

**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!

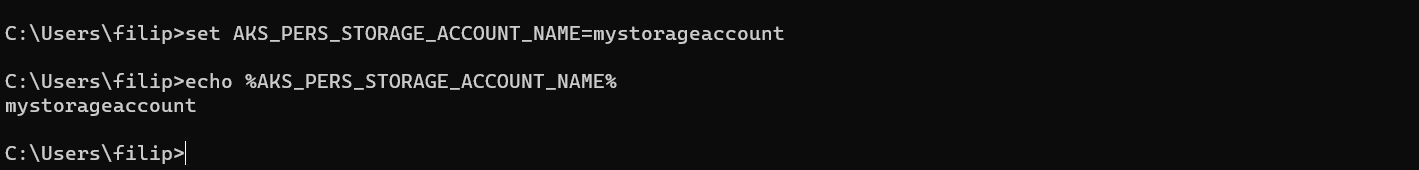
* 1. Login to Azure and connect to your AKS cluster.  
       
     Using Azure CLI through command prompt.  
       
     Text

     Description automatically generatedminikube start starts a local Kubernetes cluster on your machine with a single node configuration.  
     Text

     Description automatically generated
  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





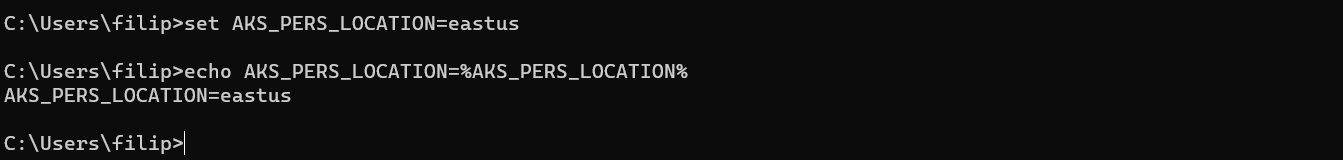
AKS\_PERS\_RESOURCE\_GROUP=myAKSShare

Text

Description automatically generated

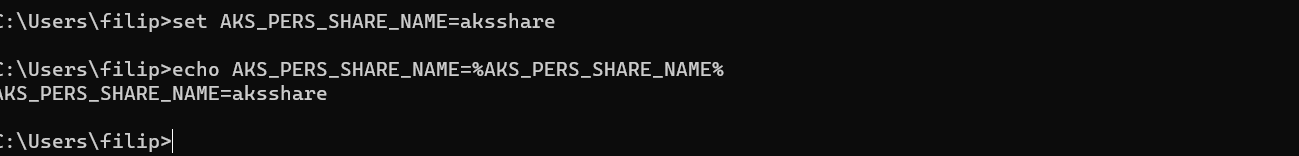


AKS\_PERS\_LOCATION=eastus





AKS\_PERS\_SHARE\_NAME=aksshare





# Create a resource group



az group create --name $AKS\_PERS\_RESOURCE\_GROUP --location $AKS\_PERS\_LOCATION

Text

Description automatically generated



# 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

Graphical user interface, text

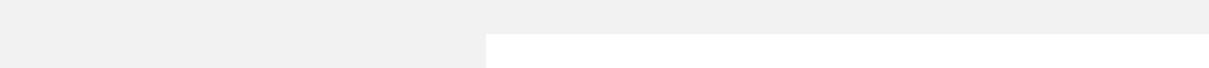
Description automatically generated

Text

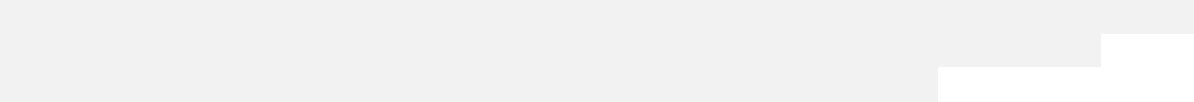
Description automatically generated

Text

Description automatically generated



The name used previously was already taken so I will have to change the value of the variable, so so I can proceed further.

* 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)
* Text

  Description automatically generated

This is used to connect the storage account.

* Create the file share



az storage share create -n $AKS\_PERS\_SHARE\_NAME --connection-string



$AZURE\_STORAGE\_CONNECTION\_STRING

Text

Description automatically generated

Got warnings. C:\Users\filip>echo Failed to load python executable.

Failed to load python executable. Text

Description automatically generated

Checking if there is a mistake along the way.

