Running Applications on Azure Kubernetes Service: A Tutorial

Shohruh MIRYUSUPOV*

February 21, 2024

Abstract

This tutorial is designed to guide professionals through the intricacies of deploying applications within the Azure Kubernetes Service (AKS) on Windows containers. Adapted to cater to both beginners and seasoned users of Azure or Kubernetes, the tutorial outlines the process of setting up the Azure Container Registry (ACR) for Docker image storage, creating an AKS cluster, and smoothly deploying applications onto this cluster.

The tutorial is structured to ensure ease of understanding and practical application, featuring step-by-step instructions enriched with concrete examples. It utilizes a range of tools and technologies, including Powershell, Docker, Azure CLI (az), Kubernetes command-line tool (kubectl), and YAML configurations, providing a hands-on approach to learning and implementation.

Moreover, the tutorial goes beyond the basics by including appendices that guide users in selecting the most suitable virtual machine for computational needs and choosing appropriate storage options on Microsoft Azure.

Whether you are looking to deepen your understanding of AKS or seeking practical guidance for your next application deployment on Azure, this tutorial offers the insights and step-by-step procedures to ensure the successful deployment of the solution to the Microsoft Azure Cloud.

*shohruh@miryusupov.com

Contents

Introduction

This tutorial serves as a guide to deploying applications within the AKS. It details the procedure for setting up the ACR to store Docker images, creating an AKS cluster, and deploying an application onto this cluster. The instructions are designed to be straightforward, ensuring even those new to Azure or Kubernetes can follow along and successfully deploy their applications, complete with concrete examples using `Powershell`, `docker`, Azure CLI (`az`), Kubernetes command-line tool (`kubectl`), and `YAML` configurations. In Appendix we show how one can find a virtual machine that is appropriate for computational needs and how to select a storage on Microsoft Azure.

Definitions

In this section we establish the fundamental concepts and terminologies used throughout the tutorial. By defining these terms such as Docker Image, Azure CLI, and AKS Cluster, the tutorial ensures that readers have a shared understanding of the essential elements involved in deploying applications in AKS.

Table 1 allows to navigate faster navigate, if necessary, through the list of definitions.

Docker	**Kubernetes**	**Microsoft Azure**
Docker	Kubernetes	Azure CLI
Docker Image	Kubernetes cluster	Azure Container Registry (ACR)
Container	Cluster Master	Azure Kubernetes Service (AKS)
	Pods	Azure File Storage
	Deployment	AKS Cluster
	CronJobs	SMB Protocol
	ReplicaSet	Manifest
	Replica	
	Volume	
	Persistent Volume (PV)	
	Persistent Volume Claim (PVC)	
	Kubelet	
	Node	

Table 1: Definitions grouped by Docker, Kubernetes, and Microsoft Azure

Docker Docker [1], [3] is a platform as a service (PaaS) product that uses OS-level virtualization to deliver software in packages called containers. Containers are isolated environments that run applications, ensuring that they work uniformly regardless of the environment.

Docker Image A lightweight, standalone, executable package that includes everything needed to run a piece of software, including the code, runtime, libraries, environment variables, and config files [1], [4].

Azure CLI Azure CLI [25], [26], short for Azure Command Line Interface, is a set of commands used to manage Azure resources. With Azure CLI, you can interact with Azure services and manage Azure resources such as compute, network, storage, and more. Installation details can be found in [27].

Azure Container Registry (ACR) A managed Docker container registry service used for storing private Docker container images, similar to Docker Hub, but hosted within Azure for better integration and security with Azure services [4].

Azure Kubernetes Service (AKS) A managed container orchestration service, based on Kubernetes, that facilitates deploying, managing, and scaling containerized applications on Microsoft Azure [4]. Azure Kubernetes Service

Azure File Storage A Microsoft Azure service providing cloud file storage via the Server Message Block (SMB) protocol and accessible from anywhere using REST APIs or Azure management tools [4].

AKS Cluster A set of node machines for running containerized applications in AKS. Each AKS cluster includes at least one master machine and multiple worker machines or nodes [4].

Kubernetes Kubernetes is an open-source container orchestration system that automates various aspects of software deployment, scaling, and management [5]. A Kubernetes cluster, which is a set of running Linux containers, can be visualized as having two parts: the control plane and the compute machines (or nodes). The control plane manages the state of the cluster, including applications and workloads, while the compute machines actually run these applications.

Kubernetes cluster A Kubernetes cluster consists of a set of worker machines, called nodes, that run containerized applications [5]. Every cluster has at least one worker node. The worker node(s) host the Pods that are the components of the application workload.

Cluster Master The Cluster Master [24] in AKS serves as the brain of the Kubernetes cluster, orchestrating containerized applications' deployment, scaling, and management. It automates the distribution and scheduling of applications across the worker nodes, handles cluster-wide networking, and maintains the overall desired state of the cluster.

Container A standard unit of software that packages up code and all its dependencies so the application runs quickly and reliably from one computing environment to another [1], [4].

Pods The smallest deployable units of computing that can be created and managed in Kubernetes. A pod encapsulates one or more containers, storage resources, a unique network IP, and options that govern how the container(s) should run [4], [5].

Execution The process of running the application within the container, leveraging the resources and environment provided by the container platform [4], [5].

Deployment In Kubernetes, a Deployment provides declarative updates for Pods and ReplicaSets. You describe a desired state in a Deployment, and the Deployment Controller changes the actual state to the desired state at a controlled rate [4].

`cronjobs` `cronjobs` are a Kubernetes resource that allows users to run jobs (i.e., tasks or containers) on a recurring schedule, similar to cron tasks in Unix/Linux systems. A CronJob object specifies a job template and a schedule in the Cron format. Kubernetes then ensures that the jobs are created and executed based on this schedule. CronJobs are useful for automating regular maintenance tasks, running batch jobs, or executing tasks that need to happen at specific times or intervals [4], [7].

ReplicaSet A ReplicaSet's purpose is to maintain a stable set of replica Pods running at any given time. It is often used to guarantee the availability of a specified number of identical Pods [4], [5].

Replica A replica [4], [5] is an instance of a pod, which runs a set of containers. In Kubernetes, a pod represents the smallest deployable unit that can be created, scheduled, and managed. It's often a wrapper for a single container, but it can also contain multiple containers that need to work together.

Volume A directory containing data, accessible to the containers in a pod in Kubernetes, allowing for data persistence and sharing between containers [5].

Persistent Volume (PV) In Kubernetes, a PersistentVolume [8] is a piece of storage in the cluster that has been provisioned by an administrator or dynamically provisioned using Storage Classes. It is a resource in the cluster just like a node is a cluster resource. PVs are volume plugins like Volumes, but have a lifecycle independent of any individual Pod that uses the PV.

Persistent Volume Claim (PVC) A PersistentVolumeClaim [8] is a request for storage by a user. It is similar to a Pod. Pods consume node resources and PVCs consume PV resources. Pods can request specific levels of resources (CPU and Memory). Similarly, PVCs can request specific size and access modes (e.g., they can be mounted once read/write or many times read-only).

Kubelet Kubelet [15] is a fundamental component of Kubernetes, a container orchestration system. It runs on each node in a Kubernetes cluster and has the following responsibilities:

- Node registration: it registers the node with the cluster, adding it to the available pool of resources for running workloads.
- Pod lifecycle management.
- Resource monitoring and reporting.
- Executing control plane instructions.
- Volume management: it manages the mounting and unmounting of data volumes for pods, allowing for data persistence or sharing between containers within a pod.
- Log management.

Node In AKS, a node is essentially a virtual machine (VM) that serves as a worker machine for running containerized applications [6]. These nodes provide necessary resources like CPU, memory, and storage, and are managed by the Kubernetes master. Each node runs the kubelet agent to manage container states based on the master's instructions. Nodes in AKS can vary in size and type, and can be organized into node pools for managing different workloads. While AKS handles node-level operations like scaling, patching, and updating, users are responsible for the applications and containers running on these nodes. *Nodes are the hardware resources that host and run the pods, while pods are logical units that group one or more containers together, sharing specific resources like storage and networking.*

SMB Protocol SMB [18], [17], which stands for Server Message Block, is a network communication protocol used for providing shared access to files, printers, and serial ports among nodes on a network. Initially developed by IBM, it was popularized by Microsoft as it became the standard protocol for file and print sharing in Windows operating systems.

Manifest A manifest [6] in the context of computing, specifically within Kubernetes, is a YAML (Yet Another Markup Language) file that defines the desired state of an object that the user wants to create on a Kubernetes cluster, such as pods, services, and volume claims.

`SSH keys` `SSH keys` are a pair of cryptographic keys that can be used to authenticate to an SSH server as an alternative to password-based logins. An SSH key pair consists of a private key and a public key. Private Key is kept secret and secure by the user. It is used to decrypt data sent to it that has been encrypted with the public key. On the other hand, public key can be shared with anyone and is used to encrypt data before it is sent to the private key holder.

