

Exercise: Storage

Kubernetes is a free and open-source container orchestration platform. It provides services and management capabilities needed to efficiently deploy, operate, and scale containers in a cloud or cluster environment.

When managing containerized environments, Kubernetes storage is useful for storage administrators, because it allows them to maintain multiple forms of persistent and non-persistent data in a Kubernetes cluster. This makes it possible to create dynamic storage resources that can serve different types of applications.

Practice 1: Direct provisioning of Azure File storage

Note: Try not to do a copy/paste on commands requests unless you are instructed to do so.

Copy/paste will not help you to learn Kubernetes!

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

```
PS /home/vladimir> az account set --subscription c031718c-8f31-4c8d-ae56-634904ec50f9
PS /home/vladimir> az aks get-credentials --resource-group testStorage --name cluster1
Merged "cluster1" as current context in /home/vladimir/.kube/config
```

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

```
PS /home/vladimir> kubectl get pods --namespace=default
No resources found in 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=myAKSShare

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

```
PS /home/vladimir> az group create --name $AKS_PERS_RESOURCE_GROUP --location $AKS_PERS_LOCATION
{
  "id": "/subscriptions/c031718c-8f31-4c8d-ae56-634904ec50f9/resourceGroups/myAKSShare",
  "location": "eastus",
  "managedBy": null,
  "name": "myAKSShare",
  "properties": {
    "provisioningState": "Succeeded"
  },
  "tags": null,
  "type": "Microsoft.Resources/resourceGroups"
}
```

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`

```
PS /home/vladimir> 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 --all
{
  "accessTier": "Hot",
  "allowBlobPublicAccess": true,
  "allowCrossTenantReplication": null,
  "allowSharedKeyAccess": null,
  "allowedCopyScope": null,
  "azureFilesIdentityBasedAuthentication": null,
  "blobRestoreStatus": null,
  "creationTime": "2023-04-06T08:50:53.787577+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": "2023-04-06T08:50:53.960170+00:00"
      }
    }
  },
  "file": {
```

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`

```
PS /home/vladimir> az storage share create -n $AKS_PERS_SHARE_NAME --connection-string $AZURE_STORAGE_CONNECTION_STRING
{
  "created": true
}
```

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`

```
PS /home/vladimir> echo Storage account name: $AKS_PERS_STORAGE_ACCOUNT_NAME
Storage
account
name:
aksstorage321
PS /home/vladimir> echo Storage account key: $STORAGE_KEY
Storage
account
key:
UnhTm+hju6pXAmQ0pDlToh3u5007HCpPruoUC/KyFktvNaBu4sSSxcLXSEIJGD2S+cJ4vfGrUjUE+AstJVkdeQ==
```

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.

Storage account name: aksstorage321

Storage account key:

UnhTm+hju6pXAmQOpDItoH3u5007HCpPruoUC/KyFktvNaBu4sSSxcLXSEIJGD2S+cJ4vfGrUjUE+AStJvkdeQ==

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

```
PS /home/vladimir> kubectl create secret generic azure-secret --from-literal=azurestorageaccountname=$AKS_PERS_STORAGE_ACCOUNT_NAME --from-literal=azurestorageaccountkey=$STORAGE_KEY
secret/azure-secret created
```

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

```
PS /home/vladimir> kubectl get secret -A
```

NAMESPACE	NAME	TYPE	DATA	AGE
default	azure-secret	Opaque	2	43s
kube-system	bootstrap-token-wsyau4	bootstrap.kubernetes.io/token	4	30m
kube-system	konnnectivity-certs	Opaque	3	30m

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**.

```
PS /home/vladimir> kubectl apply -f azure-files-pod.yaml
pod/mypod created
```

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.

```
Volumes:
  azure:
    Type:          AzureFile (an Azure File Service mount on the host and bind mount to the pod)
    SecretName:    azure-secret
    ShareName:     aksshare
    ReadOnly:      false
  kube-api-access-vcmgb:
    Type:          Projected (a volume that contains injected data from multiple sources)
    TokenExpirationSeconds: 3607
    ConfigMapName:  kube-root-ca.crt
    ConfigMapOptional: <nil>
    DownwardAPI:    true
```

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

```
PS /home/vladimir> kubectl exec -it mypod -- bash
error: Internal error occurred: error executing command in container: failed to exec in container: failed to start e
xec "129bac8177bb73c49e07b59a71e4d990617df40157db910b6d66dcadb6ad2d57": OCI runtime exec failed: exec failed: contai
ner_linux.go:380: starting container process caused: exec: "bash": executable file not found in $PATH: unknown
PS /home/vladimir> kubectl exec -it mypod -- sh
/ # ls
```

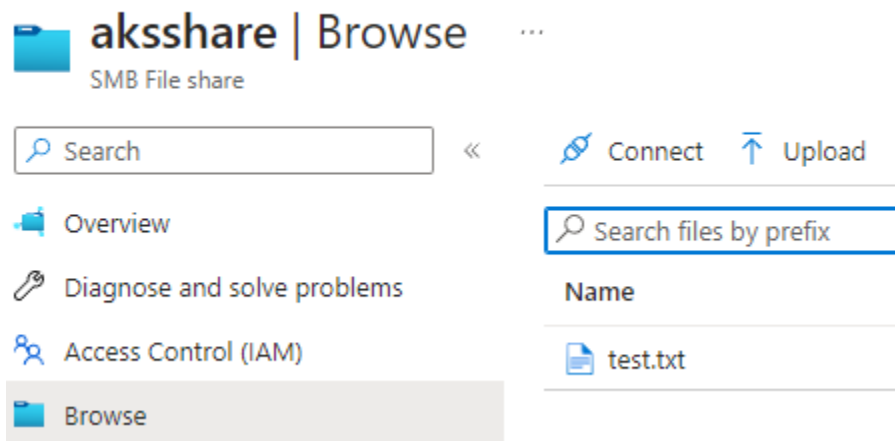
14. Go to /mnt/azure and create a blank file test.txt file.

```
/ # cd /mnt/azure
/mnt/azure # touch test.txt
/mnt/azure # ls
test.txt
/mnt/azure #
```

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.

[Home](#) > [Storage accounts](#) > [aksstorage321](#) | [File shares](#) > [aksshare](#)



aksshare | Browse ...

SMB File share

Search

Connect Upload

Overview

Diagnose and solve problems

Access Control (IAM)

Browse

Search files by prefix

Name
test.txt

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

```
PS /home/vladimir> kubectl delete pod mypod
pod "mypod" deleted
```

When I deleted the pod, the Azure File share was not deleted. The data stored in the file share persisted even after the pod was deleted. This is because the Azure File share is a separate storage resource that is not tied to the pod. If I create a new pod and mount the same Azure File share, I will be able to access the same data that was stored in the file share before the previous pod was deleted.

Practice 2: Provisioning Azure File storage using PVs and PVCs

Note: Try not to do a copy/paste on commands requests unless you are instructed to do so.

Copy/paste will not help you to learn Kubernetes!

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:

```
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?

Azure Files only supports ReadWriteMany access mode, which means that multiple pods can mount and read/write to the same file share simultaneously.

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**.

```
PS /home/vladimir> kubectl apply -f azurefile-mount-options-pv.yaml
persistentvolume/azurefile created
PS /home/vladimir> kubectl apply -f azurefile-mount-options-pvc.yaml
persistentvolumeclaim/azurefile created
```

8. Verify your PersistentVolumeClaim is created and bound to the PersistentVolume. Run **kubectl get pvc azurefile**.

```
PS /home/vladimir> kubectl get pvc azurefile
NAME          STATUS    VOLUME   CAPACITY   ACCESS MODES   STORAGECLASS   AGE
azurefile     Bound    azurefile  5Gi        RWX              azurefile      32s
```

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**.

```
PS /home/vladimir> kubectl apply -f azure-files-pod.yaml
pod/mypod created
```

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.

```
Volumes:
  azure:
    Type:          PersistentVolumeClaim (a reference to a PersistentVolumeClaim in the same namespace)
    ClaimName:     azurefile
    ReadOnly:      false
  kube-api-access-zkc2l:
    Type:          Projected (a volume that contains injected data from multiple sources)
    TokenExpirationSeconds: 3607
    ConfigMapName:  kube-root-ca.crt
    ConfigMapOptional: <nil>
    DownwardAPI:    true
```

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

```
PS /home/vladimir> kubectl exec -it mypod -- sh
/ #
```

