Microsoft Azure

Kubernetes on Azure
101

Rui Félix Pereira
Mar, 2019

Content

Kubernetes 101
Kubernetes on Azure overview
Kubernetes objects
Resources

Introduction

Kubernetes: the industry-leading orchestrator

Portable

Public, private, hybrid, multi-cloud

Extensible

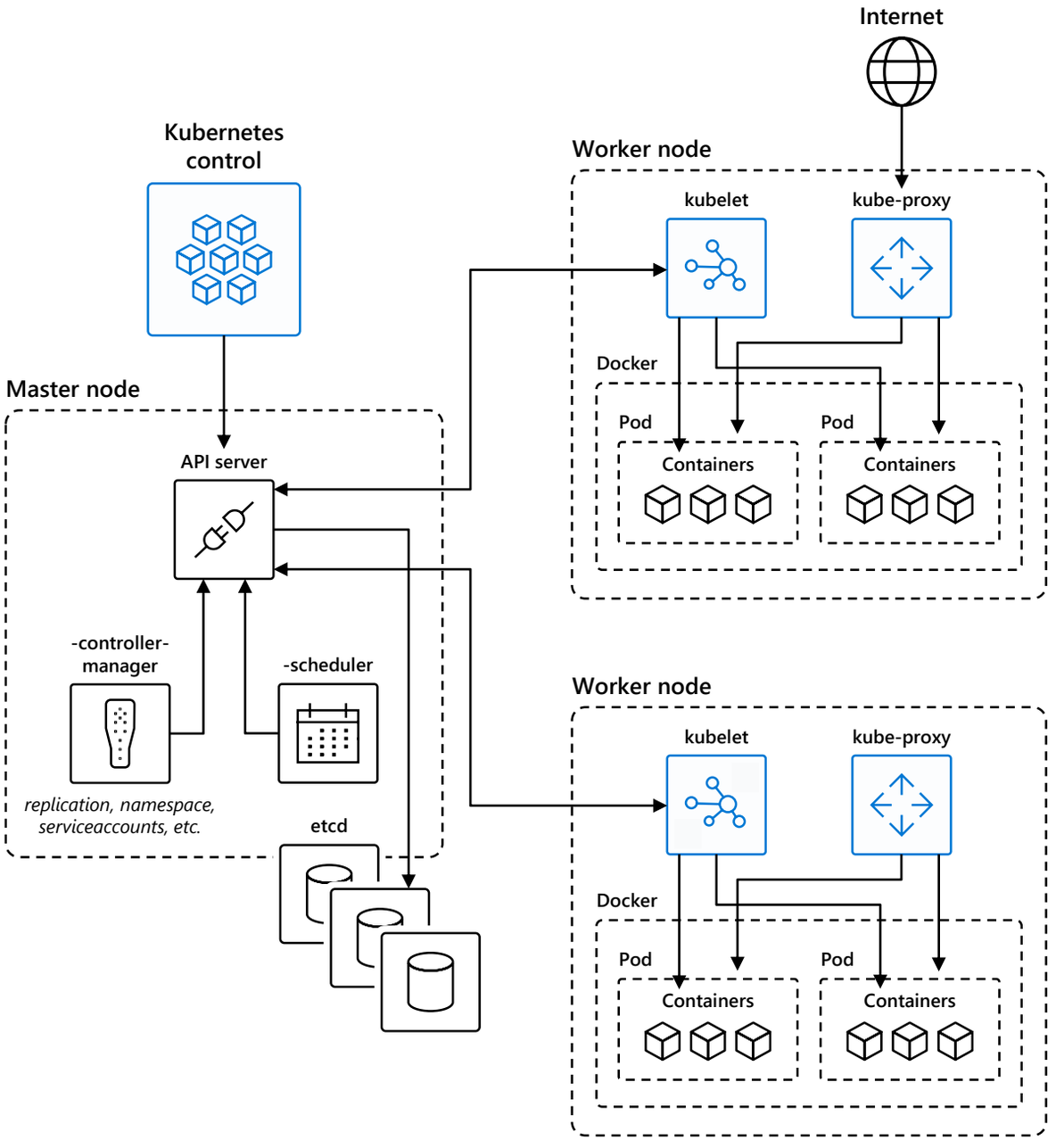
Modular, pluggable, hookable, composable

Self-healing

Auto-placement, auto-restart, auto-replication, auto-scaling

Kubernetes 101

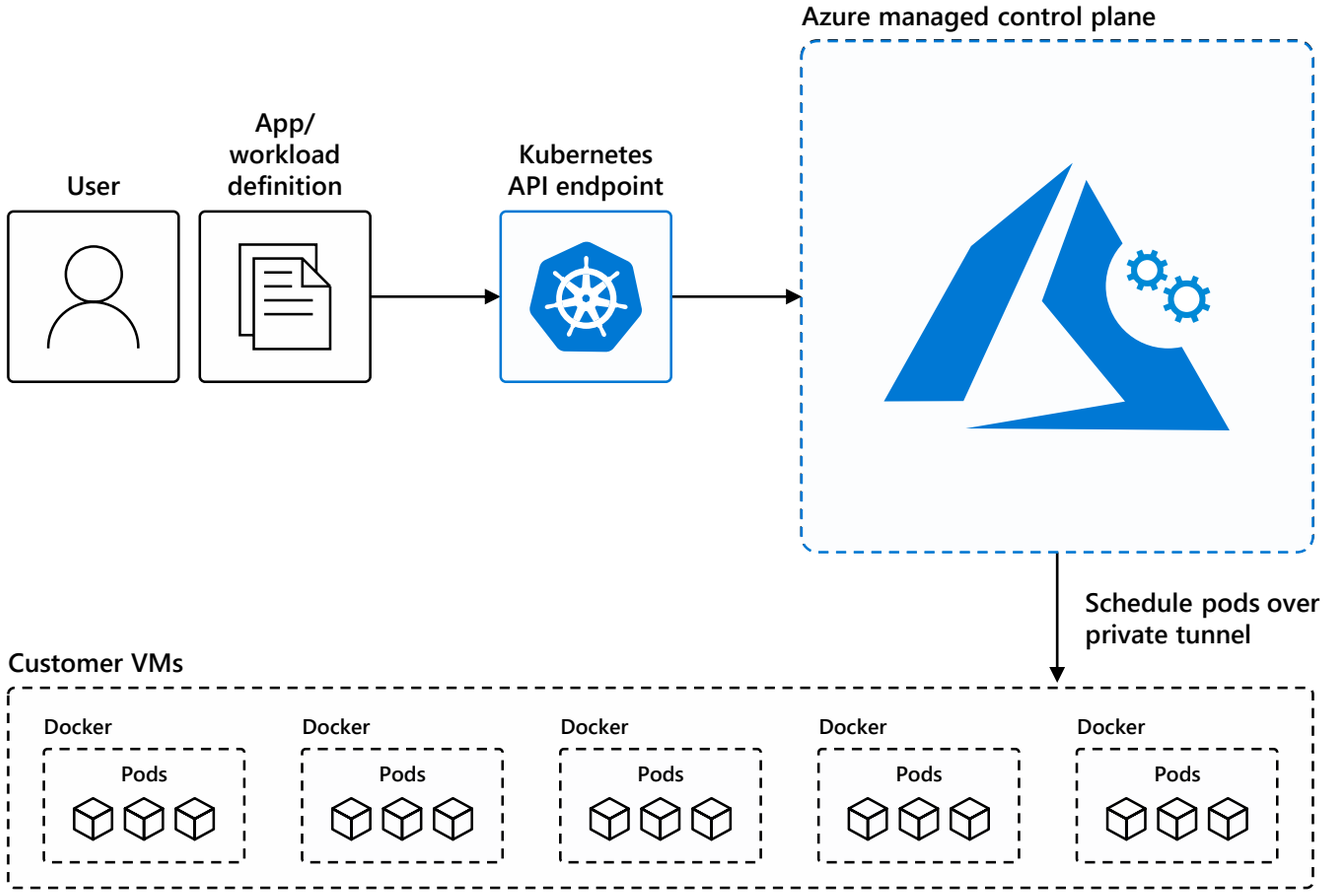
1. Kubernetes users communicate with API server and apply desired state
2. Master nodes actively enforce desired state on worker nodes
3. Worker nodes support communication between containers
4. Worker nodes support communication from the Internet



