**🧪 Kubernetes Lab Guide: Creating and Managing Users with Client Certificates and RBAC**

**🧠 Scenario Analogy**

| **Real World** | **Kubernetes Equivalent** |
| --- | --- |
| Company Building | Kubernetes Cluster |
| Reception | Kubernetes Admin |
| Security Officer | Kubernetes API Server |
| Bob’s Badge | bob.crt (signed certificate) |
| Chip in Badge | Identity verified by Kubernetes CA |
| Entry Permissions | RBAC roles and bindings |
| Production Floor | Namespace + Pod Access |

**🎯 Lab Objectives**

* Generate a new user credential (certificate-based).
* Add the user to kubeconfig.
* Verify access control (authentication and authorization).
* Bind appropriate RBAC permissions.
* Clean up after testing.

**🔧 Prerequisites**

Ensure the following tools and permissions:

* Access to Kubernetes control plane node.
* kubectl configured as admin.
* openssl installed on the system.

**📁 Directory Setup**

bash

CopyEdit

mkdir ~/k8s-user-lab && cd ~/k8s-user-lab

**✅ Step 1: Generate Private Key & CSR for Bob**

**🛠️ Command:**

bash

CopyEdit

openssl genrsa -out bob.key 2048

openssl req -new -key bob.key -out bob.csr -subj "/CN=bob/O=developers"

* CN=bob: sets the username as bob.
* O=developers: adds Bob to the developers group (optional but useful for group-based RBAC).

**✅ Step 2: Sign Bob’s CSR with Kubernetes CA**

**🔍 Locate CA files (on control plane node):**

bash

CopyEdit

ls /etc/kubernetes/pki/ca.\*

**🖋️ Sign:**

bash

CopyEdit

openssl x509 -req -in bob.csr \

-CA /etc/kubernetes/pki/ca.crt \

-CAkey /etc/kubernetes/pki/ca.key \

-CAcreateserial \

-out bob.crt -days 365

This generates bob.crt, a valid certificate signed by the cluster's CA.

**✅ Step 3: Backup Admin Kubeconfig (Precaution)**

bash

CopyEdit

cd ~/.kube

cp config config\_bak

**✅ Step 4: Add Bob to Kubeconfig**

**4.1 Add Credentials**

bash

CopyEdit

kubectl config set-credentials bob \

--client-certificate=~/k8s-user-lab/bob.crt \

--client-key=~/k8s-user-lab/bob.key

**4.2 Get Cluster Name**

bash

CopyEdit

kubectl config get-clusters

Assume it's kubernetes.

**4.3 Add Bob’s Context**

bash

CopyEdit

kubectl config set-context bob-context \

--cluster=kubernetes \

--user=bob \

--namespace=raman

**4.4 Switch to Bob's Context**

bash

CopyEdit

kubectl config use-context bob-context

kubectl config current-context

**🔍 Step 5: Test Access (Expect Denied)**

bash

CopyEdit

kubectl get pods

**❌ Expected:**

plaintext

CopyEdit

Error from server (Forbidden): User "bob" cannot list resource "pods" in API group "" in the namespace "raman"

This confirms Bob is authenticated (valid cert) but not yet authorized (no RBAC).

**✅ Step 6: Grant Access using RBAC**

**6.1 Return to Admin Context**

bash

CopyEdit

kubectl config use-context kubernetes-admin@kubernetes

**6.2 Create Role (to list pods in raman namespace)**

yaml

CopyEdit

# role-pod-reader.yaml

apiVersion: rbac.authorization.k8s.io/v1

kind: Role

metadata:

name: pod-reader

namespace: raman

rules:

- apiGroups: [""]

resources: ["pods"]

verbs: ["get", "list", "watch"]

bash

CopyEdit

kubectl apply -f role-pod-reader.yaml

**6.3 Create RoleBinding**

yaml

CopyEdit

# rolebinding-bob.yaml

apiVersion: rbac.authorization.k8s.io/v1

kind: RoleBinding

metadata:

name: bob-pod-access

namespace: raman

subjects:

- kind: User

name: bob

apiGroup: rbac.authorization.k8s.io

roleRef:

kind: Role

name: pod-reader

apiGroup: rbac.authorization.k8s.io

bash

CopyEdit

kubectl apply -f rolebinding-bob.yaml

**🔍 Step 7: Switch to Bob Context and Test Access**

bash

CopyEdit

kubectl config use-context bob-context

kubectl get pods

**✅ Expected:**

* If there are pods: lists the pods in raman namespace.
* If none: shows empty list but no error.