Focus on Clusters and Nodes

Figure 1 shows the structure of Kubernetes cluster and the relationship between the Cluster Master and nodes processes. Each Node runs (multiple) Pods and is managed by the Master. The scheduling of Pods is performed automatically by the Master and this takes into account the available resources on the Nodes [24] [23]. Every Kubernetes Node runs at least:

- A container runtime (like Docker, rkt) that will take care of pulling all your containers from a registry.
- Kubelet, that acts as a bridge between the Kubernetes Master and the Nodes; it manages the Pods and the containers running on a machine.

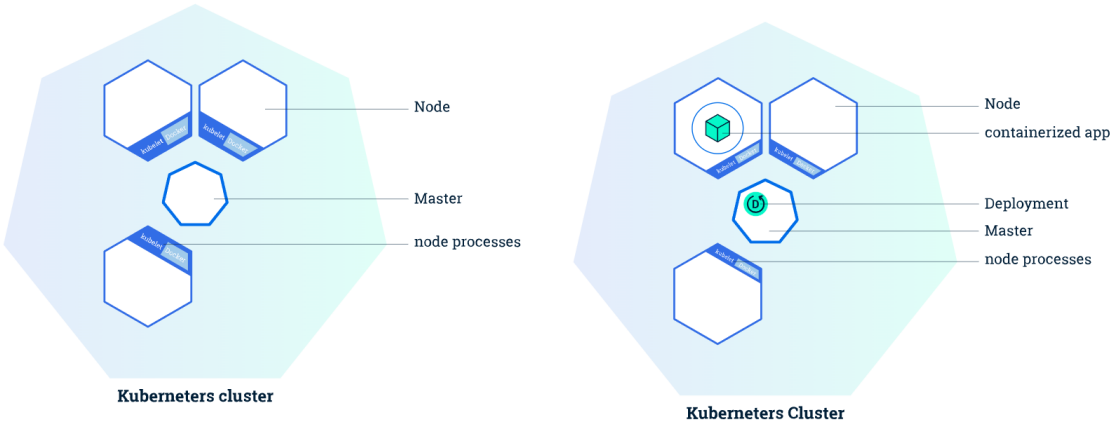

Figure 1: Cluster diagram and deployment [23]

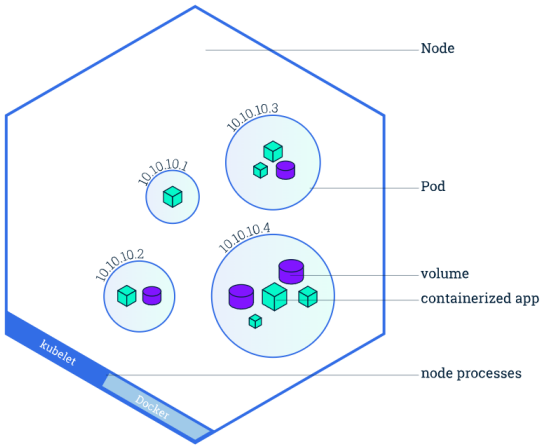

Figure 2: Node structure [23]

Pipeline

This section visually describes the deployment process from pushing a Docker image to ACR, to deploying it on AKS, and managing persistent storage with PVCs. ,

Figure 3 illustrates the general pipeline of deployment of the docker image to Azure Kubernetes Service. The process starts by pushing a Docker image (The application packaged with all its dependencies) to ACR (a service to store and manage container images). From there, AKS Cluster deploys and manages the Kubernetes service, which schedules Pods (the smallest deployable unit in Kubernetes, which can contain one or more containers). Pods mount storage from a PVC (a request for storage by a user that can be bound to a Persistent Volume) bound to a PV (Represents storage provisioned from Azure File Storage that is persistent and not tied to the pod lifecycle), which is backed by Azure File Storage. Containers within the pods execute the application.

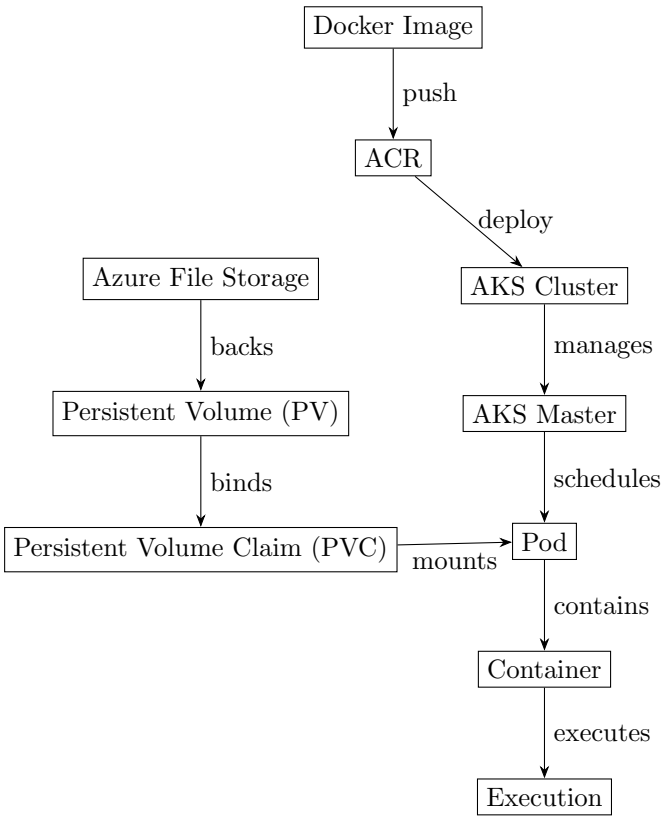

Figure 3: General pipeline

1 AKS Cluster

1.1 Prerequisites and Description

Before starting, ensure you have the Azure CLI installed and are logged into your Azure account. You will also need Docker installed on your local machine to create and push a container image.

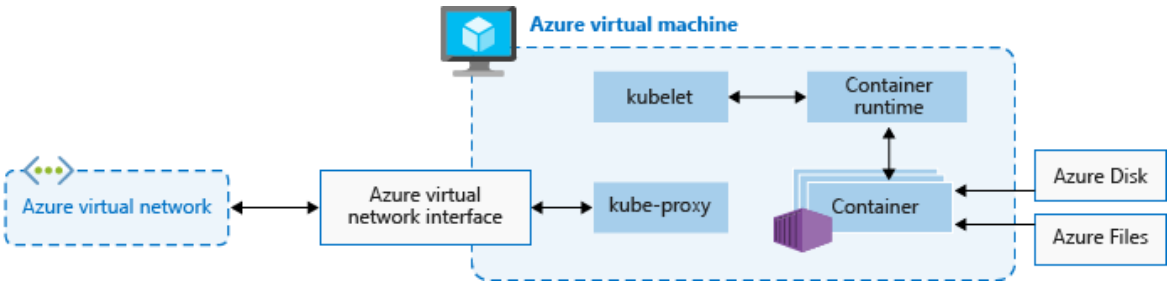

Figure 4: The diagram depicts the architecture and components involved in a Kubernetes node hosted on an Azure VM within AKS cluster. Source: MS Azure

Figure 4 depicts the way the node in AKS operates. The Azure VM acts as the node, which is part of an Azure virtual network, and is interfaced through the Azure virtual network interface. The node contains several key components:

`kubelet` : This is the primary node agent that communicates with the Kubernetes control plane, managing the state of containers on the VM as per the desired specifications.

Container runtime : It is the underlying software that is used to run containers. The kubelet interacts with the container runtime to control the lifecycle of the containers.

kube-proxy : This component maintains network rules on the node, allowing for network communication to the containers from within or outside of the Kubernetes cluster.

Container : The containers run the application workloads. Each container is an isolated environment that includes the application and its dependencies.

Azure Disk : This represents the persistent storage option for stateful applications that require data to be retained across container restarts or node redeployments.

Azure Files : This is another storage option that can be used for shared storage scenarios and can be accessed by containers running in AKS.

1.2 Setup Azure CLI and Create Resource Group

The commands provided in this section outline the process for setting up an AKS cluster, including the registration for Windows container support and the management of node pools. These steps form the basis for deploying scalable and secure containerized applications on Azure.

Figure 5 [16] illustrates the Azure Resource Manager (ARM) as the central service in Azure's architecture for managing the lifecycle of various resources like Data Stores, Web Apps, Virtual Machines, and more. Different interfaces such as the Azure portal, Azure PowerShell, Azure CLI, and REST clients interact with ARM via SDKs to perform resource management tasks, with all actions authenticated for security. ARM provides a unified and programmatic approach to deploying, managing, and monitoring Azure resources.

