Exercise: Kubernetes 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.

```
Requesting a Cloud Shell.Succeeded.
Connecting terminal...

lyubomir [ ~ ]$ az account set --subscription 3b42ed89-1120-4053-a858-c2baec72c542
lyubomir [ ~ ]$ az aks get-credentials --resource-group rg1 --name akc1
Merged "akc1" as current context in /home/lyubomir/.kube/config
```

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

```
lyubomir [ ~ ]$ kubectl get pods
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=lyubostorage

AKS_PERS_RESOURCE_GROUP=rg1

AKS_PERS_LOCATION=eastus

AKS_PERS_SHARE_NAME=aksshare

```
lyubomir [ ~ ]$ AKS_PERS_STORAGE_ACCOUNT_NAME=lyubostorage
lyubomir [ ~ ]$ AKS_PERS_RESOURCE_GROUP=rg1
lyubomir [ ~ ]$ AKS_PERS_LOCATION=eastus
lyubomir [ ~ ]$ AKS_PERS_SHARE_NAME=aksshare
```

Create a resource group

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

```
lyubomir [ ~ ]$ az group create --name $AKS_PERS_RESOURCE_GROUP --location $AKS_PERS_LOCATION
{
  "id": "/subscriptions/3b42ed89-1120-4053-a858-c2baec72c542/resourceGroups/rg1",
  "location": "eastus",
  "managedBy": null,
  "name": "rg1",
  "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

```
lyubomir [ ~ ]$ 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 false.
{
  "accessTier": "Hot",
  "allowBlobPublicAccess": true,
  "allowCrossTenantReplication": true,
  "allowSharedKeyAccess": true,
  "allowedCopyScope": null,
  "azureFilesIdentityBasedAuthentication": null,
  "blobRestoreStatus": null,
  "creationTime": "2023-04-19T23:00:44.077168+00:00",
  "customDomain": null,
  "defaultToOAuthAuthentication": false,
  "dnsEndpointType": "Standard",
  "enableHttpsTrafficOnly": true,
  "enableNfsV3": null,
  "encryption": {
    "encryptionIdentity": null,
    "keySource": "Microsoft.Storage",
    "keyVaultProperties": null,
    "requireInfrastructureEncryption": false,
    "services": {
      "blob": {
        "enabled": true,
        "keyType": "Account",
        "lastEnabledTime": "2023-04-19T23:00:44.233442+00:00"
      },
      "file": {
        "enabled": true,
```

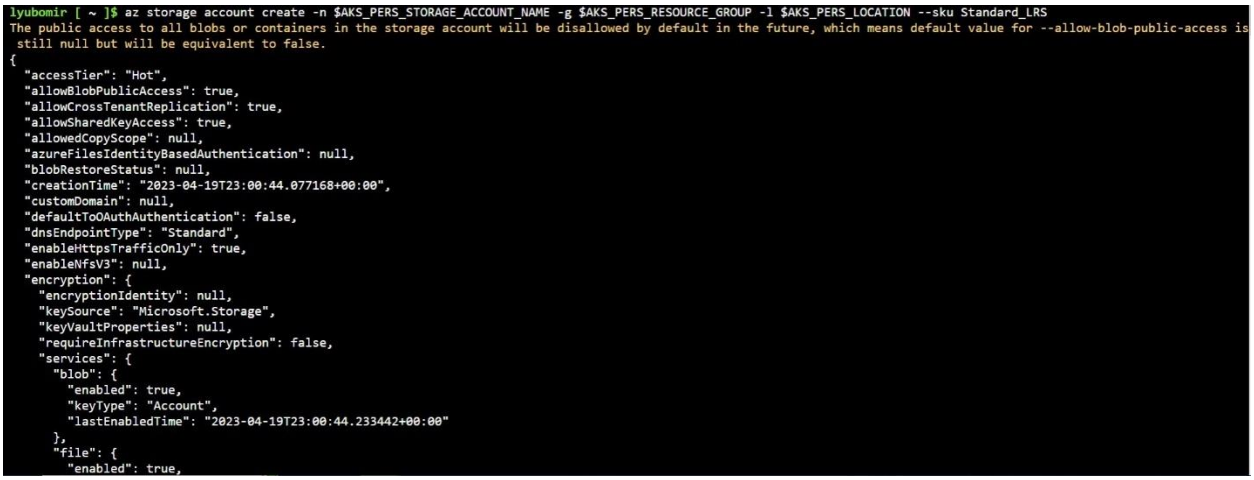
Export the connection string as an environment variable, this is used when creating the Azure file share

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



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.

```
lyubomir [ ~ ]$ kubectl get secret -A
NAMESPACE     NAME                  TYPE                             DATA   AGE
default       azure-secret          Opaque                           2      33m
kube-system   ama-logs-secret       Opaque                           2      72m
kube-system   bootstrap-token-3qci1f   bootstrap.kubernetes.io/token   4      72m
kube-system   konnectivity-certs    Opaque                           3      72m
```

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:

```yaml
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.