**🧹 Step 8: Cleanup**

**Delete Role and RoleBinding**

bash

CopyEdit

kubectl config use-context kubernetes-admin@kubernetes

kubectl delete -f rolebinding-bob.yaml

kubectl delete -f role-pod-reader.yaml

**Delete Bob’s kubeconfig entries**

bash

CopyEdit

kubectl config delete-context bob-context

kubectl config delete-user bob

**Delete generated files**

bash

CopyEdit

cd ~/k8s-user-lab

rm -f bob.key bob.crt bob.csr ca.srl

cd ~/.kube

mv config\_bak config

**📝 Summary**

| **Step** | **Task** |
| --- | --- |
| 1 | Generate key & CSR |
| 2 | Sign CSR with cluster CA |
| 3 | Add user to kubeconfig |
| 4 | Create RBAC Role & RoleBinding |
| 5 | Validate access |
| 6 | Clean up objects and restore config |

**📚 Bonus Notes**

* Kubernetes **does not store user accounts** internally.
* AuthN: Based on certificates (CN and O) + ca.crt trust.
* AuthZ: RBAC determines what resources the user can access.

**🧪 Kubernetes RBAC Lab: Namespace-Specific Permissions Using ServiceAccount**

**🧠 Overview**

This lab demonstrates how **RBAC (Role-Based Access Control)** restricts access to Kubernetes resources on a **per-namespace basis** using a **ServiceAccount**.

You will:

* Create a namespace for isolation.
* Define a Role with read-only permissions on Pods.
* Create a ServiceAccount and bind it to the Role.
* Validate access using kubectl auth can-i.

**🔧 Prerequisites**

* A working Kubernetes cluster (minikube, kubeadm-based, or cloud-hosted).
* kubectl configured as an admin.
* Familiarity with basic Kubernetes objects (Pod, Namespace, ServiceAccount).

**📁 Step 1: Create a Namespace**

Namespaces allow logical partitioning of resources. We’ll create test-namespace.

bash

CopyEdit

kubectl create namespace test-namespace

**✅ Verify:**

bash

CopyEdit

kubectl get namespaces

**📁 Step 2: Create a Role with Read Access to Pods**

RBAC Role defines what actions are allowed within a **specific namespace**.

**📝 Create pod-reader-role.yaml:**

yaml

CopyEdit

apiVersion: rbac.authorization.k8s.io/v1

kind: Role

metadata:

namespace: test-namespace

name: pod-reader

rules:

- apiGroups: [""]

resources: ["pods"]

verbs: ["get", "list", "watch"]

You can also add "create" if you want to test elevated access scenarios.

**🛠️ Apply Role:**

bash

CopyEdit

kubectl apply -f pod-reader-role.yaml

**✅ Verify:**

bash

CopyEdit

kubectl get roles -n test-namespace

kubectl describe role pod-reader -n test-namespace

**📁 Step 3: Create a ServiceAccount**

ServiceAccounts represent non-human identities (e.g., for pods or automation).

bash

CopyEdit

kubectl create serviceaccount test-sa -n test-namespace

**✅ Verify:**

bash

CopyEdit

kubectl get serviceaccounts -n test-namespace

**📁 Step 4: Create a RoleBinding for the ServiceAccount**

RBAC RoleBinding links the Role to a specific subject (user, group, or service account).

**📝 Create pod-reader-rolebinding.yaml:**

yaml

CopyEdit

apiVersion: rbac.authorization.k8s.io/v1

kind: RoleBinding

metadata:

name: pod-reader-binding

namespace: test-namespace

subjects:

- kind: ServiceAccount

name: test-sa

namespace: test-namespace

roleRef:

kind: Role

name: pod-reader

apiGroup: rbac.authorization.k8s.io

✅ kind: ServiceAccount and namespace: test-namespace are required to correctly identify the subject.

