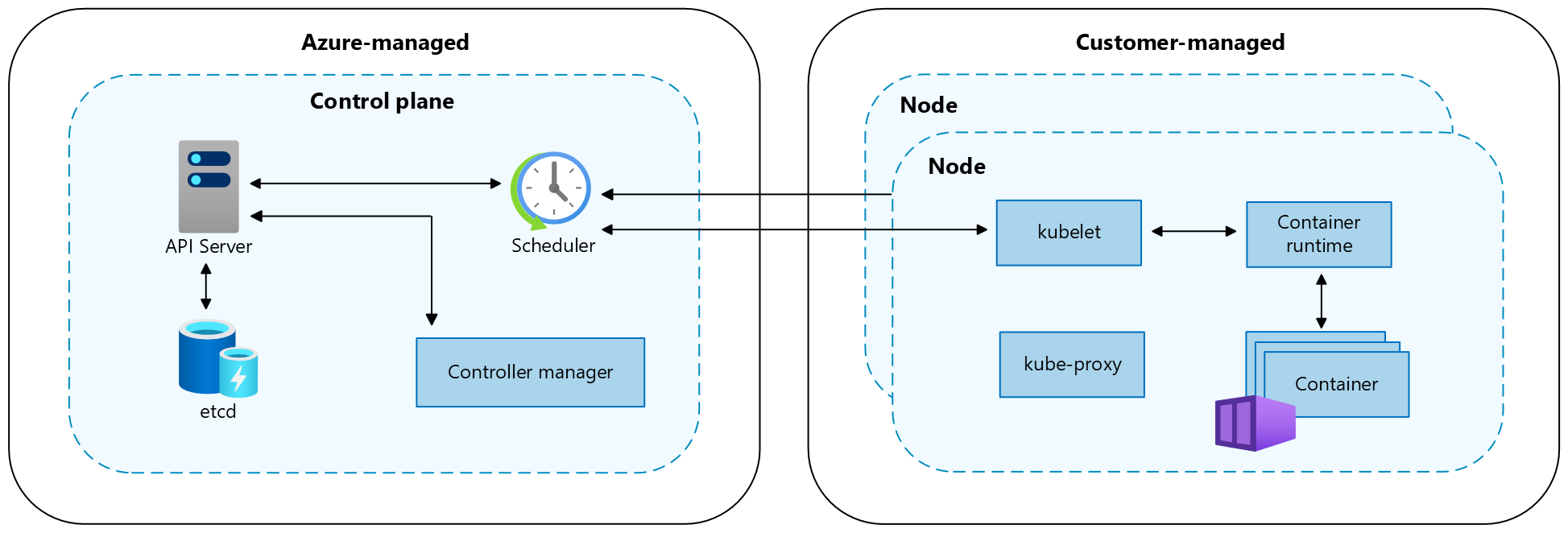
**What is AKS?**

AKS is a managed Kubernetes service that simplifies deploying, managing, and scaling containerized applications using Kubernetes. For more information, see [What is Azure Kubernetes Service (AKS)?](https://learn.microsoft.com/en-us/azure/aks/what-is-aks)



**Control plane**

The Azure managed control plane is composed of several components that help manage the cluster:

A screenshot of a computer

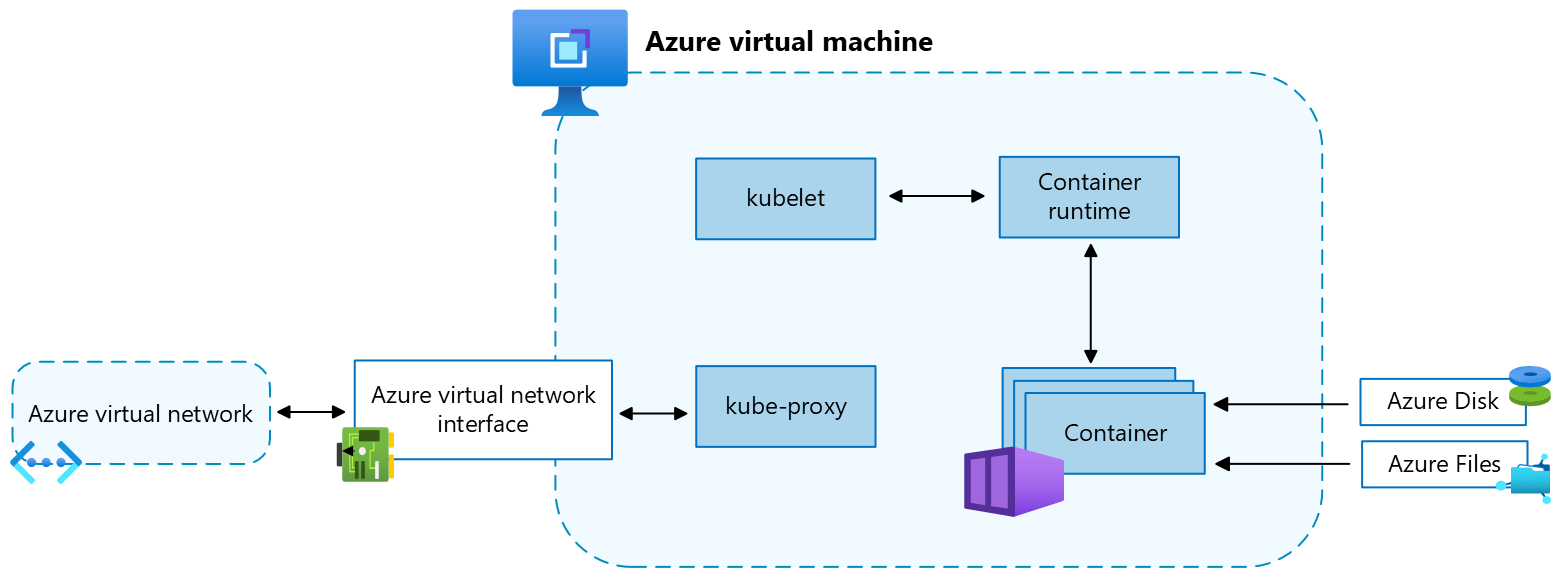
AI-generated content may be incorrect.