Kubernetes on Azure overview

How managed Azure Kubernetes Service works

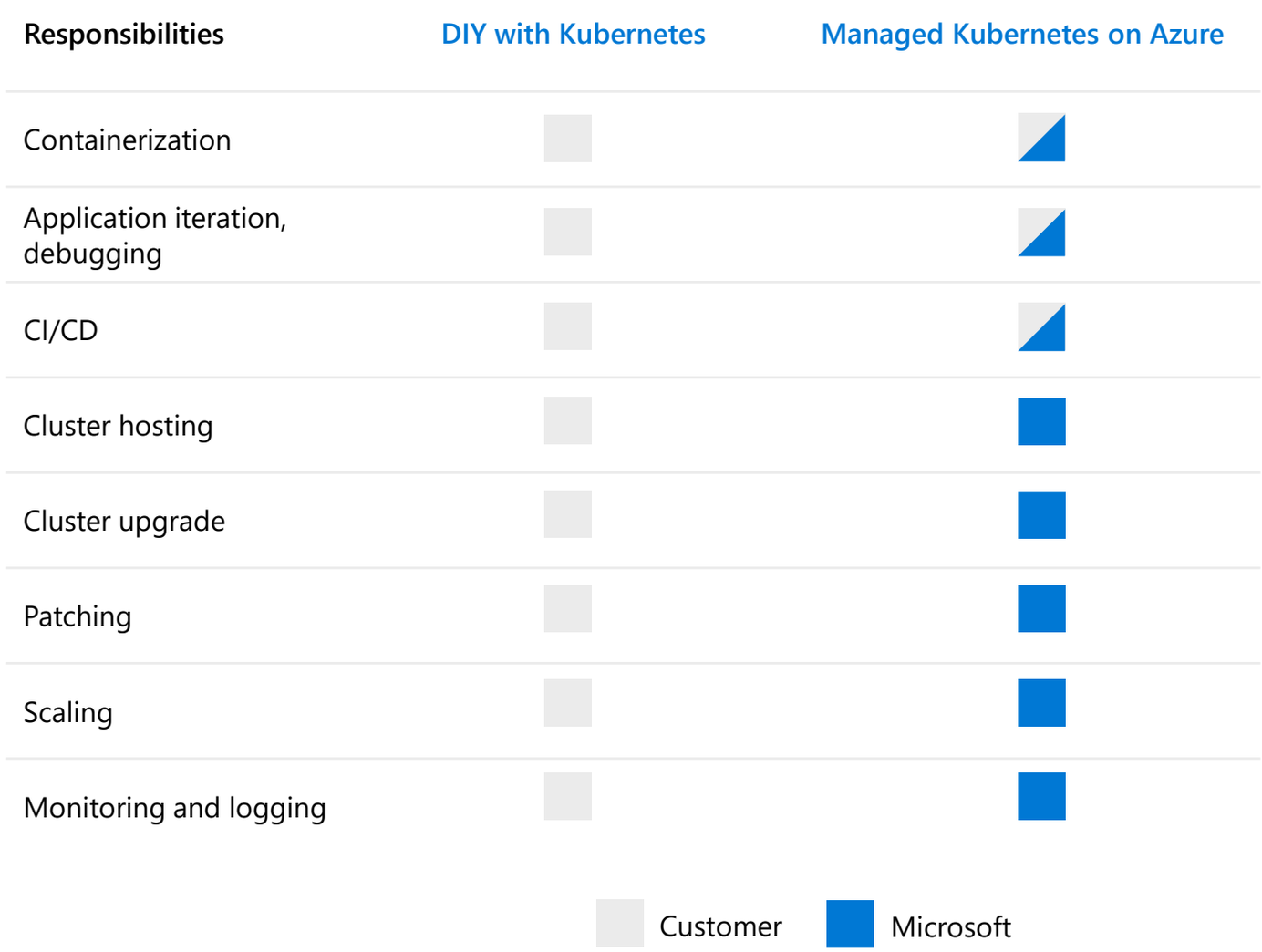
- Automated upgrades, patches
- High reliability, availability
- Easy, secure cluster scaling
- Self-healing
- API server monitoring
- At no charge



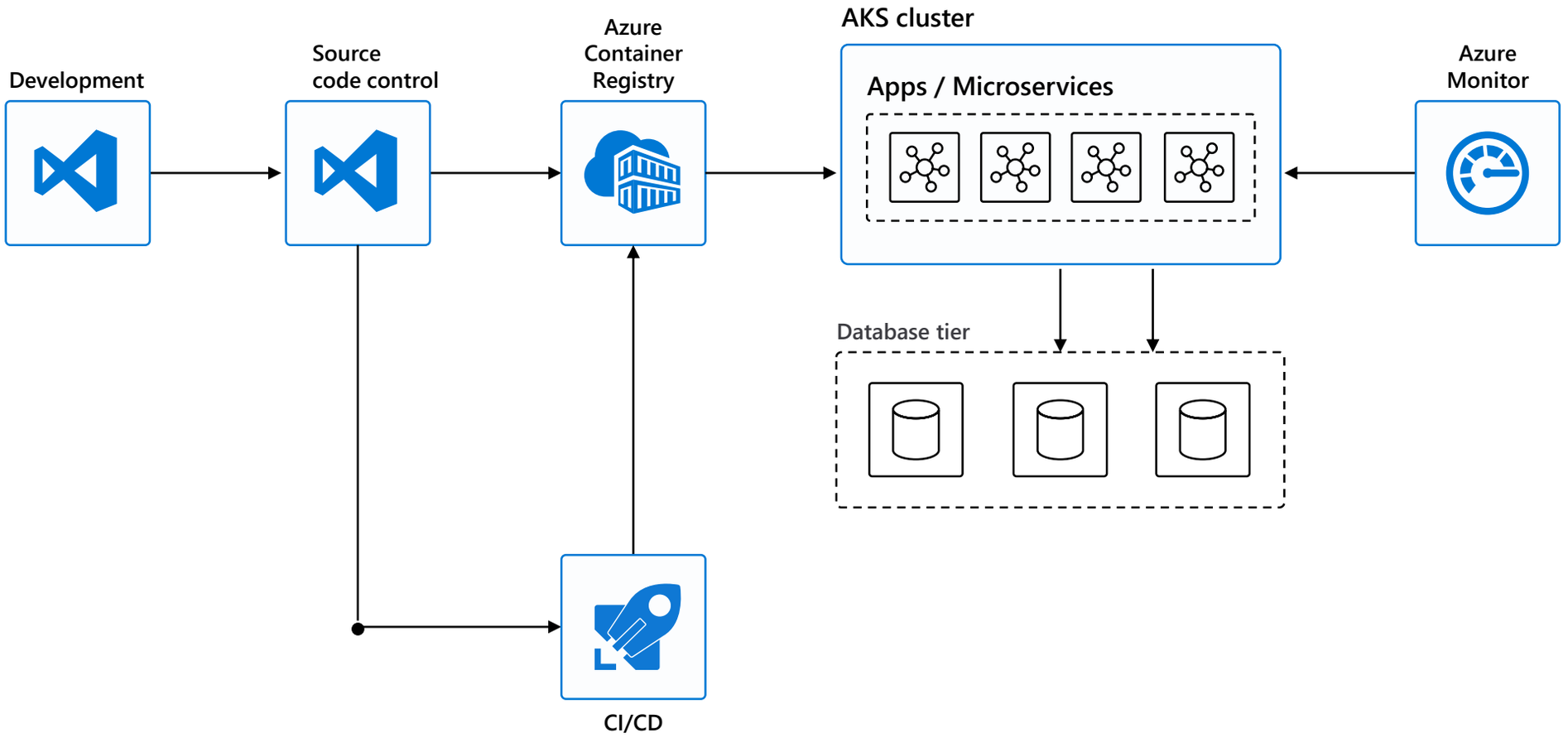
From infrastructure to innovation

Managed Kubernetes empowers you to achieve more

Focus on your containers and code, not the plumbing of them



Typical DevOps with Kubernetes

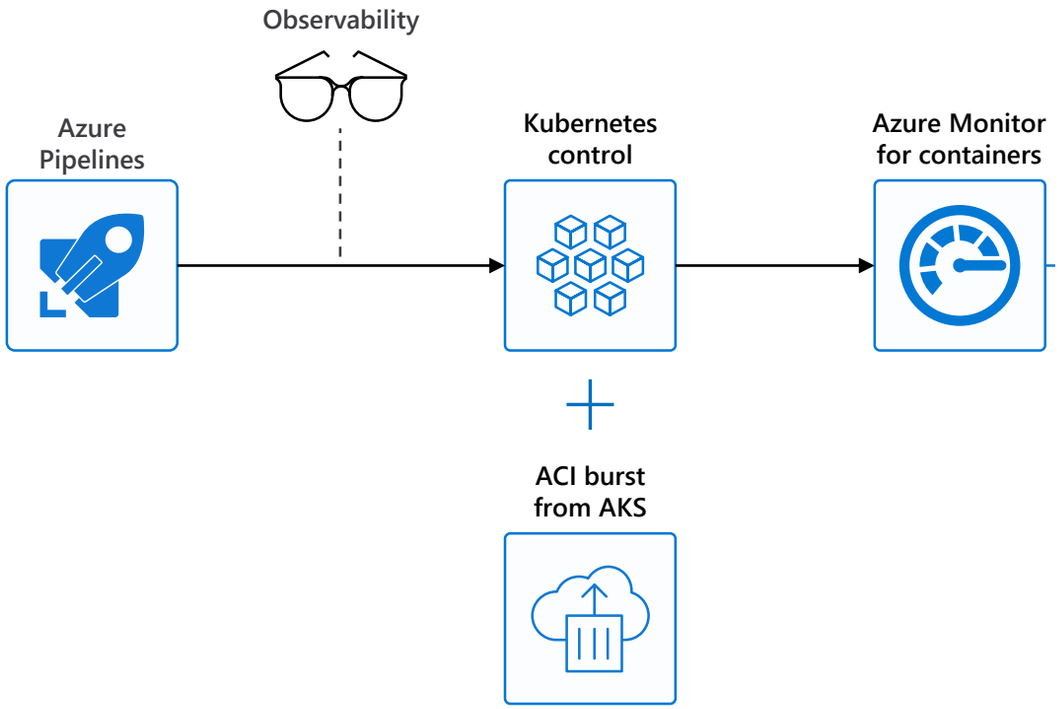


Azure makes Kubernetes easy

Deploy and manage Kubernetes with ease

Task	The Old Way	With Azure
Create a cluster	Provision network and VMs Install dozens of system components including etcd Create and install certificates Register agent nodes with control plane	`az aks create`
Upgrade a cluster	Upgrade your master nodes Cordon/drain and upgrade worker nodes individually	`az aks upgrade`
Scale a cluster	Provision new VMs Install system components Register nodes with API server	`az aks scale`

Azure Monitor for containers



Visualization
Visualize overall health and performance from clusters to containers with drill downs and filters

Insights
Provide insights with multi-cluster health roll up view

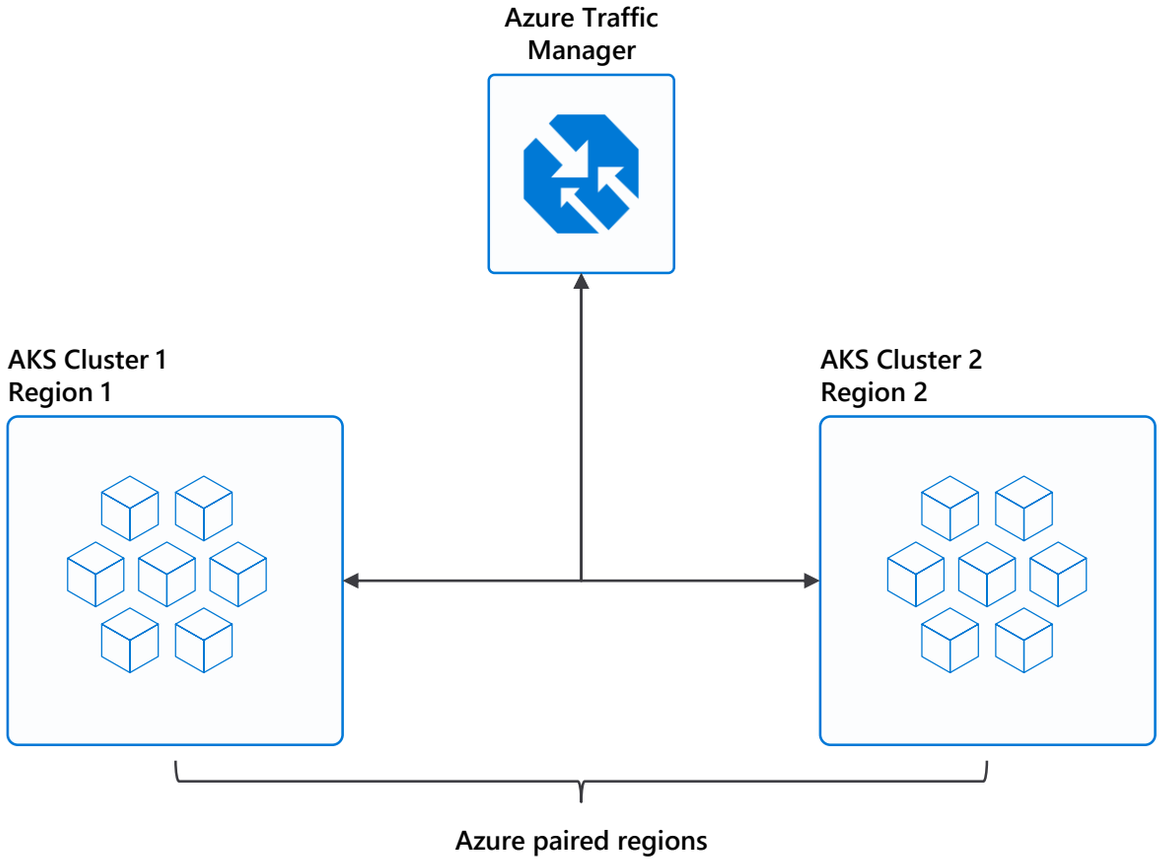
Monitor & Analyze
Monitor and analyze Kubernetes and container deployment performance, events, health, and logs

Response
Native alerting with integration to issue managements and ITSM tools

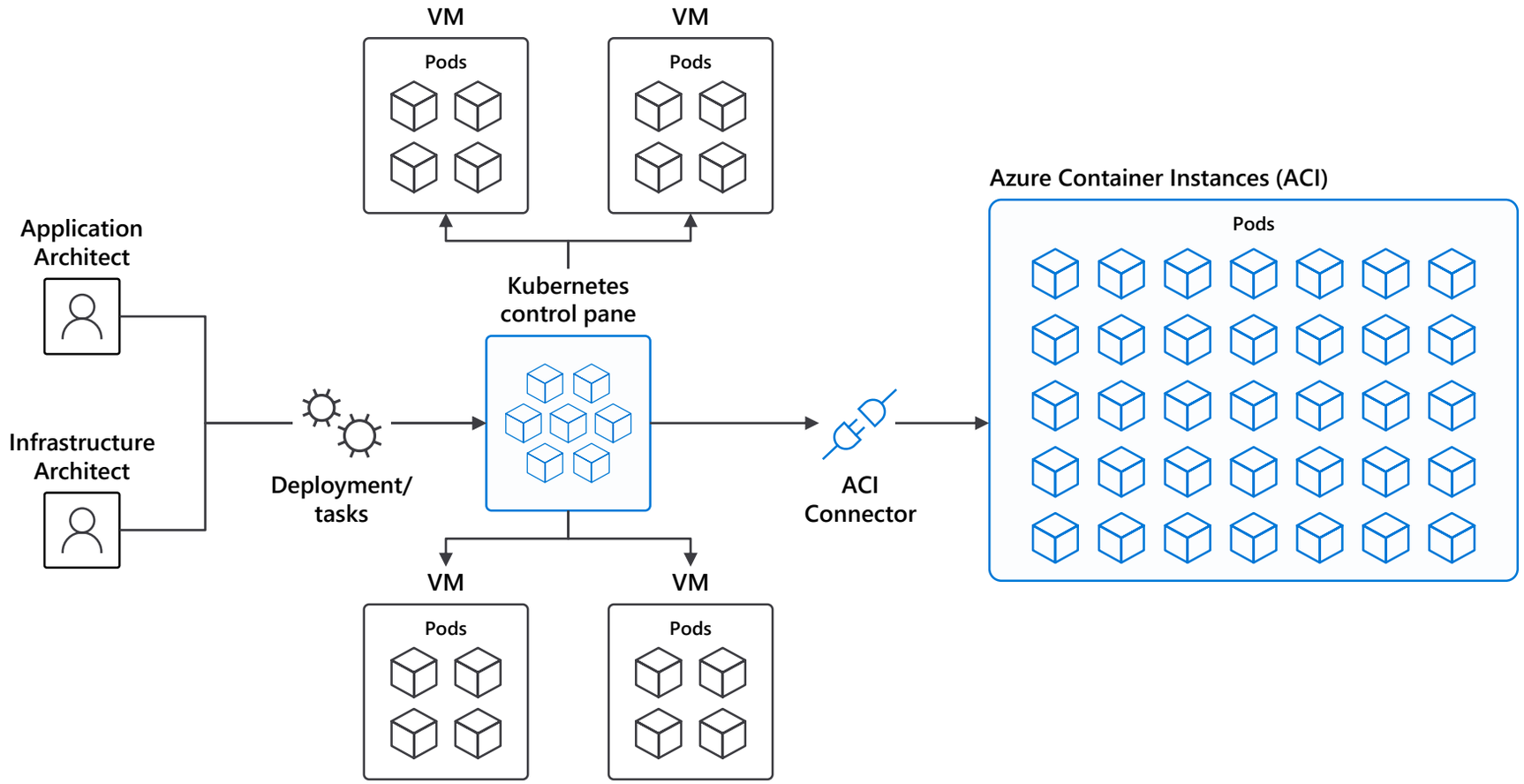
Observability
Observe live container logs on container deployment status

Multi-Region Clusters

- Minimize downtime risk
- One live region
  - Another backup
  - Or weighted traffic
- A/B testing



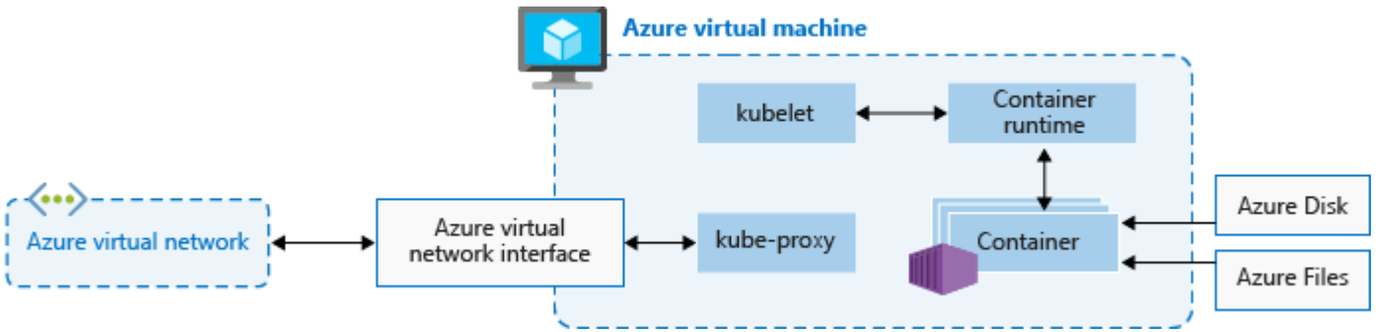
Bursting with the ACI Connector/ Virtual Kubelet



Kubernetes objects

Nodes and node pools

- Azure VM size for your nodes defines how many CPUs, how much memory, and the size and type of storage available (such as high-performance SSD or regular HDD)



Lab 1: Nodes and node pools

Create a managed Azure Kubernetes Service cluster

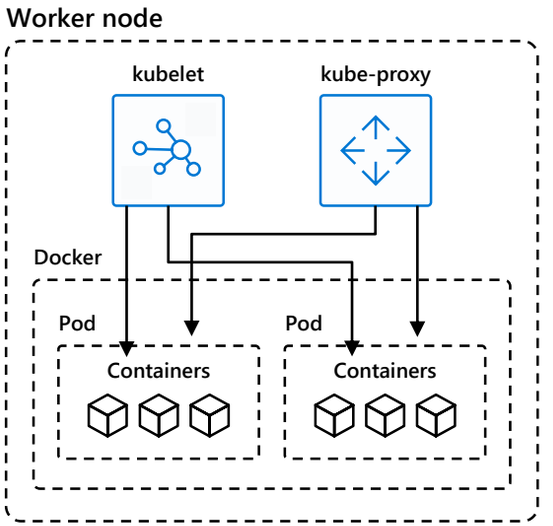
Task	With Azure

Create a cluster	`az group create --name $(RESOURCE_GROUP) --location $(LOCATION) --output table`
	`az aks create \`
	`  --resource-group $(RESOURCE_GROUP) \`
	`  --name $(CLUSTER_NAME) \`
	`  --node-count 2 \`
	`  --node-vm-size Standard_A2_v2 \`
	`  --generate-ssh-keys`
	`  --no-wait`

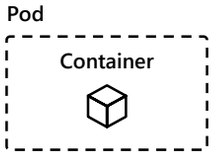
Validate cluster creation	`sudo az aks install-cli`
	`az aks show \`
	`  --resource-group $(RESOURCE_GROUP) \`
	`  --name $(CLUSTER_NAME)`
	`az aks get-credentials \`
	`  --resource-group $(RESOURCE_GROUP) \`
	`  --name $(CLUSTER_NAME)`
	`kubectl get nodes`
	`kubectl cluster-info`

Pods

- Basic building block of Kubernetes
- Encapsulates an application container (or multiple containers), storage resources, a unique network IP, and options that govern how the container(s) should run
- Represents a unit of deployment: a single instance of an application in Kubernetes



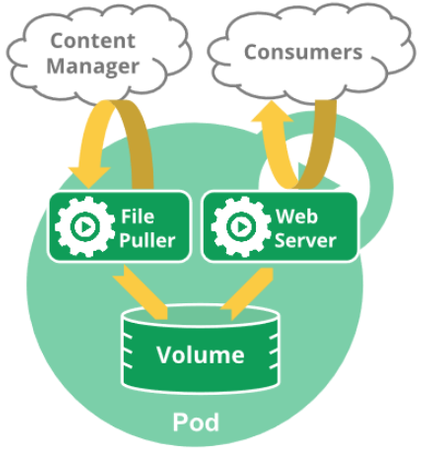
Pods - Single container deployment



- Pods that run a single container.
- The “one-container-per-Pod” model is the most common Kubernetes use case
- You can think of a Pod as a wrapper around a single container, and Kubernetes manages the Pods rather than the containers directly

```
apiVersion: v1
kind: Pod
metadata:
  name: helloWeb
spec:
  containers:
    image: nginx
    ports:
    - containerPort: 80
```

Pods - Multiple containers deployment

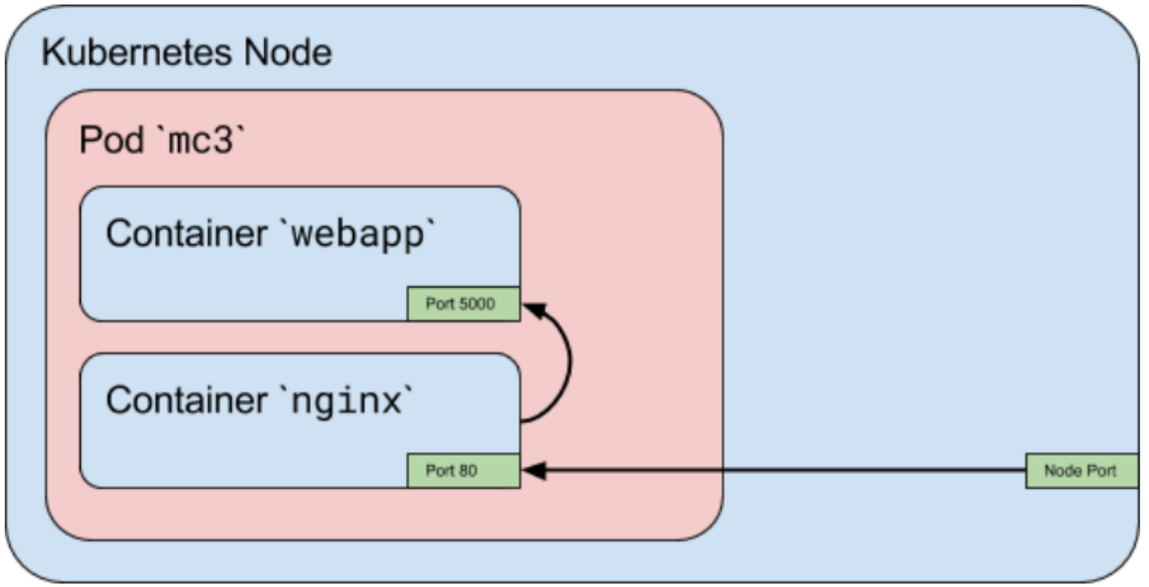


- Pods that run multiple containers that need to work together
- A Pod might encapsulate an application composed of **multiple co-located containers** that are tightly coupled and **need to share resources**

```
apiVersion: v1
kind: Pod
spec:
  volumes:
  - name: html
    emptyDir: {}
  containers:
  - name: webserver
    image: nginx
    volumeMounts:
    - name: html
      mountPath: /usr/share/nginx/html
  - name: content
    image: debian
    volumeMounts:
    - name: html
      mountPath: /html
```

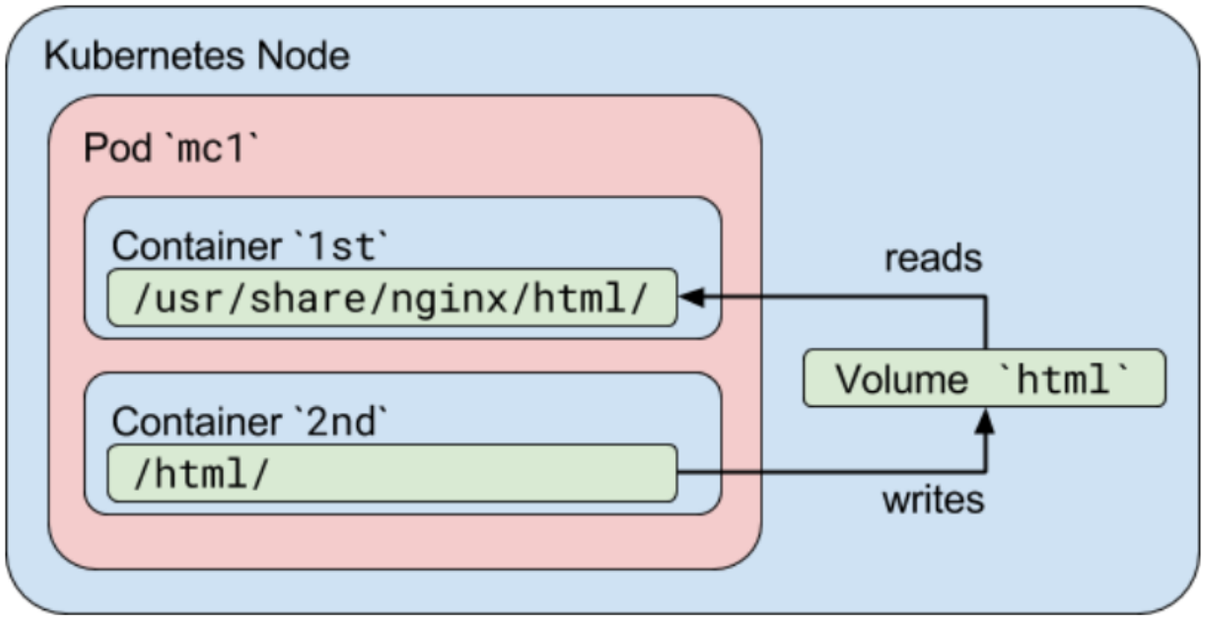
Pods - Network

- Each Pod is assigned a unique IP address
- Every container in a Pod shares the network namespace, including the IP address and network ports
- Containers inside a Pod can communicate with one another using localhost
- When containers in a Pod communicate with entities outside the Pod, they must coordinate how they use the shared network resources (such as ports)



Pods - Storage

- A Pod can specify a set of shared storage volumes
- All containers in the Pod can access the shared volumes, allowing those containers to share data
- Volumes also allow persistent data in a Pod to survive in case one of the containers within needs to be restarted



Pods – Resource limits

- When Containers have resource requests specified, the scheduler can make better decisions about which nodes to place Pods on
- When Containers have their limits specified, contention for resources on a node can be handled in a specified manner

```
apiVersion: v1
kind: Pod
metadata:
  name: frontend
spec:
  containers:
  - name: db
    image: mysql
    resources:
      requests:
        memory: "64Mi"
        cpu: "250m"
      limits:
        memory: "128Mi"
```

Deployments

- A Deployment controller provides declarative updates for Pods and ReplicaSets
- You describe a desired state in a Deployment object, and the Deployment controller changes the actual state to the desired state at a controlled rate

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: hello-world-deployment
  labels:
    app: hello-app
spec:
  replicas: 3
  selector:
    matchLabels:
      app: hello-app
    spec:
      containers:
      - name: hello-world
        image: hello
```

Lab 2: Deployments

Deploy sample application

Task	With Azure
Deploy sample app	`kubectl create -f azure-vote.yaml`

Validate new app	`# get-deployments`
	`kubectl get deployments`
	
	`# describe-deployments`
	`kubectl describe deployments`
	
	`# get-replicasets`
	`kubectl get rs`
	
	`# get-pods`
	`kubectl get pods --show-labels`

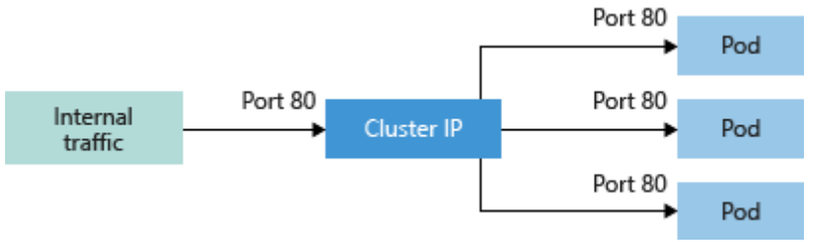
Services

- Services logically group a set of pods together and provide network connectivity
- Allow to expose endpoints (public or private)
- Several Service types are available

```
kind: Service
metadata:
  name: azure-vote-front
spec:
  type: LoadBalancer
  ports:
  - port: 80
    targetPort: 8080
  selector:
    app: azure-vote-front
```

Services – Cluster IP

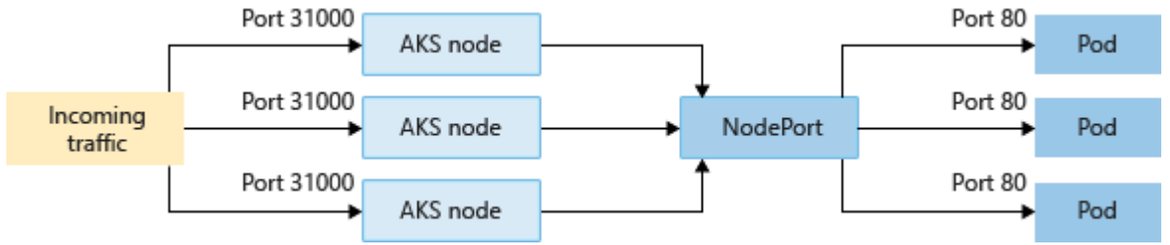
- Creates an internal IP address for use within the AKS cluster
- Good for internal-only applications that support other workloads within the cluster




```
kind: Service
metadata:
  name: azure-vote-back
spec:
  ports:
  - port: 6379
  selector:
    app: azure-vote-back
```

Services – NodePort

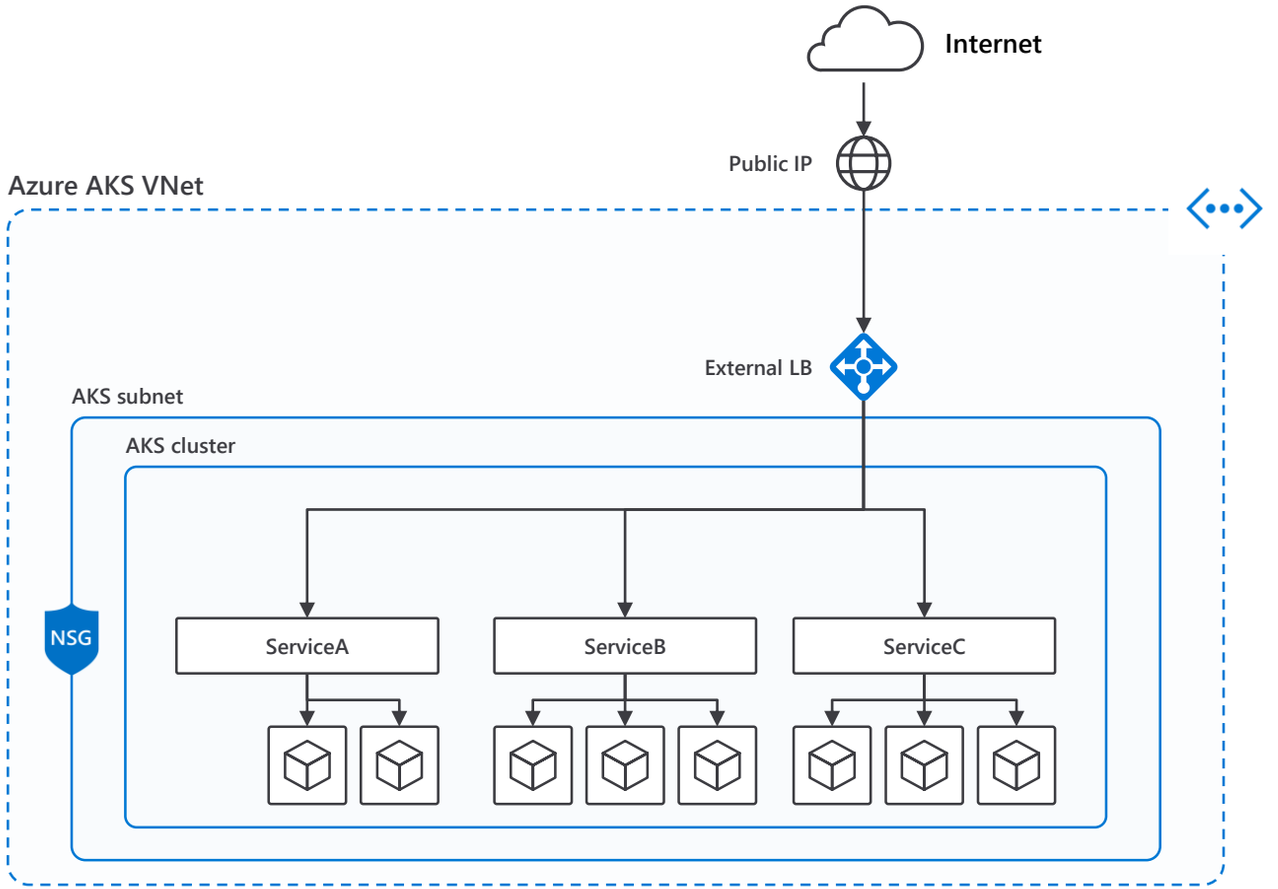
- Creates a port mapping on the underlying node that allows the application to be accessed directly with the node IP address and port




```
kind: Service
metadata:
  name: azure-vote-back
spec:
  type: NodePort
  ports:
  - port: 6379
  selector:
    app: azure-vote-back
```

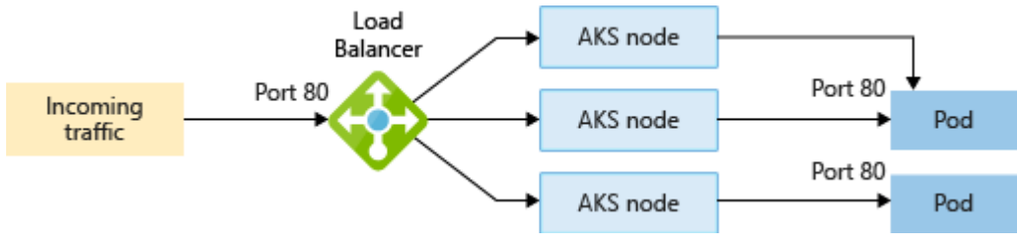
Services – Load balancer (external)

- Creates an Azure load balancer resource, configures a Public external IP address, and connects the requested pods to the load balancer backend pool
- To allow customers traffic to reach the application, load balancing rules are created on the desired ports



Services – Load balancer (external)

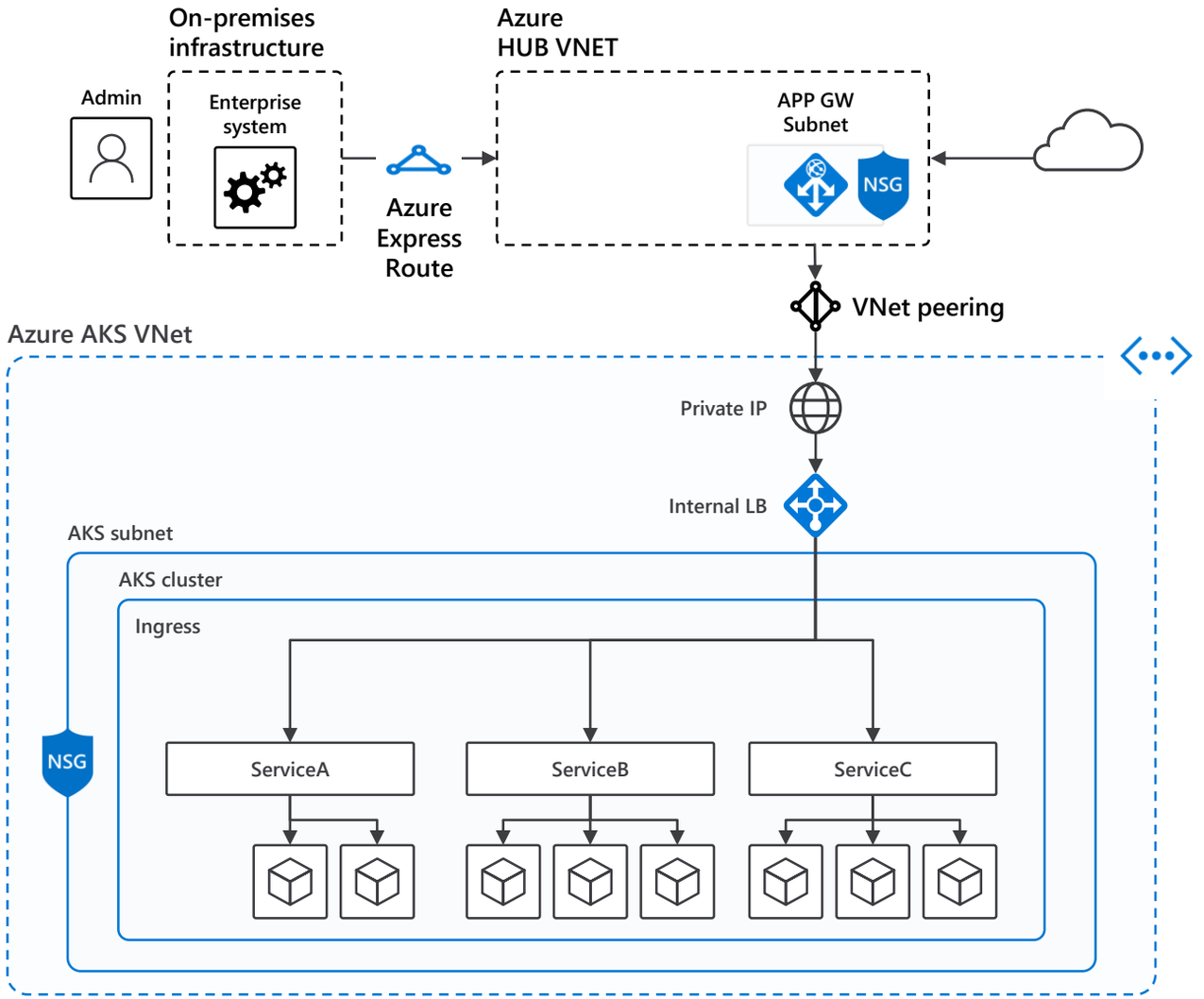
- Creates an Azure load balancer resource, configures a Public external IP address, and connects the requested pods to the load balancer backend pool
- To allow customers traffic to reach the application, load balancing rules are created on the desired ports




