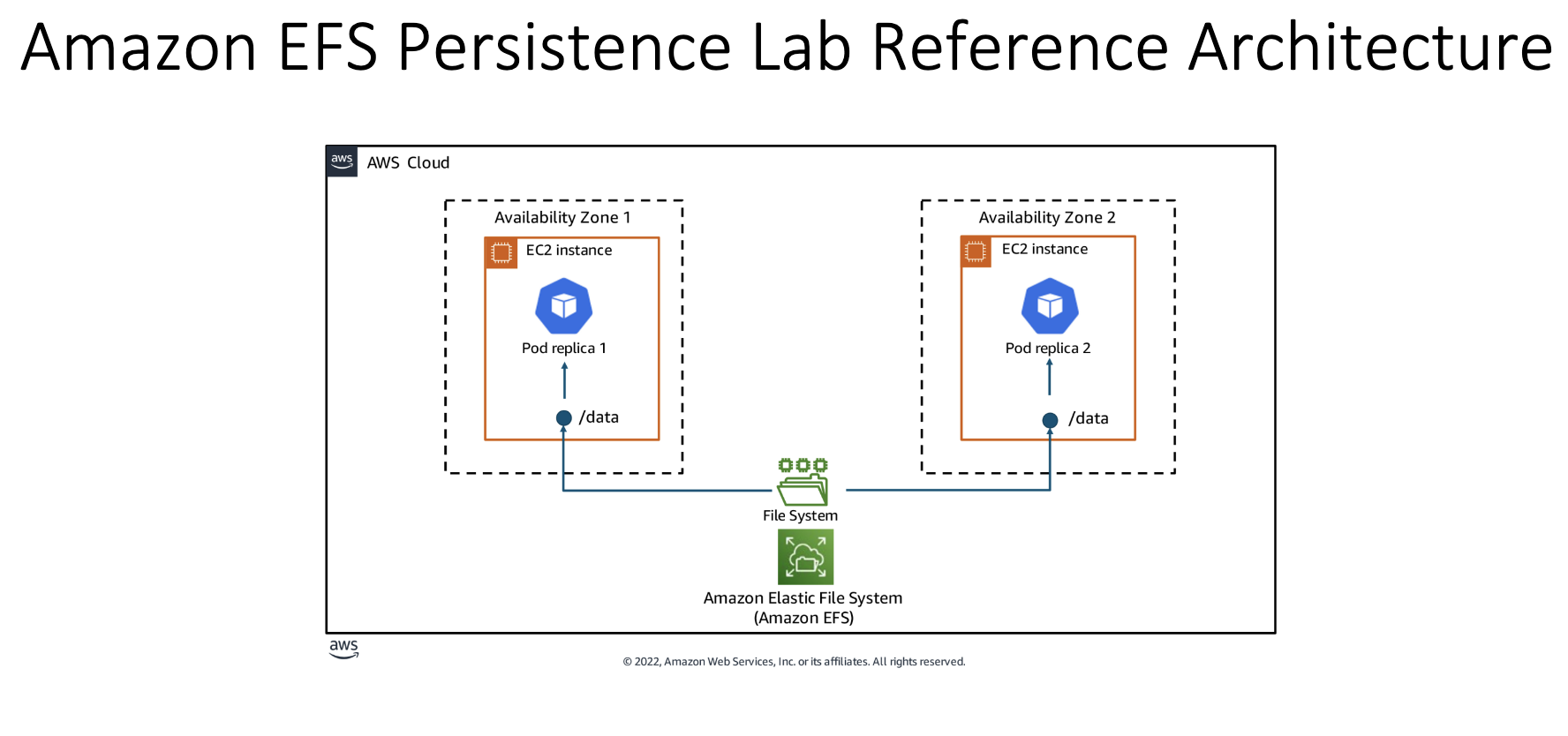
# Lab 5: Configuring Storage in Amazon EKS

Objectives-

1. Set up Amazon EFS for distributed workloads.
2. Configure persistent volumes and claims.
3. Learn the Amazon EFS CSI driver's role.
4. View and update stored files.



Two pod replicas scheduled to different Amazon Elastic Compute Cloud (Amazon EC2) instances that share a persistent volume store hosted in Amazon Elastic File System (EFS).

