**🔬 Lab Guide: Using Kubernetes Secrets with a Deployment (Namespace: raman)**

**✅ Lab Prerequisites**

1. A running Kubernetes cluster (You’re using kubeadm, so assumed ready).
2. Namespace raman exists. If not:

bash

CopyEdit

kubectl create namespace raman

1. kubectl is aliased as k.

**🧩 Step 1: Base64 Encode Sensitive Data**

Kubernetes Secrets require the data to be base64-encoded.

You encoded:

bash

CopyEdit

# Username

echo 'ramankhanna' | base64

# Output:

cmFtYW5raGFubmEK

# Password

echo 'ramankhanna123' | base64

# Output:

cmFtYW5raGFubmExMjMK

These values will be used in the secret manifest.

**📄 Step 2: Create the Kubernetes Secret Manifest**

Create a file named secret.yml:

yaml

CopyEdit

apiVersion: v1

kind: Secret

metadata:

name: my-secrets

namespace: raman

type: Opaque

data:

username: cmFtYW5raGFubmEK

password: cmFtYW5raGFubmExMjMK

🔐 **Note:** The type: Opaque is used for generic key-value pairs.

**🚀 Apply the Secret**

bash

CopyEdit

k create -f secret.yml

**🧾 Verify the Secret**

bash

CopyEdit

# List secrets in 'raman' namespace

k get secrets -n raman

# Describe to verify keys

k describe secret my-secrets -n raman

**📄 Step 3: Create the Deployment Manifest**

Filename: deploy2.yml

yaml

CopyEdit

apiVersion: apps/v1

kind: Deployment

metadata:

name: myapp-deployment

namespace: raman

labels:

app: myapp

type: front-end

spec:

replicas: 3

selector:

matchLabels:

app: myapp

type: front-end

template:

metadata:

labels:

app: myapp

type: front-end

spec:

containers:

- name: httpd-container

image: httpd

env:

- name: SECRET\_USERNAME

valueFrom:

secretKeyRef:

name: my-secrets

key: username

- name: SECRET\_PASSWD

valueFrom:

secretKeyRef:

name: my-secrets

key: password

**🚀 Deploy it:**

bash

CopyEdit

k create -f deploy2.yml

**🔍 Step 4: Verify Deployment**

bash

CopyEdit

# Check pod status

k get pods -n raman

Expected: 3 pods of myapp-deployment running.

**🔎 Step 5: Verify Secret in Pod (Environment Variable)**

Choose one pod and exec into it:

bash

CopyEdit

# Replace with actual pod name if needed

k exec -it <pod-name> -n raman -- /bin/bash

If you're using httpd (Apache), the container might not have /bin/bash. Use /bin/sh:

bash

CopyEdit

k exec -it <pod-name> -n raman -- /bin/sh

**Inside the container:**

Check environment variables:

sh

CopyEdit

echo $SECRET\_USERNAME

# Output: ramankhanna

echo $SECRET\_PASSWD

# Output: ramankhanna123

If you're seeing **encoded values**, that means the key in the Secret was put under stringData instead of base64 under data.

**🧪 Step 6: (Optional) View the Secret Data (Decoded) from Cluster**

bash

CopyEdit

# Get raw secret

k get secret my-secrets -n raman -o yaml

# Decode username:

echo 'cmFtYW5raGFubmEK' | base64 -d

# Decode password:

echo 'cmFtYW5raGFubmExMjMK' | base64 -d

**🧼 Cleanup Resources (Optional)**

bash

CopyEdit

k delete deployment myapp-deployment -n raman

k delete secret my-secrets -n raman

**🔄 Summary**

| **Resource** | **Purpose** |
| --- | --- |
| **Secret** | Stores username, password securely |
| **Deployment** | Reads secret keys as environment variables |
| **Pod** | Consumes env vars via secretKeyRef |
| **Namespace** | Isolated environment for resources |

**🧠 Notes**

* Kubernetes secrets are **base64-encoded**, not encrypted by default.
  + For encryption at rest, enable KMS or EncryptionConfiguration in kube-apiserver.
* Avoid committing Secrets to version control directly — use tools like **SealedSecrets**, **Vault**, or **SOPS** for better security.
* env.secretKeyRef is one method; secrets can also be mounted as volumes.

**🧪 LAB: Shared NFS Storage using PV + PVC + AWS EFS in Kubernetes**

**🎯 Objective**

Deploy an NGINX-based Kubernetes deployment where **multiple replicas share a central NFS-backed volume** using ReadWriteMany (RWX) access via AWS EFS (or NFS server).