Text

Description automatically generated

Still getting the error. I will try with recreating the previous step - Export the connection string as an environment variable, this is used when creating the Azure file share. I believe the mistake was here.

Text

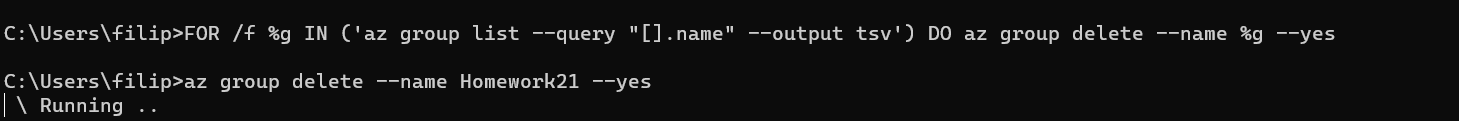
Description automatically generated

Text

Description automatically generated

I probably didn’t set it up correctly.

**Could NOT locate the problem and started over.**



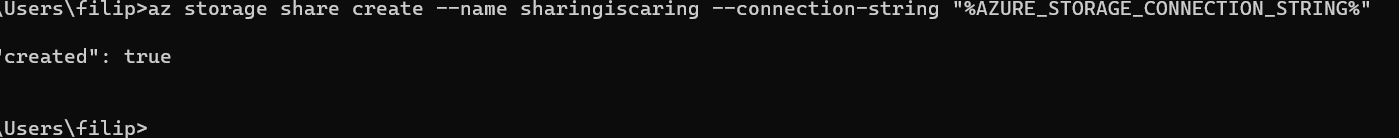
**I will skip the previous steps and continue once everything is in order.**

They are all set.

**Text

Description automatically generated**

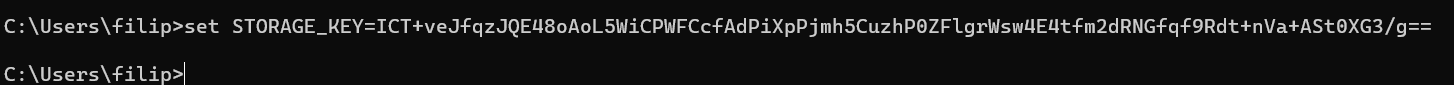
Somehow managed to create the storage share file.

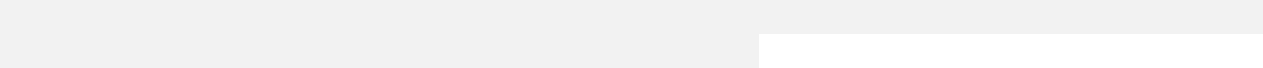
****

# 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)

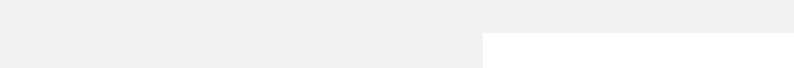




# Echo storage account name and key



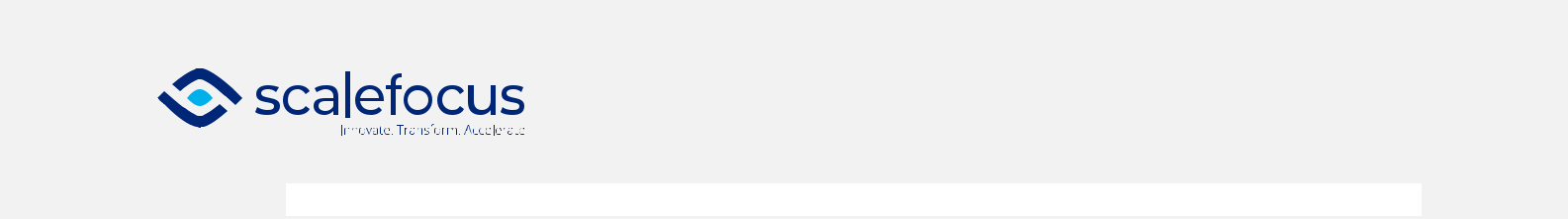
echo Storage account name: $AKS\_PERS\_STORAGE\_ACCOUNT\_NAME



echo Storage account key: $STORAGE\_KEY

Text

Description automatically generated

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.