```
kind: Service
metadata:
  name: azure-vote-front
spec:
  type: LoadBalancer
  ports:
  - port: 80
    targetPort: 8080
  selector:
    app: azure-vote-front
```

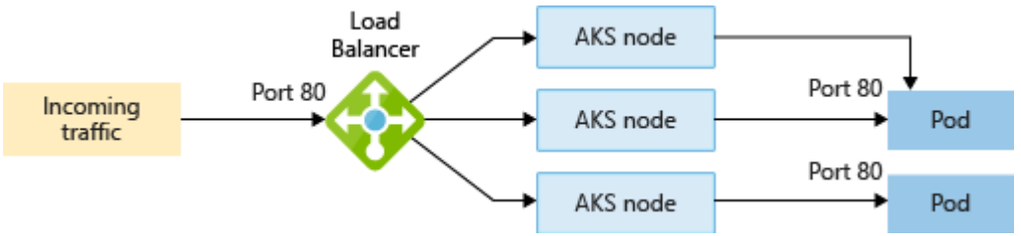
Services – Load balancer (internal)

- Creates an Azure load balancer resource, configures a VNET external IP address, and connects the requested pods to the load balancer backend pool
- To allow customers traffic to reach the application, load balancing rules are created on the desired ports



Services – Load balancer (internal)

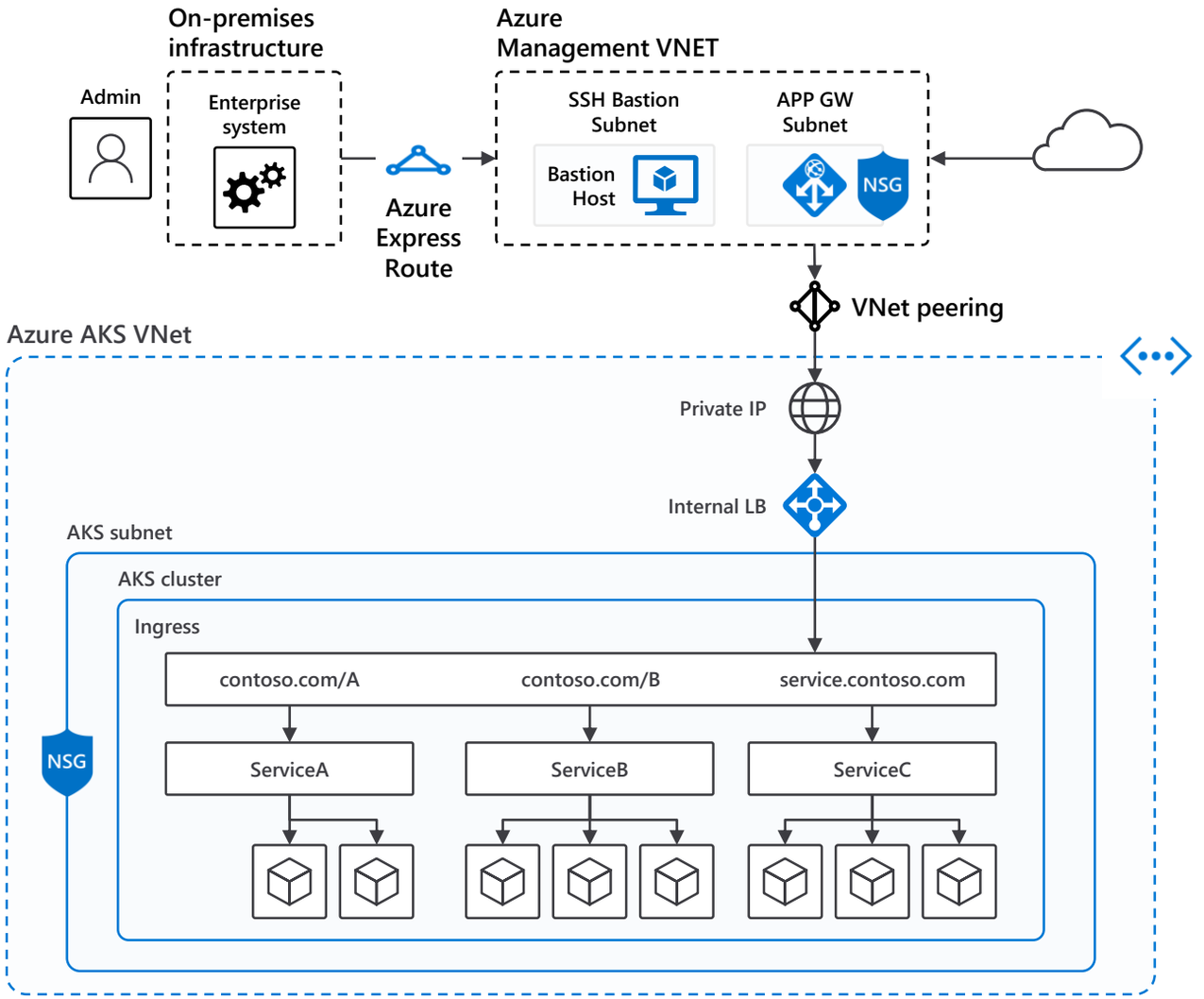
- Creates an Azure load balancer resource, configures a VNET external IP address, and connects the requested pods to the load balancer backend pool
- To allow customers traffic to reach the application, load balancing rules are created on the desired ports




```
apiVersion: v1
kind: Service
metadata:
  name: azure-vote-front-internal
  annotations:
    service.beta.kubernetes.io/azure-load-balancer-internal: "true"
spec:
  type: LoadBalancer
  ports:
  - port: 80
  selector:
    app: azure-vote-front
```

Ingress controllers

- When you create a LoadBalancer type Service, an underlying Azure load balancer resource is created. The load balancer is configured to distribute traffic to the pods in your Service on a given port. The LoadBalancer only works at layer 4 - the Service is unaware of the actual applications, and can't make any additional routing considerations.
- Ingress controllers work at layer 7, and can use more intelligent rules to distribute application traffic. A common use of an Ingress controller is to route HTTP traffic to different applications based on the inbound URL.



Lab 3: Services

Expose different types of services

Task	With Azure

Expose public service	`kubectl create -f azure-vote-service-public.yaml`
	
	`# get public IP`
	`kubectl get service azure-vote-front –watch`

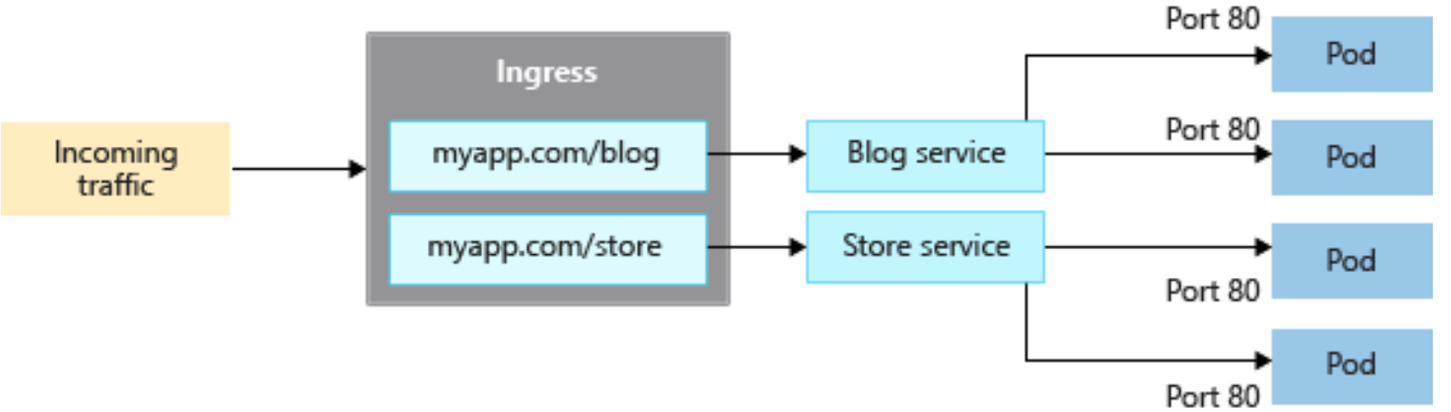
Expose private (Cluster IP) service	`kubectl create -f azure-vote-service-private.yaml`
	
	`# get IP`
	`kubectl get services`

Expose NodePort service	`kubectl create -f nodeport-service.yaml`
	
	`# get IP`
	`kubectl describe service azure-vote-front-node`

Ingress controllers

- When you create a LoadBalancer type Service, an underlying Azure load balancer resource is created. The load balancer is configured to distribute traffic to the pods in your Service on a given port. The LoadBalancer only works at layer 4 - the Service is unaware of the actual applications, and can't make any additional routing considerations.

- Ingress controllers work at layer 7, and can use more intelligent rules to distribute application traffic. A common use of an Ingress controller is to route HTTP traffic to different applications based on the inbound URL.

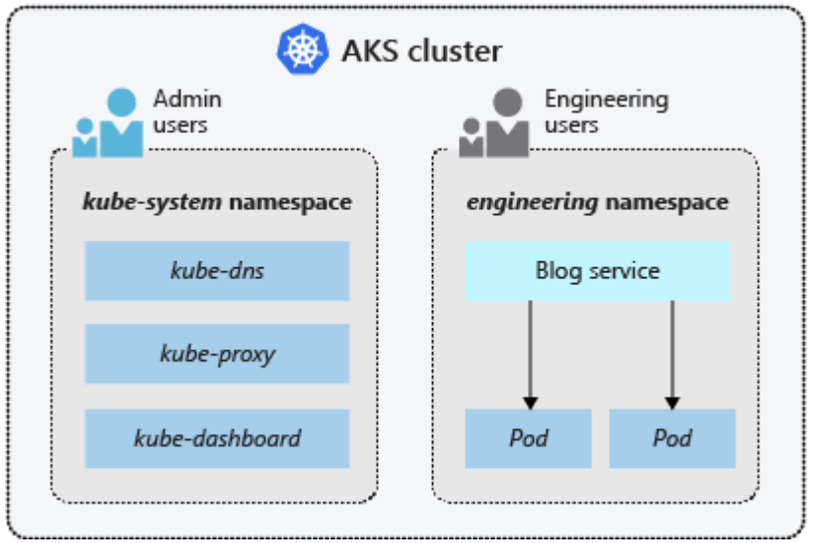


Namespaces

- Namespaces provide a scope for names. Names of resources need to be unique within a namespace
- Namespaces are a way to divide cluster resources between multiple users (via resource quota)
- A service DNS entry is of the form <service-name>.<namespace-name>.svc.cluster.local, which means that if a container just uses <service-name>, it will resolve to the service which is local to a namespace

```
$ kubectl --namespace=<namespace-name> get pods

$ kubectl config set-context $(kubectl config current-
context) --namespace=<namespace-name>
```