```
lyubomir [ ~ ]$ 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.

```
lyubomir [ ~ ]$ kubectl describe pod mypod
Name:             mypod
Namespace:        default
Priority:         0
Service Account:  default
Node:             aks-agentpool-34592492-vmss000001/10.224.0.5
Start Time:       Wed, 19 Apr 2023 23:22:05 +0000
Labels:           <none>
Annotations:      <none>
Status:           Running
IP:               10.244.0.14
IPs:
  IP:  10.244.0.14
Containers:
  mypod:
    Container ID:   containerd://484339cec275b4ea8c4bd6048d94f1a0d2ef9dc4a65560eab79097c74fce5486
    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:      Wed, 19 Apr 2023 23:22:08 +0000
    Ready:          True
    Restart Count:  0
    Limits:
```

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

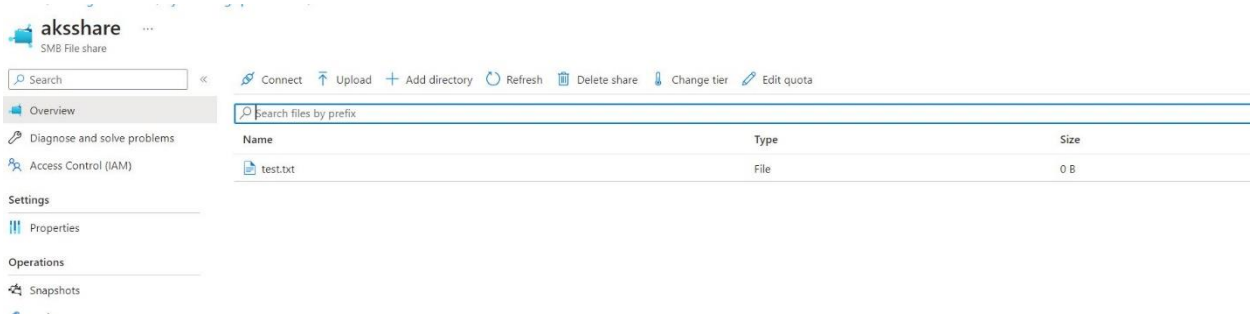
mypod – bash

```
lyubomir [ ~ ]$ kubectl exec -it mypod -- sh
/ # cd /mnt/azure
/mnt/azure # touch test.txt
```

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?

```
lyubomir [ ~ ]$ kubectl delete pod mypod
pod "mypod" 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.

```
Requesting a Cloud Shell.Succeeded.
Connecting terminal...

lyubomir [ ~ ]$ az account set --subscription 3b42ed89-1120-4053-a858-c2baec72c542
lyubomir [ ~ ]$ az aks get-credentials --resource-group rg1 --name akc1
Merged "akc1" as current context in /home/lyubomir/.kube/config
```

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

```
lyubomir [ ~ ]$ kubectl get pods --namespace=default
No resources found in 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
```

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:

```yaml
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.

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

```
lyubomir [ ~ ]$ 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.

```
lyubomir [ ~ ]$ kubectl get pvc azurefile
NAME        STATUS   VOLUME      CAPACITY   ACCESS MODES   STORAGECLASS   AGE
azurefile   Bound    azurefile   5Gi        RWX                           16s
```

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.

```
lyubomir [ ~ ]$ 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.

```
lyubomir [ ~ ]$ kubectl describe pod mypod
Name:             mypod
Namespace:        default
Priority:         0
Service Account:  default
Node:             aks-agentpool-34592492-vmss000001/10.224.0.5
Start Time:       Thu, 20 Apr 2023 00:19:20 +0000
Labels:           <none>
Annotations:      <none>
Status:           Running
IP:               10.244.0.17
IPs:
  IP:  10.244.0.17
Containers:
  mypod:
    Container ID:   containerd://927d10681a5bedcd9396f90f0807a2cba7dc5a99bea13fcb415988c3c212ccea
    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:      Thu, 20 Apr 2023 00:19:21 +0000
    Ready:          True
    Restart Count:  0
    Limits:
      cpu:     250m
      memory:  256Mi
    Requests:
      cpu:        100m
      memory:     128Mi
    Environment:  <none>
    Mounts:
      /mnt/azure from azure (rw)
      /var/run/secrets/kubernetes.io/serviceaccount from kube-api-access-k6w56 (ro)
```

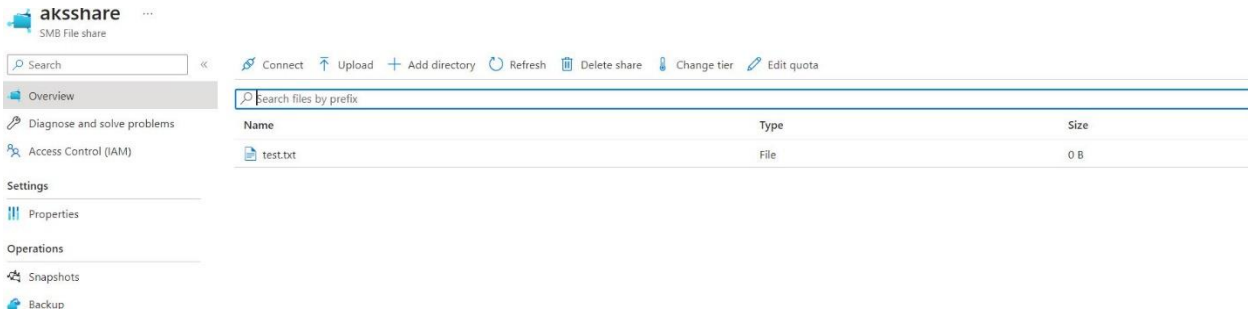
13. Now exec to the pod and try to access the mounted file share. Run the following command kubectl exec -it mypod – bash

```
lyubomir [ ~ ]$ kubectl exec -it mypod -- sh
/ # cd /mnt/azure
/mnt/azure # touch test.txt
```

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?

```
lyubomir [ ~ ]$ kubectl delete pod mypod
pod "mypod" 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.

```
Requesting a Cloud Shell.Succeeded.
Connecting terminal...

lyubomir [ ~ ]$ az account set --subscription 3b42ed89-1120-4053-a858-c2baec72c542
lyubomir [ ~ ]$ az aks get-credentials --resource-group rg1 --name akc1
Merged "akc1" as current context in /home/lyubomir/.kube/config
```

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

```
lyubomir [ ~ ]$ kubectl get pods
No resources found in 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

```
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 .

```
lyubomir [ ~ ]$ 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 azure-file-pvc.yaml

```
lyubomir [ ~ ]$ kubectl apply -f azure-file-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.

```
lyubomir [ ~ ]$ kubectl get pvc my-azurefile
NAME           STATUS    VOLUME   CAPACITY   ACCESS MODES   STORAGECLASS   AGE
my-azurefile   Pending                                      my-azurefile   13s
```

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

```
kind: Pod
apiVersion: v1
```

```yaml
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 .

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

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

```
lyubomir [ ~ ]$ kubectl describe pods
Name:             mypod
Namespace:        default
Priority:         0
Service Account:  default
Node:             aks-agentpool-34592492-vmss000001/10.224.0.5
Start Time:       Thu, 20 Apr 2023 00:37:48 +0000
Labels:           <none>
Annotations:      <none>
Status:           Running
IP:               10.244.0.18
IPs:
  IP:  10.244.0.18
Containers:
  mypod:
    Container ID:   containerd://e8ca98728a847055692b35849420cbf89bd09ec20c042f03fac0d4091346faa4
    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:      Thu, 20 Apr 2023 00:37:49 +0000
    Ready:          True
    Restart Count:  0
    Limits:
      cpu:     250m
      memory:  256Mi
    Requests:
      cpu:        100m
      memory:     128Mi
    Environment:  <none>
```

```
Volumes:
  volume:
    Type:       PersistentVolumeClaim (a reference to a PersistentVolumeClaim in the same namespace)
    ClaimName:  my-azurefile
    ReadOnly:   false
  kube-api-access-4n9wv:
    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:
  Type    Reason     Age   From               Message
  ----    ------     ----  ----               -------
  Normal  Scheduled  25s   default-scheduler  Successfully assigned default/mypod to aks-agentpool-34592492-vmss000001
  Normal  Pulled     25s   kubelet            Container image "mcr.microsoft.com/oss/nginx/nginx:1.15.5-alpine" already present on machine
  Normal  Created    25s   kubelet            Created container mypod
  Normal  Started    24s   kubelet            Started container mypod
```

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

```
lyubomir [ ~ ]$ kubectl delete pod mypod
kubectl delete pvc my-azurefile
kubectl delete pv azurefile
kubectl delete sc my-azurefile
pod "mypod" deleted
persistentvolumeclaim "my-azurefile" deleted
Error from server (NotFound): persistentvolumes "azurefile" not found
storageclass.storage.k8s.io "my-azurefile" deleted
lyubomir [ ~ ]$
```

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.

```
Requesting a Cloud Shell.Succeeded.
Connecting terminal...

lyubomir [ ~ ]$ az account set --subscription 3b42ed89-1120-4053-a858-c2baec72c542
lyubomir [ ~ ]$ az aks get-credentials --resource-group rg1 --name akc1
Merged "akc1" as current context in /home/lyubomir/.kube/config
```

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

```
lyubomir [ ~ ]$ kubectl get pods
No resources found in 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 tsv6. 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

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.

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:

```
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.

kubectl describe pod mypod

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

kubectl delete pod mypod

kubectl delete pvc azure-managed-disk

```
kubectl delete pv azure-managed-disk

kubectl delete sc managed-premium
```