**Nodes**

Each AKS cluster has at least one node, which is an Azure virtual machine (VM) that runs Kubernetes node components. The following components run on each node:

A screenshot of a computer

AI-generated content may be incorrect.



**Pods**

A *pod* is a group of one or more containers that share the same network and storage resources and a specification for how to run the containers. Pods typically have a 1:1 mapping with a container, but you can run multiple containers in a pod.

**Namespaces**

Kubernetes resources, such as pods and deployments, are logically grouped into *namespaces* to divide an AKS cluster and create, view, or manage access to resources.

The following namespaces are created by default in an AKS cluster:

A screenshot of a computer

AI-generated content may be incorrect.

# LAB - Prepare an application for AKS

1. git clone https://github.com/Azure-Samples/aks-store-demo.git
2. cd aks-store-demo
3. Create docker-compose file

services:

rabbitmq:

image: rabbitmq:3.13.2-management-alpine

container\_name: 'rabbitmq'

restart: always

environment:

- "RABBITMQ\_DEFAULT\_USER=username"

- "RABBITMQ\_DEFAULT\_PASS=password"

ports:

- 15672:15672

- 5672:5672

healthcheck:

test: ["CMD", "rabbitmqctl", "status"]

interval: 30s

timeout: 10s

retries: 5

volumes:

- ./rabbitmq\_enabled\_plugins:/etc/rabbitmq/enabled\_plugins

networks:

- backend\_services

order-service:

build: src/order-service

container\_name: 'order-service'

restart: always

ports:

- 3000:3000

healthcheck:

test: ["CMD", "wget", "-O", "/dev/null", "-q", "http://order-service:3000/health"]

interval: 30s

timeout: 10s

retries: 5

environment:

- ORDER\_QUEUE\_HOSTNAME=rabbitmq

- ORDER\_QUEUE\_PORT=5672

- ORDER\_QUEUE\_USERNAME=username

- ORDER\_QUEUE\_PASSWORD=password

- ORDER\_QUEUE\_NAME=orders

- ORDER\_QUEUE\_RECONNECT\_LIMIT=3

networks:

- backend\_services

depends\_on:

rabbitmq:

condition: service\_healthy

product-service:

build: src/product-service

container\_name: 'product-service'

restart: always

ports:

- 3002:3002

healthcheck:

test: ["CMD", "wget", "-O", "/dev/null", "-q", "http://product-service:3002/health"]

interval: 30s

timeout: 10s

retries: 5

environment:

- AI\_SERVICE\_URL=http://ai-service:5001/

networks:

- backend\_services

store-front:

build: src/store-front

container\_name: 'store-front'

restart: always

ports:

- 8080:8080

healthcheck:

test: ["CMD", "wget", "-O", "/dev/null", "-q", "http://store-front:80/health"]

interval: 30s

timeout: 10s

retries: 5

environment:

- VUE\_APP\_PRODUCT\_SERVICE\_URL=http://product-service:3002/

- VUE\_APP\_ORDER\_SERVICE\_URL=http://order-service:3000/

networks:

- backend\_services

depends\_on:

- product-service

- order-service

networks:

backend\_services:

driver: bridge

1. docker compose -f docker-compose-quickstart.yml up -d
2. docker images
3. docker ps
4. <http://localhost:8080>
5. docker compose down
6. Create ACR and save the acr name in ACRNAME variable

az group create --name AKS-rg --location eastus

az acr create --resource-group AKS-rg --name $ACRNAME --sku Basic

export ACRNAME= **aksdevopsvp**

1. Build images

az acr build --registry $ACRNAME --image aks-store-demo/product-service:latest ./src/product-service/

az acr build --registry $ACRNAME --image aks-store-demo/order-service:latest ./src/order-service/

az acr build --registry $ACRNAME --image aks-store-demo/store-front:latest ./src/store-front/

1. List images

az acr repository list --name $ACRNAME --output table

**Create AKS**

Install kubectl - az aks install-cli

Create Cluster

az aks create --resource-group AKS-rg --name myAKSCluster --node-count 2 --generate-ssh-keys --attach-acr $ACRNAME

az aks get-credentials --resource-group AKS-rg --name myAKSCluster

kubectl get nodes

Get your login server address using the [az acr list](https://learn.microsoft.com/en-us/cli/azure/acr) command and query for your login server

az acr list --resource-group AKS-rg --query "[].{acrLoginServer:loginServer}" --output table

Make sure you're in the cloned *aks-store-demo* directory, and then open the aks-store-quickstart.yaml manifest file with a text editor

<https://github.com/Azure-Samples/aks-store-demo/blob/main/aks-store-quickstart.yaml>

Update the image property for the containers by replacing *ghcr.io/azure-samples* with your ACR login server name.

containers:

...

