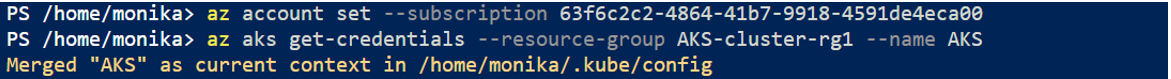
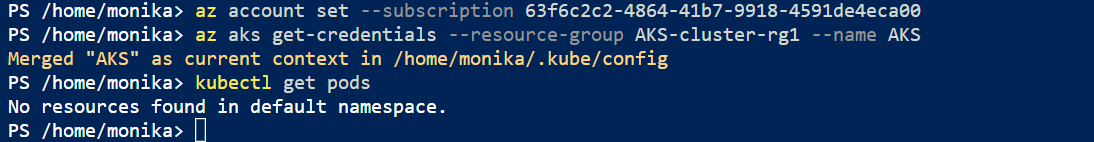
**Exercise: Storage**

**Practice 1: Direct provisioning of Azure File storage**

1. Login to Azure and connect to your AKS cluster.



2. Check if any pods run under the default namespace if so delete everything under the default namespace. 

3. In this practice we will directly provision Azure Files to a pod running inside AKS.

4. First create the Azure Files share. Run the following commands:

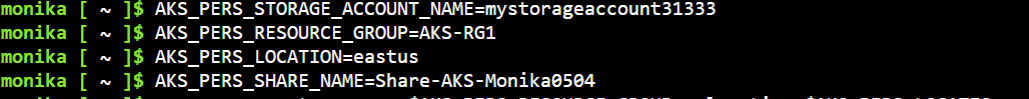
# Change these four parameters as needed for your own environment

AKS\_PERS\_STORAGE\_ACCOUNT\_NAME=mystorageaccount$RANDOM

AKS\_PERS\_RESOURCE\_GROUP= AKS-RG1

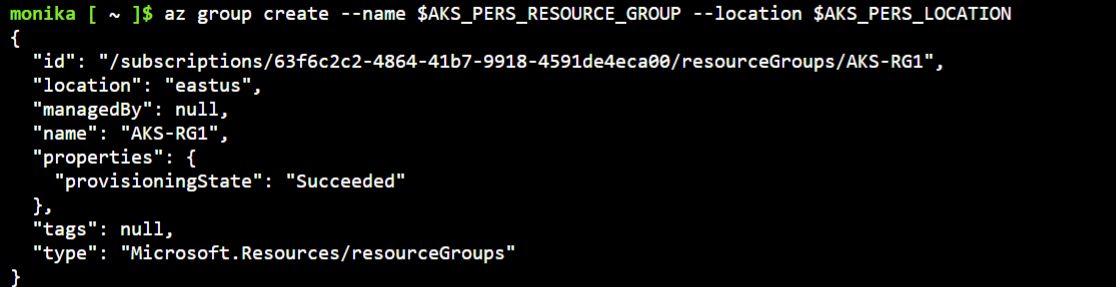
AKS\_PERS\_LOCATION=eastus

AKS\_PERS\_SHARE\_NAME=aksshare



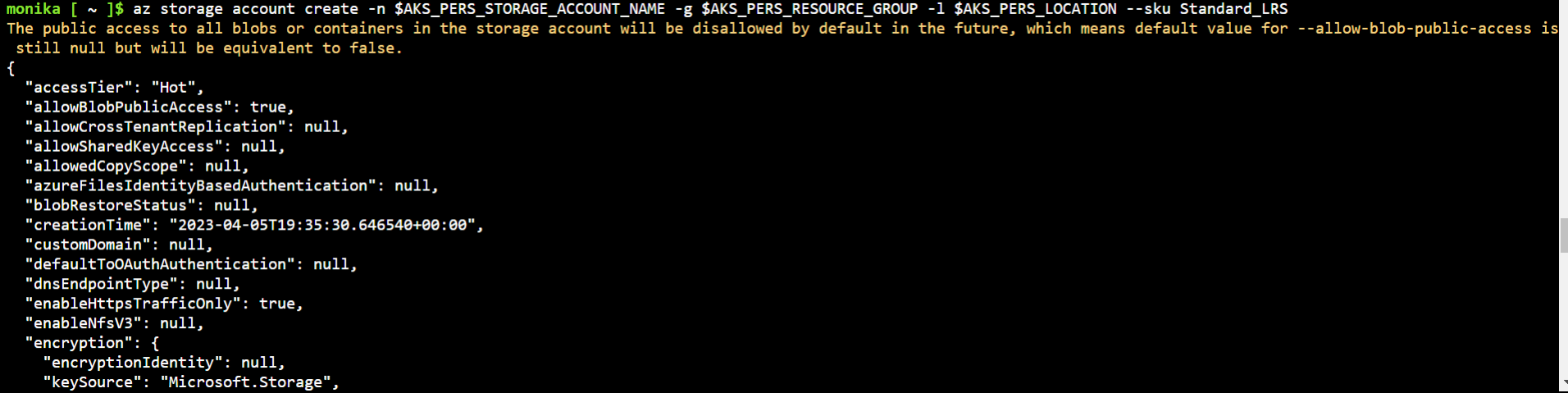
# Create a resource group

az group create --name $AKS\_PERS\_RESOURCE\_GROUP --location $AKS\_PERS\_LOCATION