**🛠️ Apply RoleBinding:**

bash

CopyEdit

kubectl apply -f pod-reader-rolebinding.yaml

**✅ Verify:**

bash

CopyEdit

kubectl get rolebindings -n test-namespace

kubectl describe rolebinding pod-reader-binding -n test-namespace

**🔍 Step 5: Test Access Using kubectl auth can-i**

We use --as to impersonate the ServiceAccount.

🔹 Format for ServiceAccounts:

ruby

CopyEdit

system:serviceaccount:<namespace>:<serviceaccount-name>

**✅ Test 1: Can the SA list pods in test-namespace?**

bash

CopyEdit

kubectl auth can-i list pods \

--namespace test-namespace \

--as system:serviceaccount:test-namespace:test-sa

**Expected Output:**

text

CopyEdit

yes

**✅ Test 2: Can the SA create pods in test-namespace?**

bash

CopyEdit

kubectl auth can-i create pods \

--namespace test-namespace \

--as system:serviceaccount:test-namespace:test-sa

**Expected Output:**

text

CopyEdit

no

We only allowed get, list, watch in the Role.

**✅ Test 3: Can the SA list pods in the default namespace?**

bash

CopyEdit

kubectl auth can-i list pods \

--namespace default \

--as system:serviceaccount:test-namespace:test-sa

**Expected Output:**

text

CopyEdit

no

The RoleBinding is **namespace-scoped** and doesn't apply outside test-namespace.

**🧹 Cleanup (Optional)**

bash

CopyEdit

kubectl delete role pod-reader -n test-namespace

kubectl delete rolebinding pod-reader-binding -n test-namespace

kubectl delete serviceaccount test-sa -n test-namespace

kubectl delete namespace test-namespace

**📝 Summary Table**

| **Object** | **Name** | **Scope** | **Purpose** |
| --- | --- | --- | --- |
| Namespace | test-namespace | Cluster-wide | Resource isolation |
| Role | pod-reader | test-namespace | Read-only access to Pods |
| ServiceAccount | test-sa | test-namespace | Non-human identity used for access |
| RoleBinding | pod-reader-binding | test-namespace | Binds Role to ServiceAccount |
| Command | kubectl auth can-i | N/A | Validates RBAC without actual execution |

**🔐 Bonus Tip: Using This ServiceAccount with a Pod**

You can mount this ServiceAccount into a pod for real usage:

yaml

CopyEdit

apiVersion: v1

kind: Pod

metadata:

name: test-pod

namespace: test-namespace

spec:

serviceAccountName: test-sa

containers:

- name: busybox

image: busybox

command: ["sleep", "3600"]

Apply and exec:

bash

CopyEdit

kubectl apply -f test-pod.yaml

kubectl exec -n test-namespace -it test-pod -- sh

**🧪 Kubernetes ConfigMap Lab: Mounting HTML Files into Nginx via ConfigMap**

**🎯 Objective**

This lab demonstrates how to:

* Create ConfigMaps from files (prod.html, dev.html)
* Mount a ConfigMap as a **volume** into a Pod
* Dynamically change Pod behavior based on ConfigMap contents
* Serve content using Nginx and verify it via curl

**🧱 Prerequisites**

* Kubernetes cluster running with access to kubectl
* Namespace raman already created
* Basic understanding of:
  + ConfigMaps
  + Pod YAML structure
  + Nginx default behavior

**🔧 Step-by-Step Lab Instructions**

**🧾 Step 1: Prepare HTML Files**

Create two HTML files that simulate content for two environments.

bash

CopyEdit

echo "hell from prod" > prod.html

echo "hell from dev" > dev.html

**✅ Verify:**

bash

CopyEdit

cat prod.html

cat dev.html

Expected Output:

csharp

CopyEdit

hell from prod

hell from dev

**🗂️ Step 2: Create ConfigMaps**

We'll create **two ConfigMaps**, each containing one HTML file.

bash

CopyEdit

kubectl create configmap prod.cmap --from-file=prod.html -n raman

kubectl create configmap dev.cmap --from-file=dev.html -n raman

**✅ Verify ConfigMaps:**

bash

CopyEdit

kubectl get configmaps -n raman

kubectl describe configmap prod.cmap -n raman

kubectl get configmap prod.cmap -n raman -o yaml > prod-cmap.yml

**📦 Step 3: Deploy a Pod with ConfigMap Mounted**

Create a file called podcm.yml with the following content:

yaml

CopyEdit

apiVersion: v1

kind: Pod

metadata:

name: nginx

namespace: raman

labels:

app: nginx

spec:

containers:

- name: nginx

image: nginx:latest

ports:

- containerPort: 80

volumeMounts:

- name: rk

mountPath: /usr/share/nginx/html

volumes:

- name: rk

configMap:

name: prod.cmap

items:

- key: prod.html

path: index.html

🔍 **Explanation**:

* **mountPath: /usr/share/nginx/html**: Nginx’s default web root.
* **items**: You map prod.html to index.html so that it serves as the default page.

**🚀 Step 4: Deploy the Pod**

bash

CopyEdit

kubectl apply -f podcm.yml

**✅ Check Pod Status:**

bash

CopyEdit

kubectl get pods -n raman -o wide

Wait for the nginx pod to be in Running status.

**🌐 Step 5: Expose the Pod (NodePort)**

bash

CopyEdit

kubectl expose pod nginx -n raman --type=NodePort --port=80 --target-port=80 --name=cmsvc

**✅ Get Service Info:**

bash

CopyEdit

kubectl get svc -n raman

Look for the cmsvc service, and note the NodePort (e.g., 31304).

**🔍 Step 6: Test Access Using curl**

Use the IP of a node in the cluster (e.g., master or worker) and the NodePort.

bash

CopyEdit

curl <NodeIP>:<NodePort>

Example:

bash

CopyEdit

curl 192.168.190.105:31304

Expected Output:

csharp

CopyEdit

hell from prod

✅ This confirms that the content served is coming from the prod.cmap.

**🔄 Step 7: Switch to dev.cmap**

Update podcm.yml to use dev.cmap instead of prod.cmap.

yaml

CopyEdit

volumes:

- name: rk

configMap:

name: dev.cmap

items:

- key: dev.html

path: index.html

**🛠 Replace the Pod with new ConfigMap:**

bash

CopyEdit

kubectl replace --force -f podcm.yml

--force deletes and recreates the pod.

**✅ Re-Test with curl:**

bash

CopyEdit

curl <NodeIP>:<NodePort>

Expected Output:

csharp

CopyEdit

hell from dev

**🧹 Optional Cleanup**

bash

CopyEdit

kubectl delete svc cmsvc -n raman

kubectl delete pod nginx -n raman

kubectl delete configmap prod.cmap dev.cmap -n raman

rm prod.html dev.html prod-cmap.yml podcm.yml

**📘 Recap: What You Learned**

| **Resource** | **Purpose** |
| --- | --- |
| ConfigMap | Injected file-based content (HTML) into pods |
| Pod | Used nginx container with mounted config |
| VolumeMount | Mounted HTML file at Nginx web root |
| Service | Exposed the pod externally via NodePort |
| curl | Verified which file was being served |

**🔐 Bonus: Serving Multiple Pages**

You can configure the volume like this to serve **both files**:

yaml

CopyEdit

configMap:

name: dev.cmap

items:

- key: dev.html

path: dev.html

Then access via curl <NodeIP>:<NodePort>/dev.html.

**🧪 Lab Guide: Deploy Fluentd DaemonSet with ServiceAccount for Log Collection**

**🎯 Objective**

Deploy **Fluentd** as a **DaemonSet** to collect logs from all Kubernetes nodes and optionally send them to an Elasticsearch backend. The guide also demonstrates how to:

* Run the DaemonSet on **all nodes**, including control-plane (optional)
* Mount host log directory (/var/log) into containers
* Use a **dedicated ServiceAccount** with proper namespace scoping

**🧱 Prerequisites**

* Running Kubernetes cluster with at least one control plane and one worker
* kubectl configured and working
* Sufficient privileges to create resources in kube-system namespace

**🛠 Step-by-Step Instructions**

**🔹 Step 1: Create a ServiceAccount**

Create a ServiceAccount named fluentd-sa in the kube-system namespace.