- name: order-service

image: <acrName>.azurecr.io/aks-store-demo/order-service:latest

...

- name: product-service

image: <acrName>.azurecr.io/aks-store-demo/product-service:latest

...

- name: store-front

image: <acrName>.azurecr.io/aks-store-demo/store-front:latest

...

kubectl apply -f aks-store-quickstart.yaml

kubectl get pods

**Basic K8s labs**

https://labs.play-with-k8s.com/

ctrl + insert to copy and shift + insert to paste

**DEPLOYMENT**

apiVersion: apps/v1

kind: Deployment

metadata:

name: nginx-deployment

spec:

replicas: 3

selector:

matchLabels:

app: nginx

template:

metadata:

labels:

app: nginx

spec:

containers:

- name: nginx-container

image: nginx:latest

ports:

- containerPort: 80

**POD AND SERVICE**

apiVersion: v1

kind: Pod

metadata:

name: nginx

labels:

app.kubernetes.io/name: proxy

spec:

containers:

- name: nginx

image: nginx:stable

ports:

- containerPort: 80

name: http-web-svc

---

apiVersion: v1

kind: Service

metadata:

name: nginx-service

spec:

selector:

app.kubernetes.io/name: proxy

ports:

- name: name-of-service-port

protocol: TCP

port: 80

targetPort: http-web-svc

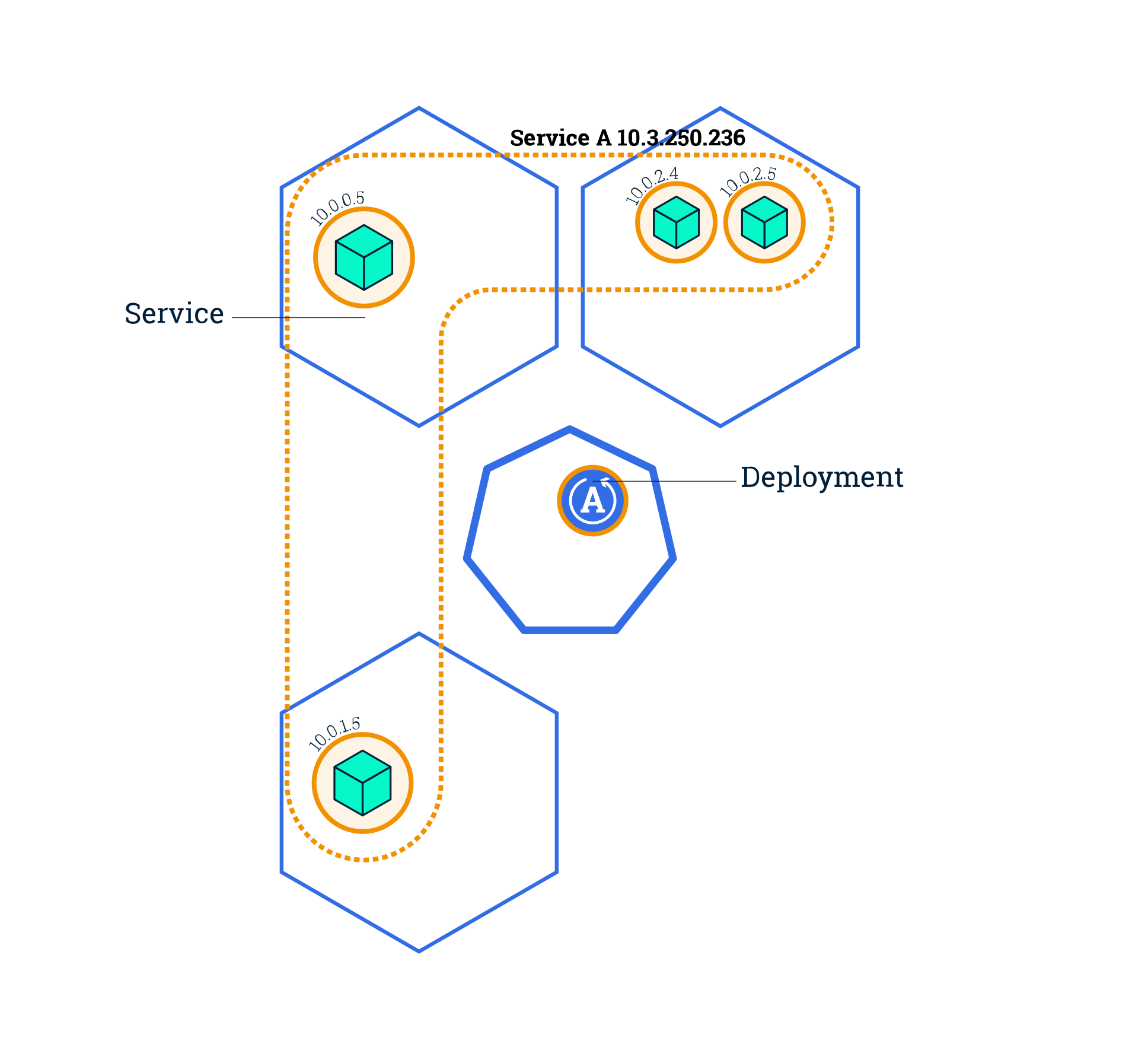
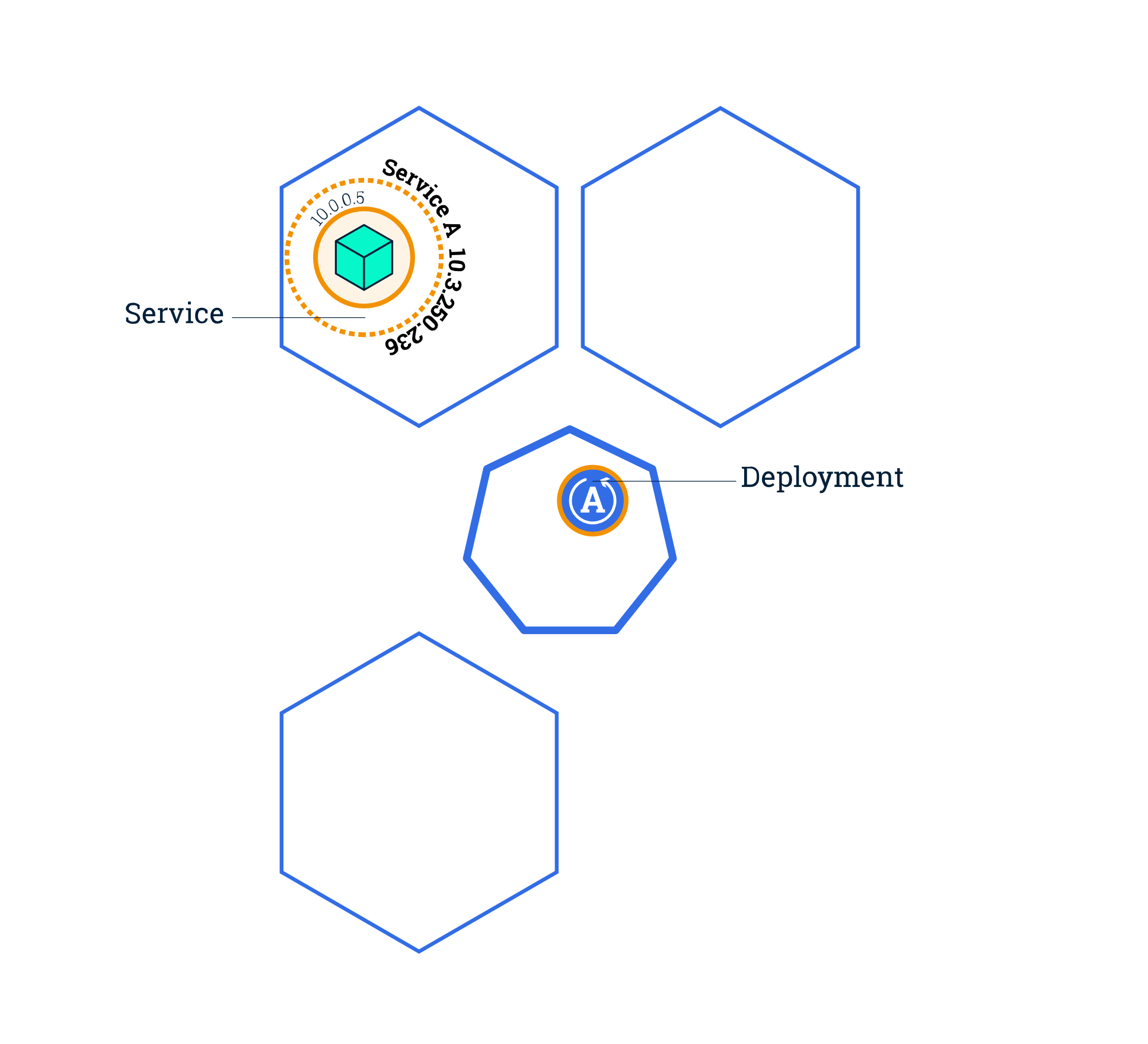
# Scaling