**🧱 LAB Components**

| **Component** | **Role** |
| --- | --- |
| **NFS Server / AWS EFS** | Centralized file storage, accessible by all nodes |
| **PersistentVolume (PV)** | Defines the NFS mount as a volume Kubernetes can claim |
| **PersistentVolumeClaim (PVC)** | Requests a chunk of storage from the defined PV |
| **Deployment (10 Pods)** | All pods mount the shared volume (RWX) |
| **NodePort Service** | Exposes the NGINX deployment for external access |

**🧑‍💻 Step 1: Set Up the NFS Server (for Lab or mimic EFS setup)**

**On your NFS server node (or EFS mount target)**

**A. Install NFS Tools**

For a **Kubernetes worker/master node** to access EFS or NFS:

bash

CopyEdit

sudo apt update

sudo apt install -y nfs-common

**B. Mount the AWS EFS or NFS Server**

bash

CopyEdit

# Create mount point

sudo mkdir /efs

# Mount EFS or NFS manually

sudo mount -t nfs4 -o nfsvers=4.1,rsize=1048576,wsize=1048576,hard,timeo=600,retrans=2,noresvport 172.31.12.142:/ /efs

Replace 172.31.12.142 with your actual **EFS mount target** IP (or your own NFS server).

**C. Verify Mount**

bash

CopyEdit

df -h | grep efs

cd /efs

ls -la

Ensure the directory is accessible and writable.

**📦 Step 2: Define PersistentVolume (PV) and PersistentVolumeClaim (PVC)**

Create a file called nfs-pv-pvc.yml:

yaml

CopyEdit

apiVersion: v1

kind: PersistentVolume

metadata:

name: nfs-pv-raman

spec:

capacity:

storage: 11Mi

accessModes:

- ReadWriteMany

persistentVolumeReclaimPolicy: Retain

mountOptions:

- hard

- nfsvers=4.1

nfs:

path: /

server: 172.31.12.142

---

apiVersion: v1

kind: PersistentVolumeClaim

metadata:

name: nfs-raman-pvc

spec:

accessModes:

- ReadWriteMany

resources:

requests:

storage: 5Mi

volumeName: nfs-pv-raman

**Apply PV and PVC**

bash

CopyEdit

kubectl apply -f nfs-pv-pvc.yml

**🚀 Step 3: Create the Deployment Using PVC**

Create a file called nfs-deploy.yml:

yaml

CopyEdit

apiVersion: apps/v1

kind: Deployment

metadata:

name: nfs-raman

spec:

replicas: 10

selector:

matchLabels:

role: nfs-raman

template:

metadata:

labels:

role: nfs-raman

spec:

containers:

- name: nginx

image: nginx:1.14.2

ports:

- containerPort: 80

volumeMounts:

- name: nfs

mountPath: /usr/share/nginx/deploydata

volumes:

- name: nfs

persistentVolumeClaim:

claimName: nfs-raman-pvc

**Apply Deployment**

bash

CopyEdit

kubectl apply -f nfs-deploy.yml

**🌐 Step 4: Expose the Deployment as a NodePort Service**

bash

CopyEdit

kubectl expose deploy nfs-raman --name nfs-svc --type NodePort --target-port 80 --port 80

**🧾 Step 5: Verification**

**A. Verify PV and PVC**

bash

CopyEdit

kubectl get pv

kubectl get pvc

Expected:

* PV nfs-pv-raman status: **Bound**
* PVC nfs-raman-pvc status: **Bound**

**B. Verify Pods**

bash

CopyEdit

kubectl get pods -l role=nfs-raman

You should see 10 pods in **Running** state.

**C. Verify Mount in Pod**

Pick one pod:

bash

CopyEdit

kubectl exec -it <pod-name> -- /bin/sh

Inside container:

bash

CopyEdit

cd /usr/share/nginx/deploydata

touch from-pod-1.txt

exit

Now check on another pod:

bash

CopyEdit

kubectl exec -it <another-pod-name> -- /bin/sh

ls /usr/share/nginx/deploydata

# Should show: from-pod-1.txt

✅ This confirms that **shared volume is mounted** and **RWX is working**.

**🌍 Step 6: Access the Application in Browser**

**A. Get NodePort**

bash

CopyEdit

kubectl get svc nfs-svc

Look for the PORT(S) like:

ruby

CopyEdit

80:<NodePort>/TCP

E.g., 80:31234/TCP ⇒ NodePort is 31234

**B. Get Node Public IP**

For AWS:

bash

CopyEdit

# Or get from EC2 dashboard

curl ifconfig.me

Browse:

php-template

CopyEdit

http://<Node\_Public\_IP>:<NodePort>

You should see **NGINX Welcome Page**.

**🔐 Step 7: Security Group Validation (AWS Specific)**

Ensure **EFS mount target** has **inbound rules** that allow:

* **NFS (2049)** from your worker/master nodes (source: security group or IP range).
* If using a **custom NFS server**, ensure port 2049 is open between nodes and server.

**🧼 Step 8: Cleanup (Optional)**

bash

CopyEdit

kubectl delete deployment nfs-raman

kubectl delete service nfs-svc

kubectl delete pvc nfs-raman-pvc

kubectl delete pv nfs-pv-raman

**📘 Summary Table**

| **Component** | **Description** |
| --- | --- |
| **nfs-pv-raman** | Exposes / from 172.31.12.142 (EFS/NFS) |
| **nfs-raman-pvc** | Requests 5Mi RWX from the PV |
| **nfs-raman** | NGINX pods that mount shared volume |
| **nfs-svc** | NodePort service to access NGINX externally |