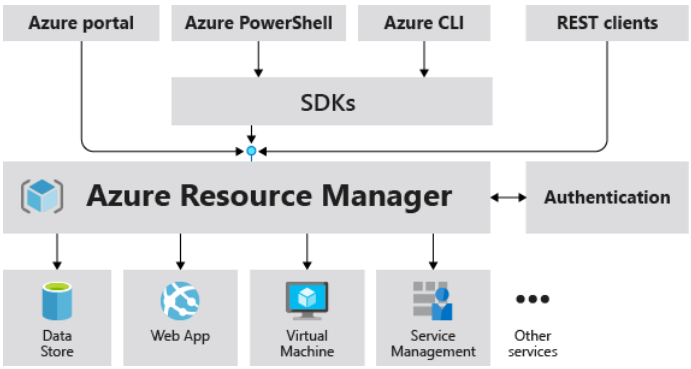

Figure 5: Azure Resource Manager [16]

First, log into Azure and create a new resource group. Resource groups help organize your Azure resources.

```
az login
az group create --name MYResourceGroup --location francecentral
```

1.3 Create Azure Container Registry

Create an ACR [11] to store Docker images. ACR allows you to build, store, and manage images for all types of container deployments.

Figure 6 showcases the workflow of deploying a Docker container using Azure services. We start with a Local PC, where a Docker image is created. This image is then pushed to the ACR, a private repository for managing and storing Docker container images. Finally, the Docker image can be pulled from ACR to Azure Container Instances (ACI), which is a service that allows you to run containers directly on the Azure cloud, without needing to manage virtual machines or additional services.

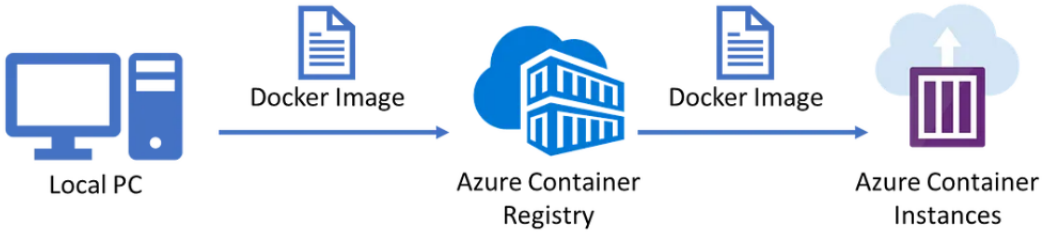

Figure 6: Azure Container Registry pipeline.

```
az acr create --resource-group MYResourceGroup --name your-registry-name --sku Basic --admin-enabled true
```

Replace `MYResourceGroup` with the name of actual resoource group.

1.4 Setting up AKS Cluster and Node Pools

1.4.1 Deploy AKS Cluster

First, create an AKS cluster named `kubermy` within the resource group `MYResourceGroup`. This cluster will be the foundation for deploying containerized applications. Monitoring addons are enabled to provide insights into the cluster's performance, and `SSH keys` are generated for secure access.

```
az aks create ‘
    --resource-group MYResourceGroup ‘
    --name kubermy ‘
    --node-count 1 ‘
    --enable-addons monitoring ‘
    --kubernetes-version 1.27.7 ‘
    --generate-ssh-keys ‘
    --location francecentral
```

1.4.2 Enable Windows Preview

Before adding a Windows node pool, enable the Windows preview feature for AKS. This is necessary for clusters that will host Windows containers.

```
az feature register --name WindowsPreview --namespace Microsoft.ContainerService
az provider register -n Microsoft.ContainerService
```

1.5 Add Node Pools

1.5.1 Windows Node Pool

Add a Windows node pool named `winnp` to your AKS cluster. This node pool will allow you to deploy Windows containers alongside Linux containers in the same Kubernetes cluster.

```
az aks nodepool add ‘
    --resource-group MYResourceGroup ‘
    --cluster-name myAKSCluster ‘
    --os-type Windows ‘
    --name winnp ‘
    --node-count 1 ‘
    --node-vm-size Standard_D16s_v3
```

Replace `myAKSCluster` with the name of actual AKS cluster.
More details in about node vm `Standard_D16s_v3` can be found in Appendix E.

1.5.2 Deleting Node Pools

To demonstrate node pool management, the following command deletes an example Windows node pool named `winnp`. Adjust the command as necessary for your specific deployment requirements.

```
az aks nodepool delete ‘
    --resource-group MYResourceGroup ‘
    --cluster-name myAKSCluster ‘
    --name winnp
```

1.6 Delete AKS Cluster

For completeness, the following command illustrates how to delete the AKS cluster named `myAKSCluster`. Use this command with caution, as it will remove all resources associated with the cluster.

```
az aks delete --name myAKSCluster --resource-group MYResourceGroup --yes --no-wait
```

1.7 Cluster Autoscaler

The Cluster Autoscaler [28] can automatically adjust the number of nodes in a node pool based on the demands of your workloads.

First we list the pods:

```
az aks nodepool list ‘
  --resource-group myResourceGroup ‘
  --cluster-name myAKSCluster ‘
  --output table

```

Check whether autoscaler is enabled:

```
az aks nodepool show ‘
  --resource-group myResourceGroup ‘
  --cluster-name myAKSCluster ‘
  --name myNodePool
```

And if autoscaling is disabled, i.e. `"enableAutoScaling": true`, then

```
az aks nodepool update ‘
  --resource-group CEBPLResourceGroup ‘
  --cluster-name cebplAKSCluster ‘
  --name winnp ‘
  --enable-cluster-autoscaler ‘
  --min-count 1 ‘
  --max-count 3

```

With autoscaling enabled, Windows-based node pool will be more cost-effective, scaling down to 1 node during periods of low demand. However, because the minCount is set to 1, there will always be at least one node running, incurring costs even when idle. This is a limitation when considering cost optimization for Windows node pools, as they cannot scale to 0 like Linux node pools.

2 Azure Storage and Azure share file

2.1 Azure Storage

Figure 7 shows four primary types of Azure Storage, that include:

Blob Storage : Optimized for storing massive amounts of unstructured data, such as text or binary data, Blob Storage is ideal for serving images or documents directly to a browser, storing files for distributed access, and streaming video and audio. It can be mounted as a file system on Linux, allowing for seamless integration with Linux-based applications and services. However, *it is not natively mountable on Windows machines.*

File Storage : Provides shared storage for applications using the standard SMB protocol. It is ideal for lift-and-shift applications, and for sharing files across cloud and on-premises deployments. It allows to *mount directly on Windows VMs.*

Queue Storage : Supports messaging for workflow processing and for communication between components of cloud services.

Table Storage : Offers highly available, massively scalable storage, that is ideal for applications requiring a broad, flat namespace that stores structured, non-relational data.

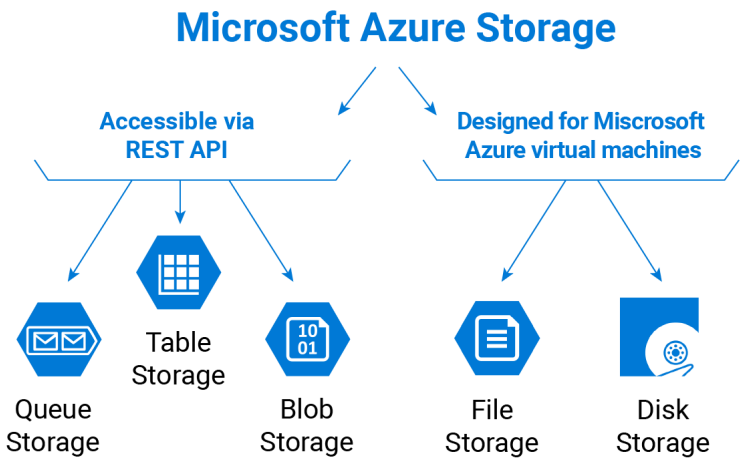

Figure 7: Storage types in Microsoft Azure Storage.

2.2 Azure share file

Azure File Share [9] provides a scalable and easily accessible file storage system that integrates seamlessly with cloud and on-premises architectures. In Kubernetes environments, especially for applications requiring access to a shared file system or maintaining state across pod restarts or scaling, persistent storage is crucial. Without it, data stored within a pod would be lost upon termination or failure, leading to data loss and inconsistency.

Figure 8 shows the Cluster Master with the API Server managing the storage resources, where a PV is provisioned from Azure Managed Disk or Azure Files. This PV is then claimed by a PVC from a Pod running on a Node within the AKS cluster, allowing the Pod to access and utilize the persistent storage as needed for stateful applications.

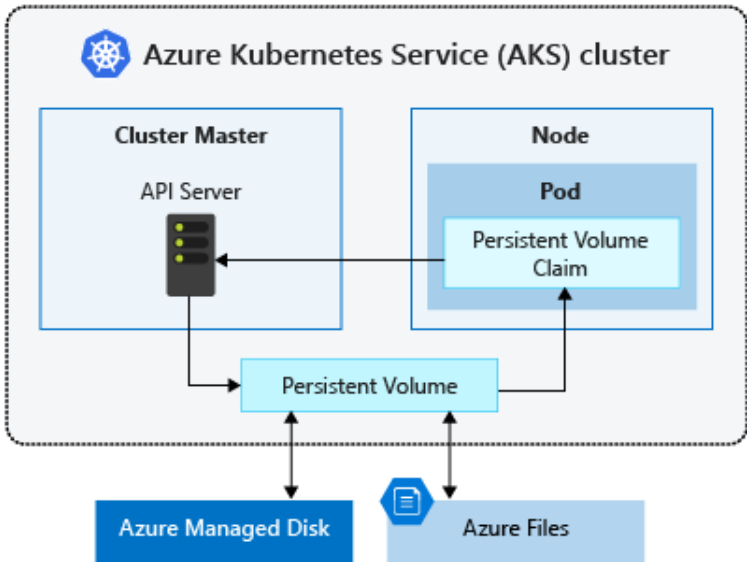

Figure 8: The storage architecture within an AKS cluster.