**Amazon EFS: a service that provides a place to store files that can be accessed by multiple applications at once, even if they're running on different servers.**

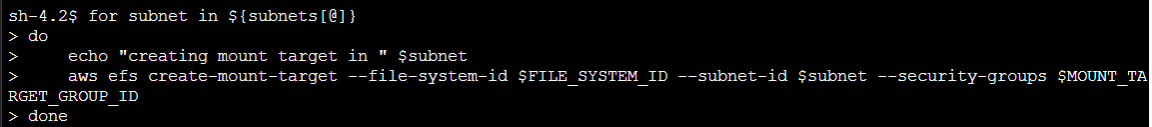
**Task 1: Install and review Amazon EFS setup**

1.1 Connect to Bastion host

1.2 To create an EFS file system and save the EFS Id to a shell variable, enter the following command:



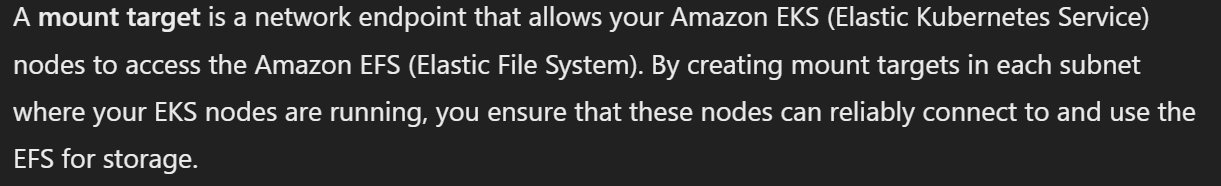
1.3 To identify the subnets hosting your EKS nodes, create a mount target in each of them using:



**SUBNET LEVEL - MOUNT TARGET**

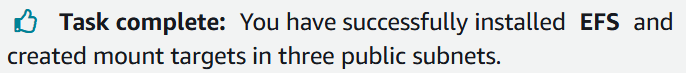
Output-



From GPT- 

My thought-

Creating mount targets in each of the subnets where your EKS nodes are located ensures that your EFS file system is highly available and can be accessed from multiple Availability Zones. This setup provides redundancy and helps maintain access to the EFS file system even if one Availability Zone experiences issues.



**Task 2: Create persistence components and install the Amazon EFS CSI driver**

Steps in easy way:

Install the Amazon EFS CSI Driver: This tool helps Kubernetes talk to Amazon EFS.

Create an EFS File System: Think of this as setting up a shared hard drive that can be used by your application.

Set Up StorageClass, PersistentVolume (PV), and PersistentVolumeClaim (PVC):

StorageClass: This tells Kubernetes how to use the EFS storage.

PersistentVolume (PV): This is a piece of storage in your cluster.

PersistentVolumeClaim (PVC): This is a request for storage by your application.

Connect Your Application to the Storage: The application (like a product catalog) will use the PVC to save and access its data.

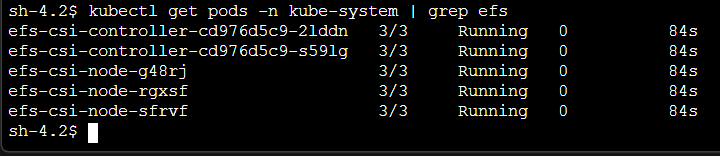
2.1 Enter these commands to install EFS CSI driver using HELM:

helm repo add aws-efs-csi-driver https://kubernetes-sigs.github.io/aws-efs-csi-driver/

helm repo update

helm upgrade --install aws-efs-csi-driver --namespace kube-system --set image.tag=v2.0.0 aws-efs-csi-driver/aws-efs-csi-driver

