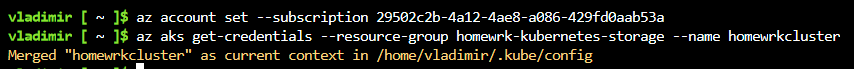
**Exercise: Storage**

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**

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



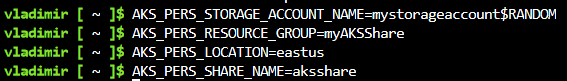
1. Check if any pods run under the default namespace if so delete everything under the default namespace.

* There are no pods under my default namespace.

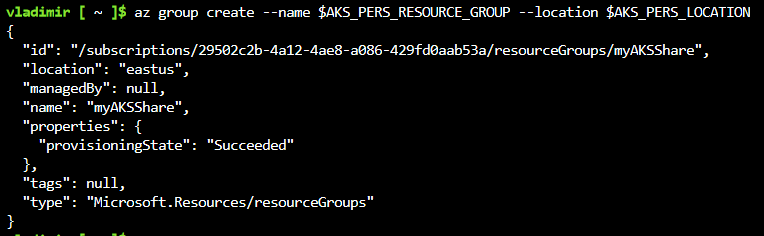


1. In this practice we will directly provision Azure Files to a pod running inside AKS.
2. First create the Azure Files share. Run the following commands:

# Change these four parameters as needed for your own environment



# Create a resource group



# Create a storage account

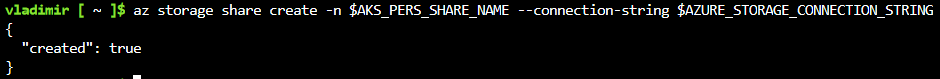
A picture containing graphical user interface

Description automatically generated

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



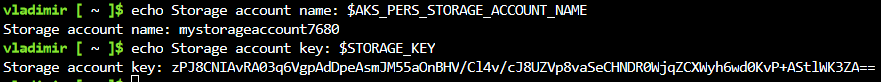
# Create the file share



# Get storage account key



# Echo storage account name and key

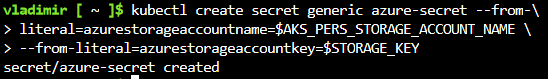


1. Make a note of the storage account name and key shown at the end of the script output. These values are

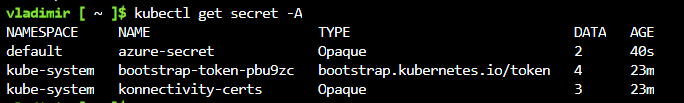
needed when you create the Kubernetes volume in one of the following steps.

* Storage account name: mystorageaccount7680
* Storage account key: zPJ8CNIAvRA03q6VgpAdDpeAsmJM55aOnBHV/Cl4v/cJ8UZVp8vaSeCHNDR0WjqZCXWyh6wd0KvP+AStlWK3ZA==

1. 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.
2. Run the following (single) command to create the secret:



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



1. 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

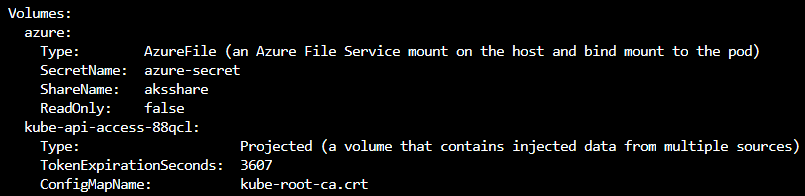
      shareName: aksshare

      readOnly: false

1. Run **kubectl apply -f azure-files-pod.yaml**.



1. You now have a running pod with an Azure Files share mounted at /mnt/azure.
2. You can use **kubectl describe pod mypod** to verify the share is mounted successfully. Search for the Volumes section of the output.



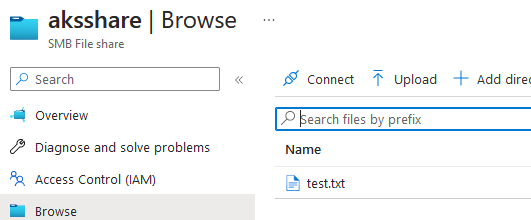
1. Now exec to the pod and try to access the mounted file share. Run the following command **kubectl exec -it mypod -- sh**

****

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



1. Go to the portal and locate your Azure storage provisioned for this practice.
2. Under the Files section, check the contents of the Azure file share and check if test.txt file exists.



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



* The file still exists under aksshare

**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.

* I’m still logged in from the previous exercise and the default namespace is empty

1. Now we will provision Azure files storage to a pod using PV and PVC.
2. Create an 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

1. Note the access mode. Can you use other mode with Azure files?
2. 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

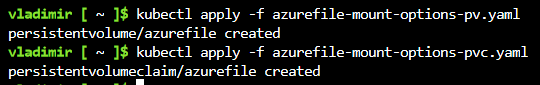
storageClassName: ""

resources:

requests:

storage: 5Gi

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

****

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

**azurefile.**

Text

Description automatically generated

1. 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

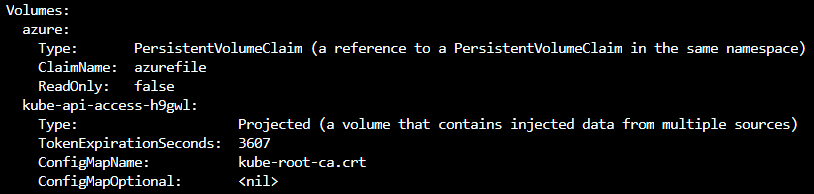
    persistentVolumeClaim:

      claimName: azurefile

1. Run **kubectl apply -f azure-files-pod.yaml**.



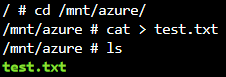
1. You now have a running pod with an Azure Files share mounted at /mnt/azure.
2. You can use **kubectl describe pod mypod** to verify the share is mounted successfully. Search for the Volumes section of the output.



