Kubernetes Storage

Practice 1: Direct provisioning of Azure File storage

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

```
PS /home/andrijana> az account set --subscription 836f56df-cca0-4866-b552-adbe26a742da
PS /home/andrijana> az aks get-credentials --resource-group myAKSShare --name myCluster
```

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=myAKSShare
AKS_PERS_LOCATION=eastus AKS_PERS_SHARE_NAME=aksshare

```
PS /home/andrijana> $AKS_PERS_STORAGE_ACCOUNT_NAME='mystorageAKS'
PS /home/andrijana> $AKS_PERS_RESOURCE_GROUP='myAKSShare'
PS /home/andrijana> $AKS_PERS_LOCATION='eastus'
PS /home/andrijana> $AKS_PERS_SHARE_NAME='aksshare'
```

- Later on, I changed the storage account name because it has to be with lowercase letters and numbers, and I've used uppercase.

```
PS /home/andrijana> $AKS_PERS_STORAGE_ACCOUNT_NAME='mystorageaks77'
```

Create a resource group
az group create --name $AKS_PERS_RESOURCE_GROUP --location $AKS_PERS_LOCATION

```
PS /home/andrijana> az group create --name $AKS_PERS_RESOURCE_GROUP --location $AKS_PERS_LOCATION
{
  "id": "/subscriptions/836f56df-cca0-4866-b552-adbe26a742da/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/andrijana> 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 be equivalent to fals
{
  "accessTier": "Hot",
  "allowBlobPublicAccess": true,
  "allowCrossTenantReplication": null,
  "allowSharedKeyAccess": null,
  "allowedCopyScope": null,
  "azureFilesIdentityBasedAuthentication": null,
  "blobRestoreStatus": null,
  "creationTime": "2023-04-09T08:23:27.768615+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": "2023-04-09T08:23:27.924828+00:00"
      },
      "file": {
        "enabled": true,
        "keyType": "Account",
        "lastEnabledTime": "2023-04-09T08:23:27.924828+00:00"
      },
      "queue": null,
```

Terminal container button

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)

```
PS /home/andrijana> $AZURE_STORAGE_CONNECTION_STRING = az storage account show-connection-string -n $AKS_PERS_STORAGE_ACCOUNT_NAME -g $AKS_PERS_RESOURCE_GROUP -o tsv
PS /home/andrijana>
```

Create the file share
az storage share create -n $AKS_PERS_SHARE_NAME --connection-string $AZURE_STORAGE_CONNECTION_STRING

```
PS /home/andrijana> 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)

```
PS /home/andrijana> $STORAGE_KEY=$(az storage account keys list --resource-group $AKS_PERS_RESOURCE_GROUP --account-name $AKS_PERS_STORAGE_ACCOUNT_NAME --query "[0].value" -o tsv)
PS /home/andrijana>
```

Echo storage account name and key

echo Storage account name: $AKS_PERS_STORAGE_ACCOUNT_NAME

```
PS /home/andrijana> echo Storage account name: $AKS_PERS_STORAGE_ACCOUNT_NAME
Storage
account
name:
mystorageaks77
```

echo Storage account key: $STORAGE_KEY

```
PS /home/andrijana> echo Storage account key: $STORAGE_KEY
Storage
account
key:
[redacted]/otTyZST8zFMC+ASt9ikN+A==
```

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

```
PS /home/andrijana> 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/andrijana> kubectl get secret -A
NAMESPACE     NAME                    TYPE                             DATA   AGE
default       azure-secret            Opaque                           2      22s
kube-system   ama-logs-secret         Opaque                           2      63m
kube-system   bootstrap-token-ptse59  bootstrap.kubernetes.io/token    4      63m
kube-system   konnectivity-certs      Opaque                           3      63m
```

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:

```
io.k8s.api.core.v1.Pod (v1@pod.json)
apiVersion: v1
kind: Pod
metadata:
  name: mypod
spec:
  containers:
    - name: mypod
      image: :mcr.microsoft.com/oss/nginx/nginx:1.15.5-alpine
      resources:
        requests:
          memory: "128Mi"
          cpu: "100m"
        limits:
          memory: "256Mi"
          cpu: "250m"
      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/andrijana> 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.

