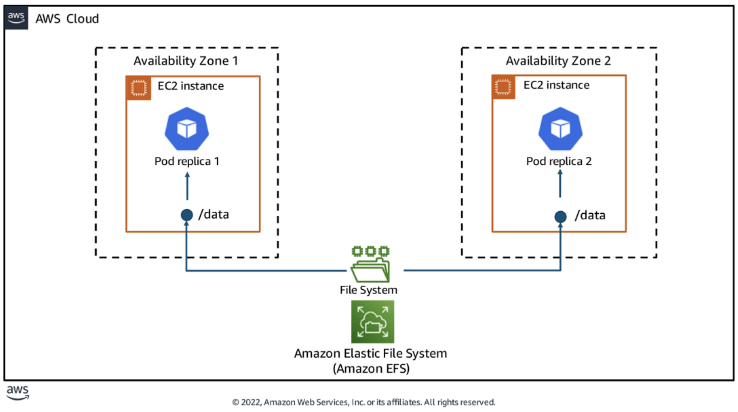
Lab 5: Configuring Storage in Amazon EKS

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.

Amazon EFS Persistence Lab Reference Architecture



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:

```
sh-4.2$ FILE_SYSTEM_ID=$(aws efs create-file-system | jq --raw-output '.FileSystemId')
sh-4.2$ echo "The File System ID is $FILE_SYSTEM_ID"
The File System ID is fs-03d03fbfbc213febe
```

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

```
sh-4.2$ for subnet in ${subnets[@]}
> do
>     echo "creating mount target in " $subnet
>     aws efs create-mount-target --file-system-id $FILE_SYSTEM_ID --subnet-id $subnet --security-groups $MOUNT_TA
RGET_GROUP_ID
> done
```

Output-

```
creating mount target in  subnet-0f2df34dba68eab21
{
    "OwnerId": "917759929138",
    "MountTargetId": "fsmt-097903ccd4321e34a",
    "FileSystemId": "fs-03d03fbfbc213febe",
    "SubnetId": "subnet-0f2df34dba68eab21",
    "LifeCycleState": "creating",
    "IpAddress": "10.10.71.65",
    "NetworkInterfaceId": "eni-0a9a6674c03cea132",
    "AvailabilityZoneId": "use1-az1",
    "AvailabilityZoneName": "us-east-1a",
    "VpcId": "vpc-0cec511b4435e219e"
}
creating mount target in  subnet-0219c4185c366f070
{
    "OwnerId": "917759929138",
    "MountTargetId": "fsmt-0c154db9c4ed24531",
    "FileSystemId": "fs-03d03fbfbc213febe",
    "SubnetId": "subnet-0219c4185c366f070",
    "LifeCycleState": "creating",
    "IpAddress": "10.10.111.205",
    "NetworkInterfaceId": "eni-0bf80c28a3904712a",
    "AvailabilityZoneId": "use1-az2",
    "AvailabilityZoneName": "us-east-1b",
    "VpcId": "vpc-0cec511b4435e219e"
}
creating mount target in  subnet-08c8477701a38f591
{
    "OwnerId": "917759929138",
    "MountTargetId": "fsmt-0293fbd5bf2b5b17f",
    "FileSystemId": "fs-03d03fbfbc213febe",
    "SubnetId": "subnet-08c8477701a38f591",
    "LifeCycleState": "creating",
    "IpAddress": "10.10.153.153",
    "NetworkInterfaceId": "eni-053e6cf4750ab2b85",
    "AvailabilityZoneId": "use1-az4",
    "AvailabilityZoneName": "us-east-1c",
    "VpcId": "vpc-0cec511b4435e219e"
}
sh-4.2$
```

From GPT-

A **mount target** is a network endpoint that allows your Amazon EKS (Elastic Kubernetes Service) nodes to access the Amazon EFS (Elastic File System). By creating mount targets in each subnet where your EKS nodes are running, you ensure that these nodes can reliably connect to and use the EFS for storage.

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 complete: You have successfully installed **EFS** and created mount targets in three public subnets.

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:

```
sh-4.2$ kubectl get pods -n kube-system | grep efs
efs-csi-controller-cd976d5c9-21ddn   3/3     Running   0          84s
efs-csi-controller-cd976d5c9-s591g   3/3     Running   0          84s
efs-csi-node-g48rj                   3/3     Running   0          84s
efs-csi-node-rgxsf                   3/3     Running   0          84s
efs-csi-node-sfrvf                   3/3     Running   0          84s
sh-4.2$
```

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):

```yaml
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: efs-sc
provisioner: XXXX
---
apiVersion: v1
kind: PersistentVolume
metadata:
  name: efs-pvc
spec:
  capacity:
    storage: 5Gi
  volumeMode: Filesystem
  accessModes:
    - XXXX
  persistentVolumeReclaimPolicy: Retain
  storageClassName: efs-sc
  csi:
    driver: XXXX
    volumeHandle: EFS_VOLUME_ID
---
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: efs-storage-claim
  namespace: workshop
spec:
  accessModes:
    - XXXX
  storageClassName: efs-sc
  resources:
    requests:
      storage: 5Gi
```

```yaml
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: efs-sc
provisioner: efs.csi.aws.com
---
apiVersion: v1
kind: PersistentVolume
metadata:
  name: efs-pvc
spec:
  capacity:
    storage: 5Gi
  volumeMode: Filesystem
  accessModes:
    - ReadWriteMany
  persistentVolumeReclaimPolicy: Retain
  storageClassName: efs-sc
  csi:
    driver: efs.csi.aws.com
    volumeHandle: EFS_VOLUME_ID
---
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: efs-storage-claim
  namespace: workshop
spec:
  accessModes:
    - ReadWriteMany
  storageClassName: efs-sc
  resources:
    requests:
      storage: 5Gi
```

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:

Question 1 :

19. Lines 5 and 20 define the **provisioner** and **driver** . Replace **XXXX** on each line with one of the following:

- **Option A** : `efs.csi.aws.com`
- **Option B** : `ebs.csi.aws.com`

+++Hint **Hint:** What service does the CSI driver support? +++

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

```
sh-4.2$ sudo sed -i "s/EFS_VOLUME_ID/$FILE_SYSTEM_ID/g" /lab/eks-app-mesh-polyglot-demo/eks-app-mesh-polygl
ot-demo/workshop/efs-pvc.yaml
```

2.5 Manifest file after changes

```
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: efs-sc
provisioner: efs.csi.aws.com
---
apiVersion: v1
kind: PersistentVolume
metadata:
  name: efs-pvc
spec:
  capacity:
    storage: 5Gi
  volumeMode: Filesystem
  accessModes:
    - ReadWriteMany
  persistentVolumeReclaimPolicy: Retain
  storageClassName: efs-sc
  csi:
    driver: efs.csi.aws.com
    volumeHandle: fs-0b6fbc35750e2aebc
---
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: efs-storage-claim
  namespace: workshop
spec:
  accessModes:
    - ReadWriteMany
  storageClassName: efs-sc
  resources:
    requests:
      storage: 5Gi
-4.2$
```

2.6 Apply manifest file changes to Cluster using:

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

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

```
sh-4.2$ helm upgrade --reuse-values -f /lab/eks-app-mesh-polyglot-demo/eks-app-mesh-polyglot-demo/workshop
/helm-chart/values-efs.yaml productcatalog /lab/eks-app-mesh-polyglot-demo/eks-app-mesh-polyglot-demo/work
shop/helm-chart/
Release "productcatalog" has been upgraded. Happy Helming!
NAME: productcatalog
LAST DEPLOYED: Fri May 31 15:38:45 2024
NAMESPACE: default
STATUS: deployed
REVISION: 2
```

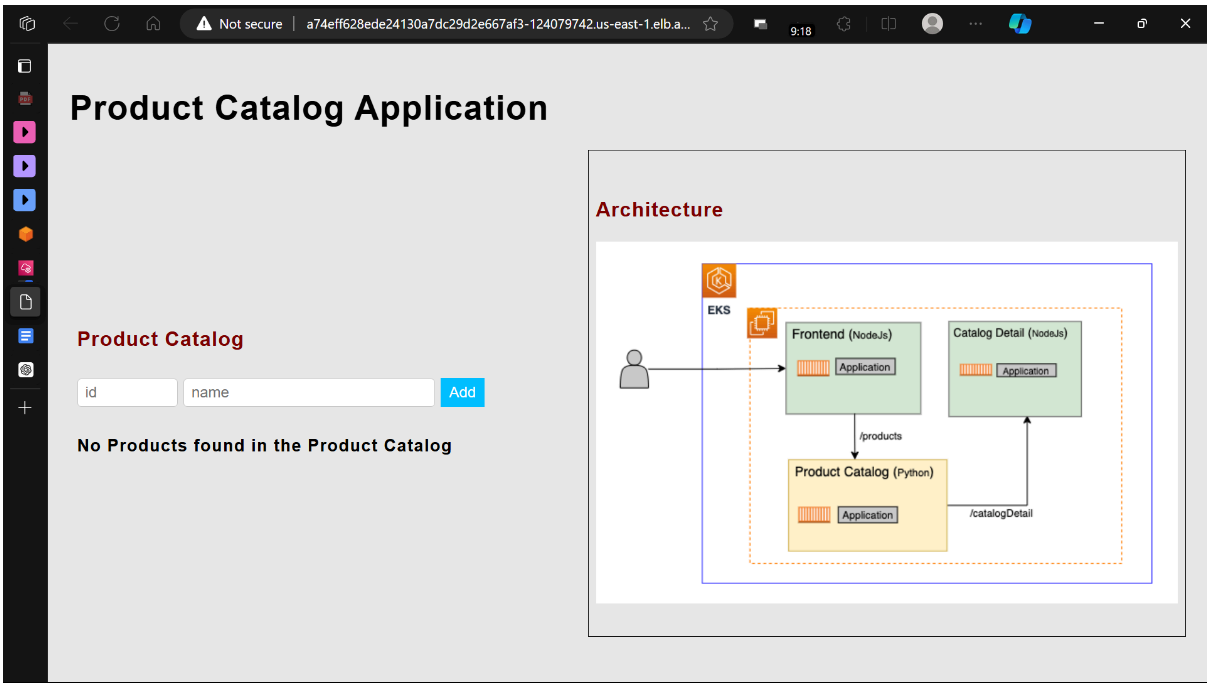
RESULT TILL NOW - The pod for prodcatalogservice references the PVC resource named efs-storage-claim created earlier and mounts the backing PersistentVolume to a local directory named /products.

The PVC has been associated with the PersistentVolume.

Task complete: You successfully configured persistent storage for an application running on Amazon EKS using Amazon EFS and the Amazon EFS CSI driver.

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.

```
root@prodcatalog-7fc4b697f6-r2hvp:/app# cat /products/products.txt
cat: /products/products.txt: No such file or directory
```

3.4 add a product in the actual application

Product Catalog

id | name | Add

Product ID	Product Name
1	EKS-Course

Catalog Detail

Vendors : ABC.com

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-

```
1 EKS-Course
```

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.

```
sh-4.2$ kubectl scale --replicas=2 deployment/prodcatalog -n workshop
deployment.apps/prodcatalog scaled
```

3.7 Check the status of creation of 2nd pod

```
sh-4.2$ kubectl get pods -n workshop -l app=prodcatalog
NAME                            READY   STATUS              RESTARTS   AGE
prodcatalog-7fc4b697f6-r2hvp    1/1     Running             0          99m
prodcatalog-97d647b69-6ntrt     0/1     ContainerCreating   0          21s
```

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:

```
cat /products/products.txt
```

output -

```
1 EKS-Course
```

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.

The original pod wrote the product *EKS-Course* with ID *1* to the Amazon EFS volume.
This data was persisted on Amazon EFS.
A new pod was created by scaling up the deployment.
The new pod mounts the same Amazon EFS volume.
Running *cat /products/products.txt* in the new pod shows that it can access the data written by the first pod.