Steps

1. **Create Azure Storage Account**:

```
az storage account create --name mystorage --resource-group MYResourceGroup --location
↪ francecentral --sku Standard_LRS
```

where `Standard_LRS` (Standard Locally Redundant Storage) in Azure ensures that your data is replicated three times within a single storage cluster in a single data center in the same region.

Replace `mystorage` with the name of actual storage.

2. **Create Azure File Share of size 100GB**:

```
az storage share create --name myfileshare --account-name mystorage --quota 100
```

Replace `myfileshare` with the name of actual fileshare.

3. **Retrieve Storage Account Key**:

```
STORAGE_KEY=$(az storage account keys list --resource-group MYResourceGroup --account-name
↪ mystorage --query "[0].value" -o tsv)
```

4. **Create Kubernetes Secret**:

```
kubectl create secret generic azure-secret --from-literal=azurestorageaccountname=mystorage --
↪ from-literal=azurestorageaccountkey=$STORAGE_KEY
```

5. **Deploy PV and PVC**. Detailed YAML manifest configurations can be found in Appendix A.1 and A.2:

```
kubectl apply -f azurefile-pv.yaml
kubectl apply -f azurefile-pvc.yaml
```

6. **Verify PVC Binding**:

```
kubectl get pvc azurefile
```

7. **Force Delete PVC (If Needed)**:

```
kubectl patch pvc azurefile -p "{\"metadata\":{\"finalizers\":[]}}"
```

2.3 Access to Azure File Share

In order to have access to Azure File Storage you need Access keys. Retrieve the keys.

```
az storage account keys list --account-name mystorage --output table
```

- Choose a key to use: You can use either of these keys to access your Azure File Share.
- On your Windows system, open 'File Explorer'. Right-click on 'This PC' and choose 'Map network drive'.
- Select a drive letter of your choice. For the folder, enter the file share path in the format: `mystorage.file.core.windows.net\myfileshare`
- When prompted, enter the storage account name as the username. Use the key value copied earlier as the password.

3 Application Deployment

3.1 Prepare Application for Deployment

Build your Docker image [2] and push it to the ACR. Replace `your-registry-name` with your ACR name. Make sure that the `.Dockerfile` is ready, an example can be found in Appendix I.

```
docker build -t your-registry-name.azurecr.io/my-app:latest .
az acr login --name your-registry-name
docker push your-registry-name.azurecr.io/my-app:latest
```

The script shows how to build a Docker image, log in to ACR, and push the image to the ACR . First, it builds a Docker image named `my-app:latest` from the current directory. Second, it logs into an ACR specified by `your-registry-name`, and then pushes the newly created image to the same ACR. The `-t` flag tags the built image with the name `your-registry-name.azurecr.io/my-app` and the tag `latest`. In particular, `your-registry-name.azurecr.io` typically refers to the address of a specific container registry (in this case, ACR) This process insures that the Docker images are stored in Azure, making them accessible for deployment in cloud environments.

We can list all ACRs in the Azure's subscription.

```
az acr list --output table
az acr list --resource-group [ResourceGroupName] --output table
```

3.2 Deploy Application

kubectl is the command line tool for interacting with Kubernetes clusters. Kubernetes, often abbreviated as K8s, is an open-source platform designed to automate deploying, scaling, and operating application containers.

1. Deploy the application to AKS using a deployment configuration file. This creates a pod running your application. More details on `YAML` manifest configuration can be found in Appendix C.1.

```
kubectl apply -f deployment.yaml
```

2. Verify the deployment and watch the pod's status to ensure the application runs correctly.

```
kubectl get deployment my-app-deployment
kubectl get pods -l app=my-app --watch
```

where `-l` means to filter the list of pods to only show those with a label `app` that has the value `my-app`. And the flag `-watch` is used to continuously monitor the state of the specified resources in real-time.

The name of deployment corresponds to the name define in the `YAML` manifest, see in Appendix C.1. It can also be found in CLI:

```
kubectl get deployments
```

3. If you need to access the PowerShell within a running pod, use the following command. Replace `[your-pod-name]` with the name of your pod.

```
kubectl exec -it [your-pod-name] -- powershell
```

4. Delete the deployment if necessary, or scale it down to prevent the application from restarting.

```
kubectl delete deployment my-app-deployment
kubectl scale deployment my-app-deployment --replicas=0
```

3.3 Managing `cronjobs`

The `kubectl` command is used to create a `cronjob` object which, upon the scheduled time, spawns a `Job` object. This, in turn, creates one or more pods within the Kubernetes cluster to execute the tasks. Figure 9 describes the cronjob's pipeline [14].

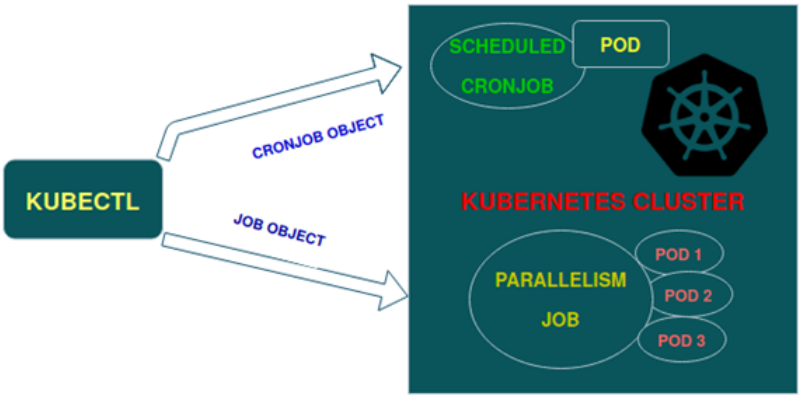

Figure 9: `cronjob`'s scheme.

Main steps to deploy and manage `cronjob`:

1. Deploy `cronjob` to schedule the application tasks. The `cronjob` configuration is defined in `cronjob.yaml`. More details on `YAML` manifest configuration can be found in Appendix B.1.

```
kubectl apply -f cronjob.yaml
```

Apply a configuration to a resource by file name or stdin. `-f` means filename that contains a configuration [19].

2. Inspect the cron job to verify its configuration and schedule.

```
kubectl describe cronjob my-app-cronjob
```

3. For manual testing of the `cronjob`, create a job from the `cronjob` definition and observe its execution.

```
    kubectl create job --from=cronjob/my-app-cronjob test-my-job
    kubectl get job test-my-job
    kubectl get pods --selector=job-name=test-my-job
    kubectl logs <pod-name>
    kubectl delete job test-my-job

```

   Replace `myapp-cronjob` with the name of actual `cronjon`.

4. Retrieve AKS credentials to interact with your cluster.

```
    az aks get-credentials --resource-group MYResourceGroup --name myAKSCluster

```

5. View all pods in wide format to check their statuses and node allocations.

```
    kubectl get pods -o wide

```

   where `-o wide` is used to outpute more detailed information like the node name on which each pod is running, the pod's IP address etc.

6. Based on the schedule defined in the `cronjob`, it will create Jobs at specific times. We can check if any Jobs have been created by `cronjob` and if a Job has been created, we can check its logs to see if it executed successfully.

```
    kubectl get jobs
    kubectl logs job/<job-name>

```

3.4 Accessing the Application

Execute the following command to access the PowerShell interface of the running container. Replace `[your-pod-name]` with the name of your pod.

```
kubectl exec -it [your-pod-name] -- powershell
```

3.5 Cleanup

After testing, you can delete the deployment or scale it down to zero replicas to stop the application.

```
kubectl delete deployment my-app-deployment
kubectl scale deployment my-app-deployment --replicas=0
```

4 Managing AKS Clusters and Node Pools

4.1 Listing AKS Clusters

To view all AKS clusters within your Azure subscription, use the following command. This can help you verify the existence and status of your clusters.

```
az aks list --output table
```

4.2 Managing Node Pools

4.2.1 Listing Node Pools

To list all node pools associated with a specific AKS cluster, use the command below. This provides a quick overview of the resources allocated to your cluster.

```
az aks nodepool list --resource-group MYResourceGroup --cluster-name myAKSCluster --output table
```

4.2.2 Scaling Node Pools

Adjust the number of nodes in a specific node pool to meet your application's demands. This example scales the Windows node pool to one node.

```
az aks nodepool scale --resource-group MYResourceGroup --cluster-name myAKSCluster --name winnp --node-
    ↪ count 1
```

4.3 Inspecting Cluster Nodes

4.3.1 Listing Nodes

View details about the nodes in your Kubernetes cluster directly using `kubectl`.

```
kubectl get nodes -o wide
```

4.3.2 Describing Nodes

For a more detailed view of a specific node, including its status, labels, and events, use the describe command.

```
kubectl describe node akswinnp000000
```

4.4 Deploying and Managing Applications

4.4.1 Listing Application Pods

To see the pods associated with your application, use the following label selector:

```
kubectl get pods -l app=my-app
```

4.4.2 Integrating ACR

Ensure your AKS cluster can securely pull images from your ACR by updating the AKS configuration.

```
az aks update -n myAKSCluster -g MYResourceGroup --attach-acr your-registry-name
```

4.5 Troubleshooting

4.5.1 Checking ACR Repositories

If you encounter issues with image pulls, verify the repositories and tags in your ACR.

```
az acr repository list --name your-registry-name --output table
az acr repository show-tags --name your-registry-name --repository my-app --output table
```

Summary

By following the steps outlined in this tutorial, you should now have a solid understanding of the process required to deploy containerized applications in AKS. From setting up the ACR to storing your Docker images, to managing persistent storage with Azure Files or Azure Managed Disks, and orchestrating these components within an AKS cluster, you are equipped to leverage the powerful capabilities of Azure for your containerized workloads.