```
apiVersion: v1
kind: Namespace
metadata:
  name: engineering
```

Lab 4: Namespaces

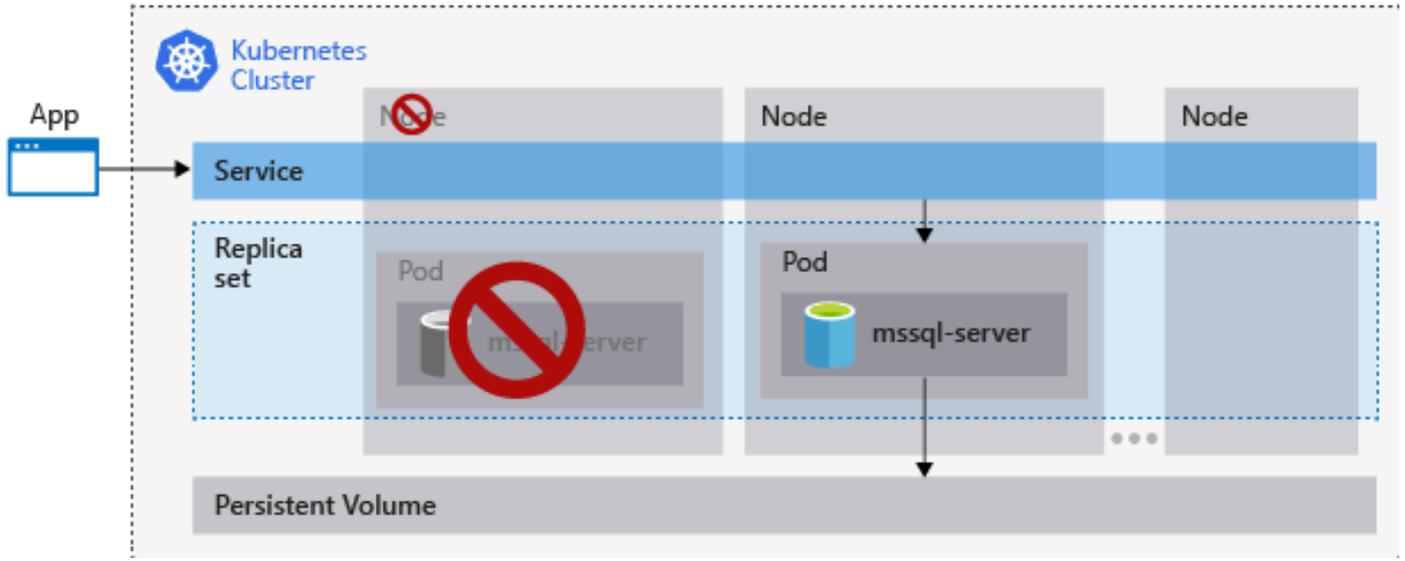
Create new namespace

Task	With Azure
Check existing namespaces	`kubectl get namespaces`
Create namespace	`kubectl create namespace aznamespace`

Check new namespace	`kubectl get namespaces`
	
	`# get pods in the new namespace`
	`kubectl --namespace=aznamespace get pods`

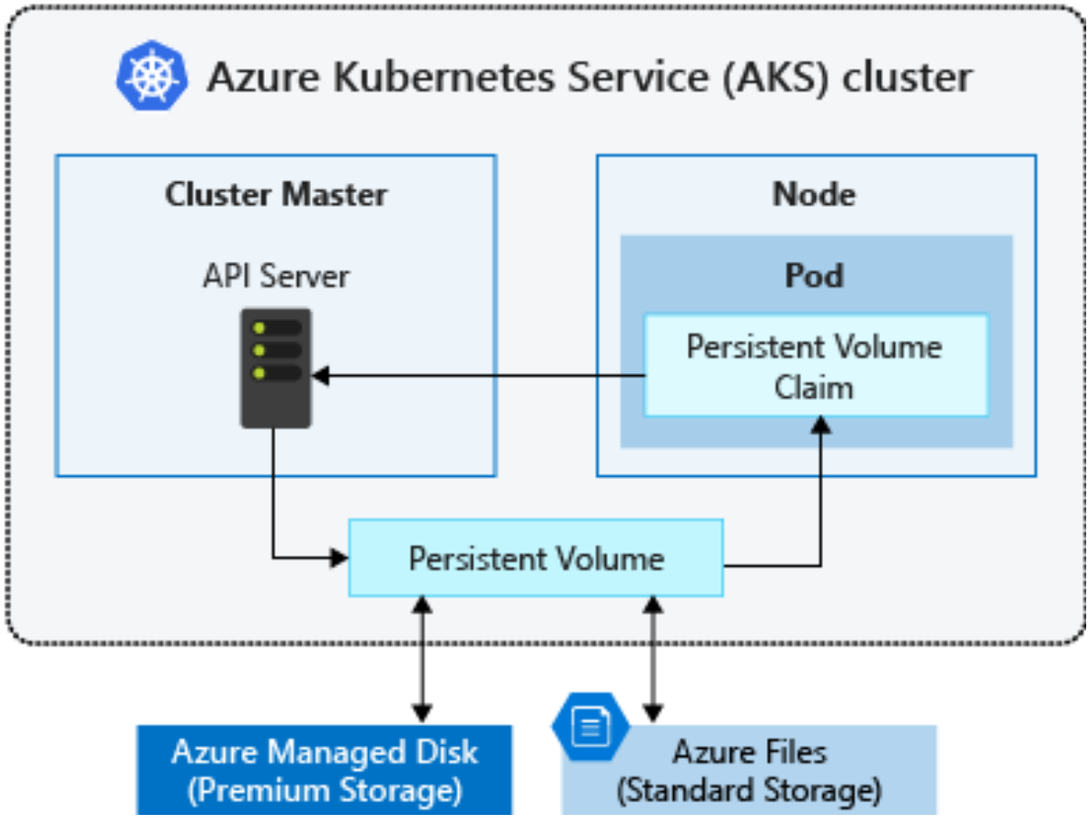
Storage and persistence

- In some cases persistent storage is a need for sharing data between containers or to guarantee high availability where data must resist to container failures



Storage and persistence on AKS

- In some cases persistent storage is a need for sharing data between containers or to guarantee high availability where data must resist to container failures



Storage – Storage classes

- A storage class is used to define how a unit of storage is dynamically created with a persistent volume
- Each AKS cluster includes two pre-created storage classes, both configured to work with Azure disks: Standard and Premium

```
kind: StorageClass
apiVersion: storage.k8s.io/v1beta1
metadata:
     name: azure-disk-standard
provisioner: kubernetes.io/azure-disk
parameters:
  storageaccounttype: Standard_LRS
  kind: Managed
---
kind: StorageClass
apiVersion: storage.k8s.io/v1beta1
metadata:
     name: azure-disk-premium
provisioner: kubernetes.io/azure-disk
parameters:
  storageaccounttype: Premium_LRS
  kind: Managed
```

Storage – Persistent volumes

- A persistent volume claim (PVC) is used to automatically provision storage based on a storage class

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: my-data-pv-claim
  annotations:
    volume.beta.kubernetes.io/storage-class: default
spec:
  accessModes:
  - ReadWriteOnce
  resources:
    requests:
      storage: 2Gi
```

Storage – Using persistent volumes

- Using a persistent volume claim (PVC) in a Pod and mounting it in a mount point

```
kind: Pod
apiVersion: v1
metadata:
  name: task-pv-pod
spec:
  volumes:
    - name: task-pv-storage
      persistentVolumeClaim:
       claimName: my-data-pv-claim
  containers:
    - name: task-pv-container
      image: nginx
      ports:
        - containerPort: 80
          name: "http-server"
      volumeMounts:
        - mountPath: "/usr/share/nginx/html"
          name: task-pv-storage
```

Lab 5: Persistent Volumes

Create and use new persistent volume

Task	With Azure

Check storage classes	`kubectl get storageclasses`
	
	`# create new storage class if needed`
	`kubectl create -f sample-storageclass.yaml`

Create persistent volume	`kubectl create -f sample-pvc.yaml`
	
	`# get persistent volumes`
	`kubectl get pv`
	`kubectl describe pv`

Use persistent volume	`kubectl apply -f test-persistent-volumes.yaml`
	
	`# get pods using persistent volume`
	`kubectl get pod task-pv-pod`
	
	`# access persistent volume (write something, destroy the pod and repeat the test again)`
	`kubectl exec -it task-pv-pod -- /bin/bash`

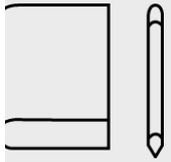
Best support for your enterprise need

Kubernetes 101 Docs
aka.ms/LearnAKS

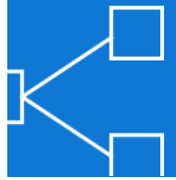
Best practices
aka.ms/aks/bestpractices

Hear from experts
aka.ms/k8s/lightboard

Case studies
aka.ms/aks/casestudy



Microservices architecture
aka.ms/aks/microservices



Try for free
aka.ms/aks/trial

Feedback on the roadmap? Tell us at https://aka.ms/aks/feedback

Microsoft Azure

Thank you.