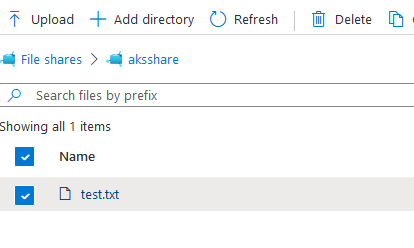
1. Now exec to the pod and try to access the mounted file share. Run the following command **kubectl exec -it mypod – bash**

****

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



1. Go to the portal and locate your Azure storage provisioned for this practice.
2. Under the Files section, check the contents of the Azure file share and check if test.txt file exists.



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

* The file still exists

**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.

* I’m still connected from the previous exercises and the default namespace is empty

1. 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

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



1. 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

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



1. 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.



1. 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

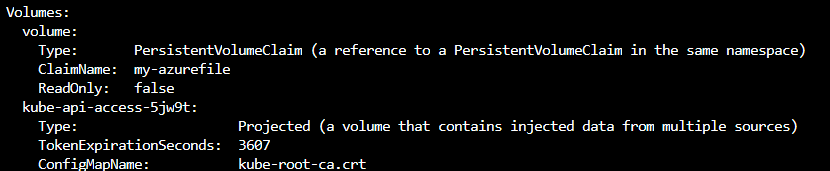
    persistentVolumeClaim:

      claimName: my-azurefile

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

****

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



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

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

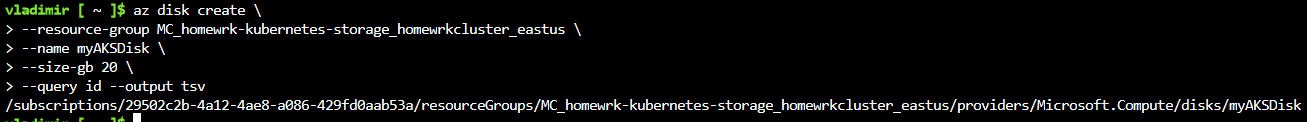
1. Login to Azure and connect to your AKS cluster.
2. Check if any pods run under the default namespace if so delete everything under the default namespace.

-I’m still logged in from the previous exercises and the default namespace is empty

1. In this practice we will directly provision Azure Disk to a pod running inside AKS.
2. 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** .



1. Now create a disk using:



1. 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.
2. 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

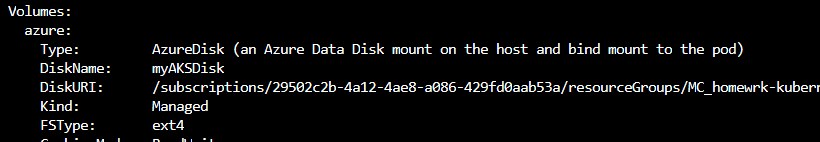
      diskName: myAKSDisk

  diskURI: /subscriptions/295…

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



1. You now have a running pod with an Azure Disk mounted at /mnt/azure.
2. You can use **kubectl describe pod mypod** to verify the share is mounted successfully. Search for the Volumes section of the output.



1. Now exec to the pod and try to access the mounted volume. Run the following command **kubectl exec -it mypod – bash**
2. Go to /mnt/azure and try create a blank file test.txt file.

**Text

Description automatically generated**

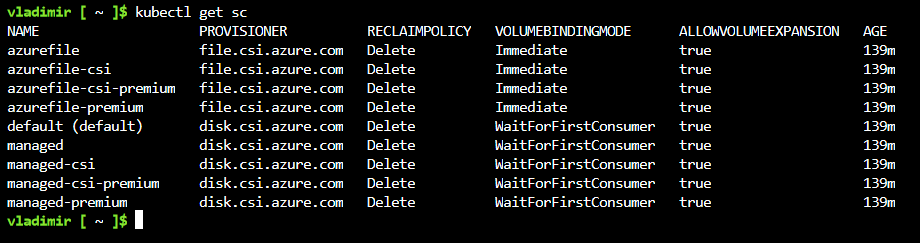
1. Delete everything created by this practice.

**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.

-I’m still logged in from the previous exercises and the default namespace is empty

1. 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.**

****

1. 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.
2. 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:

apiVersion: v1

kind: PersistentVolumeClaim

metadata:

  name: azure-managed-disk

spec:

  accessModes:

  - ReadWriteOnce

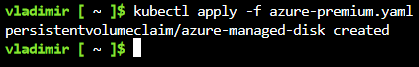
  storageClassName: managed-premium

  resources:

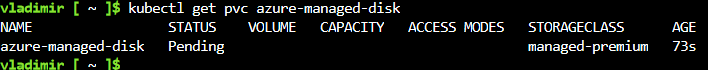
    requests:

      storage: 5Gi

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

****

1. Check the status of your PVC.



1. 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

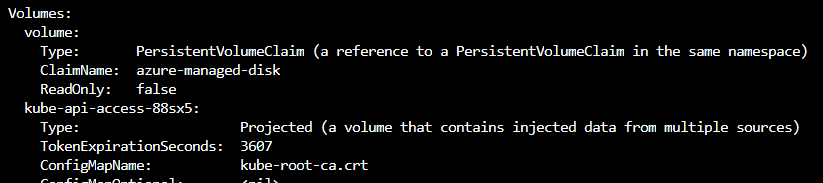
    persistentVolumeClaim:

      claimName: azure-managed-disk

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

****

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



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