In Kubernetes, you can *scale* a workload depending on the current demand of resources. This allows your cluster to react to changes in resource demand more elastically and efficiently.

When you scale a workload, you can either increase or decrease the number of replicas managed by the workload, or adjust the resources available to the replicas in-place.

* **Horizontal scaling**: [Running multiple instances of your app](https://kubernetes.io/docs/tutorials/kubernetes-basics/scale/scale-intro/)
* **Vertical scaling**: [Resizing CPU and memory resources assigned to containers](https://kubernetes.io/docs/tasks/configure-pod-container/resize-container-resources/)

**Horizontal**



Scaling out a Deployment will ensure new Pods are created and scheduled to Nodes with available resources

**kubectl scale deployments/<deployment-name> --replicas=4**

**kubectl get pods -o wide**

**Vertical**

Resizing Pods cpu / mem in place

* Resize policies allow for a more fine-grained control over how pod's containers are resized for CPU and memory resources.
* For example, the container's application may be able to handle CPU resources resized without being restarted, but resizing memory may require that the application hence the containers be restarted.
* To enable this, the Container specification allows users to specify a resizePolicy. The following restart policies can be specified for resizing CPU and memory:
* NotRequired: Resize the container's resources while it is running.
* RestartContainer: Restart the container and apply new resources upon restart.
  + If resizePolicy[\*].restartPolicy is not specified, it defaults to NotRequired.