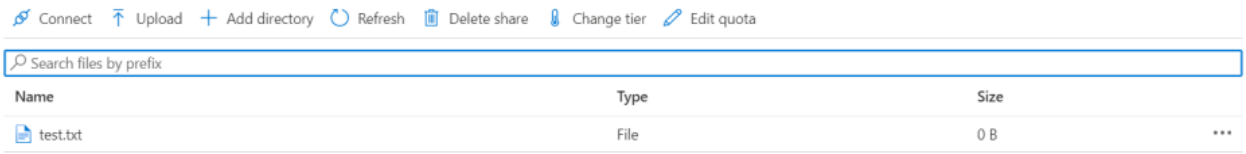
12. You can use kubectl describe pod mypod to verify the share is mounted successfully. Search for the Volumes section of the output.

```
PS /home/andrijana> kubectl describe pod mypod
Name:             mypod
Namespace:        default
Priority:         0
Service Account:  default
Node:             aks-agentpool-32754409-vmss000000/10.224.0.4
Start Time:       Sun, 09 Apr 2023 08:57:17 +0000
Labels:           <none>
Annotations:      <none>
Status:           Running
IP:               10.244.1.19
IPs:
  IP:  10.244.1.19
Containers:
  mypod:
    Container ID:   containerd://17f0004c203aaa9848d4211b86d79da234977919a3c4f1490149f8e7f72993a9
    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
```

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.
ku
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?
    - The file still stays in the Azure File share

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:

```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
```

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

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:

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

```
rks > Homework 18 - Kubernetes > ! azurefile-mount-options-p
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: azurefile
spec:
  accessModes:
    - ReadWriteMany
  storageClassName: azurefile-csi
  volumeName: azurefile
  resources:
    requests:
      storage: 5Gi
```

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

```
PS /home/andrijana> kubectl get pvc azurefile
NAME        STATUS    VOLUME      CAPACITY   ACCESS MODES   STORAGECLASS    AGE
azurefile   Pending   azurefile   0                         azurefile-csi   3m26s
PS /home/andrijana>
```

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:
  - name: mypod
    image: :mcr.microsoft.com/oss/nginx/nginx:1.15.5-alpine
    resources:
      requests:
        memory: "128Mi"
        cpu: "100m"
      limits:
        memory: "256Mi"
        cpu: "250m"
    volumeMounts:
    - name: azure
      mountPath: /mnt/azure
      volumes:
      - name: azure
        azureFile:
          secretName: azure-secret
          shareName: aksshare
          readOnly: false
```

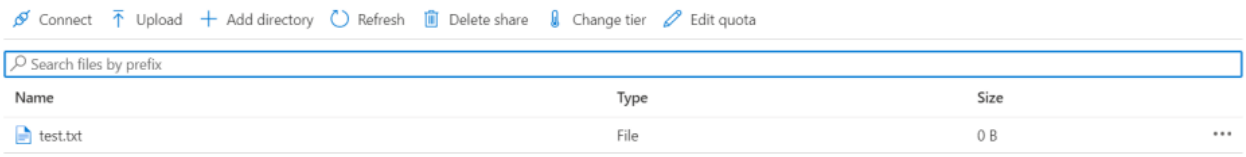
10. Run kubectl apply -f azure-files-pod.yaml.

```
PS /home/andrijana> kubectl apply -f azure-files-pod.yaml
pod/mypod configured
```

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.

- Yes, test.txt still exists - we have 2 txt files in Azure file share

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



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:

```
io.k8s.api.storage.v1.StorageClass (v1@storageclass.json)
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/andrijana> kubectl get pods
No resources found in default namespace.
PS /home/andrijana> kubectl apply -f azure-file-sc.yaml
storageclass.storage.k8s.io/my-azurefile created
PS /home/andrijana>
```

5. 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:

```yaml
io.k8s.api.core.v1.PersistentVolumeClaim (v1@persistentvolumeclaim.json)
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.