# Create a storage account

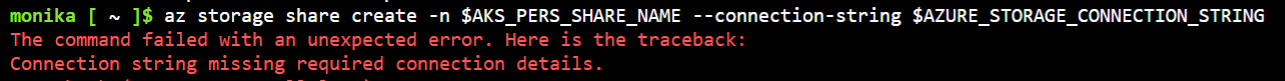
az storage account create -n $AKS\_PERS\_STORAGE\_ACCOUNT\_NAME -g $AKS\_PERS\_RESOURCE\_GROUP –l $AKS\_PERS\_LOCATION --sku Standard\_LRS



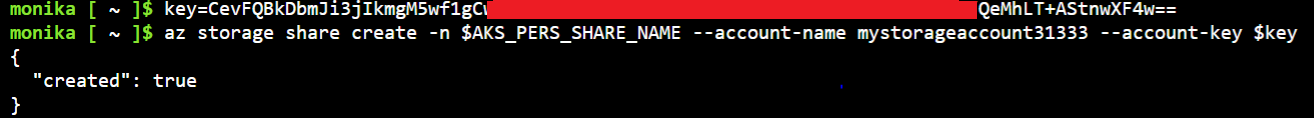
# Export the connection string as an environment variable, this is used when creating the Azure file share  
export AZURE\_STORAGE\_CONNECTION\_STRING=$(az storage account show-connection-string –n $AKS\_PERS\_STORAGE\_ACCOUNT\_NAME -g $AKS\_PERS\_RESOURCE\_GROUP -o tsv)



# Create the file share  
az storage share create -n $AKS\_PERS\_SHARE\_NAME --connection-string $AZURE\_STORAGE\_CONNECTION\_STRING



The provided command failed because of an unexpected error and I found a another solution to create the file share:

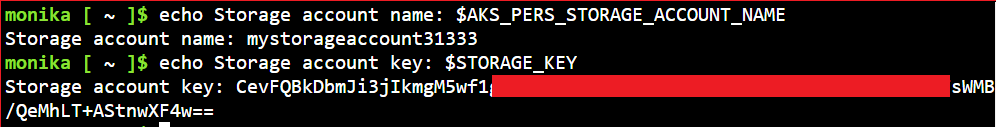


Instead of using the connection string, I used the account name and the account key from the storage account and the file share was successfully created

# Get storage account key  
STORAGE\_KEY=$(az storage account keys list --resource-group $AKS\_PERS\_RESOURCE\_GROUP --account-name $AKS\_PERS\_STORAGE\_ACCOUNT\_NAME --query "[0].value" -o tsv)



# Echo storage account name and key  
echo Storage account name: $AKS\_PERS\_STORAGE\_ACCOUNT\_NAME  
echo Storage account key: $STORAGE\_KEY



5. Make a note of the storage account name and key shown at the end of the script output. These values are needed when you create the Kubernetes volume in one of the following steps.

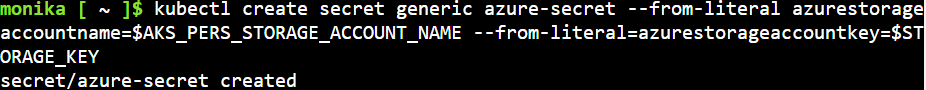
6. Now we will need to create a Kubernetes secret that will be used to mount the Az File Share to the pod. You need to hide this information from the pod’s definition and K8S secret is the best way to do it.

7. Run the following (single) command to create the secret:

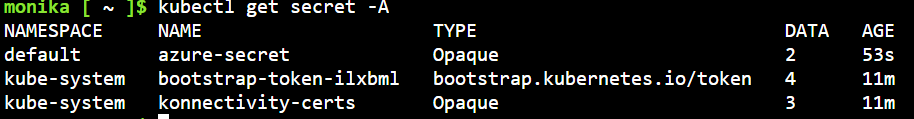
kubectl create secret generic azure-secret --from- \

literal=azurestorageaccountname=$AKS\_PERS\_STORAGE\_ACCOUNT\_NAME \

--from-literal=azurestorageaccountkey=$STORAGE\_KEY



8. Check if secret was created. Run kubectl get secret -A.



9. Now we can create the pod and mount the Azure File. Create a new file named azure-files-pod.yaml with the following contents:

apiVersion: v1

kind: Pod

metadata:

name: mypod

spec:

containers:

- image: mcr.microsoft.com/oss/nginx/nginx:1.15.5-alpine

name: mypod

resources:

requests:

cpu: 100m

memory: 128Mi

limits:

cpu: 250m

memory: 256Mi

volumeMounts:

- name: azure

mountPath: /mnt/azure

volumes:

- name: azure

azureFile:

secretName: azure-secret

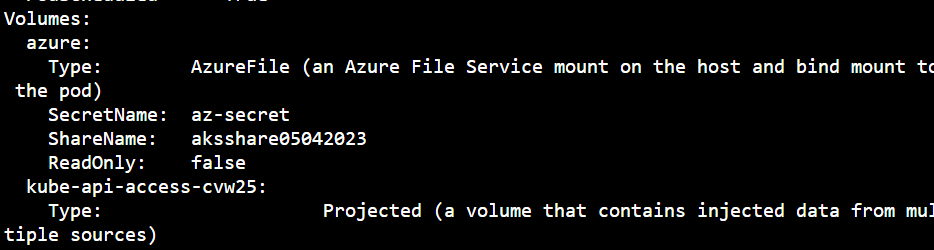
shareName: aksshare

readOnly: false

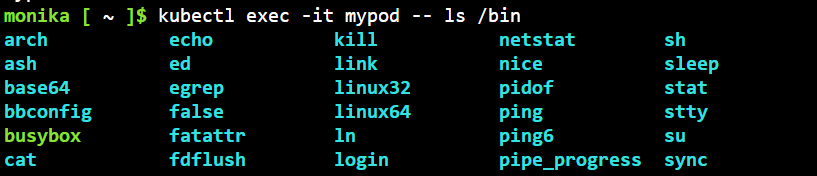
10. Run kubectl apply -f azure-files-pod.yaml.  
  
we are updating(applying) the pod that was created earlier to the azure file.

11. You now have a running pod with an Azure Files share mounted at /mnt/azure.

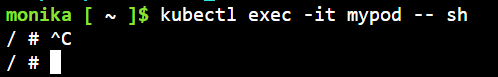
12. You can use **kubectl describe pod mypod** to verify the share is mounted successfully.  
 Search for the Volumes section of the output.



13. Now exec to the pod and try to access the mounted file share.  
 Run the following command kubectl exec -it mypod – bash

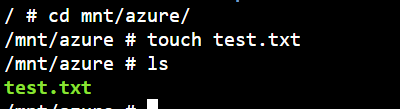


Exec starts a shell session inside containers running in your Kubernetes cluster



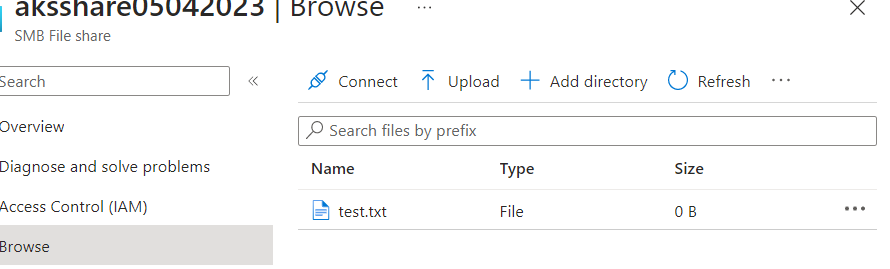
Bash was not installed on my container, so I logged in using the sh shell

14. Go to /mnt/azure and create a blank file test.txt file.



15. Go to the portal and locate your Azure storage provisioned for this practice.

16. Under the Files section, check the contents of the Azure file share and check if test.txt file exists.



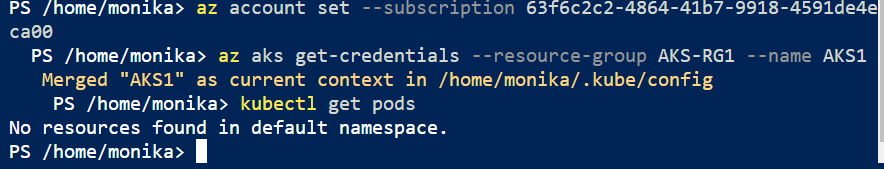
17. Delete the mypod. What happens to the Azure File share?

- After mypod is deleted the fileshare and the file still exist in the Azure Portal

**Practice 2: Provisioning Azure File storage using PVs and PVCs**

1. Login to Azure and connect to your AKS cluster.

2. Check if any pods run under the default namespace if so delete everything under the default namespace.



3. Now we will provision Azure files storage to a pod using PV(persistent volume) and PVC(persistent volume claim)

4. Create a azurefile-mount-options-pv.yaml file with a PersistentVolume like this:

apiVersion: v1

kind: PersistentVolume

metadata:

name: azurefile

spec:

capacity:

storage: 5Gi

accessModes:

- ReadWriteMany

azureFile:

secretName: azure-secret

shareName: aksshare

readOnly: false

mountOptions:

- dir\_mode=0777

- file\_mode=0777

- uid=1000

- gid=1000

- mfsymlinks

- nobrl

5. Note the access mode. Can you use other mode with Azure files?

Yes, we can use other access modes with Azure files such as ReadWriteMany, ReadWriteOnce, and ReadOnlyMany.

6. Now create a azurefile-mount-options-pvc.yaml file with a PersistentVolumeClaim that uses the

PersistentVolume like this:

apiVersion: v1

kind: PersistentVolumeClaim

metadata:

name: azurefile

spec:

accessModes:

- ReadWriteMany

storageClassName: ""

resources:

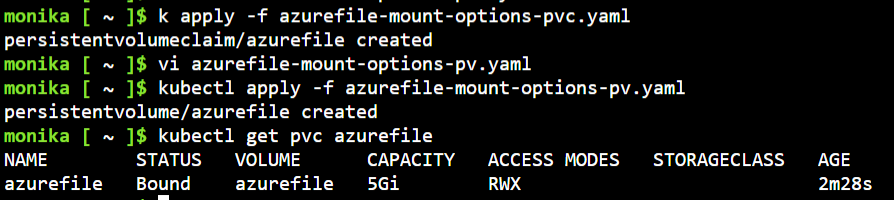
requests:

storage: 5Gi

7. Execute kubectl apply -f azurefile-mount-options-pv.yaml and kubectl apply -f azurefile-mount-optionspvc.yaml.

8. Verify your PersistentVolumeClaim is created and bound to the PersistentVolume.

Run kubectl get pvc azurefile.



9. Now we can embed the PVC info inside our pod definition.   
Create the following file azure-files-pod.yaml with following content:

apiVersion: v1

kind: Pod

metadata:

name: mypod

spec:

containers:

- image: mcr.microsoft.com/oss/nginx/nginx:1.15.5-alpine

name: mypod

resources:

requests:

cpu: 100m

memory: 128Mi

limits:

cpu: 250m

memory: 256Mi

volumeMounts:

- name: azure

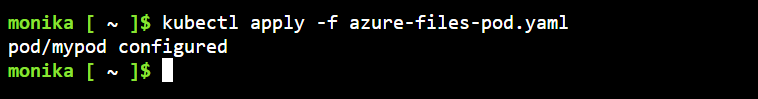
mountPath: /mnt/azure

volumes:

- name: azure

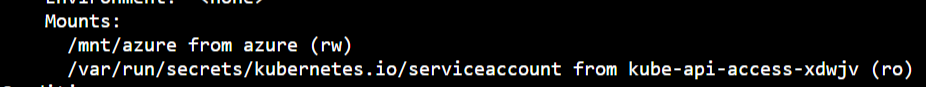
persistentVolumeClaim:

claimName: azurefile

10. Run kubectl apply -f azure-files-pod.yaml. 

11. You now have a running pod with an Azure Files share mounted at /mnt/azure.

12. You can use kubectl describe pod mypod to verify the share is mounted successfully. Search for the Volumes section of the output.



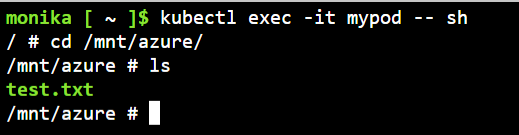
13. Now exec to the pod and try to access the mounted file share.

Run the following command kubectl exec -it mypod -- bash

14. Go to /mnt/azure and create a blank file test.txt file.

15. Go to the portal and locate your Azure storage provisioned for this practice.

16. Under the Files section, check the contents of the Azure file share and check if test.txt file exists.



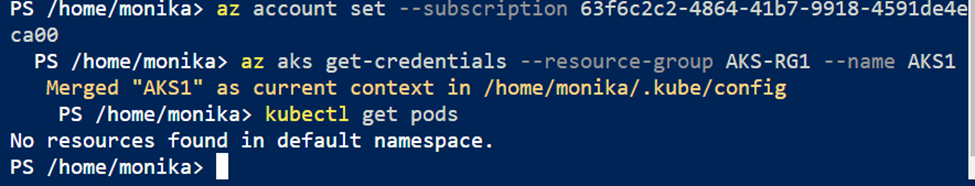
17. Delete the mypod the pv and pvc you have created so far. What happens to the Azure File share?

The file and the fileshare still exist

**Practice 3: Provisioning Azure file storage using Storage Classes**

1. Login to Azure and connect to your AKS cluster.

2. Check if any pods run under the default namespace if so delete everything under the default namespace.



3. Now we will provision file storage using the definition of storage classes. Create a file named azure-file-sc.yaml and copy in the following example manifest:

kind: StorageClass

apiVersion: storage.k8s.io/v1

metadata:

name: my-azurefile

provisioner: kubernetes.io/azure-file

mountOptions:

- dir\_mode=0777

- file\_mode=0777

- uid=0

- gid=0

- mfsymlinks

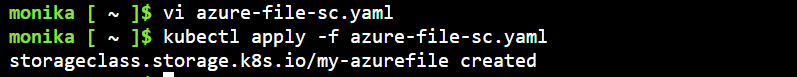
- cache=strict

- actimeo=30

parameters:

skuName: Standard\_LRS

4. Create the storage class with kubectl apply -f azure-file-sc.yaml .



5. Now we will create the PVC that will consume the storage class defined previously. Create a file named azurefile-pvc.yaml and copy in the following YAML:

apiVersion: v1

kind: PersistentVolumeClaim

metadata:

name: my-azurefile

spec:

accessModes:

- ReadWriteMany

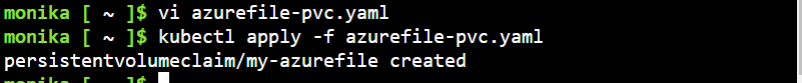
storageClassName: my-azurefile

resources:

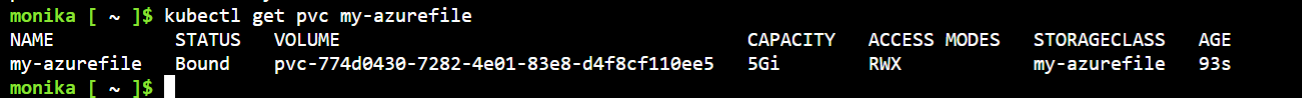
requests:

storage: 5Gi

6. Create the persistent volume claim with the kubectl apply -f azure-file-pvc.yaml.



7. Once completed, the file share will be created. A Kubernetes secret is also created that includes connection information and credentials. You can use the kubectl get pvc my-azurefile command to view the status of the PVC.



8. Now we will create the pod that consumes the PVC. Create a file named azure-pvc-files.yaml, and copy in the mvfollowing YAML. Make sure that the claimName matches the PVC created in the last step:

kind: Pod

apiVersion: v1

metadata:

name: mypod

spec:

containers:

- name: mypod

image: mcr.microsoft.com/oss/nginx/nginx:1.15.5-alpine

resources:

requests:

cpu: 100m

memory: 128Mi

limits:

cpu: 250m

memory: 256Mi

volumeMounts:

- mountPath: "/mnt/azure"

name: volume

volumes:

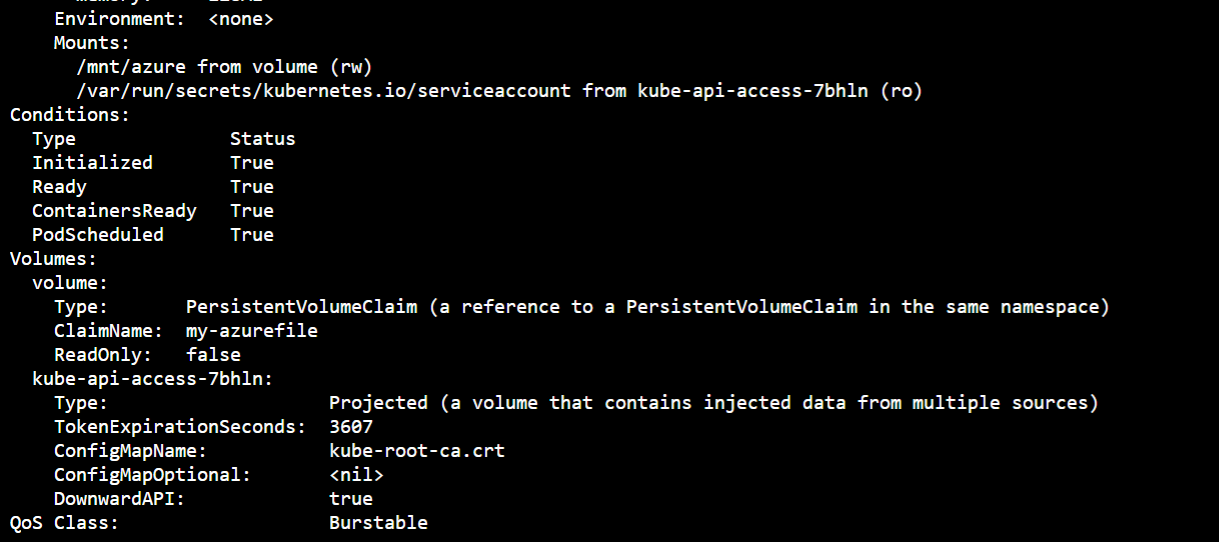
- name: volume

persistentVolumeClaim:

claimName: my-azurefile

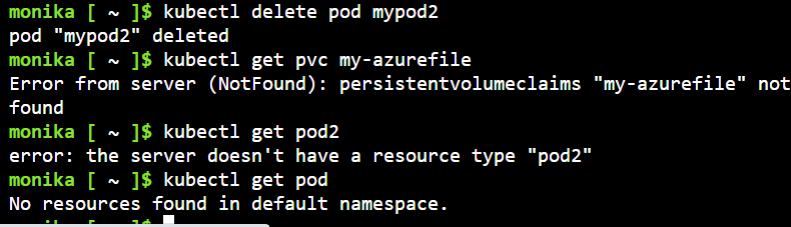
9. Create the pod with kubectl apply -f azure-pvc-files.yaml .



10. Do a describe on the pod and check the volumes mounted.

11. Delete everything created under this practice including the storage class.

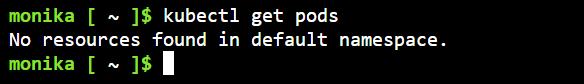




**Practice 4: Direct provisioning of Azure Disk storage**

1. Login to Azure and connect to your AKS cluster.

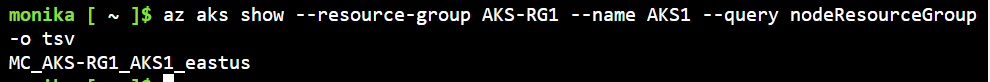
2. Check if any pods run under the default namespace if so delete everything under the default namespace.



3. In this practice we will directly provision Azure Disk to a pod running inside AKS.

4. First create the disk in the node resource group.

First, get the node resource group name with az aks show --resource-group myResourceGroup --name myAKSCluster --query nodeResourceGroup -o tsv .



5. Now create a disk using:

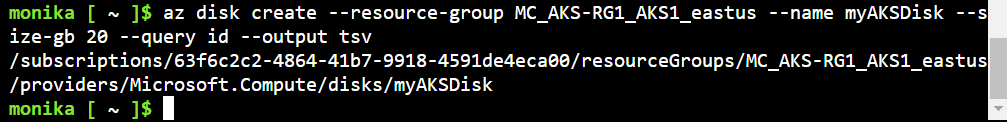
az disk create \

--resource-group MC\_myResourceGroup\_myAKSCluster\_eastus \

--name myAKSDisk \

--size-gb 20 \

--query id --output tsv



6. Make a note of the disk resource ID shown at the end of the script output. This value is needed when you create the Kubernetes volume in one of the following steps.

7. Now we can create the pod and mount the Azure Disk.  
 Create a new file named azure-disk-pod.yaml with the following contents:

apiVersion: v1

kind: Pod

metadata:

name: mypod

spec:

containers:

- image: mcr.microsoft.com/oss/nginx/nginx:1.15.5-alpine

name: mypod

resources:

requests:

cpu: 100m

memory: 128Mi

limits:

cpu: 250m

memory: 256Mi

volumeMounts:

- name: azure

mountPath: /mnt/azure

volumes:

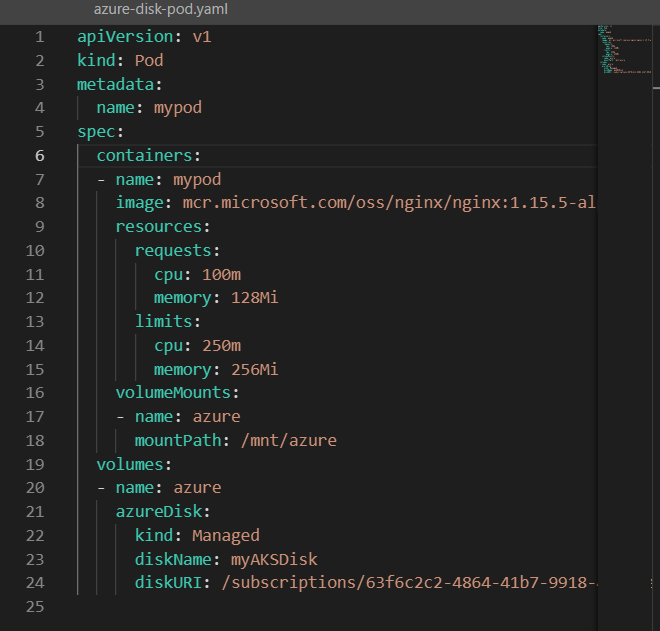
- name: azure

azureDisk:

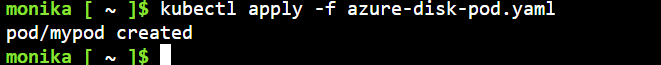
kind: Managed

diskName: myAKSDisk

diskURI: <!!!!!!!!!!!!! Put the Disk resource id noted before!!!>

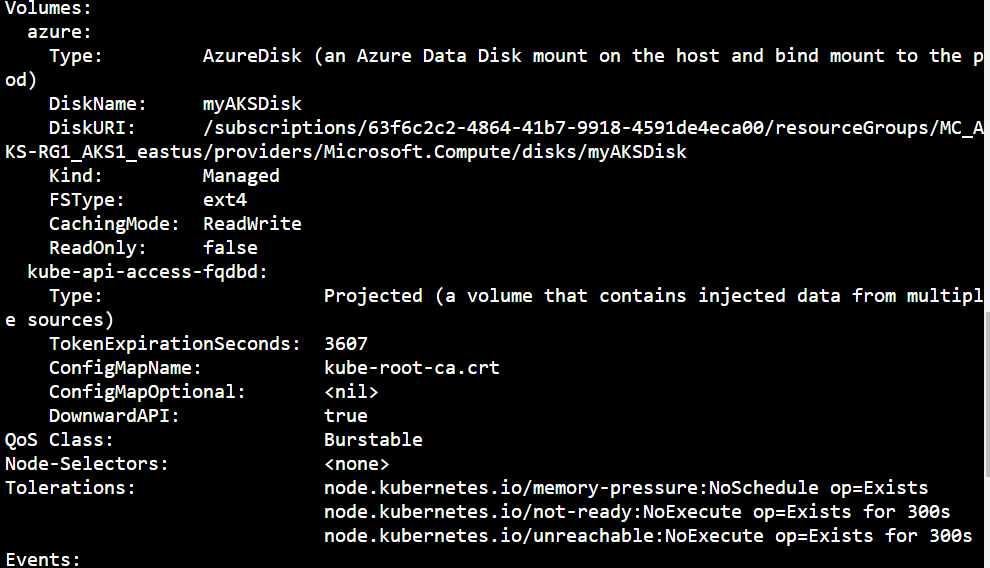


8. Run kubectl apply -f azure-disk-pod.yaml.



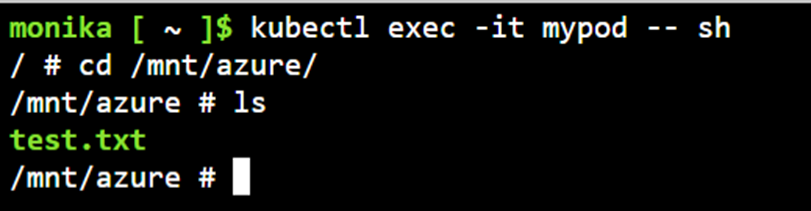
9. You now have a running pod with an Azure Disk mounted at /mnt/azure.

10. You can use kubectl describe pod mypod to verify the share is mounted successfully. Search for the Volumes section of the output.

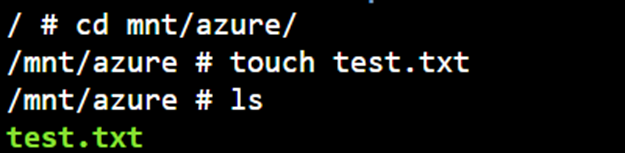


11. Now exec to the pod and try to access the mounted volume.

Run the following command kubectl exec -it mypod – bash



12. Go to /mnt/azure and try create a blank file test.txt file.



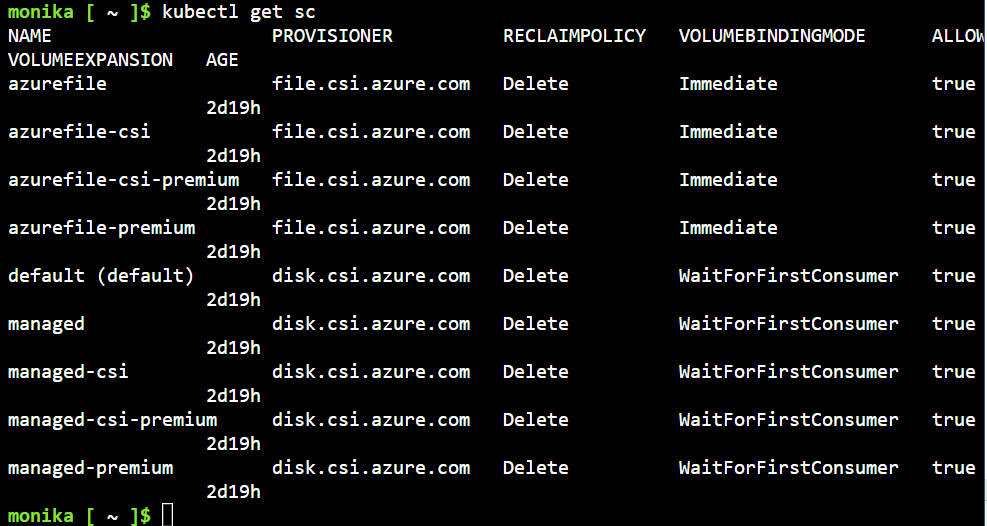
13. Delete everything created by this practice

**Practice 5: Provisioning Azure Disk storage using Storage Classes**

1. Login to Azure and connect to your AKS cluster.

2. Check if any pods run under the default namespace if so delete everything under the default namespace.

3. Now we will provision Azure disk and attach it to a running pod but this time using dynamic provisioning with storage classes. List the available storage classes, run **kubectl get sc.**



4. Examine the output. Each AKS cluster includes four pre-created storage classes, two of them configured to work with Azure disks, default and managed-premium. We will use the managed-premium in our PVC definition since it uses premium type of disks.