*apiVersion: v1*

*kind: Pod*

*metadata:*

*name: qos-demo-5*

*namespace: qos-example*

*spec:*

*containers:*

*- name: qos-demo-ctr-5*

*image: nginx*

*resizePolicy:*

*- resourceName: cpu*

*restartPolicy: NotRequired*

*- resourceName: memory*

*restartPolicy: RestartContainer*

*resources:*

*limits:*

*memory: "200Mi"*

*cpu: "700m"*

*requests:*

*memory: "200Mi"*

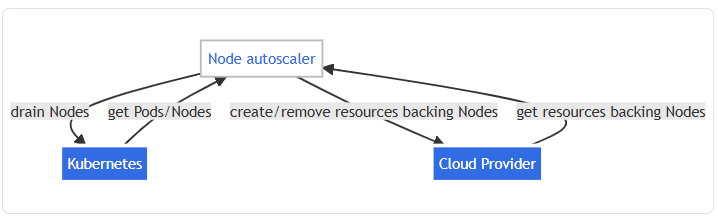
*cpu: "700m"*

<https://kubernetes.io/docs/tasks/run-application/horizontal-pod-autoscale-walkthrough/>

**Node Autoscaling**

If there are Pods in a cluster that can't be scheduled on existing Nodes, new Nodes can be automatically added to the cluster—*provisioned*—to accommodate the Pods.

az aks scale --resource-group myResourceGroup --name myAKSCluster --node-count 3



# Storage

Applications running in Azure Kubernetes Service (AKS) might need to store and retrieve data. While some application workloads can use local, fast storage on unneeded, emptied nodes, others require storage that persists on more regular data volumes within the Azure platform.

Multiple pods might need to:

* Share the same data volumes.
* Reattach data volumes if the pod is rescheduled on a different node.

**Volumes**

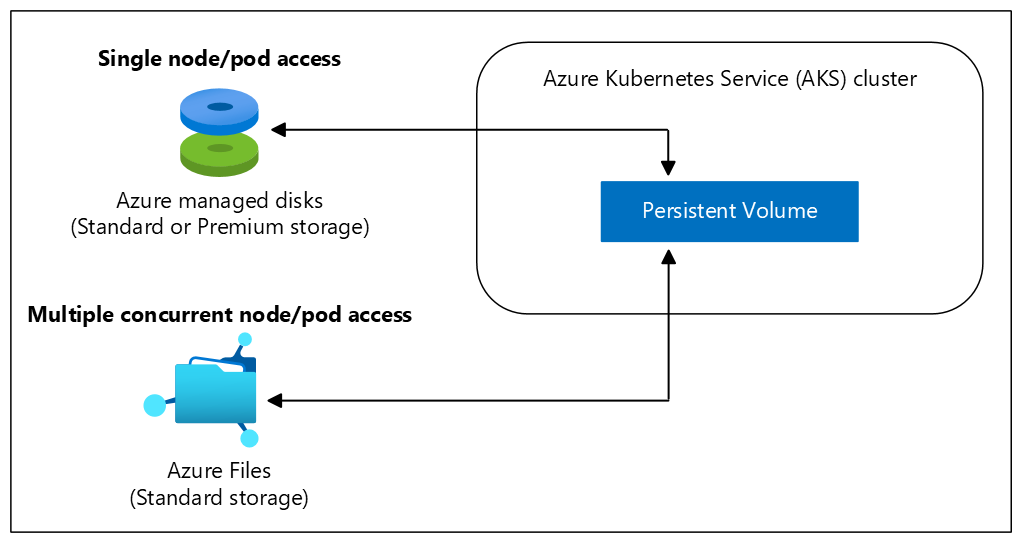
Kubernetes typically treats individual pods as ephemeral, disposable resources. Applications have different approaches available to them for using and persisting data. A *volume* represents a way to store, retrieve, and persist data across pods and through the application lifecycle.

**Persistent volumes**

Volumes defined and created as part of the pod lifecycle only exist until you delete the pod. Pods often expect their storage to remain if a pod is rescheduled on a different host during a maintenance event, especially in StatefulSets. A *persistent volume* (PV) is a storage resource created and managed by the Kubernetes API that can exist beyond the lifetime of an individual pod.

You can use the following Azure Storage services to provide the persistent volume:

* [Azure Disk](https://learn.microsoft.com/en-us/azure/aks/azure-csi-disk-storage-provision)
* [Azure Files](https://learn.microsoft.com/en-us/azure/aks/azure-csi-files-storage-provision)
* [Azure Container Storage](https://learn.microsoft.com/en-us/azure/storage/container-storage/container-storage-introduction)
* [Storage classes](https://learn.microsoft.com/en-us/azure/aks/concepts-storage#storage-classes)



**Storage classes**

To specify different tiers of storage, such as premium or standard, you can create a *storage class*.

A storage class also defines a *reclaim policy*. When you delete the persistent volume, the reclaim policy controls the behavior of the underlying Azure Storage resource. The underlying resource can either be deleted or kept for use with a future pod.