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

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.

```
persistentvolumeclaim/my-azurefile created
PS /home/andrijana> kubectl get pvc my-azurefile
NAME           STATUS   VOLUME                                     CAPACITY   ACCESS MODES   STORAGECLASS   AGE
my-azurefile   Bound    pvc-5512dfcb-ec7d-401c-b0f4-02436ef45085   5Gi        RWX            my-azurefile   24s
```

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:

```
io.k8s.api.core.v1.Pod (v1@pod.json)
apiVersion: v1
kind: Pod
metadata:
  name: mypod
spec:
  containers:
    - name: mypod
      image: mcr.microsoft.com/oss/nginx/nginx:1.15.5-alpine
      resources:
        requests:
          memory: "128Mi"
          cpu: "100m"
        limits:
          memory: "256Mi"
          cpu: "250m"
      volumeMounts:
        - name: volume
          mountPath: /mnt/azure
  volumes:
    - name: volume
      persistentVolumeClaim:
        claimName: my-azurefile
```

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

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

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

```
pod/mypod created
PS /home/andrijana> kubectl describe pod mypod
Name:             mypod
Namespace:        default
Priority:         0
Service Account:  default
Node:             aks-agentpool-32754409-vmss000000/10.224.0.4
Start Time:       Sun, 09 Apr 2023 10:07:15 +0000
Labels:           <none>
Annotations:      <none>
Status:           Running
IP:               10.244.1.20
IPs:
  IP:  10.244.1.20
Containers:
  mypod:
    Container ID:   containerd://ed1ecab565111c11c874b57208109a0a0079cbc6f58acf08dd1144e13cd1be33
    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:      Sun, 09 Apr 2023 10:07:16 +0000
    Ready:          True
    Restart Count:  0
    Limits:
      cpu:     250m
      memory:  256Mi
    Requests:
      cpu:        100m
```

The volumes:

```
Volumes:
  volume:
    Type:       PersistentVolumeClaim (a reference to a PersistentVolumeClaim in the same namespace)
    ClaimName:  my-azurefile
    ReadOnly:   false
  kube-api-access-bvk5j:
    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

Events:
```

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

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.

```
PS /home/andrijana> kubectl get pods
No resources found in default namespace.
PS /home/andrijana>
```

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/andrijana> az aks show --resource-group myAKSShare --name myCluster --query nodeResourceGroup -o tsv
MC_myAKSShare_myCluster_eastus
```

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/andrijana> az disk create --resource-group MC_myAKSShare_myCluster_eastus --name myAKSDisk --size-gb 20 --query id --output tsv
/subscriptions/836f56df-cca0-4866-b552-adbe26a742da/resourceGroups/MC_myAKSShare_myCluster_eastus/providers/Microsoft.Compute/disks/myAKSDisk
```

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/836f56df-cca0-4866-b552-adbe26a742da/resourceGroups/MC_myAKSShare_myCluster_eastus/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:

```yaml
io.k8s.api.core.v1.Pod (v1@pod.json)
apiVersion: v1
kind: Pod
metadata:
  name: mypod
spec:
  containers:
    - name: mypod
      image: mcr.microsoft.com/oss/nginx/nginx:1.15.5-alpine
      resources:
        requests:
          memory: "128Mi"
          cpu: "100m"
        limits:
          memory: "256Mi"
          cpu: "250m"
      volumeMounts:
        - name: azure
          mountPath: /mnt/azure
  volumes:
    - name: azure
      azureDisk:
        kind: Managed
        diskName: myAKSDisk
        diskURI: "/subscriptions/
        836f56df-cca0-4866-b552-adbe26a742da/resourceGroups/
        MC_myAKSShare_myCluster_eastus/providers/Microsoft.Compute/
        disks/myAKSDisk"
```