5. Now we will create the PVC that will consume the storage class defined previously. Create a file named azurepremium.yaml and copy in the following YAML:

apiVersion: v1

kind: PersistentVolumeClaim

metadata:

name: azure-managed-disk

spec:

accessModes:

- ReadWriteOnce

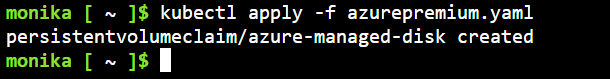
storageClassName: managed-premium

resources:

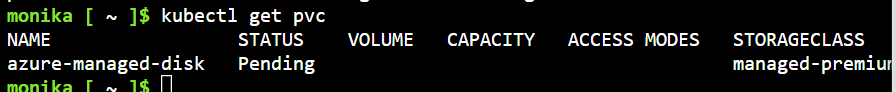
requests:

storage: 5Gi

6. Create the persistent volume claim with the kubectl apply -f azure-premium.yaml.



7. Check the status of your PVC.



8. Now we will create the pod that consumes the PVC. Create a file named azure-pvc-disk.yaml, and copy in the following YAML. Make sure that the claimName matches the PVC created in the last step:

kind: Pod

apiVersion: v1

metadata:

name: mypod

spec:

containers:

- name: mypod

image: mcr.microsoft.com/oss/nginx/nginx:1.15.5-alpine

resources:

requests:

cpu: 100m

memory: 128Mi

limits:

cpu: 250m

memory: 256Mi

volumeMounts:

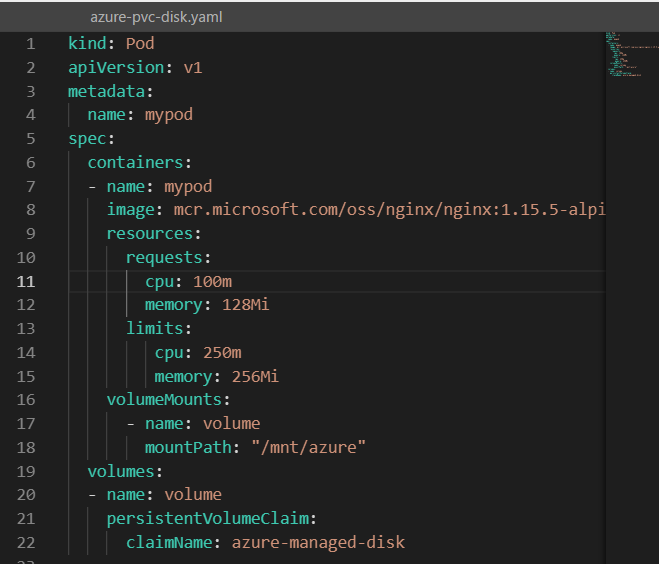
- mountPath: "/mnt/azure"

name: volume

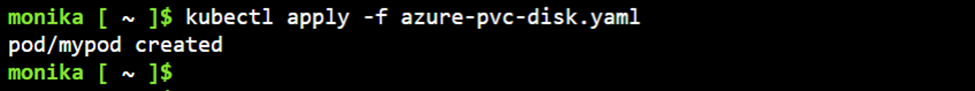
volumes:

- name: volume

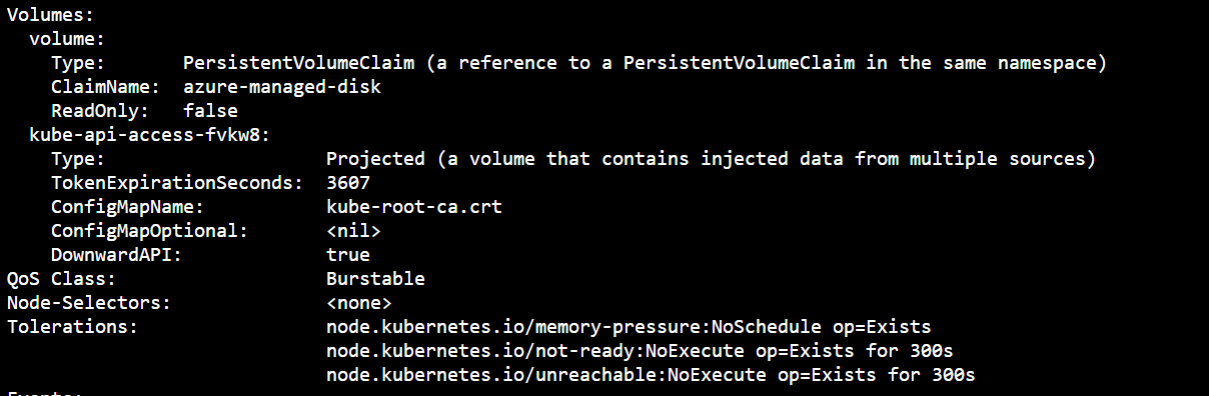
persistentVolumeClaim:

claimName: azure-managed-disk

9. Create the pod with kubectl apply -f azure-pvc-disk.yaml .



10. Do a describe on the pod and check the volumes mounted.



11. Delete everything created under this practice including the storage class.