2.2 To verify that pods have been deployed, enter the following command:

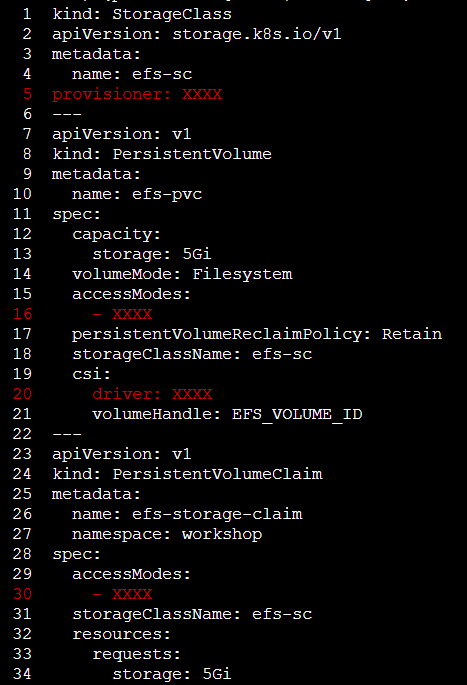


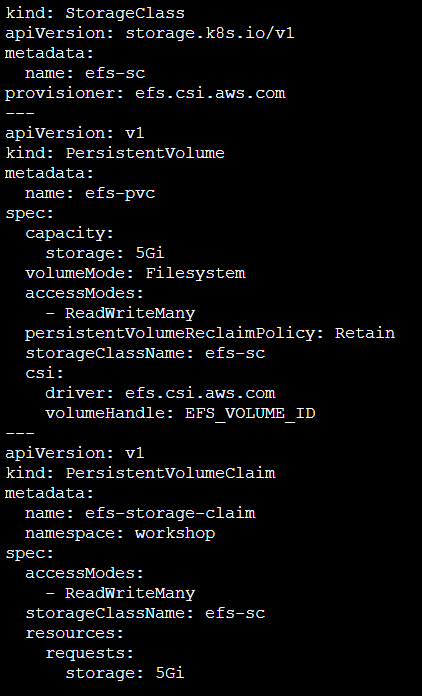
2.3 CHALLENGE: DEPLOY A PERSISTENT VOLUME

This step is presented as a challenge in the lab

We open the manifest file for PersistentVolume in editor, and change the placeholder values (provided by the lab) by the actual values, to proceed in the lab further.

Original Manifest File for PersistentVolume (kind value in line 8):

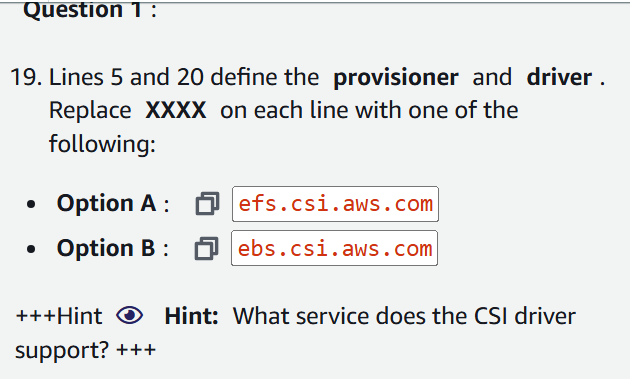




The manifest file is opened in an editor environment, and the challenge is to replace the values of “provisoner”, “driver”, “accessmodes”.

Using the hints given in the lab we have to fill these values.  
(solutions also provided in the lab)

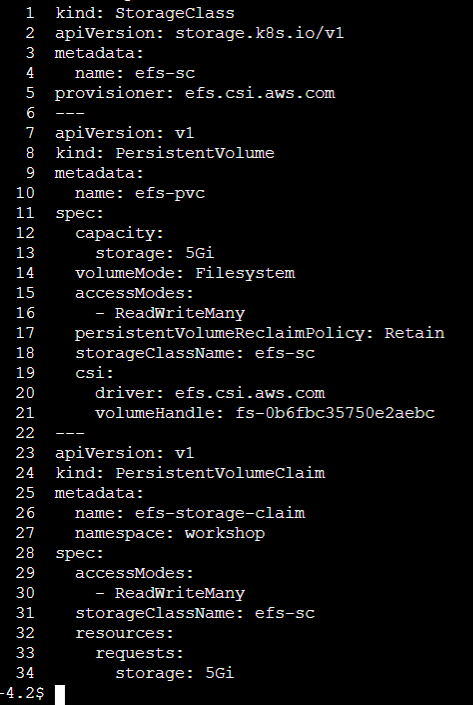
Example of the hint:



2.4 To replace EFS\_VOLUME\_ID on line 21 (of the manifest file) with the actual EFS ID, enter the following command:



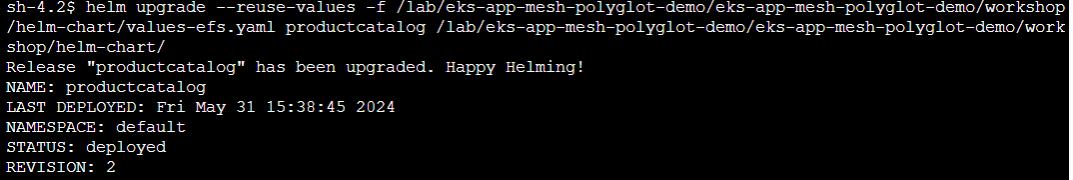
2.5 Manifest file after changes



2.6 Apply manifest file changes to Cluster using:

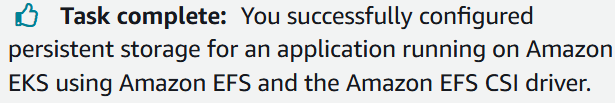
kubectl apply -f /lab/eks-app-mesh-polyglot-demo/eks-app-mesh-polyglot-demo/workshop/efs-pvc.yaml

2.7 To upgrade the application with EFS details, enter the following command:



RESULT TILL NOW - The pod for prodcatalogservice references the PVC resource named efs-storage-claim created earlier and mounts the backing PersistentVolume to a local directory named /products.

The PVC has been associated with the PersistentVolume.

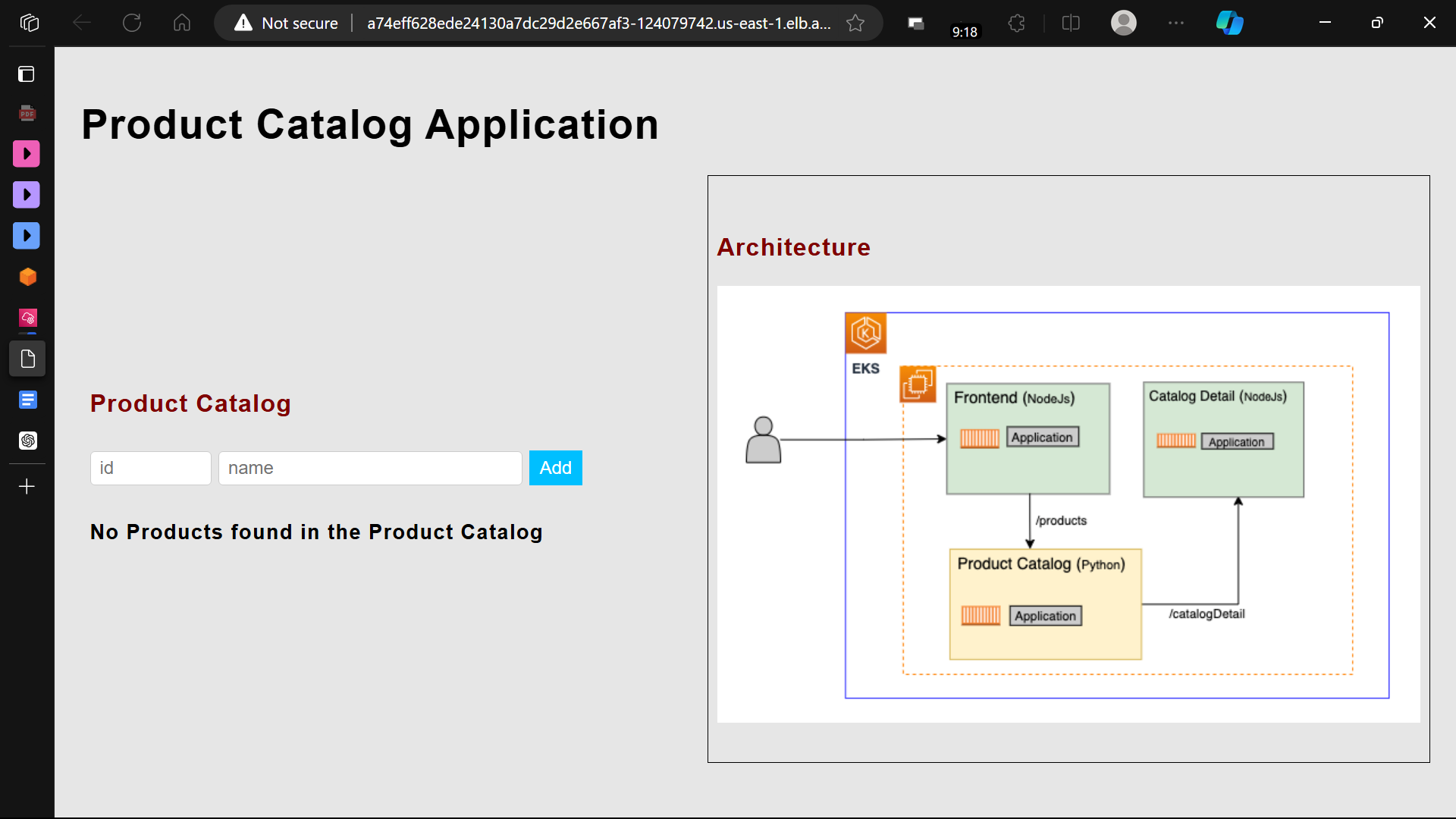


**Task 3: Test persistence**

3.1 View the application url using:

export LB\_NAME=$(kubectl get svc --namespace workshop frontend -o jsonpath="{.status.loadBalancer.ingress[\*].hostname}")

echo http://$LB\_NAME:80



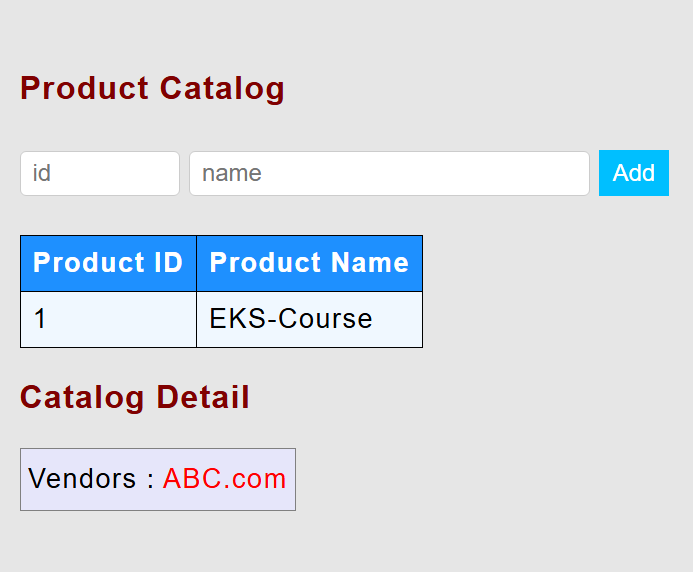
3.2 Now we check the connection of PVC with PersistentVolume

3.3 Products added to the application catalog are saved to a file called products.txt (by the lab)

That file is empty initially.



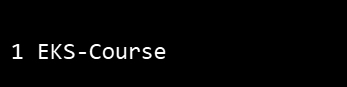
3.4 add a product in the actual application



3.5 On viewing the products.txt file again, we see that the entry has been made in the file:

Input - cat /products/products.txt (to view the products.txt file)

output-

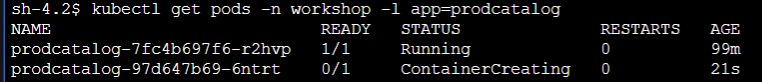


3.6 We currently have one instance of the application pod. To actually test EFS service, we will make replicas = 2.

To test if the data mounted to Amazon EFS is persistent across pods, scale up prodcatalog service to two replicas and see whether the second pod is able to successfully read the same data from the shared persistent volume.



3.7 Check the status of creation of 2nd pod



3.8 Now, we enter the 2nd pod using:

NEW\_POD=$(kubectl get pods -n workshop -l app=prodcatalog --sort-by=.metadata.creationTimestamp -o jsonpath='{.items[-1:].metadata.name}')

kubectl -n workshop exec -it $NEW\_POD -c prodcatalog -- /bin/bash

3.9 To check if EKS-Course with ID 1 exists in this new replica pod, enter the following command:



output -



**Note -** This demonstrates the persistence of data across multiple pods using the EFS storage. It shows that the newly created pod has access to the same data written by the previous pod.