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

```
error: error validating "azure-disk-pod.yaml": error valid
PS /home/andrijana> kubectl apply -f azure-disk-pod.yaml
pod/mypod created
PS /home/andrijana>
```

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

```
See 'kubectl exec --help' for usage.
PS /home/andrijana> kubectl exec -it mypod -- sh
```

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**.

```
my-azurefile        Bound    pvc-5512dfcb-ec7d-401c-b0f4-02436ef45085   5Gi        RWX            my-azurefile      32m
PS /home/andrijana> kubectl get sc
NAME                    PROVISIONER                RECLAIMPOLICY   VOLUMEBINDINGMODE      ALLOWVOLUMEEXPANSION   AGE
azurefile               file.csi.azure.com         Delete          Immediate              true                   3h
azurefile-csi           file.csi.azure.com         Delete          Immediate              true                   3h
azurefile-csi-premium   file.csi.azure.com         Delete          Immediate              true                   3h
azurefile-premium       file.csi.azure.com         Delete          Immediate              true                   3h
default (default)       disk.csi.azure.com         Delete          WaitForFirstConsumer   true                   3h
managed                 disk.csi.azure.com         Delete          WaitForFirstConsumer   true                   3h
managed-csi             disk.csi.azure.com         Delete          WaitForFirstConsumer   true                   3h
managed-csi-premium     disk.csi.azure.com         Delete          WaitForFirstConsumer   true                   3h
managed-premium         disk.csi.azure.com         Delete          WaitForFirstConsumer   true                   3h
my-azurefile            kubernetes.io/azure-file   Delete          Immediate              false                  36m
```

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 YAML:

```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/andrijana> kubectl apply -f azure-premium.yaml
persistentvolumeclaim/azure-managed-disk created
```

7. Check the status of your PVC.

```
PS /home/andrijana> kubectl get pvc
NAME                 STATUS    VOLUME                                     CAPACITY   ACCESS MODES   STORAGECLASS      AGE
azure-managed-disk   Pending                                                                         managed-premium   112s
azurefile            Pending   azurefile                                  0                         azurefile-csi     72m
my-azurefile         Bound     pvc-5512dfcb-ec7d-401c-b0f4-02436ef45085   5Gi        RWX            my-azurefile      32m
```

- Status is Bound:

```
my-azurefile         Bound     pvc-5512dfcb-ec7d-401c-b0f4-02436ef45085   5Gi        RWX            my-azurefile      32m
```

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:

```yaml
io.k8s.api.core.v1.Pod (v1@pod.json)
apiVersion: v1
kind: Pod
metadata:
  name: mypod
spec:
  containers:
    - name: mypod
      image: mcr.microsoft.com/oss/nginx/nginx:1.15.5-alpine
      resources:
        requests:
          memory: "128Mi"
          cpu: "100m"
        limits:
          memory: "256Mi"
          cpu: "250m"
      volumeMounts:
        - name: volume
          mountPath: /mnt/azure
  volumes:
    - name: volume
      persistentVolumeClaim:
        claimName: azure-managed-disk
```

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

```
my-azurefile              kubernetes.io/azure-file   Delete
PS /home/andrijana> kubectl apply -f azure-pvc-disk.yaml
pod/mypod created
PS /home/andrijana>
```

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

```
  PodScheduled      True
Volumes:
  volume:
    Type:       PersistentVolumeClaim (a reference to a PersistentVolumeClaim in the same namespace)
    ClaimName:  azure-managed-disk
    ReadOnly:   false
  kube-api-access-9jqkn:
    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.kubernetes.io/unreachable:NoExecute op=Exists for 300s
Events:
  Type    Reason                  Age   From                     Message
  ----    ------                  ----  ----                     -------
  Normal  Scheduled               12s   default-scheduler        Successfully assigned default/mypod to aks-agentpool-32754409-vmss000000
  Normal  SuccessfulAttachVolume  0s    attachdetach-controller  AttachVolume.Attach succeeded for volume "pvc-27afd623-c4ad-4cb9-8239-7573d6f1fd6a"
```

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