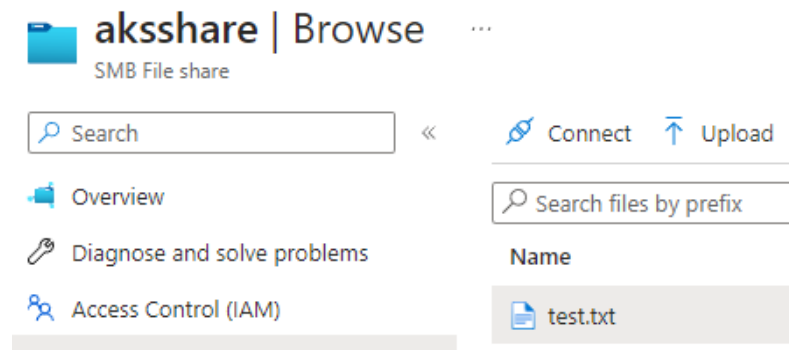
14. Go to /mnt/azure and create a blank file test.txt file.

```
/ # touch /mnt/azure/test.txt
/ # ls /mnt/azure/
test.txt
/ #
```

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.

[Home](#) > [Storage accounts](#) > [aksstorage321](#) | [File shares](#) > [aksshare](#)



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

```
PS /home/vladimir> kubectl delete --all pods
pod "mypod" deleted

PS /home/vladimir> kubectl delete pv azurefile
persistentvolume "azurefile" deleted

PS /home/vladimir> kubectl delete pvc azurefile
persistentvolumeclaim "azurefile" deleted
```

The data stored in the file share will persist even if the Kubernetes resources are deleted.

Practice 3: Provisioning Azure file storage using Storage Classes

Note: Try not to do a copy/paste on commands requests unless you are instructed to do so.

Copy/paste will not help you to learn Kubernetes!

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** .

```
PS /home/vladimir> kubectl apply -f azure-file-sc.yaml
storageclass.storage.k8s.io/my-azurefile created
```

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 azurefile-pvc.yaml**.

```
PS /home/vladimir> kubectl apply -f azurefile-pvc.yaml
persistentvolumeclaim/my-azurefile created
```

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.

```
PS /home/vladimir> kubectl get pvc my-azurefile
```

NAME	STATUS	VOLUME	CAPACITY	ACCESS MODES	STORAGECLASS	AGE
my-azurefile	Bound	pvc-b8638bca-6a7e-4871-b760-2a14b8916f49	5Gi	RWX	my-azurefile	51s

8. Now we will create the pod that consumes the PVC. Create a file named azure-pvc-files.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: 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.

```
Volumes:
  volume:
    Type:      PersistentVolumeClaim (a reference to a PersistentVolumeClaim in the same namespace)
    ClaimName: my-azurefile
    ReadOnly:  false
  kube-api-access-nzkpn:
    Type:      Projected (a volume that contains injected data from multiple sources)
    TokenExpirationSeconds: 3607
    ConfigMapName: kube-root-ca.crt
    ConfigMapOptional: <nil>
    DownwardAPI: true
```

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

```
PS /home/vladimir> kubectl delete pod mypod
pod "mypod" deleted

PS /home/vladimir> kubectl delete sc my-azurefile
storageclass.storage.k8s.io "my-azurefile" deleted

PS /home/vladimir> kubectl delete pvc my-azurefile
persistentvolumeclaim "my-azurefile" deleted
```

Practice 4: Direct provisioning of Azure Disk storage

Note: Try not to do a copy/paste on commands requests unless you are instructed to do so.

Copy/paste will not help you to learn Kubernetes!

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** .

```
PS /home/vladimir> az aks show --resource-group testStorage --name cluster1 --query nodeResourceGroup -o tsv
MC_testStorage_cluster1_westeurope
PS /home/vladimir>
```

5. Now create a disk using:

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

```
PS /home/vladimir> az disk create --resource-group MC_testStorage_cluster1_westeurope --name myAKSDisk --size-gb 20 --query id --output tsv
/subscriptions/c031718c-8f31-4c8d-ae56-634904ec50f9/resourceGroups/MC_testStorage_cluster1_westeurope/providers/Microsoft.Compute/disks/myAKSDisk
PS /home/vladimir>
```

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.

```
/subscriptions/c031718c-8f31-4c8d-ae56-634904ec50f9/resourceGroups/MC_testStorage_cluster1_westeurope/providers/Microsoft.Compute/disks/
myAKSDisk
```

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: /subscriptions/c031718c-8f31-4c8d-ae56-634904ec50f9/resourceGroups/MC_testStorage_cluster1_westeurope/providers/Microsoft.Compute/disks/myAKSdisk
```

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

```
PS /home/vladimir> kubectl apply -f azure-disk-pod.yaml
pod/mypod created
```

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.