**fluentd-sa.yml**:

yaml

CopyEdit

apiVersion: v1

kind: ServiceAccount

metadata:

name: fluentd-sa

namespace: kube-system

bash

CopyEdit

kubectl apply -f fluentd-sa.yml

✅ **Verify:**

bash

CopyEdit

kubectl get sa fluentd-sa -n kube-system

**🔹 Step 2: Define the Fluentd DaemonSet**

**daemonset.yml**:

yaml

CopyEdit

apiVersion: apps/v1

kind: DaemonSet

metadata:

name: fluentd-elasticsearch

namespace: kube-system

labels:

k8s-app: fluentd-logging

spec:

selector:

matchLabels:

name: fluentd-elasticsearch

template:

metadata:

labels:

name: fluentd-elasticsearch

spec:

serviceAccountName: fluentd-sa

tolerations:

# Allow running on control-plane nodes

- key: node-role.kubernetes.io/control-plane

operator: Exists

effect: NoSchedule

- key: node-role.kubernetes.io/master

operator: Exists

effect: NoSchedule

containers:

- name: fluentd-elasticsearch

image: quay.io/fluentd\_elasticsearch/fluentd:v2.5.2

resources:

limits:

memory: 200Mi

requests:

cpu: 100m

memory: 200Mi

volumeMounts:

- name: varlog

mountPath: /var/log

- name: containers

mountPath: /var/lib/docker/containers

readOnly: true

terminationGracePeriodSeconds: 30

volumes:

- name: varlog

hostPath:

path: /var/log

- name: containers

hostPath:

path: /var/lib/docker/containers

bash

CopyEdit

kubectl apply -f daemonset.yml

✅ **Verify Pods:**

bash

CopyEdit

kubectl get pods -n kube-system -l name=fluentd-elasticsearch -o wide

You should see one Pod per node, including the control plane if tolerations are allowed.

**🔹 Step 3: View Mounted Log Directory**

Exec into one of the Fluentd pods and check log files mounted from the host:

bash

CopyEdit

POD=$(kubectl get pods -n kube-system -l name=fluentd-elasticsearch -o jsonpath="{.items[0].metadata.name}")

kubectl exec -n kube-system $POD -- ls /var/log

Expected output includes:

nginx

CopyEdit

containers messages syslog journal ...

You can further check container logs like:

bash

CopyEdit

kubectl exec -n kube-system $POD -- tail -n 20 /var/log/syslog

**🔹 Step 4: (Optional) Define RBAC for Enhanced Permissions**

If Fluentd is to collect logs or metadata from the Kubernetes API, define a Role and RoleBinding.

**fluentd-rbac.yml**:

yaml

CopyEdit

apiVersion: rbac.authorization.k8s.io/v1

kind: ClusterRole

metadata:

name: fluentd-role

rules:

- apiGroups: [""]

resources: ["pods", "namespaces", "nodes"]

verbs: ["get", "list", "watch"]

---

apiVersion: rbac.authorization.k8s.io/v1

kind: ClusterRoleBinding

metadata:

name: fluentd-rolebinding

roleRef:

apiGroup: rbac.authorization.k8s.io

kind: ClusterRole

name: fluentd-role

subjects:

- kind: ServiceAccount

name: fluentd-sa

namespace: kube-system

bash

CopyEdit

kubectl apply -f fluentd-rbac.yml

✅ **Verify:**

bash

CopyEdit

kubectl auth can-i list pods --as=system:serviceaccount:kube-system:fluentd-sa

**🔍 Step 5: Confirm Fluentd is Running**

Use logs to confirm proper startup:

bash

CopyEdit

kubectl logs -n kube-system $POD

You should see Fluentd start logs and tailing file activity from /var/log.

**📘 Recap: What You Learned**

| **Resource** | **Purpose** |
| --- | --- |
| DaemonSet | Ensures Fluentd runs on all (or selected) nodes |
| ServiceAccount | Gives the pod an identity to access resources |
| HostPath Volumes | Mounts node file system for log scraping |
| Tolerations | Allows pod to run on master/control-plane nodes |
| RBAC | Enables safe, scoped API access for log metadata enrichment |

**🧹 Cleanup**

bash

CopyEdit

kubectl delete daemonset fluentd-elasticsearch -n kube-system

kubectl delete sa fluentd-sa -n kube-system

kubectl delete clusterrole fluentd-role

kubectl delete clusterrolebinding fluentd-rolebinding