A PV and PVC in Kubernetes

Persistent Volumes (PVs) and Persistent Volume Claims (PVCs) [9] are part of the Kubernetes storage API. They allow for storage resources to be provisioned and consumed in a way that decouples the specifics of storage from how it is used by Pods.

A.1 PersistentVolumeClaim (PVC)

A PVC is a request for storage by a user. It is similar to a Pod in that Pods consume node resources and PVCs consume PV resources.

Figure 10 illustrates the process of implementing persistent storage in an AKS cluster. It shows how Azure Managed Disks (available in Premium or Standard storage tiers) and Azure Files (also in Premium or Standard tiers) serve as the underlying persistent storage for a PV in AKS. A PVC is issued by a user or a Kubernetes process, which is then matched with a PV by AKS based on the defined Storage Class. The PVC is used by a Pod running on a Node within the AKS cluster, allowing the Pod to access the persistent storage for its data needs.

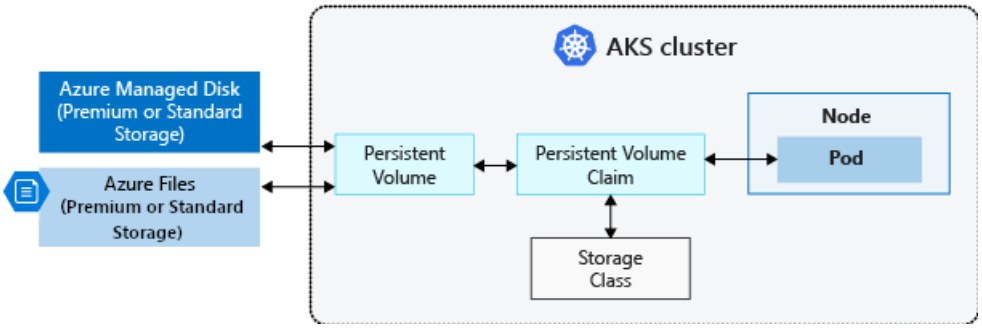

Figure 10: Persistent Storage in AKS

The following `YAML` manifest allows to create a PVC on AKS.

```yaml
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: azurefile
spec:
  accessModes:
    - ReadWriteMany  # Can be mounted by multiple nodes
  storageClassName: ""  # Empty to use the default StorageClass
  resources:
    requests:
      storage: 100Gi  # Request 100 GB of storage
  volumeName: azurefile-pv  # Binds to a specific PersistentVolume
```

- `accessModes`: Describes how the volume can be accessed. `ReadWriteMany` means the volume can be mounted as read-write by many nodes.
- `storageClassName`: Specifies the StorageClass to be used. Leaving it empty means the default StorageClass is used.
- **resources.requests.storage**: The amount of storage requested.
- `volumeName`: Specifies the binding to a specific PV. This is usually not required unless binding to a pre-provisioned PV.

A.2 PersistentVolume (PV)

A PV is a piece of storage in the cluster that has been provisioned by an administrator or dynamically provisioned using Storage Classes. In `YAML` manifest replace `myfileshare` with the name of actual file share.

```yaml
apiVersion: v1
kind: PersistentVolume
metadata:
```

```
  name: azurefile-pv
spec:
  capacity:
    storage: 100Gi  # Total capacity of the PV
  accessModes:
    - ReadWriteMany  # Matches the access mode in PVC
  persistentVolumeReclaimPolicy: Retain  # Data is retained even after PVC is deleted
  azureFile:
    secretName: azure-file-secret  # Secret containing Azure storage account info
    shareName: myfileshare  # The name of the Azure File Share
    readOnly: false  # The volume can be mounted as read-write
```

- **capacity.storage**: The amount of storage available on this PV. It should match or exceed the size requested in the PVC.
- `persistentVolumeReclaimPolicy`: Defines what happens to the PV data after the PVC is released. `Retain` keeps the data on the volume.
- `azureFile`: Specifies the Azure File Storage configuration.
- `secretName`: The name of the Kubernetes secret that contains the Azure storage account name and key.
- `shareName`: The name of the Azure file share.
- `readOnly`: Indicates whether the file share is read-only or read-write.

B `cronjob` Configuration in Kubernetes

A `cronjob` [7] in Kubernetes is used to run jobs on a time-based schedule, similar to the cron utility in Unix-like operating systems. The following YAML configuration defines a CronJob to run a containerized application on a schedule.

B.1 `cronjob`

The cron format is a simple, yet powerful scheduling syntax used to configure the time and frequency at which jobs are executed in a Unix-like operating system. The format consists of five fields that Cron converts into a set of time intervals.

```
*    *    *    *    *  command to be executed
-    -    -    -    -
|    |    |    |    |
|    |    |    |    +----- day of week (0 - 7) (0 or 7 is Sun, or use names)
|    |    |    +------- month (1 - 12)
|    |    +--------- day of month (1 - 31)
|    +------------ hour (0 - 23)
+-------------- min (0 - 59)
```

- The first field represents the minute and accepts values from 0 to 59.
- The second field represents the hour and accepts values from 0 to 23.
- The third field is the day of the month, which accepts values from 1 to 31.
- The fourth field is the month of the year, accepting values from 1 to 12 or the short names of the months (e.g., ‘Jan‘, ‘Feb‘, ‘Mar‘, etc.).
- The fifth field is the day of the week, accepting values from 0 to 7, where both 0 and 7 represent Sunday. Alternatively, you can use the short names of the days (e.g., ‘Sun‘, ‘Mon‘, ‘Tue‘, etc.).

Each field can have a single value, a list of values, a range of values, or a wildcard `*`, which represents all possible values for that field. For example, a cron schedule of `0 11 * * Mon,Wed-Fri` means the job will run at 11:00 AM on Monday, Wednesday, Thursday, and Friday. In the `YAML` below replace `my-app-cronjob`, `my-app`, `my-volume`, `your-registry-name` with corresponding actual names.

```yaml
apiVersion: batch/v1
kind: CronJob
metadata:
  name: my-app-cronjob
spec:
  schedule: "0 11 * * 1,3-5"  # At 11:00 on Monday, Wednesday, Thursday, and Friday
  jobTemplate:            # Template for the job to be run
    spec:
      template:
        spec:
          nodeSelector:
            "kubernetes.io/os": windows  # Ensure the job runs on a Windows node
          containers:
          - name: my-app
            image: your-registry-name.azurecr.io/my-app:latest  # Container image to
    ↪ use
            resources:
              requests:
                memory: "48Gi"  # Memory request for the container
                cpu: "12"       # CPU request for the container
            volumeMounts:
            - name: my-volume
              mountPath: "/data"  # Mount path for the volume inside the container
          volumes:
          - name: my-volume
            persistentVolumeClaim:
              claimName: azurefile  # PVC that the CronJob will use
          restartPolicy: OnFailure  # Job restart policy
```

- `schedule`: Specifies when the job should be run using the Cron format.
- `nodeSelector`: Ensures that the job is scheduled on nodes with the specified label, in this case, Windows nodes.
- **containers.image**: Defines which container image should be run.
- **resources.requests**: Specifies the computational resources required for the container. Here it requests 48 GiB of memory and 12 CPU units.
- `volumeMounts`: Defines the volumes to be mounted into the containers and their mount paths.
- **volumes.persistentVolumeClaim**: Links the volume defined in the pod to the PVC which provides the actual storage.
- `restartPolicy`: The policy to apply when a job fails. `OnFailure` means the job will be retried until it succeeds.

This `cronjob` is configured to run the `my-app` container daily at 11:00 AM, ensuring it has access to the required CPU and memory resources and storage volume.

C Kubernetes Deployment Configuration

A Deployment in Kubernetes [10] is used to declare replicas of pods to run in your cluster. The following YAML configuration specifies the deployment details for the application.

C.1 Deployment

In the YAML `deployment.yaml` below replace `my-app-deployment`, `my-app`, `my-volume`, `your-registry-name` with corresponding actual names.