```
Volumes:
  azure:
    Type:          AzureDisk (an Azure Data Disk mount on the host and bind mount to the pod)
    DiskName:      myAKSDisk
    DiskURI:       /subscriptions/c031718c-8f31-4c8d-ae56-634904ec50f9/resourceGroups/MC_testStorage_cluster1_westeurope/providers/Microsoft.Compute/disks/myAKSdisk
    Kind:          Managed
    FSType:         ext4
    CachingMode:   ReadWrite
    ReadOnly:      false
  kube-api-access-sg285:
    Type:          Projected (a volume that contains injected data from multiple sources)
    TokenExpirationSeconds: 3607
    ConfigMapName:  kube-root-ca.crt
    ConfigMapOptional: <nil>
    DownwardAPI:    true
```

11. Now exec to the pod and try to access the mounted volume. Run the following command **kubectl exec -it mypod -- bash**

```
PS /home/vladimir> kubectl exec -it mypod -- sh
/ #
```

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

```
/mnt/azure # touch test.txt
/mnt/azure # ls
lost+found  test.txt
/mnt/azure #
```

13. Delete everything created by this practice.

```
PS /home/vladimir> rm azure-disk-pod.yaml
PS /home/vladimir> rm azurefile-pvc.yaml
PS /home/vladimir> rm azurefile-sc.yaml
/usr/bin/rm: cannot remove 'azurefile-sc.yaml': No such file or directory
PS /home/vladimir> rm azure-file-sc.yaml
PS /home/vladimir> rm azure-pvc-files.yaml
PS /home/vladimir> ls
clouddrive  Microsoft
PS /home/vladimir>
```

Practice 5: Provisioning Azure Disk storage using Storage Classes

Note: Try not to do a copy/paste on commands requests unless you are instructed to do so.

Copy/paste will not help you to learn Kubernetes!

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**.

```
PS /home/vladimir> kubectl get sc
```

NAME	PROVISIONER	RECLAIMPOLICY	VOLUMEBINDINGMODE	ALLOWVOLUMEEXPANSION	AGE
azurefile	file.csi.azure.com	Delete	Immediate	true	6h53m
azurefile-csi	file.csi.azure.com	Delete	Immediate	true	6h53m
azurefile-csi-premium	file.csi.azure.com	Delete	Immediate	true	6h53m
azurefile-premium	file.csi.azure.com	Delete	Immediate	true	6h53m
default (default)	disk.csi.azure.com	Delete	WaitForFirstConsumer	true	6h53m
managed	disk.csi.azure.com	Delete	WaitForFirstConsumer	true	6h53m
managed-csi	disk.csi.azure.com	Delete	WaitForFirstConsumer	true	6h53m
managed-csi-premium	disk.csi.azure.com	Delete	WaitForFirstConsumer	true	6h53m
managed-premium	disk.csi.azure.com	Delete	WaitForFirstConsumer	true	6h53m

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**.

```
PS /home/vladimir> kubectl apply -f azurepremium.yaml
persistentvolumeclaim/azure-managed-disk created
```

7. Check the status of your PVC.

```
PS /home/vladimir> kubectl get pvc
```

NAME	STATUS	VOLUME	CAPACITY	ACCESS MODES	STORAGECLASS	AGE
azure-managed-disk	Pending				managed-premium	21s

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`**.

```
PS /home/vladimir> kubectl apply -f azure-pvc-disk.yaml
pod/mypod created
```

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

```
Volumes:
  volume:
    Type:      PersistentVolumeClaim (a reference to a PersistentVolumeClaim in the same namespace)
    ClaimName: azure-managed-disk
    ReadOnly:  false
  kube-api-access-nng56:
    Type:      Projected (a volume that contains injected data from multiple sources)
    TokenExpirationSeconds: 3607
    ConfigMapName: kube-root-ca.crt
    ConfigMapOptional: <nil>
    DownwardAPI: true
```

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

```
PS /home/vladimir> kubectl delete pod mypod
pod "mypod" deleted
PS /home/vladimir> kubectl delete pvc azure-managed-disk
persistentvolumeclaim "azure-managed-disk" deleted
PS /home/vladimir>
```