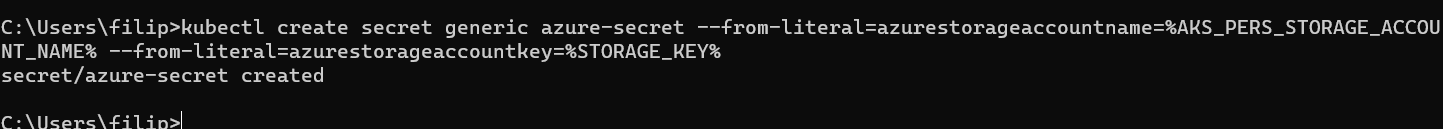
Graphical user interface

Description automatically generated with medium confidence

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:

**kubectl create secret generic azure-secret --from- \ literal=azurestorageaccountname=$AKS\_PERS\_STORAGE\_ACCOUNT\_NAME \**

**--from-literal=azurestorageaccountkey=$STORAGE\_KEY**



Secret is a piece of sensitive data that needs to be stored securely and not exposed in plain text.

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

Text

Description automatically generated

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

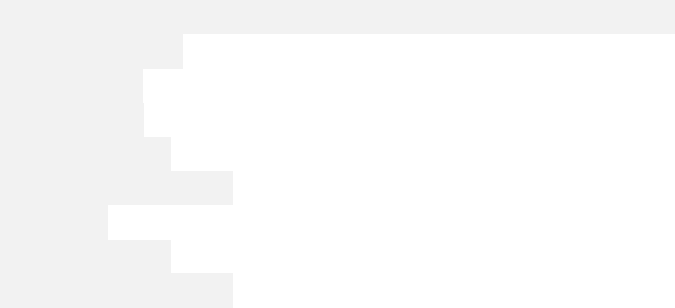
name: mypod resources: requests:

cpu: 100m

memory: 128Mi limits:

cpu: 250m

memory: 256Mi

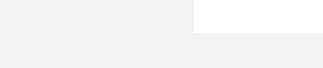


volumeMounts:



* name: azure

mountPath: /mnt/azure

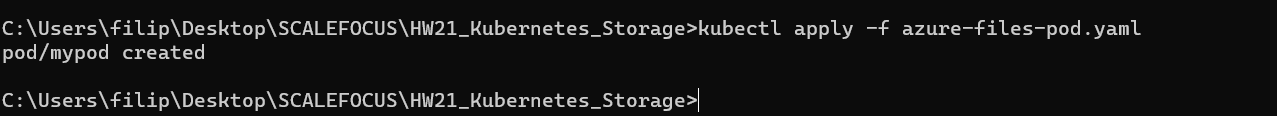


volumes:



* name: azure azureFile: secretName: azure-secret shareName: aksshare readOnly: false
* Text

  Description automatically generated
  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.