Each class is mapped to an AKS stotage option like File, Blob, Disk

The default class will be the same as managed-csi.

|  |  |
| --- | --- |
| managed- | csi Uses Azure Standard SSD locally redundant storage (LRS) to create a managed disk. The reclaim policy ensures that the underlying Azure Disk is deleted when the persistent volume that used it is deleted. The storage class also configures the persistent volumes to be expandable. |

apiVersion: storage.k8s.io/v1

kind: StorageClass

metadata:

name: managed-premium-retain

provisioner: disk.csi.azure.com

parameters:

skuName: Premium\_ZRS

reclaimPolicy: Retain

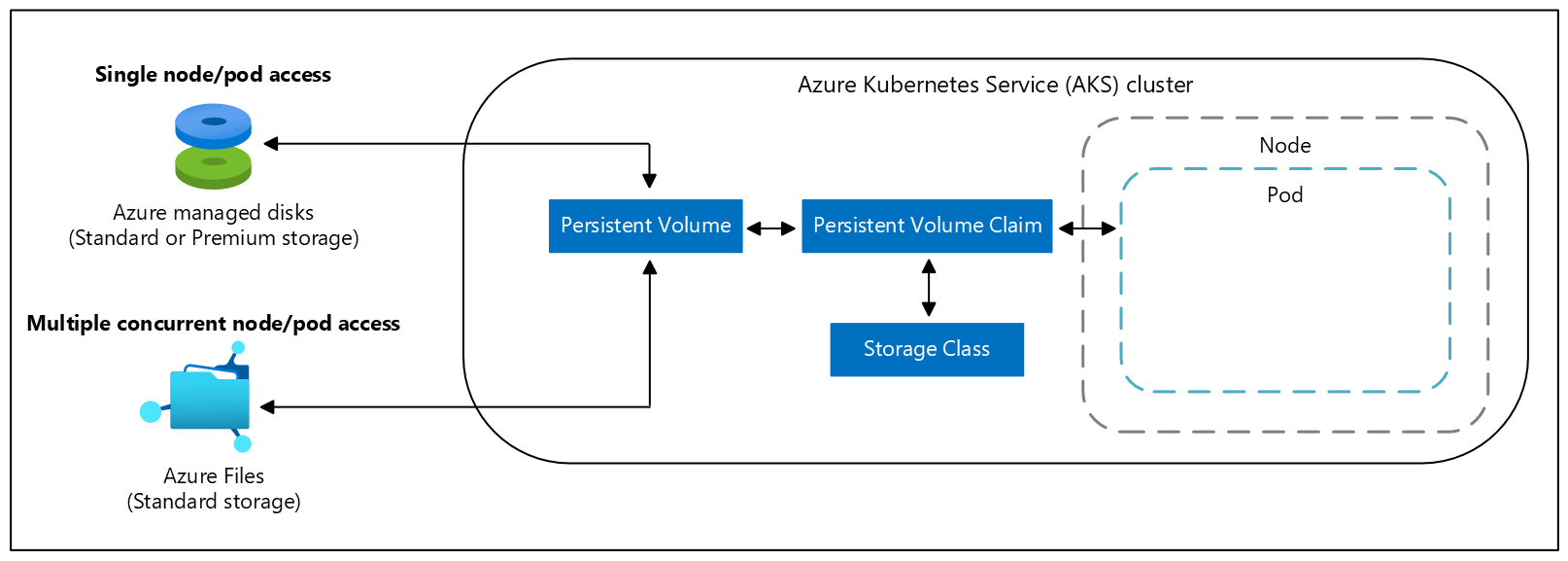
volumeBindingMode: WaitForFirstConsumer

allowVolumeExpansion: true

**Persistent volume claims**

A persistent volume claim (PVC) requests storage of a particular storage class, access mode, and size. The Kubernetes API server can dynamically provision the underlying Azure Storage resource if no existing resource can fulfill the claim based on the defined storage class.

The pod definition includes the volume mount once the volume has been connected to the pod.



Once an available storage resource has been assigned to the pod requesting storage, the persistent volume is *bound* to a persistent volume claim.

apiVersion: v1

kind: PersistentVolumeClaim

metadata:

name: azure-managed-disk

spec:

accessModes:

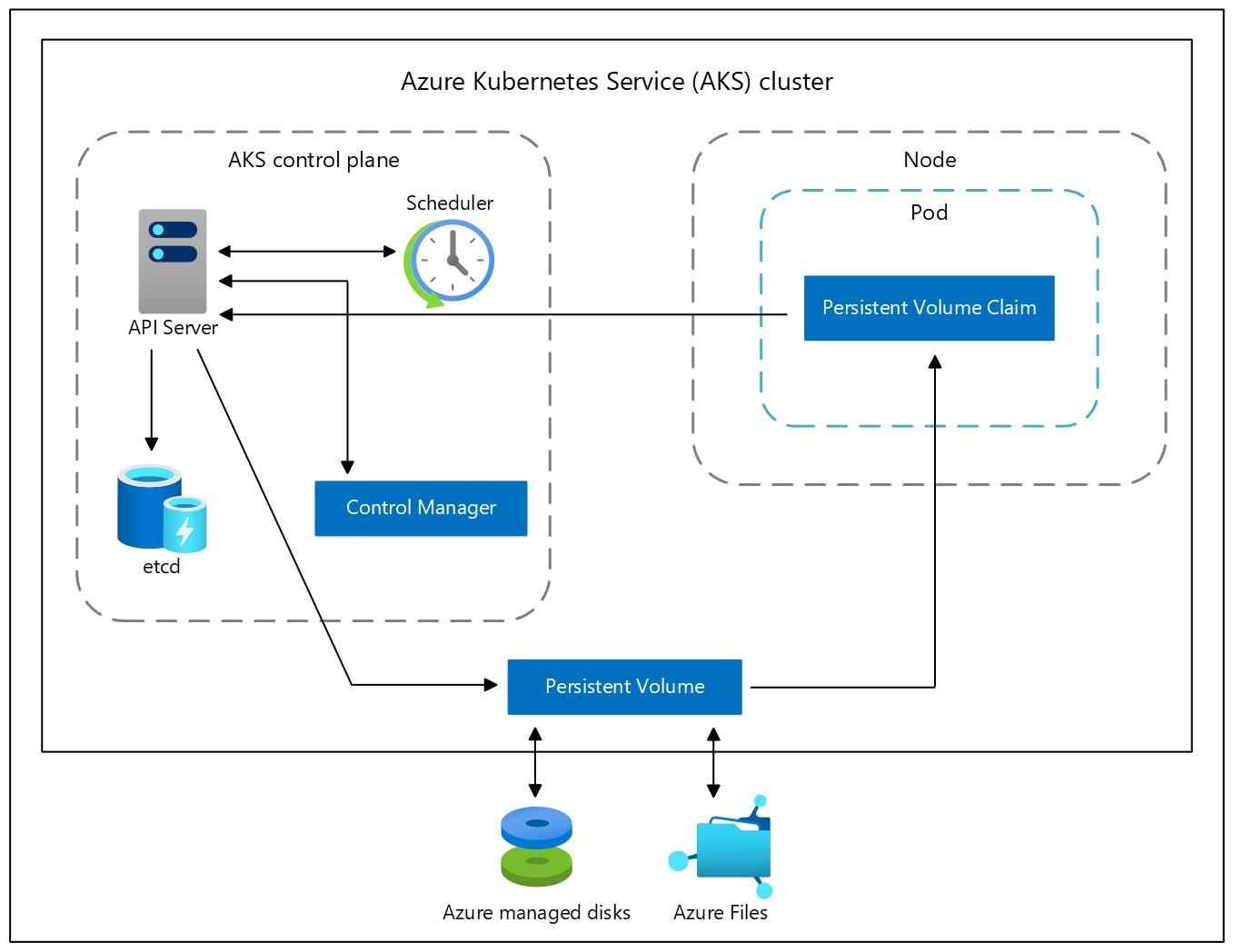
- ReadWriteOnce

storageClassName: managed-premium-retain

resources:

requests:

storage: 5Gi



# CSI

* The Container Storage Interface (CSI) is a standard for exposing arbitrary block and file storage systems to containerized workloads on Kubernetes.
* It is a standard that allows Kubernetes to integrate with different storage systems in a consistent way.
* It enables the use of third-party storage solutions without modifying Kubernetes core code.

CSI components

* **CSI Driver:** A storage provider's implementation of the CSI specification.
* **Persistent Volume (PV):** A storage unit created in Kubernetes.
* **Persistent Volume Claim (PVC):** A request for storage made by an application.
* **StorageClass:** Defines how storage should be dynamically provisioned.
* **CSI Controller Plugin:** Manages volume lifecycle operations like creation and deletion.
* **CSI Node Plugin:** Runs on each Kubernetes node and handles volume mounting and unmounting.

CSI workflow

* A user creates a **PersistentVolumeClaim (PVC)** requesting storage.
* Kubernetes checks for a matching **StorageClass** and asks the **CSI driver** to provision storage.
* The CSI **controller plugin** provisions the volume.
* The volume is bound to the **PVC**, and a **PersistentVolume (PV)** is created.
* When a Pod is scheduled, the **CSI node plugin** mounts the volume to the correct node.
* When a PVC is deleted, the CSI driver handles cleanup based on its reclaim policy.

**Create and use a volume with Azure Disks in Azure Kubernetes Service (AKS)**

*A persistent volume represents a piece of storage provisioned for use with Kubernetes pods.*

*You can use a persistent volume with one or many pods, and you can provision it dynamically or statically. This article shows you how to dynamically create persistent volumes with Azure Disks in an Azure Kubernetes Service (AKS) cluster.*

*Using Azure Disks Container Storage Interface (CSI) driver is a*[*CSI specification*](https://github.com/container-storage-interface/spec/blob/master/spec.md)*-compliant driver used by Azure Kubernetes Service (AKS) to manage the lifecycle of Azure Disk.*

az aks update --name myAKSCluster --resource-group myResourceGroup --enable-disk-driver --enable-file-driver --enable-blob-driver --enable-snapshot-controller

# LAB

Pods - <https://kubernetes.io/docs/concepts/workloads/pods/>

Statefulsets- <https://kubernetes.io/docs/concepts/workloads/controllers/statefulset/>

Service - <https://kubernetes.io/docs/concepts/services-networking/service/>

Persistent Volume -

<https://learn.microsoft.com/en-us/azure/aks/azure-csi-disk-storage-provision>

Pod, deployment, service, pv, pvc, storageclass, statefulset,…………

Toystore.com – 20 microservices

Storage – PV,PVC

DB – statefulset

Service

Deployment

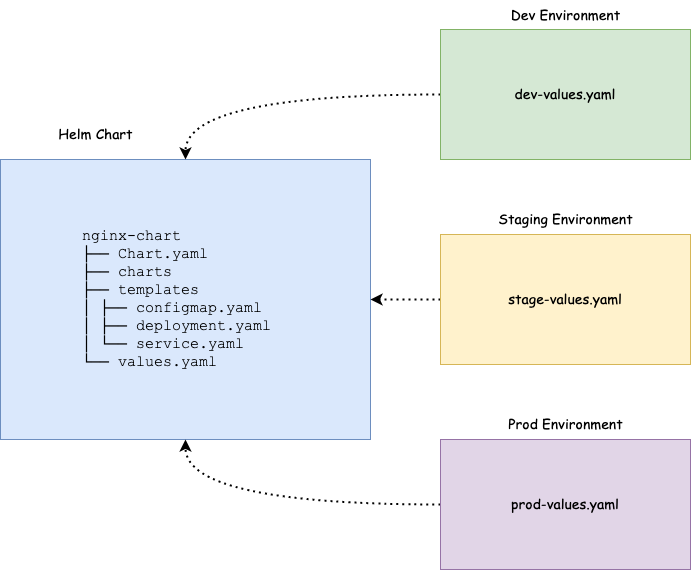
Ingress

StorageClass

# Helm

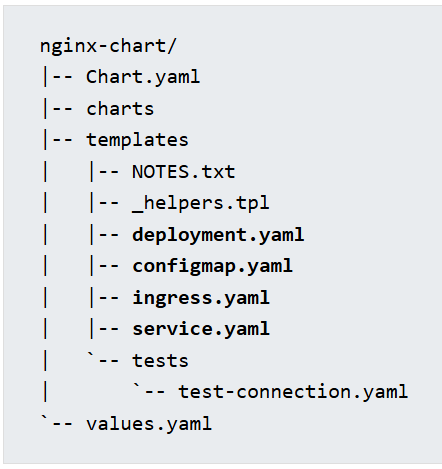
Package Manager

* [Helm](https://helm.sh/) is an open-source packaging tool that helps you install and manage the lifecycle of Kubernetes applications.
* Similar to Linux package managers like *APT* and *Yum*, Helm manages Kubernetes charts, which are packages of pre-configured Kubernetes resources.



git clone https://github.com/techiescamp/helm-tutorial.git

helm create nginx-chart



cd nginx-chart

helm lint .

helm lint /path/to/nginx-chart

helm template .

helm install --dry-run my-release nginx-chart

helm install frontend nginx-chart

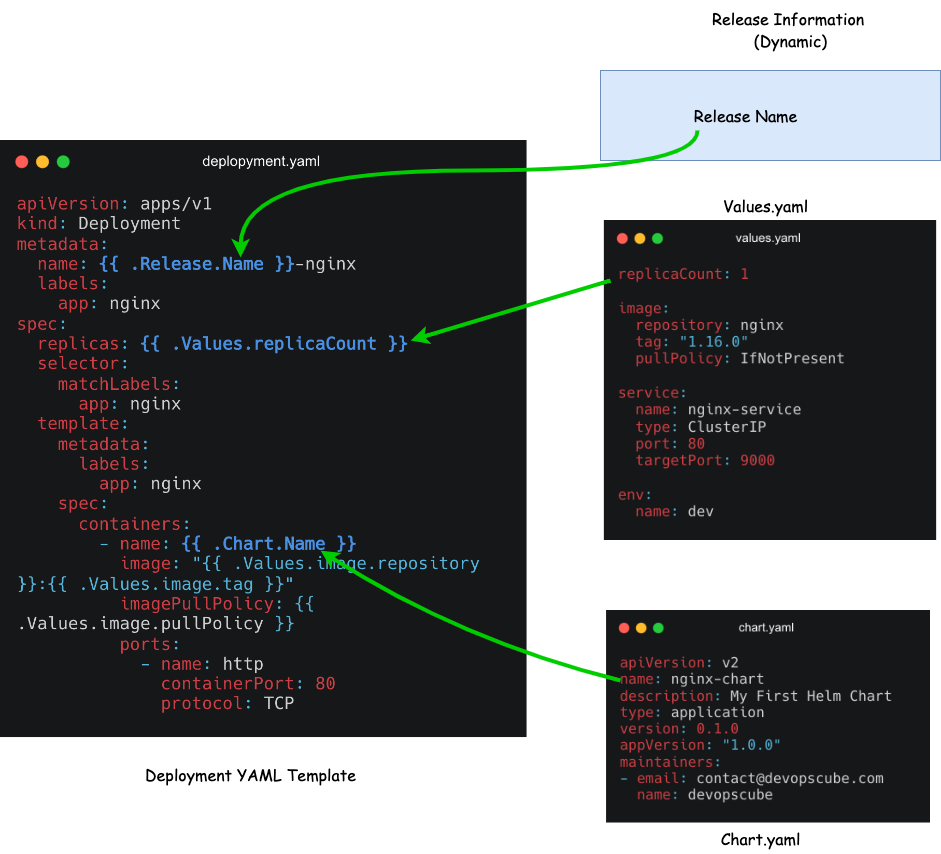
helm list

kubectl get deployment

kubectl get services

kubectl get configmap

kubectl get pods



Helm CLI automatically available on CLoudshell and uses your kubeconfig

**Prometheus Setup**

* helm repo add prometheus-community https://prometheus-community.github.io/helm-charts
* helm repo update
* helm install prometheus prometheus-community/prometheus

**Grafana setup**

<https://grafana.com/docs/grafana/latest/setup-grafana/installation/helm/>

* Configure datasource as Prmotheus
* Import dashboard – 10204
* Grafana Dashboard <https://grafana.com/grafana/dashboards/>