectl create namespace dev
```

Step 4.2 — Create Role

Create a file `role-view-pods.yaml`:

```yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  namespace: dev
  name: pod-reader
rules:
- apiGroups: [""]
  resources: ["pods"]
  verbs: ["get", "list", "watch"]
```

Apply it:

```
kubectl apply -f role-view-pods.yaml
```

Step 4.3 — Create RoleBinding

Create a file `rolebinding-sandeep.yaml`:

```yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: sandeep-pod-reader-binding
  namespace: dev
subjects:
- kind: User
  name: sandeep
  apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: Role
  name: pod-reader
  apiGroup: rbac.authorization.k8s.io
```

Apply it:

```
kubectl apply -f rolebinding-sandeep.yaml
```

Step 5 — Verify Access as Sandeep

Try again using the Sandeep config:

```
kubectl --kubeconfig=sandeep-config get pods -n dev
```

You should now be able to **view** pods in the `dev` namespace.

If you try to **create** or **delete** pods:

```
kubectl --kubeconfig=sandeep-config delete pod <pod-name> -n dev
```

It should be **forbidden**, as Sandeep only has read access.

✅**Outcome:** You successfully created a new Kubernetes user, assigned a Role, and verified access control using RBAC.

Lab 2 — ClusterRole and ClusterRoleBinding

Objective:

Learn how to give cluster-wide permissions to a user.

Step 1 — Create ClusterRole

Create a file `clusterrole-list.yaml`:

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: cluster-list-role
rules:
- apiGroups: [""]
  resources: ["pods", "services", "namespaces"]
  verbs: ["get", "list", "watch"]
```

Apply it:

```
kubectl apply -f clusterrole-list.yaml
```

Step 2 — Create ClusterRoleBinding

Create a file `clusterrolebinding-sandeep.yaml`:

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: sandeep-cluster-binding
subjects:
- kind: User
  name: sandeep
  apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: ClusterRole
  name: cluster-list-role
  apiGroup: rbac.authorization.k8s.io
```

Apply it:

```
kubectl apply -f clusterrolebinding-sandeep.yaml
```

Step 3 — Verify Access

Now try:

```
kubectl --kubeconfig=sandeep-config get pods --all-namespaces
kubectl --kubeconfig=sandeep-config get namespaces
```

You should be able to list pods and namespaces across the cluster.

✅**Outcome:** You successfully provided cluster-wide read-only access to the Sandeep user.

Lab 3 — Service Account and Pod Security

Objective:

Understand how to use **ServiceAccounts** to control permissions for Pods.

Step 1 — Create a Service Account

```
kubectl create serviceaccount pod-reader-sa -n dev
```

Step 2 — Bind Service Account with Role

Create a file `rolebinding-serviceaccount.yaml`:

```
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: sa-pod-reader-binding
  namespace: dev
subjects:
- kind: ServiceAccount
  name: pod-reader-sa
  namespace: dev
roleRef:
  kind: Role
  name: pod-reader
  apiGroup: rbac.authorization.k8s.io
```

Apply it:

```
kubectl apply -f rolebinding-serviceaccount.yaml
```

Step 3 — Create a Pod Using This ServiceAccount

Create a file `pod-with-sa.yaml`:

```
apiVersion: v1
kind: Pod
metadata:
  name: test-pod-sa
  namespace: dev
spec:
  serviceAccountName: pod-reader-sa
  containers:
  - name: busybox
    image: busybox
    command: ["sh", "-c", "sleep 3600"]
```

Apply it:

```
kubectl apply -f pod-with-sa.yaml
```

Step 4 — Verify Permissions

Exec into the Pod and try to list pods:

```
kubectl exec -it test-pod-sa -n dev -- /bin/sh
kubectl get pods
```

You'll see that the Pod can only view pods within its namespace.

✅**Outcome:** You've successfully learned how to secure Pods using ServiceAccounts.

Verification Summary

Lab	Concept	Description
1	User Creation + RoleBinding	Create user and grant limited access
2	ClusterRoleBinding	Provide cluster-wide read-only permissions
3	ServiceAccount Security	Control Pod access using ServiceAccount

---

✅ **Final Outcome:** After completing these labs, participants will understand how to: - Create users and assign permissions using RBAC. - Use ClusterRoles for broader permissions. - Secure Pods using ServiceAccounts.

These are the foundational skills for **Kubernetes security and access control** at the beginner level.