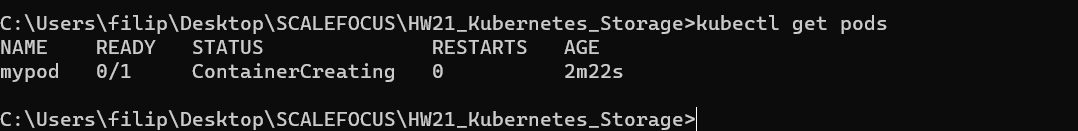
Text

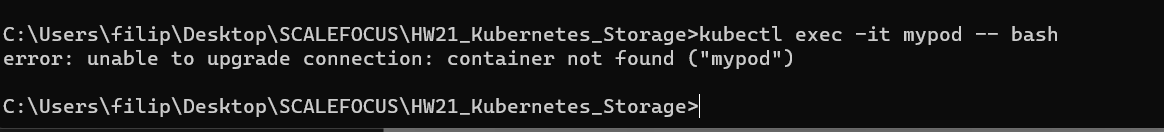
Description automatically generated

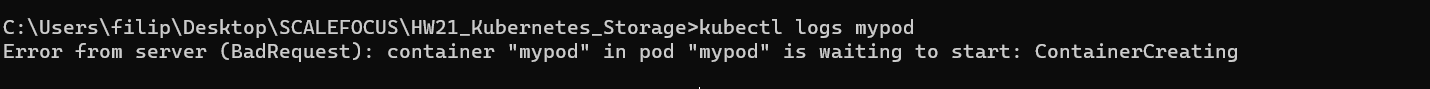
Text

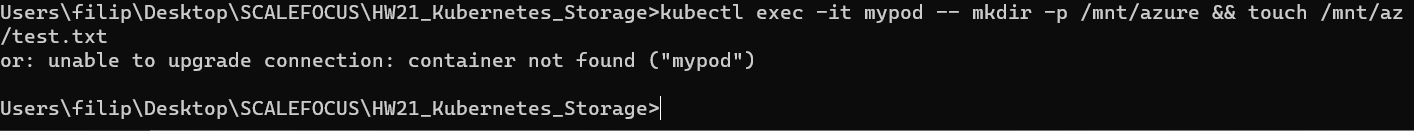
Description automatically generated

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

**It is still creating the Container.**





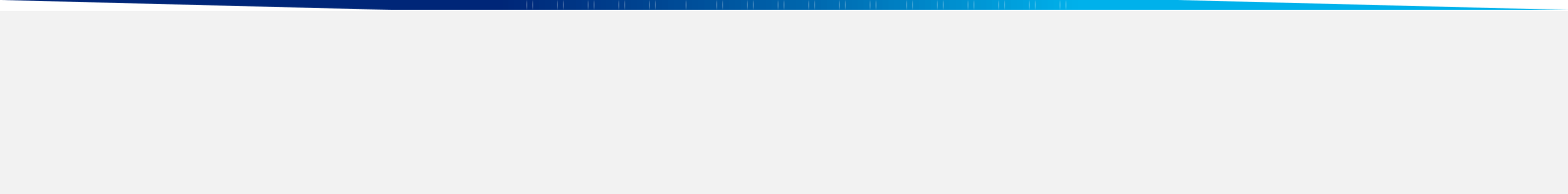
* 1. Go to /mnt/azure and create a blank file test.txt file. 

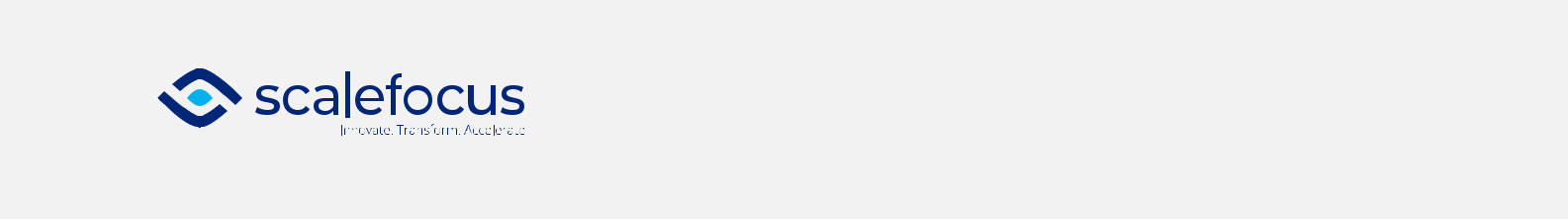
Could not create the file, The container is not created.

* 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.

Graphical user interface, text, application, email

Description automatically generated



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

It will not affect 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.

Text

Description automatically generated

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

Text

Description automatically generated

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

ReadOnlyMany: the volume can be mounted read-only by many nodes

ReadWriteMany: the volume can be mounted as read-write by many nodes

ReadWriteOnce: the volume can be mounted as read-write by a single node

* + 1. 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:

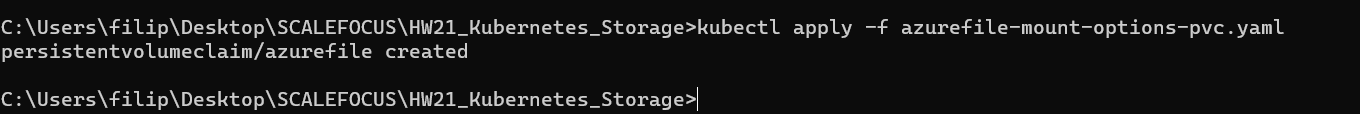
storage: 5Gi

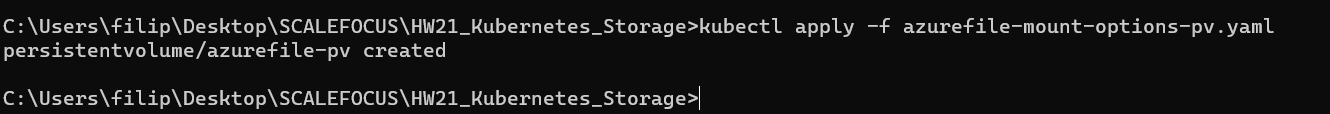


Text

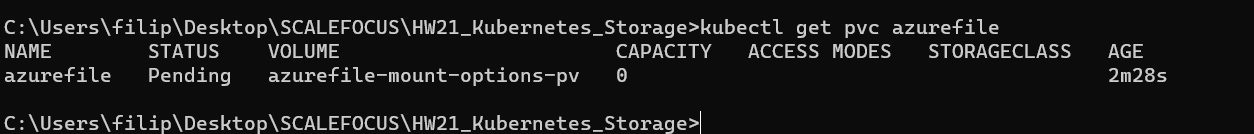
Description automatically generated

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





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



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:

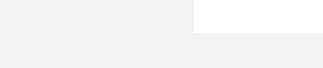
cpu: 250m

memory: 256Mi volumeMounts:



* name: azure

mountPath: /mnt/azure



volumes:

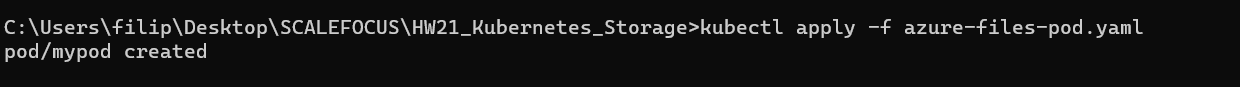


* name: azure persistentVolumeClaim: claimName: azurefile

Text

Description automatically generated

* 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.

Text

Description automatically generated

Text

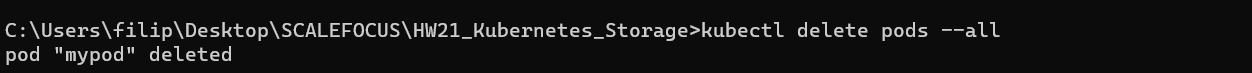
Description automatically generated

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

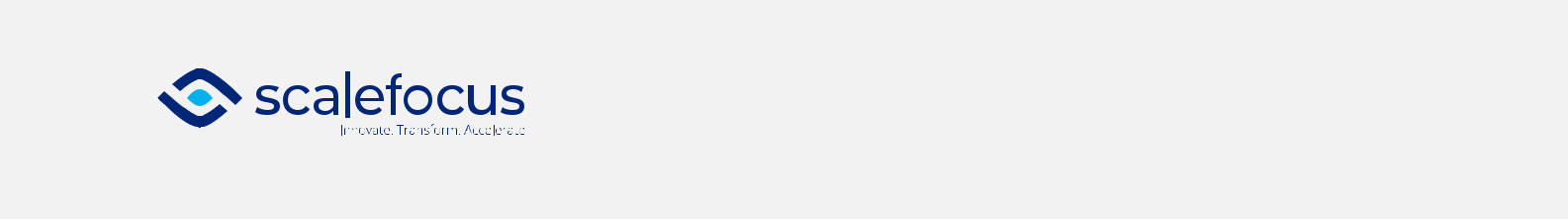
Text

Description automatically generated

* 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.
  3. 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:

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

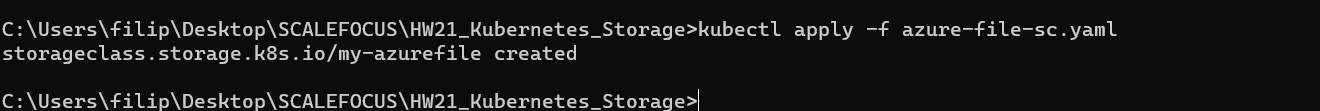
parameters:

skuName: Standard\_LRS

Text

Description automatically generated

* 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 azure-file-pvc.yaml and copy in the following YAML:



apiVersion: v1



kind: PersistentVolumeClaim



metadata:



name: my-azurefile



spec:

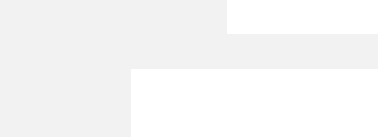


accessModes:



* ReadWriteMany storageClassName: my-azurefile resources:

requests:



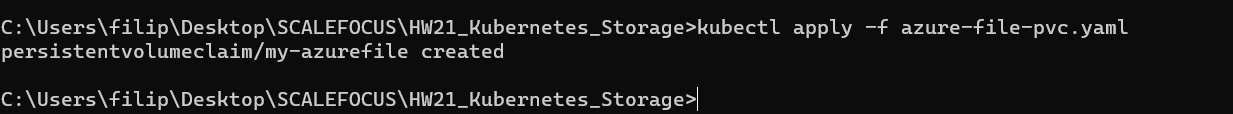
storage: 5Gi

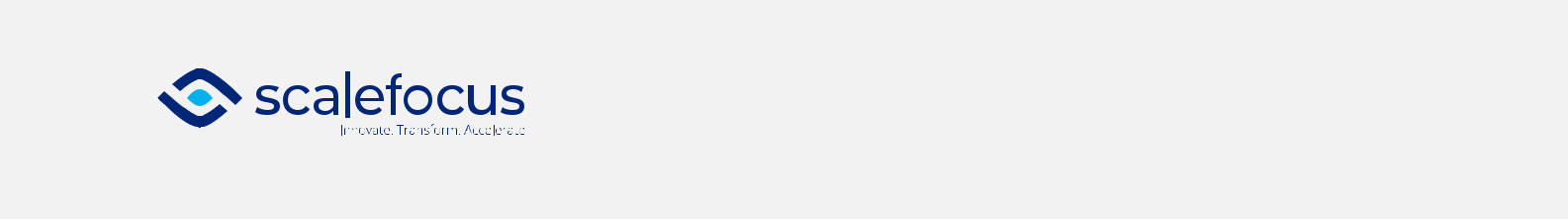
Text

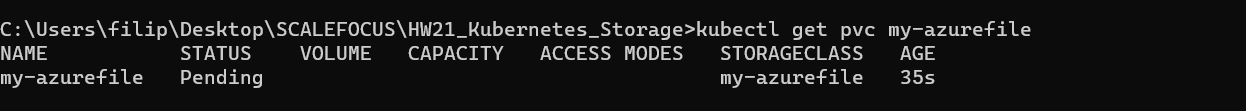
Description automatically generated



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"



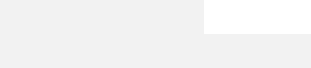
name: volume



volumes:



* name: volume persistentVolumeClaim:



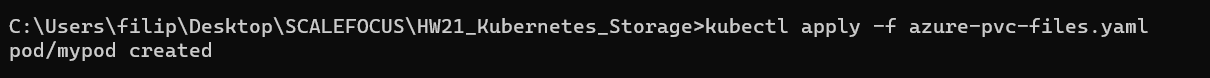
claimName: my-azurefile

Text

Description automatically generated



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



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

Text

Description automatically generated

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

Text

Description automatically generated

**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.

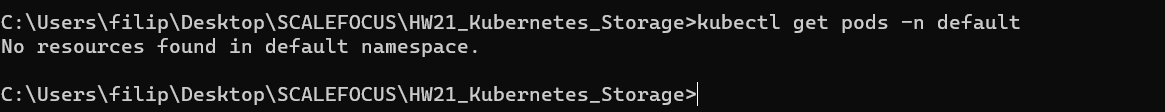
Graphical user interface, text, application, email

Description automatically generated

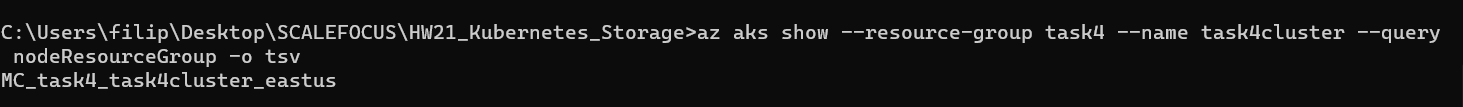
Text

Description automatically generated with medium confidence

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



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:

az disk create \



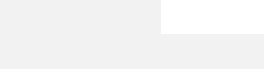
--resource-group MC\_myResourceGroup\_myAKSCluster\_eastus \



--name myAKSDisk \

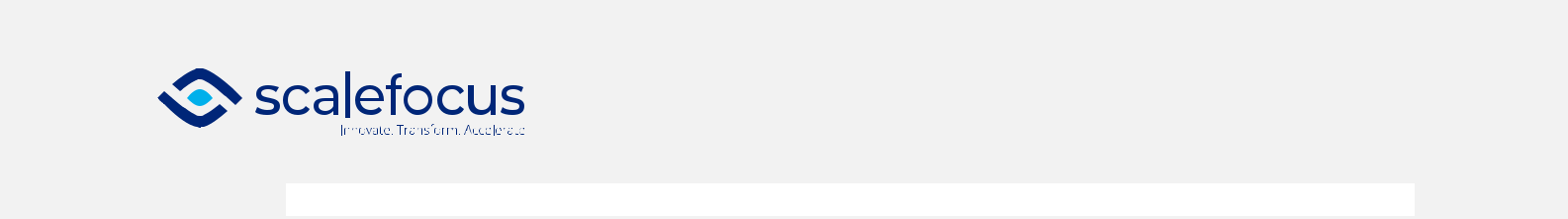


--size-gb 20 \



--query id --output tsv

Text

Description automatically generatedMake 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.

1. 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

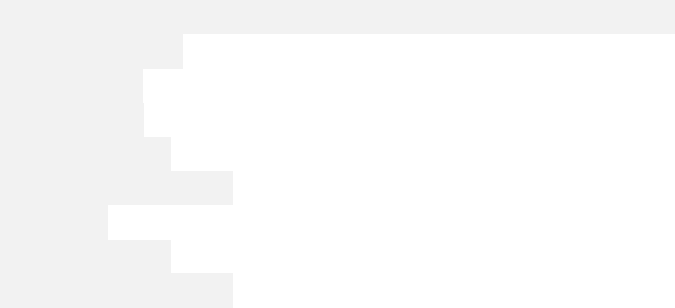
name: mypod resources: requests:

cpu: 100m

memory: 128Mi limits:

cpu: 250m

memory: 256Mi

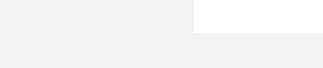


volumeMounts:



* name: azure

mountPath: /mnt/azure



volumes:



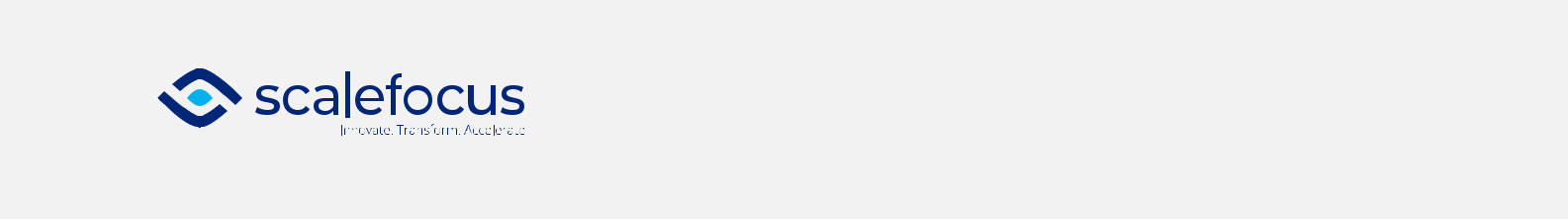
* name: azure azureDisk: kind: Managed diskName: myAKSDisk



diskURI: **<!!!!!!!!!!!!! Put the Disk resource id noted before!!!**>



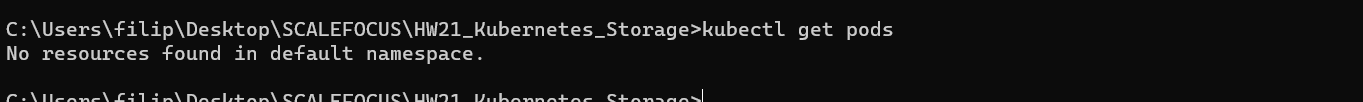
1. Run **kubectl apply -f azure-disk-pod.yaml**.
2. You now have a running pod with an Azure Disk mounted at /mnt/azure.
3. You can use **kubectl describe pod mypod** to verify the share is mounted successfully. Search for the Volumes section of the output.
4. Now exec to the pod and try to access the mounted volume. Run the following command **kubectl exec -it mypod -- bash**
5. Go to /mnt/azure and try create a blank file test.txt file.
6. 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.



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.**

Graphical user interface, text

Description automatically generated

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



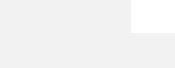
storageClassName: managed-premium



resources:



requests:

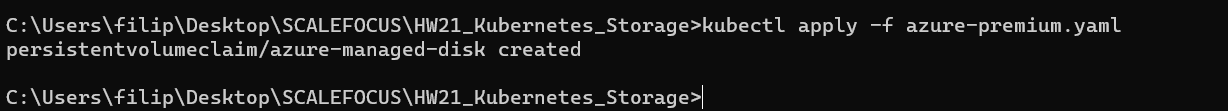


storage: 5Gi

Text

Description automatically generated

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



1. Check the status of your PVC.
2. 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:



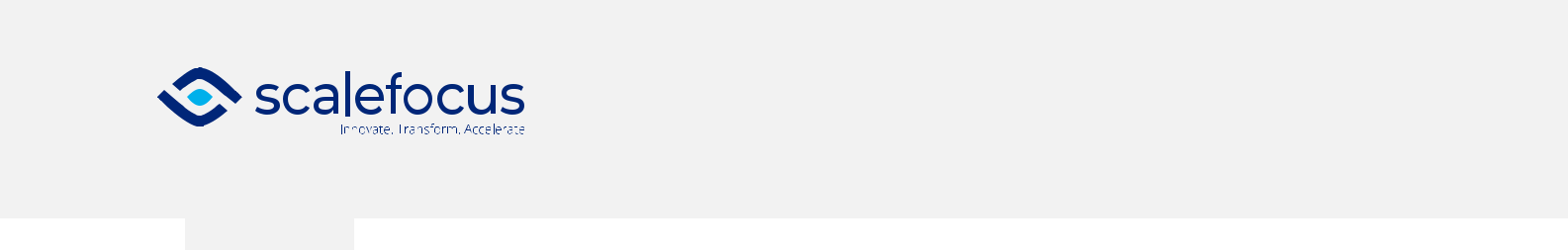
containers:



* name: mypod

image: mcr.microsoft.com/oss/nginx/nginx:1.15.5-alpine resources:

requests: cpu: 100m memory: 128Mi

limits:

cpu: 250m

memory: 256Mi

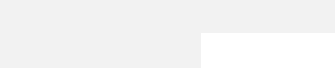


volumeMounts:



* mountPath: "/mnt/azure"

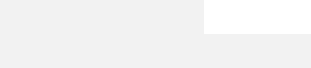
name: volume



volumes:



* name: volume persistentVolumeClaim:



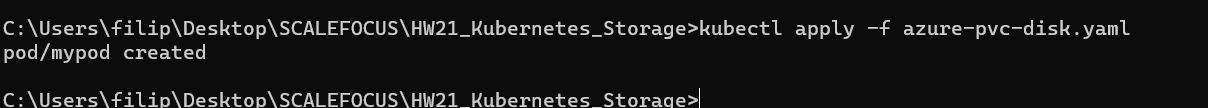
claimName: azure-managed-disk

Text

Description automatically generated



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



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

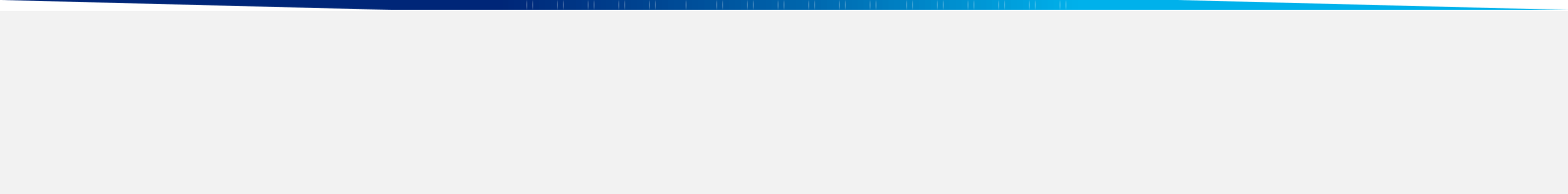
Text

Description automatically generated

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

Text

Description automatically generated



BONUS QUESTION ANSWERED BY CHATGPT 😊 😊 😊

The master node is responsible for managing the cluster and its components, and running user workloads on it can lead to performance issues and potential security vulnerabilities.

There are a few reasons why scheduling pods on the master node is not recommended:

Resource utilization: The master node is typically configured with limited resources compared to worker nodes. Running user workloads on the master node can cause resource contention, leading to performance issues and potentially impacting the stability of the cluster.

Security: Running user workloads on the master node can pose a security risk. The master node has elevated privileges and direct access to the Kubernetes API server, which means that running untrusted code on it can potentially lead to a compromise of the entire cluster.

Maintenance: The master node is responsible for managing the cluster's control plane components, such as the API server, etcd, and kube-controller-manager. Running user workloads on the master node can complicate maintenance tasks and upgrades, potentially leading to downtime and operational issues.

For these reasons, it is generally recommended to reserve the master node for control plane components and system daemons, and to schedule user workloads on worker nodes. However, there may be specific use cases where it is appropriate to schedule pods on the master node, such as for running monitoring or logging agents. In such cases, it is important to carefully consider the potential impacts on resource utilization, security, and maintenance.