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. 3.In this practice we will directly provision Azure Files to a pod running inside AKS.4
4. First create the Azure Files share. Run the following commands:

First connect your azure clusster, using this commands

```
az account set --subscription df86697d-88bc-4474-899b-64b5dfd1d8cf
az aks get-credentials --resource-group rgLearn --name mkoAKS
```

Create a resource group

```
AKS_PERS_STORAGE_ACCOUNT_NAME=aksstorage1227
AKS_PERS_RESOURCE_GROUP=rgLearn1227
AKS_PERS_LOCATION="eastus"
AKS_PERS_SHARE_NAME=1227aksshare

az group create --name $AKS_PERS_RESOURCE_GROUP --location
$AKS_PERS_LOCATION
```

PROF

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

```
az storage account create -n $AKS_PERS_STORAGE_ACCOUNT_NAME -g
$AKS_PERS_RESOURCE_GROUP -l $AKS_PERS_LOCATION --sku Standard_LRS
```

```
darko [ ~ ]$ az group create --name $AKS_PERS_RESOURCE_GROUP --location $AKS_PERS_LOCATION
{
  "id": "/subscriptions/df86697d-88bc-4474-899b-64b5dfd1d8cf/resourceGroups/rgLearn",
  "location": "westeurope",
  "managedBy": null,
  "name": "rgLearn",
  "properties": {
    "provisioningState": "Succeeded"
  },
  "tags": null,
  "type": "Microsoft.Resources/resourceGroups"
}
darko [ ~ ]$
```

Export the connection string as an environment variable, this is used when creating the Azure file shareexport

```
export AZURE_STORAGE_CONNECTION_STRING=$(az storage account show-
connection-string -n $AKS_PERS_STORAGE_ACCOUNT_NAME -g
$AKS_PERS_RESOURCE_GROUP -otsv)
```

```
}
darko [ ~ ]$ az storage account create -n $AKS_PERS_STORAGE_ACCOUNT_NAME -g $AKS_PERS_RESOURCE_GROUP -l $AKS_PERS_LOCATION --sku Standard_LRS
The public access to all blobs or containers in the storage account will be disallowed by default in the future, which means default value for --allow-blob-public-access is still null but will
ent to false.
{
  "accessTier": "Hot",
  "allowBlobPublicAccess": true,
  "allowCrossTenantReplication": null,
  "allowSharedKeyAccess": null,
  "allowedCopyScope": null,
  "azureFilesIdentityBasedAuthentication": null,
  "blobRestoreStatus": null,
  "creationTime": "2023-04-06T10:22:56.327311+00:00",
  "customDomain": null,
  "defaultToOAuthAuthentication": null,
  "dnsEndpointType": null,
  "enableHttpsTrafficOnly": true,
  "enableNfsV3": null,
  "encryption": {
    "encryptionIdentity": null,
    "keySource": "Microsoft.Storage",
    "keyVaultProperties": null,
    "requireInfrastructureEncryption": null,
    "services": {
      "blob": {
        "enabled": true,
        "keyType": "Account",
        "lastEnabledTime": "2023-04-06T10:22:56.483582+00:00"
      },
      "file": {
```

Terminal container button

Create the file share

PROF

```
az storage share create -n$AKS_PERS_SHARE_NAME --connection-
string$AZURE_STORAGE_CONNECTION_STRING
```

```
darko [ ~ ]$ export AZURE_STORAGE_CONNECTION_STRING=$(az storage account show-connection-string -n $AKS_PERS_STORAGE_ACCOUNT_NAME -g $AKS_PERS_RESOURCE_GROUP -otsv)
darko [ ~ ]$
```

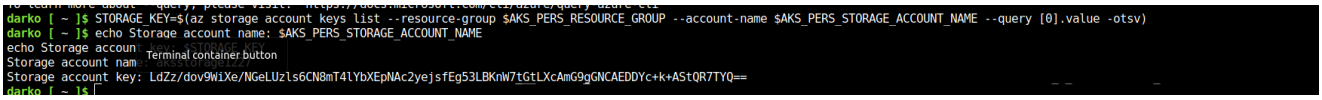
Get storage account key

```
STORAGE_KEY=$(az storage account keys list --resource-group
$AKS_PERS_RESOURCE_GROUP --account-name $AKS_PERS_STORAGE_ACCOUNT_NAME -
-query [0].value -otsv)
```

```
darko [ ~ ]$ STORAGE_KEY=$(az storage account keys list --resource-group $AKS_PERS_RESOURCE_GROUP --account-name $AKS_PERS_STORAGE_ACCOUNT_NAME --query [0].value -otsv)
darko [ ~ ]$
```

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

Create secret using this command

```
kubectl create secret generic azure-secret --from-
literal=aksstorage1227=$AKS_PERS_STORAGE_ACCOUNT_NAME --from-
literal=LdZz/dov9WiXe/NGeLUzls6CN8mT4lYbXEpNAc2yejsfEg53LBKnW7tGtLXcAmG9
gGNCAEDDYc+k+AStQR7TYQ===$STORAGE_KEY
```

Check if secret was created.

```
Run kubectl get secret -A.
```

PROF

After we created secret we can create the yaml file **azure-files-pod.yaml.**

```yaml
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
```

Upload newly created file and run to create new pod:

PROF

```
kubectl apply -f azure-files-pod.yaml.
```

```
darko [ ~ ]$ kubectl apply -f azure-files-pod.yaml
pod/mypod created
darko [ ~ ]$
```

Terminal container button

You can use kubectl describe pod mypod to verify the share is mounted successfully.

```
darko [ ~ ]$ kubectl describe pod mypod
Name:             mypod
Namespace:        default
Priority:         0
Service Account:  default
Node:             <none>
Labels:           <none>
Annotations:      <none>
Status:           Pending
IP:
IPs:              <none>
Containers:
  mypod:
    Image:      mcr.microsoft.com/oss/nginx/nginx:1.15.5-alpine
    Port:       <none>
    Host Port:  <none>
    Limits:
      cpu:     250m
      memory:  256Mi
    Requests:
      cpu:        100m
      memory:     128Mi
    Environment:  <none>
    Mounts:
      /mnt/azure from volume (rw)
      /var/run/secrets/kubernetes.io/serviceaccount from kube-api-access-z9gqs (ro)
Conditions:
  Type           Status
  PodScheduled   False
Volumes:
  volume:
    Type:       PersistentVolumeClaim (a reference to a PersistentVolumeClaim in the same namespace)
    ClaimName:  my-azurefile
    ReadOnly:   false
  kube-api-access-z9gqs:
    Type:                    Projected (a volume that contains injected data from multiple sources)
    TokenExpirationSeconds:  3607
    ConfigMapName:           kube-root-ca.crt
    ConfigMapOptional:       <nil>
    DownwardAPI:             true
QoS Class:                   Burstable
Node-Selectors:              <none>
Tolerations:                 node.kubernetes.io/memory-pressure:NoSchedule op=Exists
                             node.kubernetes.io/not-ready:NoExecute op=Exists for 300s
                             node.kubernetes.io/unreachable:NoExecute op=Exists for 300s
```

Terminal container button

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

Lets create new zurefile-mount-options-pv.yaml PersistentVolume and upload

DevOps Lab 22- K8 Storage > azurefile-mount-options-pv.yaml > ...

io.k8s.api.core.v1.PersistentVolume (v1@persistentvolume.json)

```yaml
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
```

```
kubectl apply -f azurefile-mount-options-pv.yaml
```

```
darko [ ~ ]$ kubectl apply -f azurefile-mount-options-pv.yaml
persistentvolume/azurefile created
darko [ ~ ]$
```

PROF

5. Note the access mode. Can you use other mode with Azure files?
6. Now create a azurefile-mount-options-pvc.yaml file with a PersistentVolumeClaim that uses the PersistentVolume like this:

```
EADME.md M ●   ! azurefile-mount-options-pvc.yaml U ×   ! azurefile-mount-options-pv.yaml U
Ops Lab 22- K8 Storage > ! azurefile-mount-options-pvc.yaml > {} spec > {} resources > {} requests
    io.k8s.api.core.v1.PersistentVolumeClaim (v1@persistentvolumeclaim.json)
1   apiVersion: v1
2   kind: PersistentVolumeClaim
3   metadata:
4     name: azurefile
5   spec:
6     accessModes:
7     - ReadWriteMany
8     volumeMode: Filesystem
9     storageClassName: ""
10    resources:
11     requests:
12      storage: 5Gi
13
```

```
kubectl apply -f azurefile-mount-options-pv.yaml
```

```
persistentvolume/azurefile created
darko [ ~ ]$ kubectl apply -f azurefile-mount-options-pvc.yaml
persistentvolumeclaim/azurefile created
darko [ ~ ]$
```

Mount the PersistentVolumeClaim

```
persistentvolumeclaim/azurefile created
darko [ ~ ]$ kubectl apply -f azurefile-mount-options-pvc.yaml
persistentvolumeclaim/azurefile unchanged
darko [ ~ ]$
```

Verify your PersistentVolumeClaim is created and bound to the PersistentVolume.

Run:

kubectl get pvc azurefile.

```
darko [ ~ ]$ kubectl get pvc azurefile
NAME        STATUS   VOLUME      CAPACITY   ACCESS MODES   STORAGECLASS   AGE
azurefile   Bound    azurefile   5Gi        RWX                           5m48s
darko [ ~ ]$
```

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

```
os Lab 22- K8 Storage > ! azure-files-pod.yaml > {} spec
  io.k8s.api.core.v1.Pod (v1@pod.json)
  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: azure
    volumes:
      - name: azure
        persistentVolumeClaim:
          claimName: azurefile
```

PROF

```
kubectl apply -f azure-files-pod.yaml.


kubectl describe pod mypod
```

Practice 3: Provisioning Azure file storage using Storage Classes

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:

```
darko [ ~ ]$ kubectl describe pod mypod
Name:             mypod
Namespace:        default
Priority:         0
Service Account:  default
Node:             <none>
Labels:           <none>
Annotations:      <none>
Status:           Pending
IP:
IPs:              <none>
Containers:
  mypod:
    Image:      mcr.microsoft.com/oss/nginx/nginx:1.15.5-alpine
    Port:       <none>
    Host Port:  <none>
    Limits:
      cpu:     250m
      memory:  256Mi
    Requests:
      cpu:        100m
      memory:     128Mi
    Environment:  <none>
    Mounts:
      /mnt/azure from volume (rw)
      /var/run/secrets/kubernetes.io/serviceaccount from kube-api-access-z9gqs (ro)
```

```
See 'kubectl apply -h' for help and examples
darko [ ~ ]$ kubectl apply -f azure-file-sc.yaml
storageclass.storage.k8s.io/my-azurefile created
darko [ ~ ]$
```

```
kubectl apply -f azure-file-sc.yaml
```

README.md M | azure-file-sc.yaml U | azure-files-pod.yaml

DevOps Lab 22- K8 Storage > azure-file-sc.yaml > {} parameters > skuName

io.k8s.api.storage.v1.StorageClass (v1@storageclass.json)

```yaml
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
```

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

```
README.md M    azure-file-pvc.yaml U ×    azure-file-sc.yaml U    azure-files-pod.yaml
DevOps Lab 22- K8 Storage > azure-file-pvc.yaml > {} spec > {} resources > {} requests > storage
io.k8s.api.core.v1.PersistentVolumeClaim (v1@persistentvolumeclaim.json)
1  apiVersion: v1
2  kind: PersistentVolumeClaim
3  metadata:
4    name: my-azurefile
5  spec:
6    accessModes:
7      - ReadWriteMany
8    storageClassName: my-azurefile
9    resources:
10     requests:
11         storage: 5Gi
```

```
kubectl apply -f azure-file-pvc.yaml
```

```
darko [ ~ ]$ kubectl apply -f azure-file-pvc.yaml
persistentvolumeclaim/my-azurefile created
darko [ ~ ]$
```

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.

```
kubectl get pvc my-azurefile
```

```
darko [ ~ ]$ kubectl apply -f azure-file-pvc.yaml
persistentvolumeclaim/my-azurefile created
darko [ ~ ]$ kubectl get pvc my-azurefile
NAME           STATUS   VOLUME                                     CAPACITY   ACCESS MODES   STORAGECLASS   AGE
my-azurefile   Bound    pvc-f2b1a058-2417-4748-a691-0718226a6500   5Gi        RWX            my-azurefile   54s
darko [ ~ ]$
```

```
README.md M    azure-pvc-files.yaml U ×    azure-file-pvc.yaml U    azure-file-sc.yaml U
vOps Lab 22- K8 Storage > azure-pvc-files.yaml > {} spec > [ ] volumes > {} 0 > {} persistentVolumeClaim >
io.k8s.api.core.v1.Pod (v1@pod.json)
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:
    - mountPath: "/mnt/azure"
      name: volume
  volumes:
    - name: volume
      persistentVolumeClaim:
        claimName: my-azurefile
```

PROF

```
darko [ ~ ]$ kubectl apply -f azure-file-pvc.yaml
persistentvolumeclaim/my-azurefile created
darko [ ~ ]$ kubectl get pvc my-azurefile
NAME           STATUS   VOLUME                                     CAPACITY   ACCESS MODES   STORAGECLASS   AGE
my-azurefile   Bound    pvc-f2b1a058-2417-4748-a691-0718226a6500   5Gi        RWX            my-azurefile   54s
darko [ ~ ]$ kubectl apply -f azure-pvc-files.yaml
pod/mypod configured
darko [ ~ ]$
```

Terminal

```
darko [ ~ ]$ az aks Terminal container button rgLearn --name mkoAKS --query nodeResourceGroup -o tsv
MC_rgLearn_mkoAKS_westeurope
darko [ ~ ]$
```

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

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 aks show --resource-group rgLearn --name mkoAKS --query nodeResourceGroup -o tsv

```
darko [ ~ ]$ az aks show --resource-group rgLearn --name mkoAKS --query nodeResourceGroup -o tsv
MC_rgLearn_mkoAKS_westeurope
darko [ ~ ]$
```

```
az disk create \
--resource-group MC_myResourceGroup_myAKSCluster_eastus \
--name myAKSDisk \
--size-gb 20 \
--query id --output tsv
```

```
darko [ ~ ]$ az disk create \
--resource-group MC_rgLearn_mkoAKS_westeurope \
--name myAKSDisk \
--size-gb 20 \
--query id --output tsv
/subscriptions/df86697d-88bc-4474-899b-64b5dfd1d8cf/resourceGroups/MC_rgLearn_mkoAKS_westeurope/providers/Microsoft.Compute/disks/myAKSDisk
darko [ ~ ]$
```

PROF

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

```
DevOps Lab 22- K8 Storage > ! azure-disk-pod.yaml > {} spec > [ ] volumes > {} 0 > {} azureDisk > diskURI
io.k8s.api.core.v1.Pod (v1@pod.json)
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:
    - mountPath: "/mnt/azure"
      name: azure
  volumes:
    - name: azure
      azureDisk:
        kind: Managed
        diskName: pvcRestored
        diskURI: /subscriptions/df86697d-88bc-4474-899b-64b5dfd1d8cf/resourceGroups/MC_rg
```

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

```
kubectl describe pod mypod
```

PROF

```
darko [ ~ ]$ kubectl describe pod mypod
Name:             mypod
Namespace:        default
Priority:         0
Service Account:  default
Node:             aks-agentpool-14801641-vmss000005/10.224.0.5
Start Time:       Fri, 07 Apr 2023 07:56:41 +0000
Labels:           <none>
Annotations:      <none>
Status:           Running
IP:               10.244.1.2
IPs:
  IP:  10.244.1.2
Containers:
  mypod:
    Container ID:   containerd://03adfa08ea7e00b0788f01980e2c183a5c43a6d67bb717
    Image:          mcr.microsoft.com/oss/nginx/nginx:1.15.5-alpine
    Image ID:       mcr.microsoft.com/oss/nginx/nginx@sha256:f84780a5ad654515bcd9ba2f35e20935e1246799f198683dd2c4f74d19ae9e5e
    Port:           <none>
    Host Port:      <none>
    State:          Running
      Started:      Fri, 07 Apr 2023 07:56:45 +0000
    Ready:          True
    Restart Count:  0
    Limits:
      cpu:     250m
      memory:  256Mi
    Requests:
      cpu:        100m
      memory:     128Mi
    Environment:  <none>
    Mounts:
      /mnt/azure from volume (rw)
      /var/run/secrets/kubernetes.io/serviceaccount from kube-api-access-z9gqs (ro)
Conditions:
  Type              Status
  Initialized       True
  Ready             True
  ContainersReady   True
  PodScheduled      True
Volumes:
  volume:
```

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

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

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 azure- premium.yaml and copy in the following YAM
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:

List the available storage classes, run kubectl get sc.

```
kubectl get sc
```

```
c failed: exec fa  Terminal container button  go:380: starting container process caused: exec: "bash": executable file not found in $PATH: unkr
darko [ ~ ]$ kube
NAME                    PROVISIONER                RECLAIMPOLICY   VOLUMEBINDINGMODE      ALLOWVOLUMEEXPANSION   AGE
azurefile               file.csi.azure.com         Delete          Immediate              true                   37h
azurefile-csi           file.csi.azure.com         Delete          Immediate              true                   37h
azurefile-csi-premium   file.csi.azure.com         Delete          Immediate              true                   37h
azurefile-premium       file.csi.azure.com         Delete          Immediate              true                   37h
default (default)       disk.csi.azure.com         Delete          WaitForFirstConsumer   true                   37h
managed                 disk.csi.azure.com         Delete          WaitForFirstConsumer   true                   37h
managed-csi             disk.csi.azure.com         Delete          WaitForFirstConsumer   true                   37h
managed-csi-premium     disk.csi.azure.com         Delete          WaitForFirstConsumer   true                   37h
managed-premium         disk.csi.azure.com         Delete          WaitForFirstConsumer   true                   37h
my-azurefile            kubernetes.io/azure-file   Delete          Immediate              false                  71m
darko [ ~ ]$
res    Highlight All   Match Case   Match Diacritics   Whole Words   7 of 16 matches
```

now create new file named azure-premium.yaml

```
kubectl apply -f azure-premium.yaml
```

```
darko [ ~ ]$ rm azure-premium.yaml
darko [ ~ ]$ kubectl apply -f azure-premium.yaml
persistentvolumeclaim/azure-managed-disk created
darko [ ~ ]$
res    Highlight All   Match Case   Match Diacritics   Whole Words   7 of 16 matches
```

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:

PROF

azure-pvc-disk.yaml

```
Ops Lab 22- K8 Storage > ! azure-pvc-disk.yaml > [ ] containers
    io.k8s.api.core.v1.Pod (v1@pod.json)
1   apiVersion: v1
2   kind: Pod
3   metadata:
4     name: mypod
5   spec:
6   containers:
7     - name: mypod
8       image: mcr.microsoft.com/oss/nginx/nginx:1.15.5-alpine
9   resources:
0     requests:
1       cpu: 100m
2       memory: 128Mi
```

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.