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: my-app-deployment  # Name of the deployment
spec:
  replicas: 1  # Number of replicas to run
  selector:
    matchLabels:
      app: my-app  # Label selector to identify the pods
  template:  # Template for the pods
    metadata:
      labels:
        app: my-app  # Labels assigned to the pods
    spec:
      nodeSelector:
        "kubernetes.io/os": windows  # Run the pods on Windows nodes
      containers:
      - name: my-app
        image: your-registry-name.azurecr.io/my-app:latest  # Container image to
    ↪ deploy
        resources:
          requests:
            memory: "48Gi"  # Memory request for the container
            cpu: "12"       # CPU request for the container
        volumeMounts:
        - name: my-volume
          mountPath: "/data"  # Mount path for the volume inside the container
      volumes:
      - name: my-volume
        persistentVolumeClaim:
          claimName: azurefile  # PVC that the deployment will use
```

- `replicas`: Defines the number of instances of the pod that should be running.
- **selector.matchLabels**: Determines how the Deployment finds which pods to manage.
- **template.metadata.labels**: Provides the labels for the pods that will be created.
- `nodeSelector`: Ensures the pods are scheduled on nodes with specific labels, in this case, Windows nodes.
- **containers.image**: Specifies which image should be used for creating the containers.
- **resources.requests**: Indicates the minimum amount of resources the container should have.
- `volumeMounts`: Configures the mounting of a volume within the container.
- **volumes.persistentVolumeClaim**: Connects a persistent volume claim (which provides persistent storage) to the pods.

This deployment configuration ensures that a single instance of the `my-app` container is running, with the required resources and storage volume, on a Windows node within your Kubernetes cluster.

D Integration into Github Actions

The description of the GitHub Actions workflow for deploying to AKS with Windows Containers [12], [13].

In the `YAML` below replace `myAKSCluster`, `myResourceGroup`, `myapp`, `your-registry-name` with corresponding actual names and `./path/to/cronjob.yaml` with actual path. The file `aks_deploy.yml` should be saved in `.github/workflows`.

```yaml
name: Deploy to AKS with Windows Containers

on:
  push:
    branches:
      - main

jobs:
  build-and-deploy:
    runs-on: windows-latest

    steps:
      - name: Checkout code
        uses: actions/checkout@v2

      - name: Login to Azure
        uses: azure/login@v1
        with:
          creds: ${{ secrets.AZURE_CREDENTIALS }}

      - name: Build and push Docker image
        run: |
          docker build -t ${{ secrets.REGISTRY_LOGIN_SERVER }}/myapp:latest -f ./Dockerfile .
          docker login ${{ secrets.REGISTRY_LOGIN_SERVER }} -u ${{ secrets.REGISTRY_USERNAME }} -p ${{
    ↪ secrets.REGISTRY_PASSWORD }}
          docker push ${{ secrets.REGISTRY_LOGIN_SERVER }}/myapp:latest

      - name: Set Kubernetes context
        uses: azure/aks-set-context@v1
        with:
          creds: ${{ secrets.AZURE_CREDENTIALS }}
          cluster-name: myAKSCluster
          resource-group: myResourceGroup

      - name: Deploy CronJob to AKS
        run: |
          kubectl apply -f ./path/to/cronjob.yaml
```

Workflow Name: `name: Deploy to AKS with Windows Containers`

- Specifies the name of the workflow, serving as a label for identification in the GitHub Actions UI.

Trigger: `on: push: branches: - main`

- Defines when the workflow should be triggered, configured to run on pushes to the `main` branch of the repository.

Job Definition: `jobs: build-and-deploy: runs-on: windows-latest`

- Defines a job named `build-and-deploy`.
- Specifies that the job should run on a Windows-based runner (`windows-latest`).

Steps:

1. **Checkout Code**:
   - Uses the `actions/checkout@v2` action to check out the repository code into the GitHub runner.
2. **Login to Azure**:

- Utilizes the `azure/login@v1` action for Azure authentication, requiring the `AZURE_CREDENTIALS` secret with Azure Service Principal JSON credentials.

3. **Build and Push Docker Image**:
   - Builds the Docker image from the Dockerfile and pushes it to the ACR.
4. **Set Kubernetes Context**:
   - Employs the `azure/aks-set-context@v1` action to set the Kubernetes context for the specified AKS cluster and resource group.
5. **Deploy CronJob to AKS**:
   - Executes `kubectl apply` to deploy the CronJob to the AKS cluster, with the path `./path/to/your/cronjob.yaml` replaced by the actual path to the CronJob YAML file in the repository.

The workflow is designed to automate the deployment process to AKS, with the primary requirement being the adjustment of the path to the CronJob YAML file. The rest of the workflow utilizes predefined secrets and settings.

E Selection of Azure Virtual Machine

The list of VMs available in Azure can be found in [20].

In selection of VM the following factors are important:

- Workload Requirements: Understand the demands of the applications in terms of CPU, memory, storage, and network performance.
- Scalability and Performance.
- Availability of VM Sizes in Your Region: Not all VM sizes are available in every Azure region.
- Cost: Larger VMs are more expensive.

For example, `Standard_D16s_v3`, is part of the D-series Azure VMs, optimized for applications that require a balance of CPU and memory: 16 CPU cores, 64 GB of RAM and `D16s` suggests that it supports Premium SSDs.

Figure 11 shows the pricing for `Standard_D16s_v3` vm.

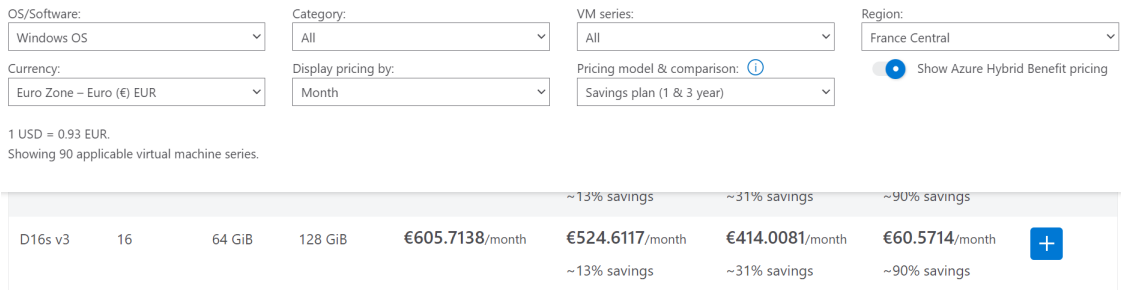

Figure 11: Prices for a `Standard_D16s_v3` vm

Figure 12 shows the price for `Standard_D16s_v3` vm in Pay as you go option [29].

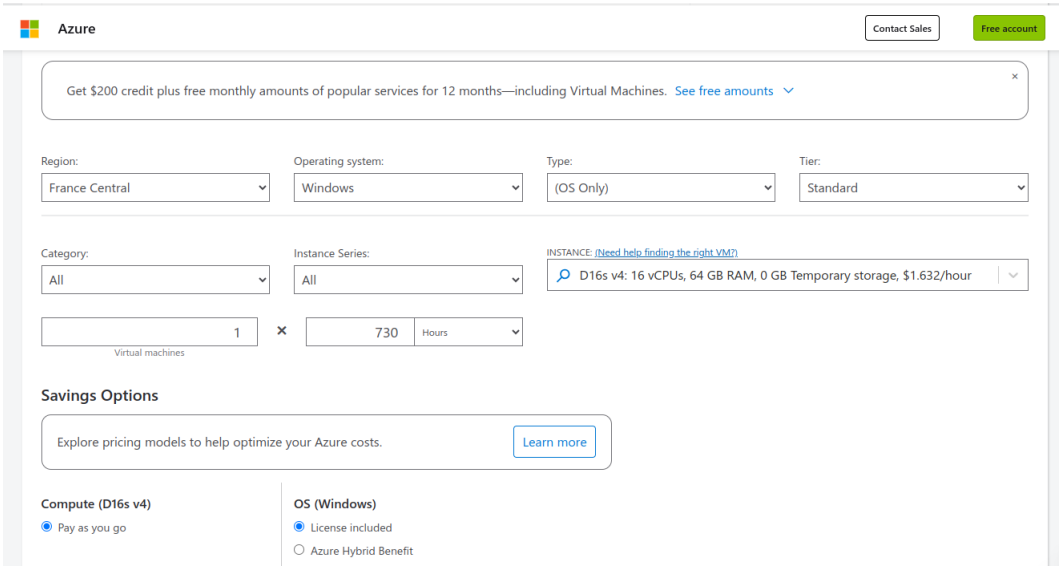

Figure 12: Prices for a `Standard_D16s_v3` vm in the Pay as you go option

F Azure Storage Selection

Figure 13 shows the prices for different options of storage [29].

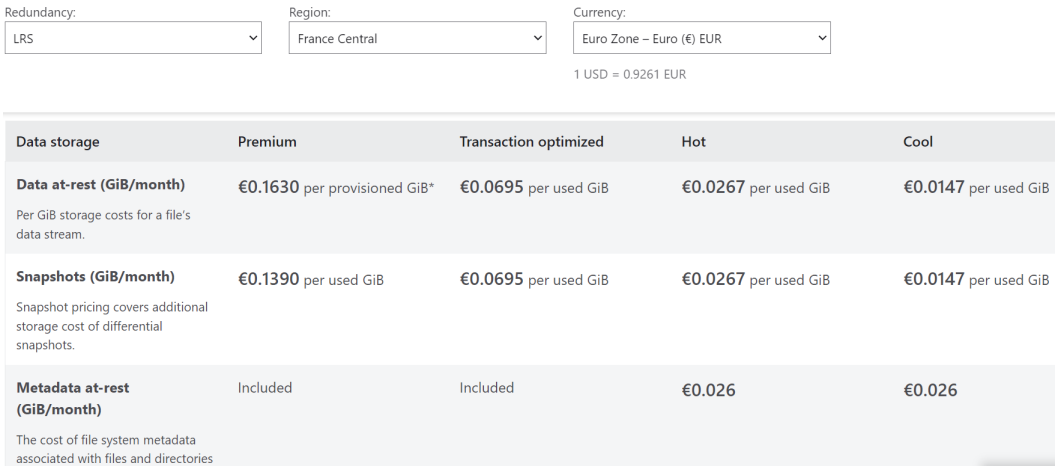

Figure 13: Prices in Azure Files

The code below allows you to extract the information about file storage account.

```
az storage account list --output table
az storage share list --account-name [StorageAccountName] --output table
az storage share show --name [ShareName] --account-name [StorageAccountName]
```

In the output, we see that the size of the file share is set to 100 GB and `AccessTier: "TransactionOptimized"` shows that "TransactionOptimized" performance tier is chosen.

```
{
  "accessTier": "TransactionOptimized",
  "deleted": null,
  "deletedTime": null,
  "lease": {
    "duration": null,
```

```
    "state": "available",
    "status": "unlocked"
  },
  "metadata": {},
  "name": "myfileshare",
  "nextAllowedQuotaDowngradeTime": null,
  "properties": {
    "etag": "\"0x8DC29BBDD441DBD\"",
    "lastModified": "2024-02-09T22:10:02+00:00",
    "quota": 100
  },
  "protocols": null,
  "provisionedBandwidth": null,
  "provisionedEgressMbps": null,
  "provisionedIngressMbps": null,
  "provisionedIops": null,
  "remainingRetentionDays": null,
  "rootSquash": null,
  "snapshot": null,
  "version": null
}
```


More details can be found in [22] and [21].

G AKS: Cost Analysis

In the context of Azure Kubernetes Service (AKS), the "Pay-As-You-Go" pricing model means the user is billed for the resources he or she uses as the resources are used, without any upfront commitment. The cost primarily consists of:

- VM Costs: The costs for the virtual machines that are part of user's AKS node pools based on their size (vCPU, RAM) and the time they are running. If the node pool has autoscaling enabled, the cost can vary depending on the number of VMs running at any given time within the autoscale range set.
- AKS Service Overhead: Azure charges a small fee for the Kubernetes management service provided by AKS includinf Azure File Share.

Figure 14 shows weekly cost of using AKS service with a selected VM. We can see that the cost of running Windows node pool costs 254€, in other words, $254€ / (24 \times 7) = 1.51€$ per hour, which approximately corresponds to the price in Figure 12.

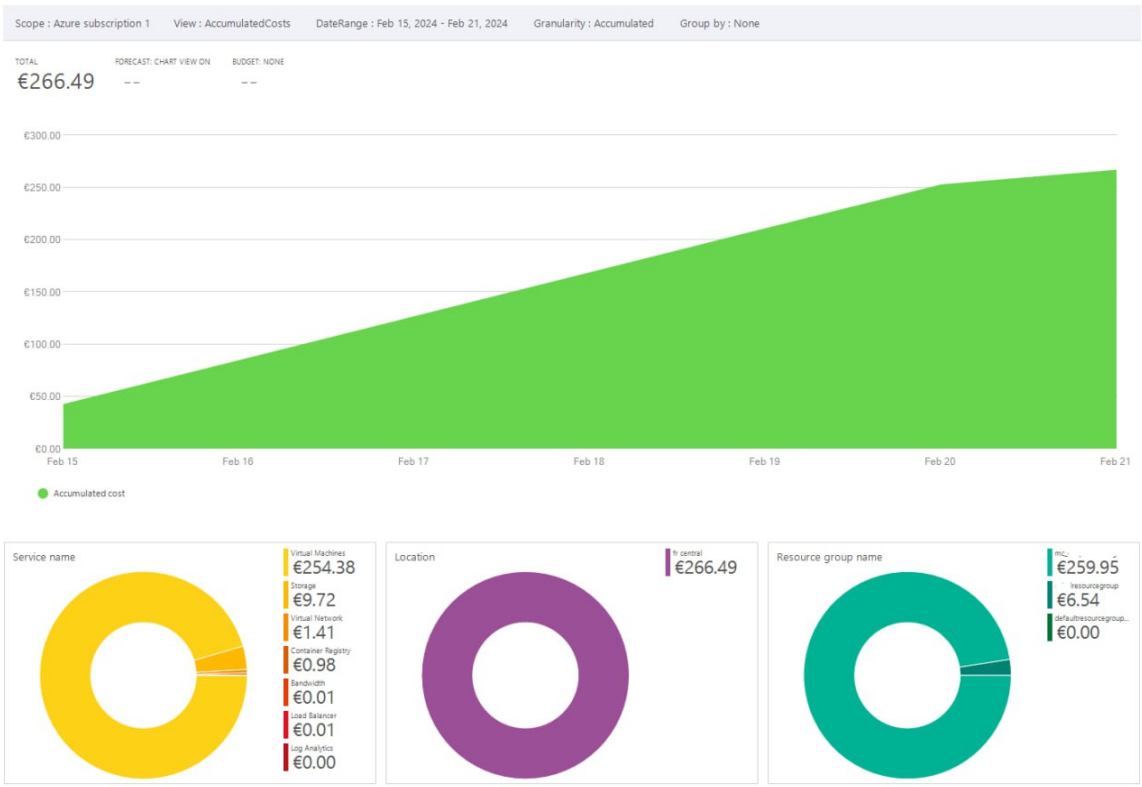

Figure 14: Cost analysis charts. The chart shows an incremental cost of using VM, storage during a period of one week

The table in Figure 15 details the cost of running one vm.

ACTUAL COST (EUR ONLY) FORECAST: CHART VIEW ON BUDGET: NONE

€533.54 -- --

Group by: Resource Granularity: None Table

Filter items 22 rows

Resource	Resource type	Location	Resource group n...	Tags	Cost
akswinnp	Virtual machine scale set	fr central	mc_cebplresourcegroup...	aks-managed-consolid...	€452.37

Service name	Meter	Cost
Virtual Machines	D16 v3/D16s v3	€452.36
Bandwidth	Intra Continent Data Transfer Out	<€0.01
Bandwidth	Inter Continent Data Transfer Out - NAM or EU To A...	€0
Bandwidth	Standard Data Transfer Out - Free	€0

Figure 15: Cost of one VM `Standard_D16s_v3`

Figure in 16 shows an incremental cost of each unity in the AKS pod. We notice that 85 % of cost os represented by `Standard_D16s_v3` VM.

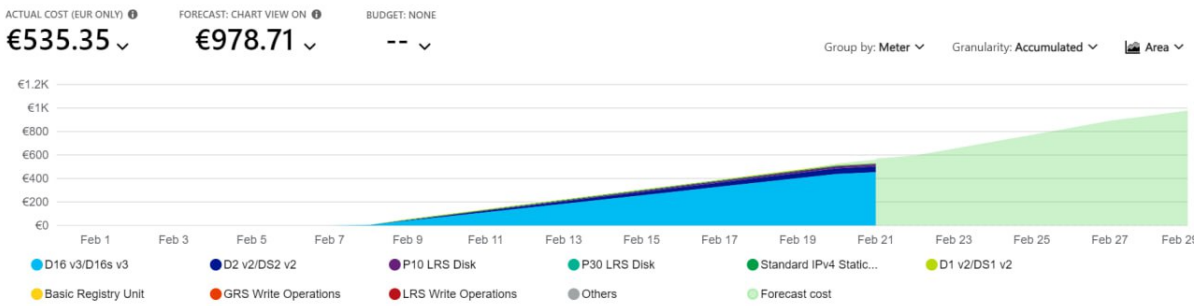

Figure 16: An incremental cost of each unity in the AKS pod.

H AKS Resources Analysis

Lets first list all the nodes and see which node they belong to by checking corresponding labels:

```
kubectl get nodes -o wide
```

and the output is

```
NAME                                STATUS   ROLES   AGE     VERSION   INTERNAL-IP   EXTERNAL-IP   OS-
    ↪ IMAGE                          KERNEL-VERSION      CONTAINER-RUNTIME
aks-nodepool1-20215633-vmss000000   Ready    agent   2d12h   v1.27.7   10.224.0.4    <none>        Ubuntu
    ↪ 22.04.3 LTS                5.15.0-1054-azure   containerd://1.7.7-1
akswinnp000000                      Ready    agent   12d     v1.27.7   10.224.0.33   <none>        Windows
    ↪  Server 2022 Datacenter   10.0.20348.2227     containerd://1.6.21+azure
```

We see that there are two node pools in AKS cluster: one running Linux (`aks-nodepool1-20215633-vmss000000`) and one running Windows (`akswinnp000000`).

This command lists all pods across all namespaces, along with the names of the nodes they are scheduled on. You can narrow down the list by specifying a particular namespace if your application runs in a specific namespace:

```
kubectl get pods -o=custom-columns=NAME:.metadata.name,NODE:.spec.nodeName -n <namespace>
```

Based on the output, we see that there are several system and infrastructure pods running on the Linux node pool, including: `CoreDNS`, `Metric Server`, `CSI (Container Storage Interface) Drivers`, `Cloud Node Manager & Azure IP Masq Agent` etc.

```
my-app-cronjob-28468020-jww68      akswinnp000000
my-app-cronjob-28472340-mh8sv      akswinnp000000
my-app-cronjob-28475220-8sgrw      akswinnp000000
ama-logs-rs-7dbd7ccf49-bfbkl          aks-nodepool1-20215633-vmss000000
ama-logs-s2mv4                        aks-nodepool1-20215633-vmss000000
ama-logs-windows-gvjsq                akswinnp000000
azure-ip-masq-agent-jxn47             aks-nodepool1-20215633-vmss000000
cloud-node-manager-qdx6c              aks-nodepool1-20215633-vmss000000
cloud-node-manager-windows-6kkgm      akswinnp000000
coredns-789789675-hzsls               aks-nodepool1-20215633-vmss000000
coredns-789789675-x7kjc               aks-nodepool1-20215633-vmss000000
coredns-autoscaler-649b947bbd-wmdk7   aks-nodepool1-20215633-vmss000000
csi-azuredisk-node-gzsp7              aks-nodepool1-20215633-vmss000000
csi-azuredisk-node-win-g4786          akswinnp000000
csi-azurefile-node-jvmjk              aks-nodepool1-20215633-vmss000000
csi-azurefile-node-win-sb7z8          akswinnp000000
konnectivity-agent-79564ddd7b-nhz65   aks-nodepool1-20215633-vmss000000
konnectivity-agent-79564ddd7b-wkj5r   aks-nodepool1-20215633-vmss000000
kube-proxy-kz6rr                      aks-nodepool1-20215633-vmss000000
metrics-server-5955767688-5q4bl       aks-nodepool1-20215633-vmss000000
metrics-server-5955767688-lxkn2       aks-nodepool1-20215633-vmss000000
```

I Dockerfile manifest

This `.Dockerfile` is designed for creating a Windows-based Docker image that sets up a Python development environment with Git, and is tailored for deploying a Python application.

```
# Use a Windows base image
ARG GITHUB_TOKEN

FROM mcr.microsoft.com/windows/servercore:ltsc2022

# Set the working directory in the container
WORKDIR C:\\myapp

# Use PowerShell for subsequent commands
SHELL ["powershell", "-Command", "$ErrorActionPreference = 'Stop';"]

# Install Python and Git
# Note: Replace the URLs with the actual URLs for Python and Git installers
RUN Invoke-WebRequest -Uri 'https://www.python.org/ftp/python/3.9.0/python-3.9.0-
    ↪ amd64.exe' -OutFile 'python_installer.exe' -UseBasicParsing; \
    Start-Process python_installer.exe -ArgumentList '/quiet InstallAllUsers=1
    ↪ PrependPath=1' -Wait; \
    Remove-Item python_installer.exe -Force; \
    Invoke-WebRequest -Uri 'https://github.com/git-for-windows/git/releases/download/
    ↪ v2.28.0.windows.1/Git-2.28.0-64-bit.exe' -OutFile 'git_installer.exe' -
    ↪ UseBasicParsing; \
    Start-Process git_installer.exe -ArgumentList '/VERYSILENT /NORESTART /NOCANCEL /
    ↪ SP-' -Wait; \
    Remove-Item git_installer.exe -Force
    # Invoke-WebRequest -Uri 'https://curl.se/ca/cacert.pem' -OutFile 'cacert.pem'; \
    # [System.Environment]::SetEnvironmentVariable('GIT_SSL_CAINFO', 'C:\codpy\cacert
    ↪ .pem', [System.EnvironmentVariableTarget]::Machine); \
    # Remove-Item cacert.pem -Force

# Add Python and Git to PATH (replace with the actual installation paths if different
    ↪ )
RUN $env:Path += ';C:\\Program Files\\Python39;C:\\Program Files\\Python39\\Scripts;C
    ↪ :\\Program Files\\Git\\cmd'; \
    [Environment]::SetEnvironmentVariable('Path', $env:Path, [
    ↪ EnvironmentVariableTarget]::Machine)

# Set the GitHub token as an environment variable
ENV GITHUB_TOKEN=$GITHUB_TOKEN

# Use the token to install the codpy package from the private repository
RUN $Env:GITHUB_TOKEN='YOURGITHUBTOKEN'; \
    git clone https://yourrepo:$Env:GITHUB_TOKEN@github.com/yourrepo/myapp.git; \
    cd myapp; \
    pip install .

# Reset the working directory to C:\app
WORKDIR C:\\app

# Copy the content of the local src directory to the working directory
COPY . C:\\app

# Install any dependencies from requirements.txt
RUN pip install --no-cache-dir -r requirements.txt
```

```
45
46 # If necessary,  copy the specific .pyd file to the site-packages directory
47 COPY ["yourapp.cp39-win_amd64.pyd", "C:/Program Files/Python39/Lib/site-packages/"]
48
49 # Command to run on container start
50 #CMD ["powershell", "-Command", "while ($true) { Start-Sleep -Seconds 3600 }"]
51
52 CMD ["python", "C:\\app\\main.py"]
```

Base Image : The image starts from a Windows Server Core base image (`ltsc2022`). This provides the underlying Windows operating system environment.

Working Directory : It sets the working directory inside the container to `C:/ myapp`, which is where subsequent commands will operate.

Shell Setting : The shell is set to PowerShell with error preferences, ensuring that subsequent commands are run using PowerShell and that any errors stop the process.

Python and Git Installation : It installs Python and Git by downloading their installers from specified URLs and running them silently. After installation, the installers are removed to keep the image size down.

Path Environment Variable : The `Dockerfile` appends the Python and Git executable paths to the system PATH environment variable, allowing these programs to be run from any location within the container.

GitHub Token : Sets an environment variable `GITHUB_TOKEN` using the build argument `ARG GITHUB_TOKEN`, which is intended to be used for secure access to private repositories.

Clone and Install Python Application : It uses the provided GitHub token to clone a Python application from a private GitHub repository. Then it changes to the application directory and installs it using pip. This process assumes that the application has a setup.py file for installation.

Reset Working Directory : The working directory is reset to `C:/ app`.

Copy Application Files : Copies the application files from the local source directory to the container's working directory.

Install Dependencies : Installs any dependencies specified in requirements.txt using pip.

Copy `.pyd` File : If necessary, copies a specific `.pyd` file (a Python extension module) to the Python site-packages directory, which is where Python looks for installed packages.

Set Startup Command : Finally, it sets the default command to run the Python application (main.py) located at `C:/ app/ main.py` when the container starts.

References

[1] Docker's glossary. https://docs.docker.com/glossary/

[2] docker build (docker image build)

[3] Use containers to Build, Share and Run your applications

[4] Microsoft Azure' glossary.

[5] Kubernetes' glossary.

[6] Core Kubernetes concepts for Azure Kubernetes Service

[7] Timer trigger for Azure Functions

[8] Persistent Volumes on Kubernetes

[9] Storage options for applications in Azure Kubernetes Service (AKS)

[10] Quickstart: Deploy an Azure Kubernetes Service (AKS) cluster using Azure CLI

[11] Azure Container Registry — Part 1 — by Anjali Dubey — Medium

[12] GitHub Action: Kubernetes Action

[13] GitHub Action Deploy to Kubernetes cluster

[14] KUBERNETES CRONJOBS - HOW TO RUN CRON JOB IN KUBERNETES?

[15] Kubernetes : kubelet

[16] What is Azure Resource Manager?

[17] Wikipedia page on: Server Message Block

[18] SMB file shares in Azure Files

[19] kubetcl commands list

[20] Virtual Machine series

[21] Understand Azure Files billing

[22] Azure Files pricing

[23] Kubernetes bootcamp

[24] Communication Master-Node

[25] Azure Command-Line Interface (CLI) documentation

[26] Azure CLI conceptual article list

[27] How to install the Azure CLI

[28] Azure Docs: Use the cluster autoscaler in Azure Kubernetes Service (AKS) - Azure Kubernetes Service

[29] MS Azure Calculator

Index