Storage Options for Azure Kubernetes Service

Applications deployed to an AKS cluster require storage to store and retrieve their data. For some workloads, these storages can be local, fast storage on the node and can be released when pods are deleted, while some other workloads may need persistent data storage hosted in Azure. Multiple pods may require sharing the same data volumes and/or reattaching the data volumes if the pods are rescheduled on a different node. It is also may be required to present sensitive data or application configuration into the pods. Figure 7-7 depicts the storage architecture for AKS clusters.

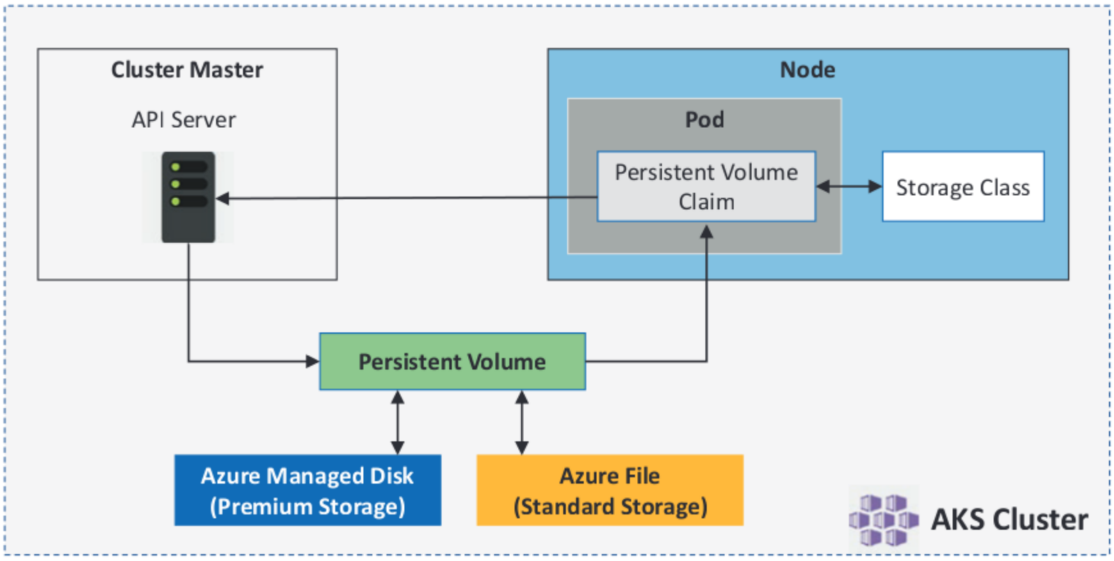

Figure 7-7. *Storage architecture in AKS*

In this section, let's explore the core concepts in AKS that explains how storage is provided to your application workloads.

Volumes

A *volume* represents a method to store, retrieve, and persist data across pods and through the application life cycle. Usually the volumes required to store and retrieve data on AKS is based on Azure storage. These data volumes can either be manually created and then assigned to pods directly or AKS can automatically create and allocate them when needed. These data volumes can either use

- **Azure Disks**: These can be used to create a Kubernetes DataDisk resource. Azure Disks can use Azure Premium storage or Azure Standard storage. For production and development workloads, it is recommended to use Premium storage. These are mounted as ReadWriteOnce and hence only available to a single node.
- **Azure Files** which are used to mount an SMB 3.0 share backed by an Azure storage account to pods. With Azure Files, you share data across multiple nodes and pods. Both Azure Premium storage and Azure Standard storage are supported with Azure Files.

Kubernetes volumes can also be leveraged to inject data into a pod for use by the containers. Additional volume types in Kubernetes include

- **emptyDir**: Used as temporary space for a pod
- **secret**: Used to inject sensitive data into pods, such as passwords
- **configMap**: Used to inject key-value pair properties into pods, such as application configuration information

Persistent Volumes

A persistent volume (PV) is created and managed by the Kubernetes API. It can exist beyond the lifetime of an individual pod, whereas traditional volumes created as part of a pod life cycle only exist until the pod is deleted. You can use either Azure Disks or Files to provide a PV.

A persistent volume can be either manually created by a cluster admin or dynamically generated by the Kubernetes API server. In case a scheduled requests storage that is not currently available, Kubernetes will create the underlying Azure Disk or Files storage and attach it to the pod. This scenario is called **Dynamic provisioning** and it uses a **StorageClass** to determine what type of Azure storage is required.

Storage Classes

In order to classify different storage tiers, you can create a **StorageClass**. The StorageClass also defines the **reclaimPolicy**. The reclaimPolicy controls the behavior of the Azure storage resource when the pod is deleted, and the persistent volume may no longer be required. When a pod is deleted, the storage resource can either be deleted or retained for use with a future pod.

There are two initial StorageClasses that can be created in AKS:

- **default**: Which utilizes Azure Standard storage to create a Managed Disk. The reclaim policy states that when the corresponding pod is deleted, the Azure Disk will also be deleted.
- **managed-premium**: Which utilizes Azure Premium storage to create a Managed Disk. The reclaim policy states that when the corresponding pod is deleted, the Azure Disk will also be deleted.

When no StorageClass is specified while creating a PV, the default StorageClass is used.

In the following YAML manifest, it is stated that Premium Managed Disks are to be used and the Azure Disk must be retained when the pod is deleted:

Listing 7-12. Defining a storage class in YAML

```
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: managed-premium-retain
provisioner: kubernetes.io/azure-disk
reclaimPolicy: Retain
parameters:
  storageaccounttype: Premium_LRS
  kind: Managed
```

Persistent Volume Claims

If you want to create either disk or file storage of a defined StorageClass, access mode, and size, define a PersistentVolumeClaim. If there are no existing resources to service the claim based on its StorageClass, the Kubernetes API server can dynamically provision the underlying storage resource. Once the volume has been connected to the pod, the pod definition will include the volume mount as well.

A PV is bound to a PersistentVolumeClaim after an available storage resource has been assigned to the pod that requests it. The mapping of persistent volumes to claims is 1:1

The following is a sample YAML manifest which denotes a PV claim with managed-premium StorageClass and a disk 5 Gi in size.

Listing 7-13. Defining a PersistentVolumeClaim in YAML

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: azure-managed-disk
spec:
  accessModes:
  - ReadWriteOnce
  storageClassName: managed-premium
  resources:
    requests:
      storage: 5Gi
```

A PV claim is specified to request the desired storage when a pod definition is created. Here you can also specify the **volumeMount** for your applications to read and write data. The following YAML manifest illustrates how the previous PV claim can be used to mount a volume at /mnt/azure.

Listing 7-14. Defining a volumeMount in a PersistentVolumeClaim in YAML

```
kind: Pod
apiVersion: v1
metadata:
  name: nginx
spec:
  containers:
    - name: myfrontend
      image: nginx
      volumeMounts:
```

```
      - mountPath: "/mnt/azure"
        name: volume
  volumes:
    - name: volume
      persistentVolumeClaim:
        claimName: azure-managed-disk
```

We have briefly discussed the storage options available for AKS workloads. Next step is to create dynamic and static volumes for AKS. The following articles from the Microsoft documentation provide a holistic overview on how to do so:

- Create a static volume using Azure Disks (`https://docs.microsoft.com/en-au/azure/aks/azure-disk-volume`).
- Create a static volume using Azure Files (`https://docs.microsoft.com/en-au/azure/aks/azure-files-volume`).
- Create a dynamic volume using Azure Disks (`https://docs.microsoft.com/en-au/azure/aks/azure-disks-dynamic-pv`).
- Create a dynamic volume using Azure Files (`https://docs.microsoft.com/en-au/azure/aks/azure-files-dynamic-pv`).

Networking in Azure Kubernetes Service

Application components in a microservices approach must work together to process their desired tasks. This application communication can be achieved using a few components provided by Kubernetes. For an example, the applications can be either exposed internally or externally, can be load balanced for high availability, and have SSL.TLS termination for ingress traffic as well as for routing of multiple components. Furthermore, developers may need you to restrict the flow of network traffic into or between pods and nodes due to security concerns.

In this section, we will dive into core networking concepts of AKS and some of the examples of providing secure network connectivity to your pods and nodes.

Kubenet vs. Azure Container Networking Interface (CNI)

An AKS cluster uses one of the following two networking models: