Microsoft

Cloud Native Transformation of Applications Using Azure Kubernetes

Week 1 – Containers, Azure Container Registry and Azure Kubernetes Service

About the Speakers

Steve Caravajal, Ph.D. is Chief Cloud Solution Architect with extensive cloud experience, and 15+ yrs with Microsoft. He is focused on leading digital transformation and cloud native modernization for Microsoft's large strategic accounts. Steve is also Cloud Security lead, SME for multi-vendor cloud strategy, and K8s certified (CKA, CKAD).

David Hoerster, a former 6-time .NET MVP, has been working with the Microsoft.NET Framework since the early 1.0 betas and has recently found his next passion in Open Source technologies. He is currently a Cloud Solutions Architect at Microsoft specializing in application development and identity. He also recently earned his CKA and CKAD.

4 Week Agenda Overview

Week 1 – July 14
Containers, Azure Kubernetes Service (AKS), Azure Container Registry (ACR)
Establish foundation to enable advanced implementations and configuration in Weeks 2-4.

Week 2 – July 21
Storage, Config-maps, Namespace, Packaging and Deployment Templates and YAML

Week 3 – July 28
AKS Networking, Managing Ingress and Container Security

Week 4 – Aug 4
Deploying a Distributed Application
Monitoring, and Service Mesh

Kubernetes Topics to be covered in 4 sessions

Overall Goal: Deploy a distributed app, with full features enabled

Nodes / Pods	ReplicaSet	Deployment
Services	Namespace / Context	Storage / Volumes
config-map	Security / Secrets / AAD / KeyVault	Ingress / Egress
Monitoring / Logging	Data Management	Networking

Kubernetes Topics – Week 1

Nodes / Pods	ReplicaSet	Deployment
Services	Namespace / Context	Storage / Volumes
config-map	Security / Secrets / AAD / KeyVault	Ingress / Egress
Monitoring / Logging	Data Management	Networking

Learning Expectations for Week1

Introduction to key concepts

- Why Containers and why Kubernetes
- Containerization Process and Walkthrough for Multi-Container App
- Azure Container Registry and container image deployment
- Overview of Kubernetes Architecture and key components
- Command line provisioning of an AKS cluster
- Deployments and Manifests
- Lab

Containers

Why Use Containers?

The Container advantages…

Standardise Build

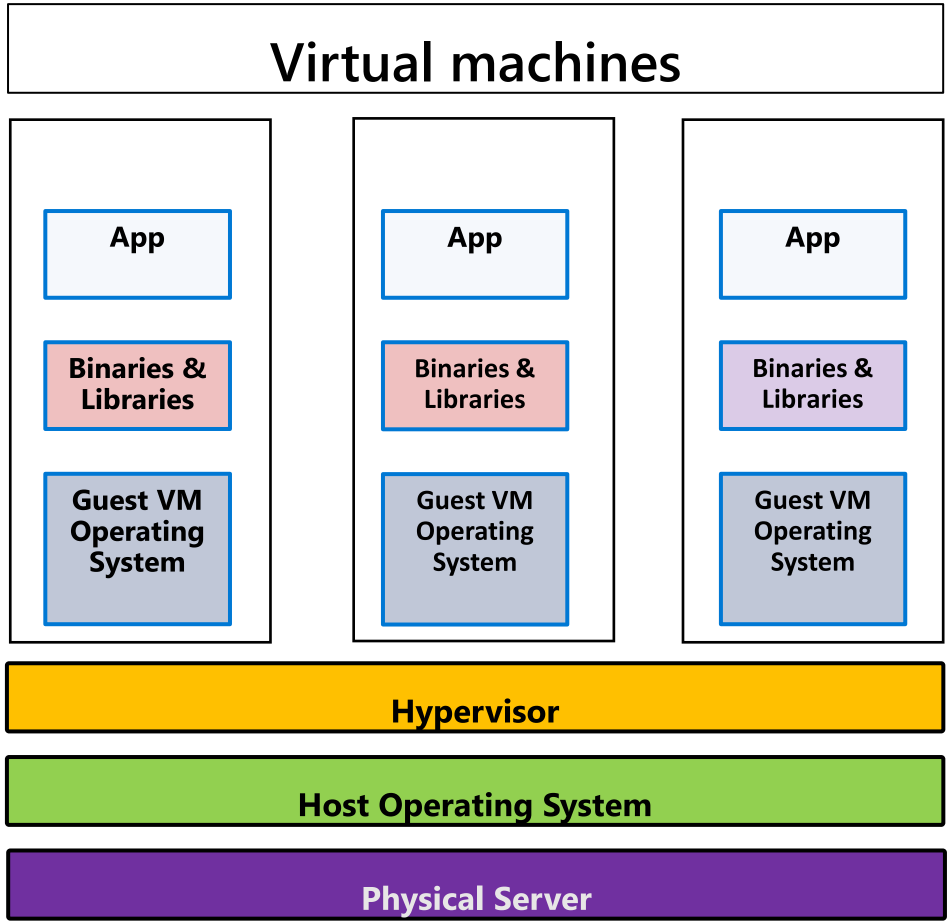
DevOps CI / CD

Run Anywhere

Rapid Deployment

Compute Density & Scale

Containers vs VMs?



Virtual machines

App

Binaries & Libraries

Guest VM Operating System

App

Binaries & Libraries

Guest VM Operating System

App

Binaries & Libraries

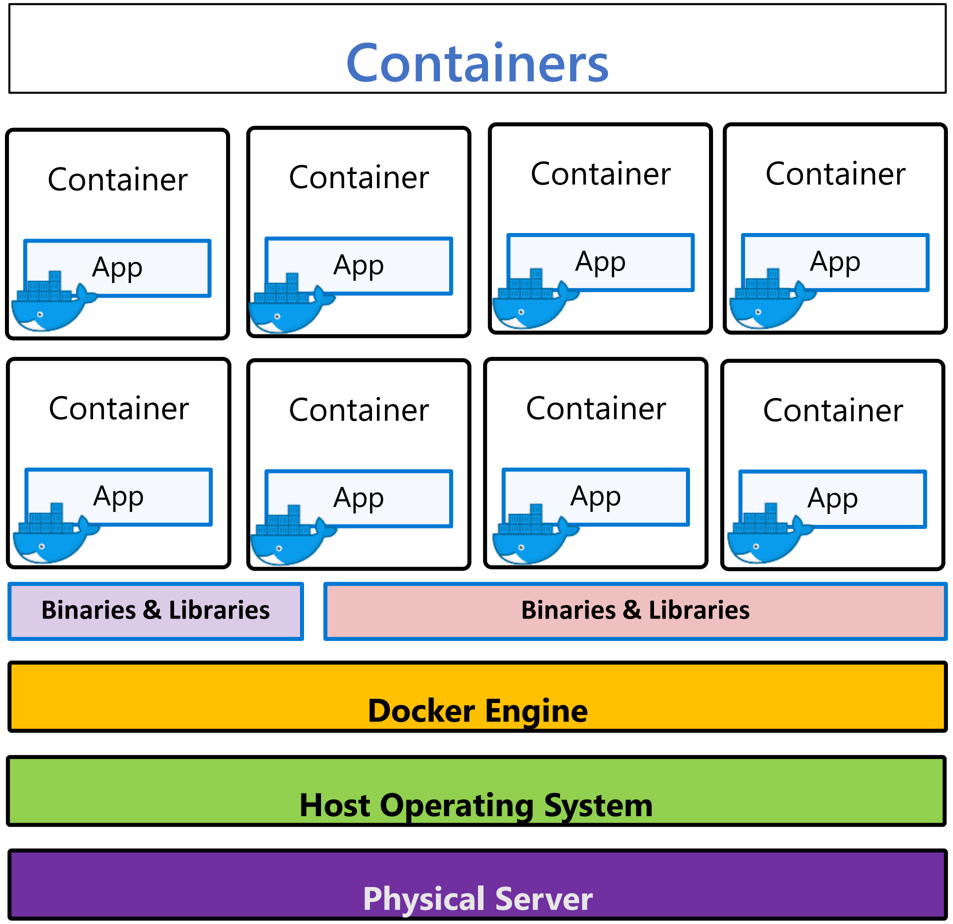
Guest VM Operating System

Hypervisor

Host Operating System

Physical Server

- Virtualize the **hardware**
- **VMs** as units of scaling
- Hypervisor **dependent**
- **Not** easily movable



Containers

Container

App

Container

App

Container

App

Container

App

Container

App

Container

App

Container

App

Container

App

Binaries & Libraries

Binaries & Libraries

Docker Engine

Host Operating System

Physical Server

- Virtualize the **operating system**
- **Applications** as units of scaling
- Platform **independent**
- **Easily** movable across environments (on-premises, multi-cloud)

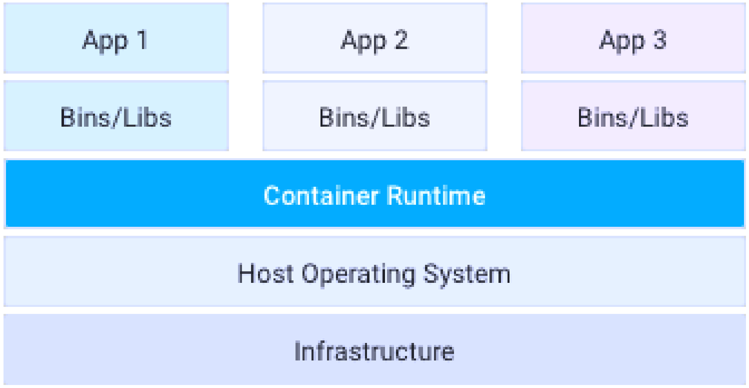
Containers vs. virtual machines

Docker containers

Shared host OS kernel

Portability

Faster provisioning and scalability

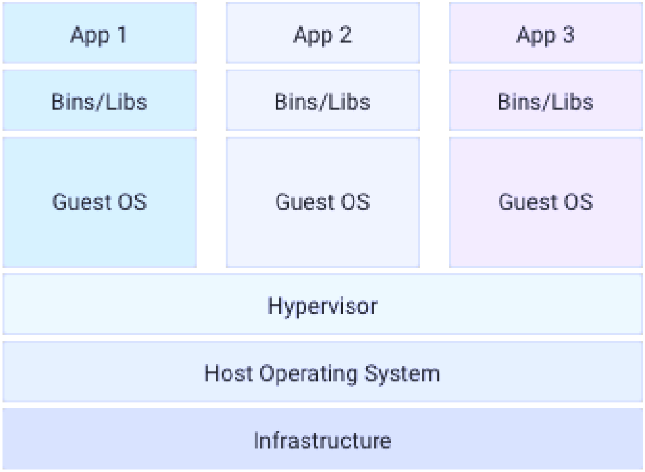


Virtual machines

Separate OS per instance

Large footprint

Slower provisioning



Let’s start the containerization process…

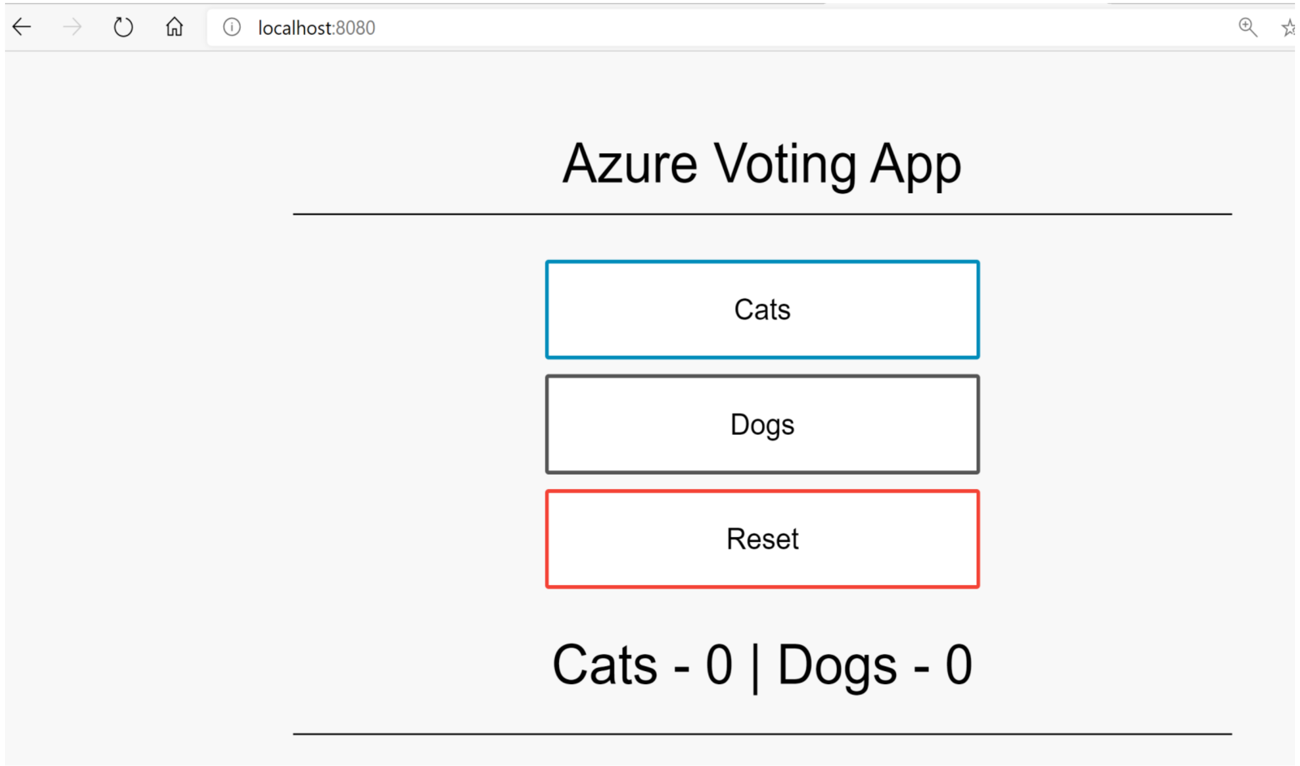
The “Hello World” of Multi-Container Apps

Azure Voting App

A multi-container application built using Python / Flask. The data component is using Redis.

A local Docker development environment and tools are used to build, test and push to Azure and private container registry.

Illustrate the containerization process, push to the cloud, and then implementation using Azure Kubernetes Service.



Azure Voting App

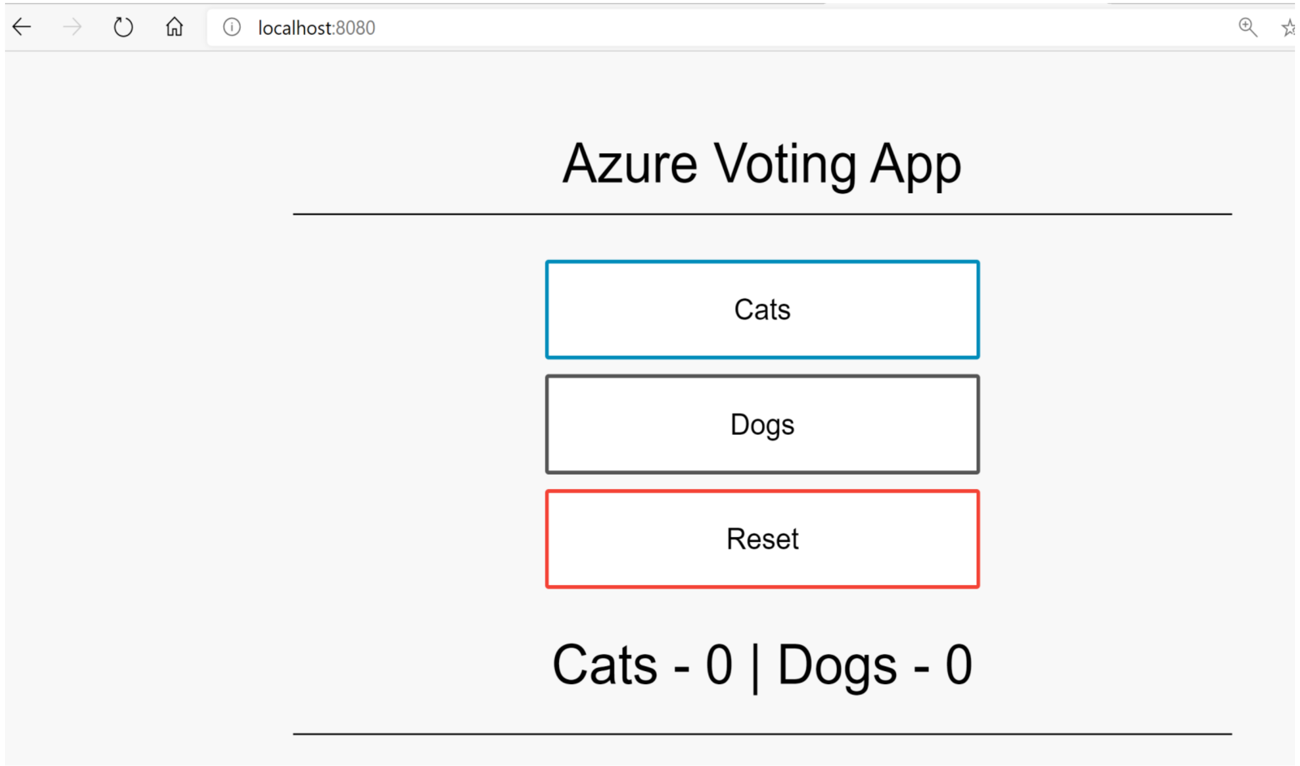
Cats - 0 | Dogs - 0

Public Code Repo
https://github.com/Azure-Samples/azure-voting-app-redis

Prepare a multi-container application for Kubernetes

General Process

1. Create A Docker Image
2. Create An Azure Container Registry
3. Push The Docker Image To ACR
4. Create A Kubernetes Cluster
5. Create Deployment Manifests
6. Deploy Your Application



localhost:8080

Azure Voting App

Cats - 0 | Dogs - 0

Public Code Repo
https://github.com/Azure-Samples/azure-voting-app-redis

1. Create Docker Image

Docker tools installed locally

Create and start containers and leave running in the background

```
docker-compose up -d
```

```yaml
version: '3'
services:
  azure-vote-back:
    image: redis
    container_name: azure-vote-back
    ports:
        - "6379:6379"

  azure-vote-front:
    build: ./azure-vote
    image: azure-vote-front
    container_name: azure-vote-front
    environment:
      REDIS: azure-vote-back
    ports:
        - "8080:80"
```

1. Create Docker Image

Docker tools installed locally

Containerization / K8s Process

See created images

```
docker images
```

```
REPOSITORY                    TAG           IMAGE ID        SIZE
azure-vote-front              latest        47607e3602d9    944MB
redis                         latest        235592615444    104MB
tiangolo/uwsgi-nginx-flask    python3.6     42c10a01539f    944MB
```

View running containers

```
docker ps
```

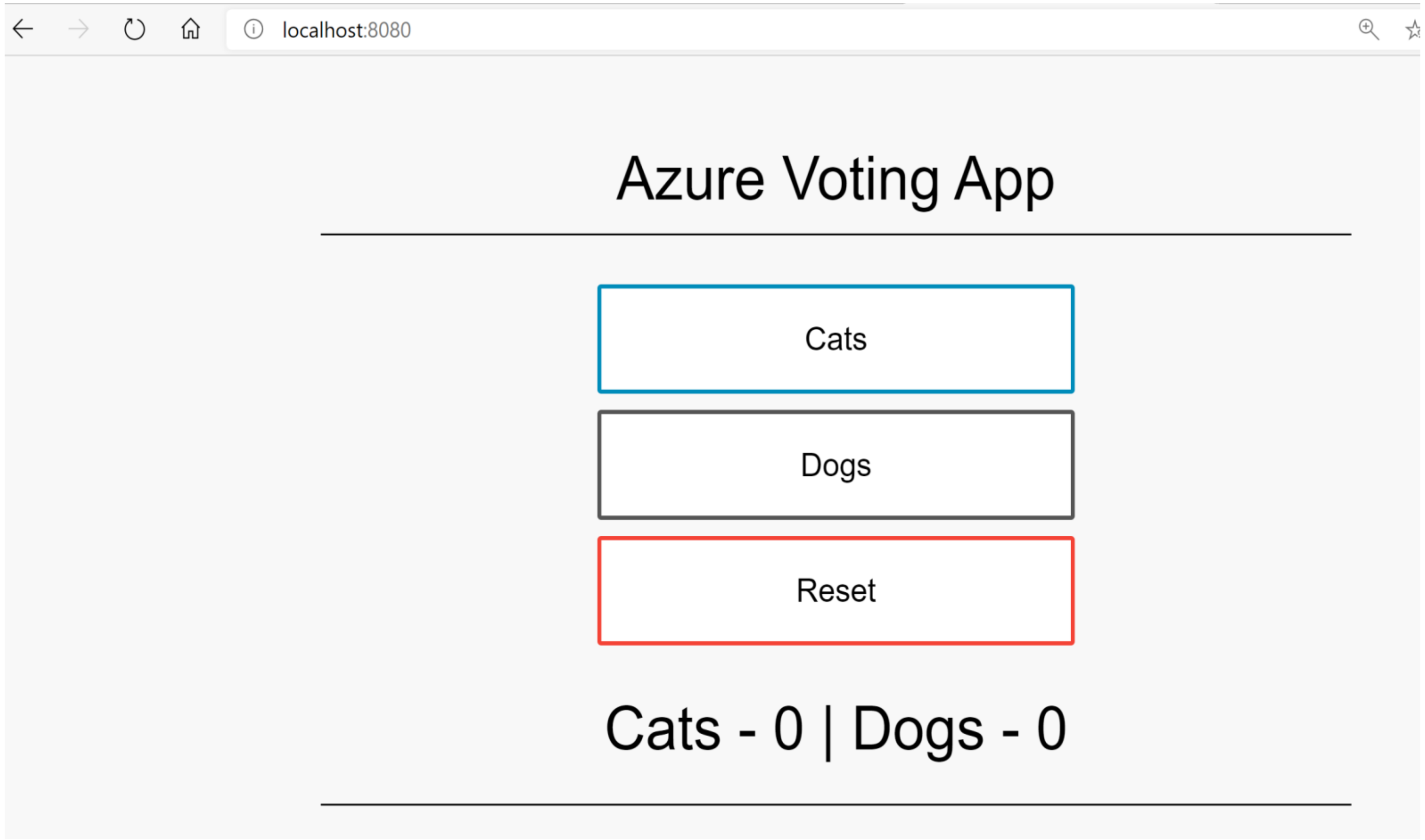
```
CONTAINER ID    IMAGE             PORTS                               NAMES
2a36c6259b7f    azure-vote-front  443/tcp, 0.0.0.0:8080->80/tcp       azure-vote-front
68aeb707d0b4    redis             0.0.0.0:6379->6379/tcp              azure-vote-back
```

1. Create Docker Image and test

Docker tools installed locally

Containerization / K8s Process

Test the application locally



localhost:8080

Azure Voting App

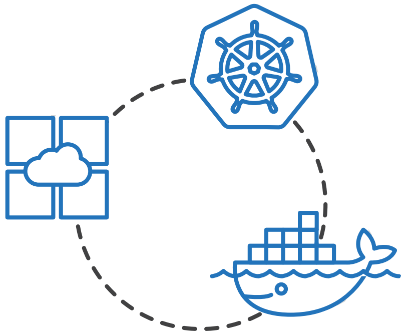
Cats - 0 | Dogs - 0

We are ready to push the container to a private repo!

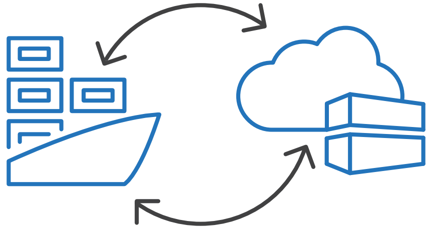
Azure Container Registry

What is an Azure container registry (ACR)?

Manage a Docker **private registry** as a first-class Azure resource



Manage images for all types of containers

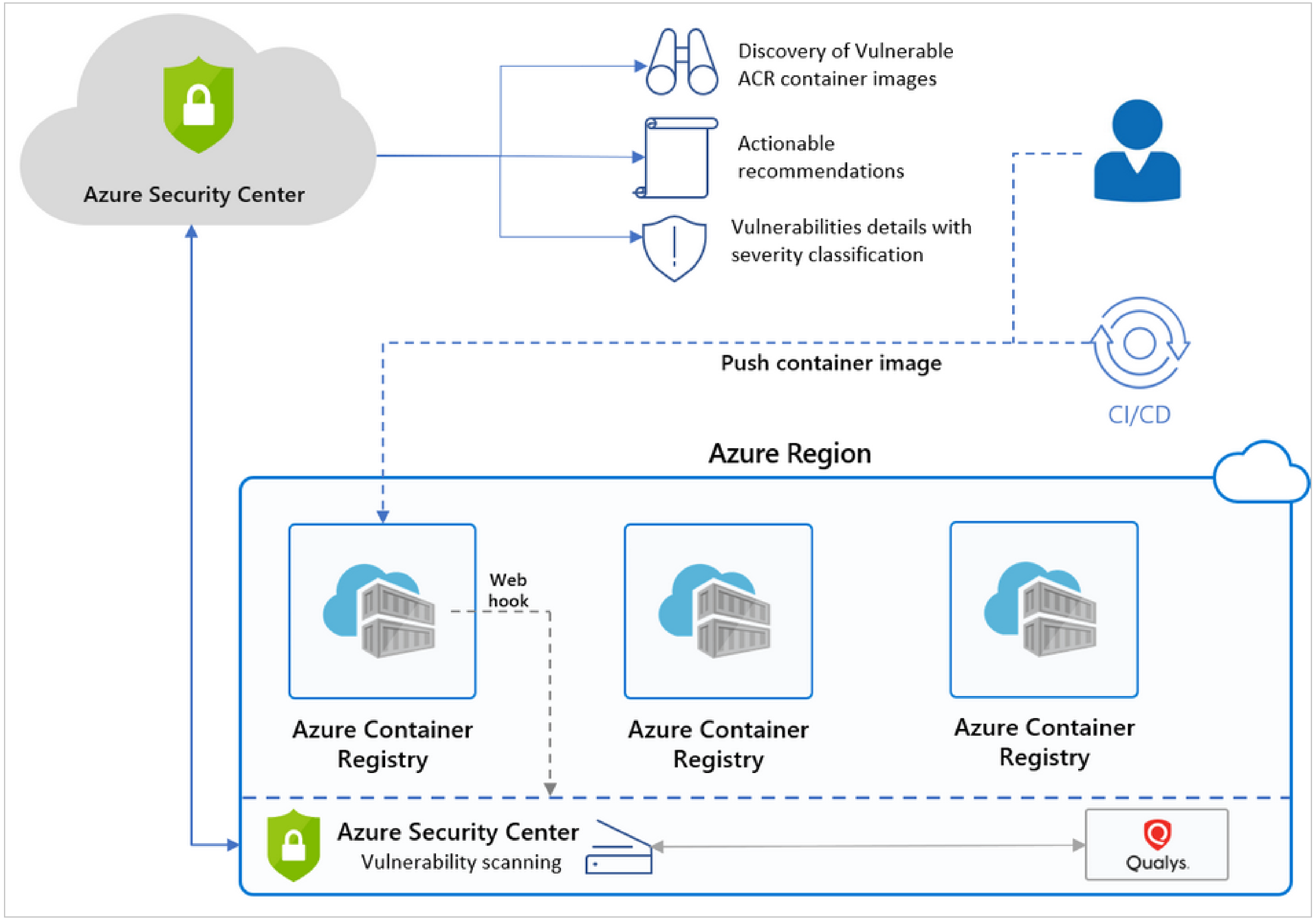


Use familiar, open-source Docker CLI tools

Azure private container registry geo-replication

Azure Container Registry
vulnerability scanning

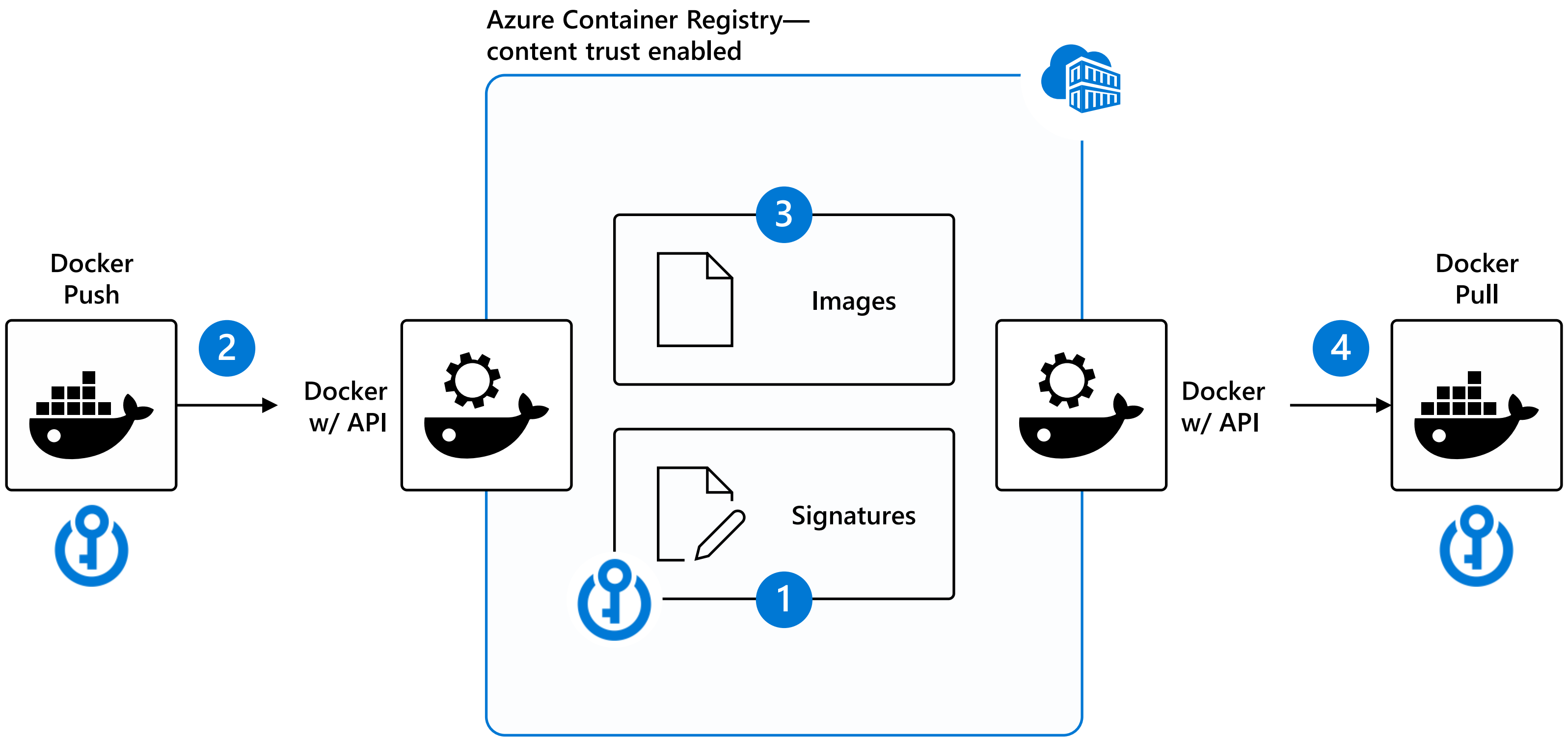
1. Developer/CI system builds container image
2. Image pushed to Azure Container Registry triggers image scan
3. Security Center scans the image and publishes to dashboard
4. Azure Container Registry scans content leveraging Qualys scanner
5. Azure Container Registry publishes the image to the repository



ACR Content Trust

securing images from source to destination

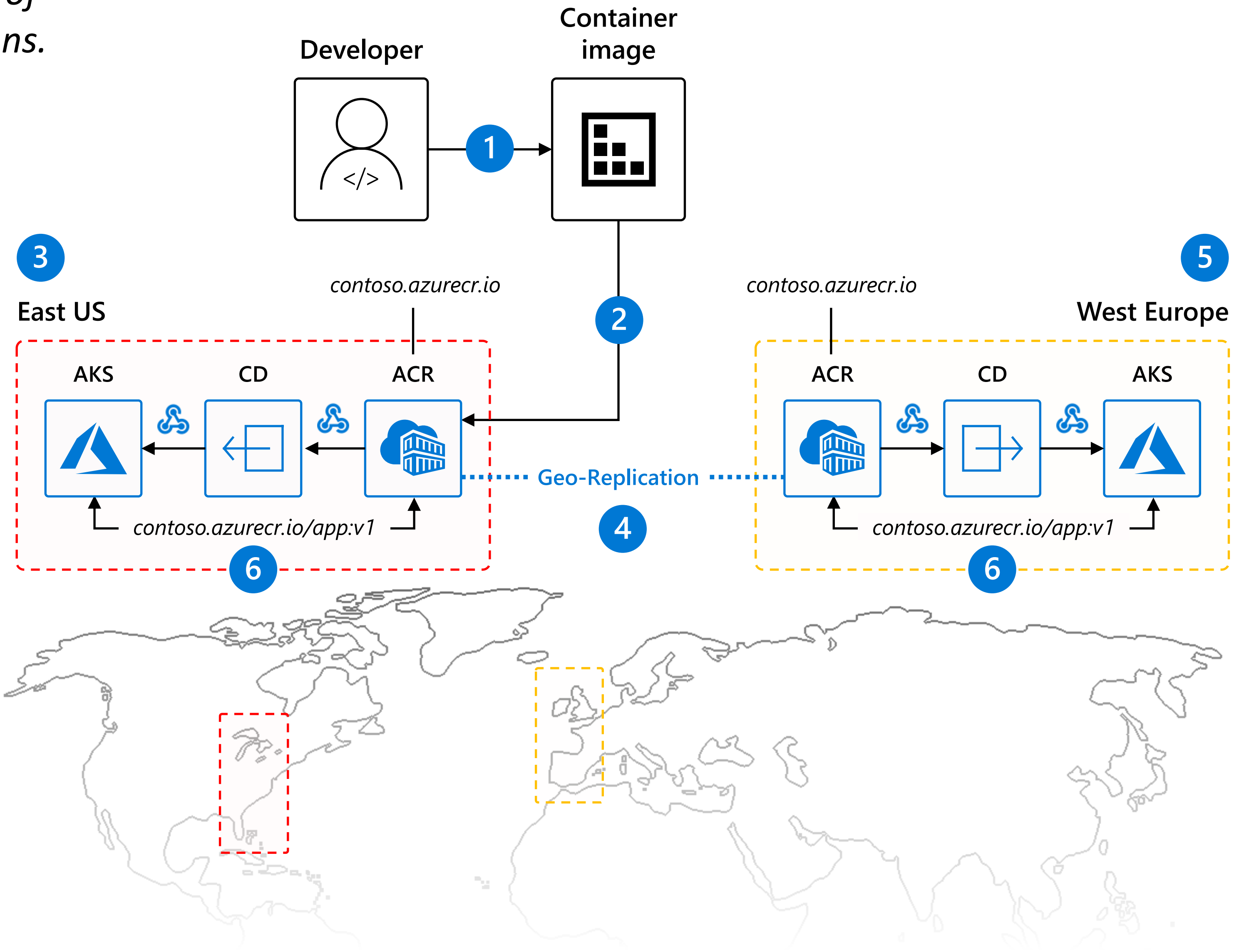
1. A set of cryptographic signing keys are associated with Azure Container Registry and used for image signing
2. Image publisher signs the image and pushes to the Azure Container Registry
3. Signed image is stored in Azure Container Registry
4. When an image consumer pulls a signed image, their Docker client verifies the integrity of the image



Azure Container Registry geo-replication

Push image to a single registry and ACR takes care of geographical replication, including local notifications.

1. US-based developer commits codes to build container image
2. Image is pushed to the nearest Azure Container Registry (ACR) region based on DNS
3. Geographical webhook triggers deployment to East US
4. ACR geo-replicates to configured regions
5. Geographical webhook triggers deployment to West Europe
6. Both AKS clusters pull from contoso.azurecr.io



Let’s create a ACR and Push the containers…

2. Create ACR Registry

Azure CLI

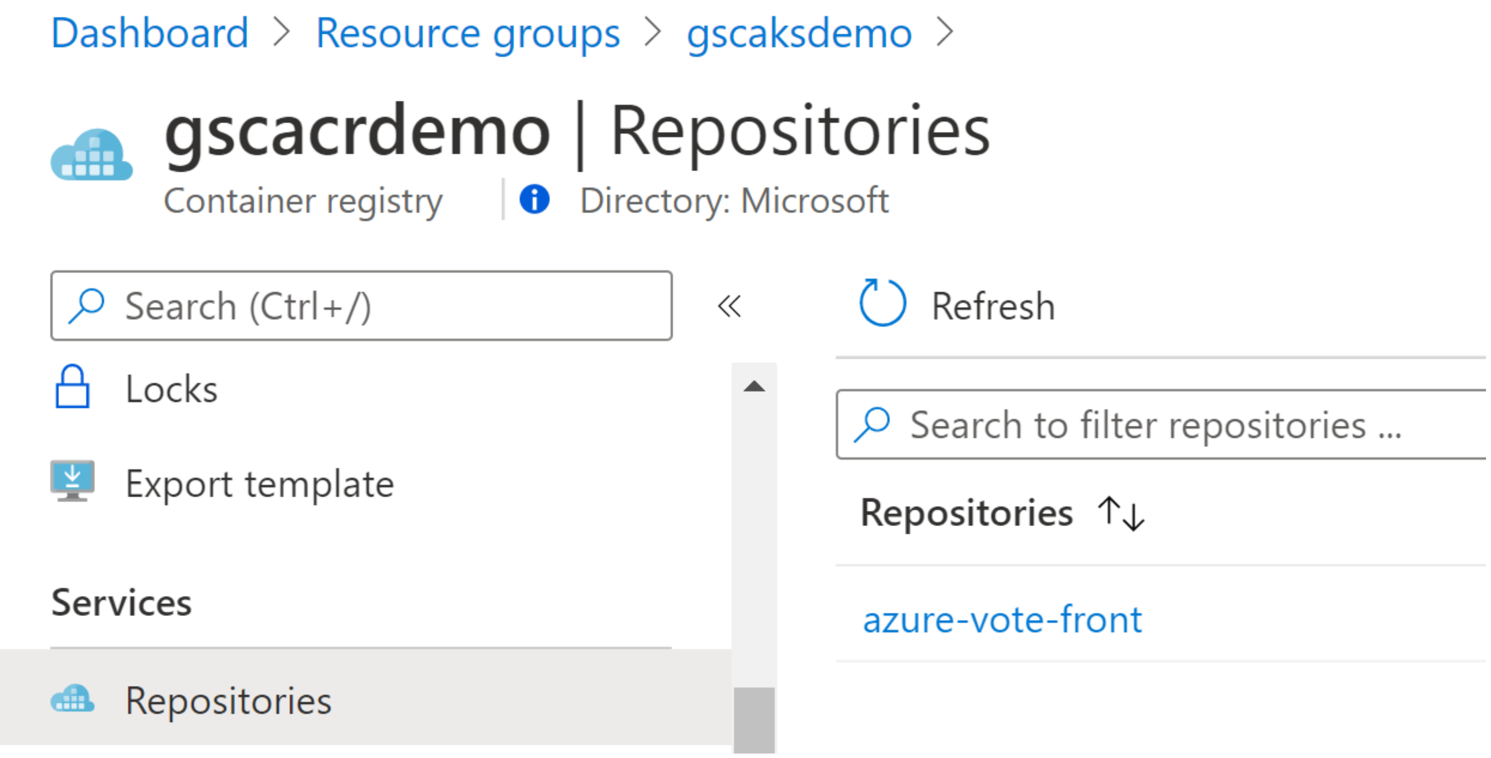
```
az acr create -g gscaksdemo --n gscacrdemo

az acr list -g gscaksdemo -o table
```

```
NAME        RESOURCE GROUP    LOCATION    SKU    LOGIN SERVER
----------  ----------------  ----------  -----  ---------------------
gscacrdemo  gscaksdemo        eastus      Basic  gscacrdemo.azurecr.io
```

3. Push local image to ACR

```
docker push gscacrdemo.azurecr.io/azure-vote-front:v1
```



Dashboard > Resource groups > gscaksdemo >

gscacrdemo | Repositories

Container registry | Directory: Microsoft

Repositories ↑↓
azure-vote-front

Ok, I’ve got containers.

And a registry with containers.

Now what?

Azure Container Hosting Options

Azure Hosts: Running Containers on Azure

App Service

Deploy web applications or Web APIs in a PaaS environment

Service Fabric

Modernize .NET applications in microservices using Windows Server

Kubernetes Service

Orchestrate and scale the Linux or Windows Containers

Container Instance

Easily run containers on Azure without managing servers

Ecosystem Partner

Bring your partner solutions that work perfectly on Azure

Azure Container Registry

Docker Hub

Choice of developer tools and clients

Why Kubernetes and AKS

Kubernetes momentum

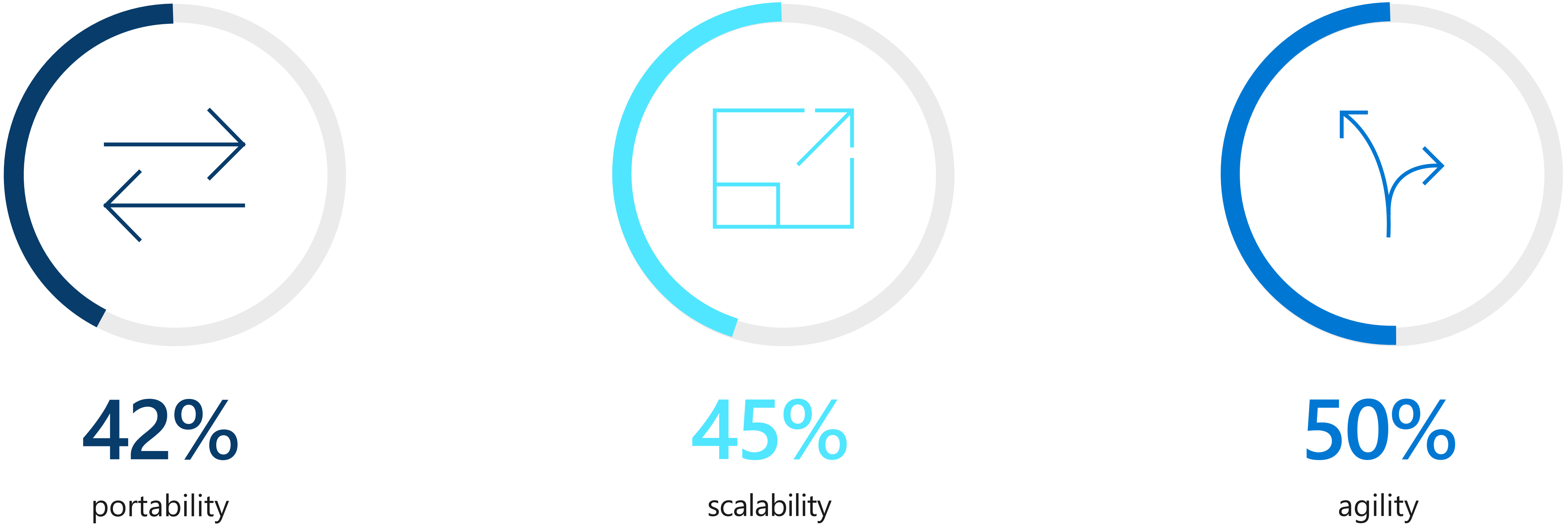
"Prediction: By 2022, more than 75% of global organizations will be running containerized applications in production, which is a significant increase from fewer than 30% in 2018.."

"Containers and Kubernetes are becoming the foundation for building cloud-native infrastructure to improve software velocity and developer productivity."

Gartner.

What’s behind the growth?

Kubernetes: the leading orchestrator shaping the future app development and management



The perceived benefits of Kubernetes

Source: Cloud Native Computing Foundation

Container Management at Scale

Cluster Management: deploy and manage cluster resources

Scheduling: where containers run

Lifecycle and Health: keep containers running despite failure

Naming and Discovery: where are my containers

Load Balancing: evenly distribute traffic

At the end of the day we need something to help us with all the orchestration..
An orchestrator!

Scaling
make se
contain
elastic i
number

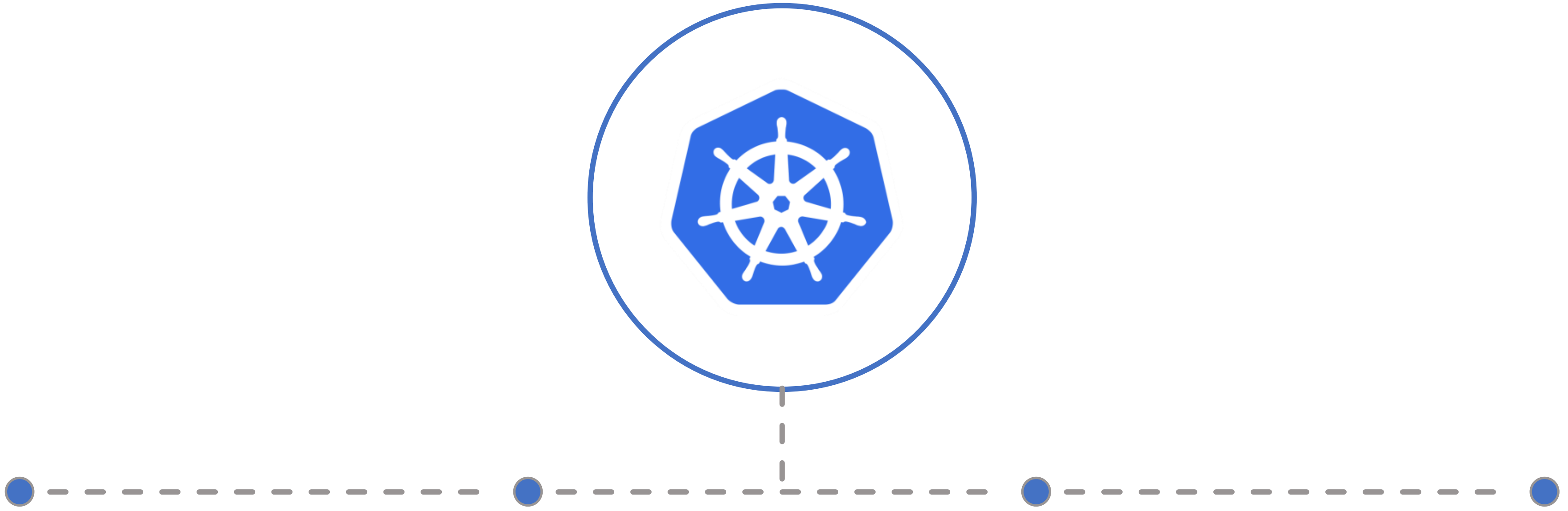
container
images

containers and
cluster

data
ers

Kubernetes: empowering you to do more

The de-facto orchestrator



Deploy your applications quickly and predictably

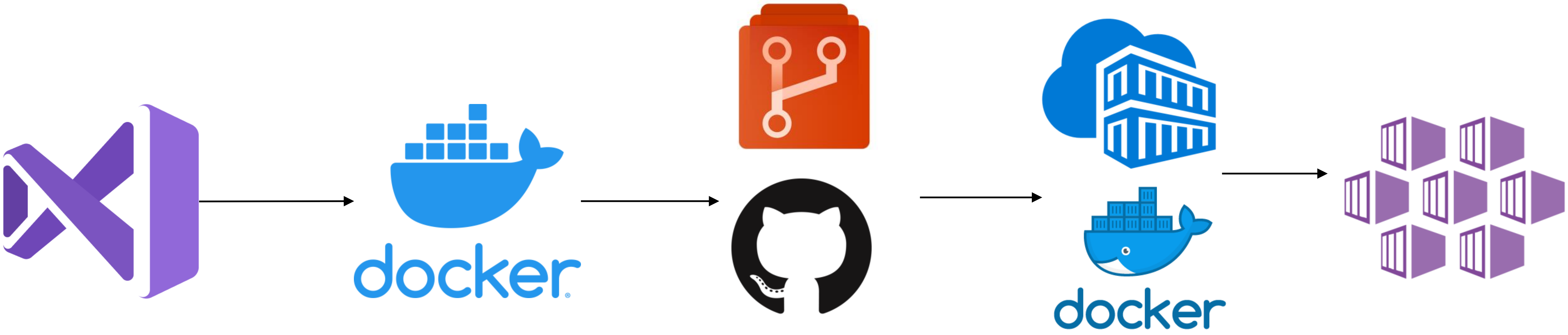
Scale your applications on the fly

Roll out new features seamlessly

Limit hardware usage to required resources only

CLOUD NATIVE
COMPUTING FOUNDATION

Deploy a Modern Applications for Azure Kubernetes Service (AKS)



Create and Push images with docker desktop

Commit Azure Repos and build Docker images

Push Docker images to Azure container registry

Deploy to AKS

AKS Managed Kubernetes

What is Azure Kubernetes Service (AKS)?

- Managed Kubernetes as a Service
- Kubernetes **Master** (Control Plane) provided as a PaaS offering and managed by Azure
- Kubernetes **Worker** Nodes (agent nodes) for application hosting containerized apps
- Customer pays compute costs for Worker Nodes only

Some Key Kubernetes Concepts for today

Node
A worker machine (VM) normally clustered, each capable of running pods

Deployment
A logical object for managing a replicated application (i.e. set of pods)

Labels and Selectors
Metadata attached to any object for configuration and selection

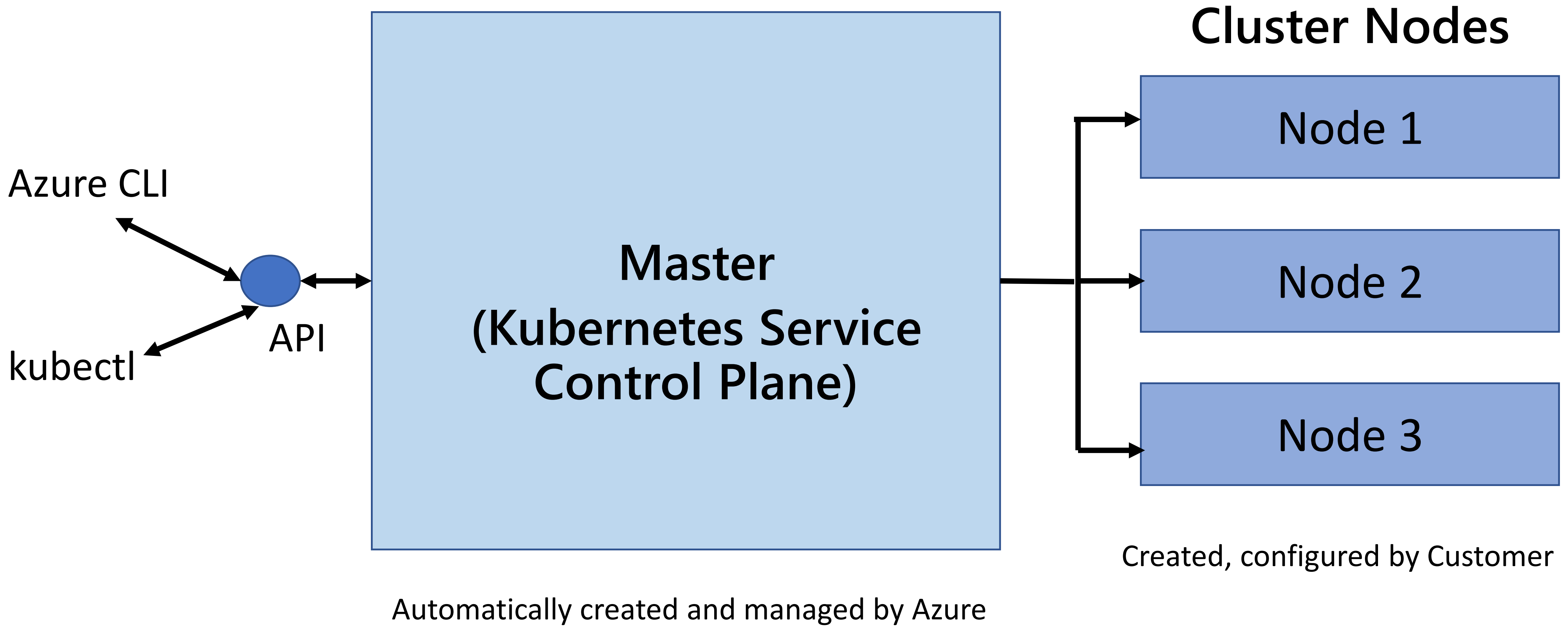
Pod
A group of one or more tightly coupled containers that is lifecycle managed

Service
Network access to a resource, e.g. pod or port. Typically load balanced

ReplicaSet
A set of one or more pods that is distributed and replicated across Nodes

Kubernetes Simplified Cluster Architecture

High Level 3-Node Cluster



Azure CLI

kubectl

API

Master
(Kubernetes Service Control Plane)

Automatically created and managed by Azure

Cluster Nodes

Node 1

Node 2

Node 3

Created, configured by Customer

Let’s create a cluster…

and see what’s created…

4. Create an AKS Cluster

Azure CLI and kubectl

```
az aks create -g gscaksdemo -n gscmyCluster --attach-acr gscacrdemo

az aks get-credentials -g gscaksdemo -n gscmyCluster

kubectl get nodes
```

```
NAME                               STATUS   Roles      AGE       VERSION
aks-nodepool1-26286499-vmss000000  Ready    agent      7m8s      v1.15.11
aks-nodepool1-26286499-vmss000001  Ready    agent      6m50s     v1.15.11
aks-nodepool1-26286499-vmss000002  Ready    agent      7m9s      v1.15.11
```

AKS Cluster Provisioning

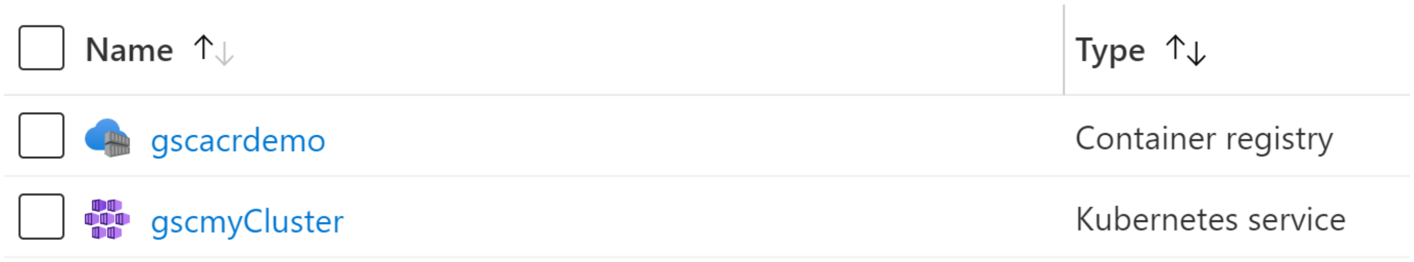
Azure CLI

```
az aks create -g gscaksdemo -n gscmyCluster --attach-acr gscacrdemo
```

- Creates a new managed Kubernetes cluster and provisions resources
- Register the application and creates a Service Principal in Azure AD
- Creates 2 Resource Groups
- Provisions several resources by default
- Configure AKS and ACR integration

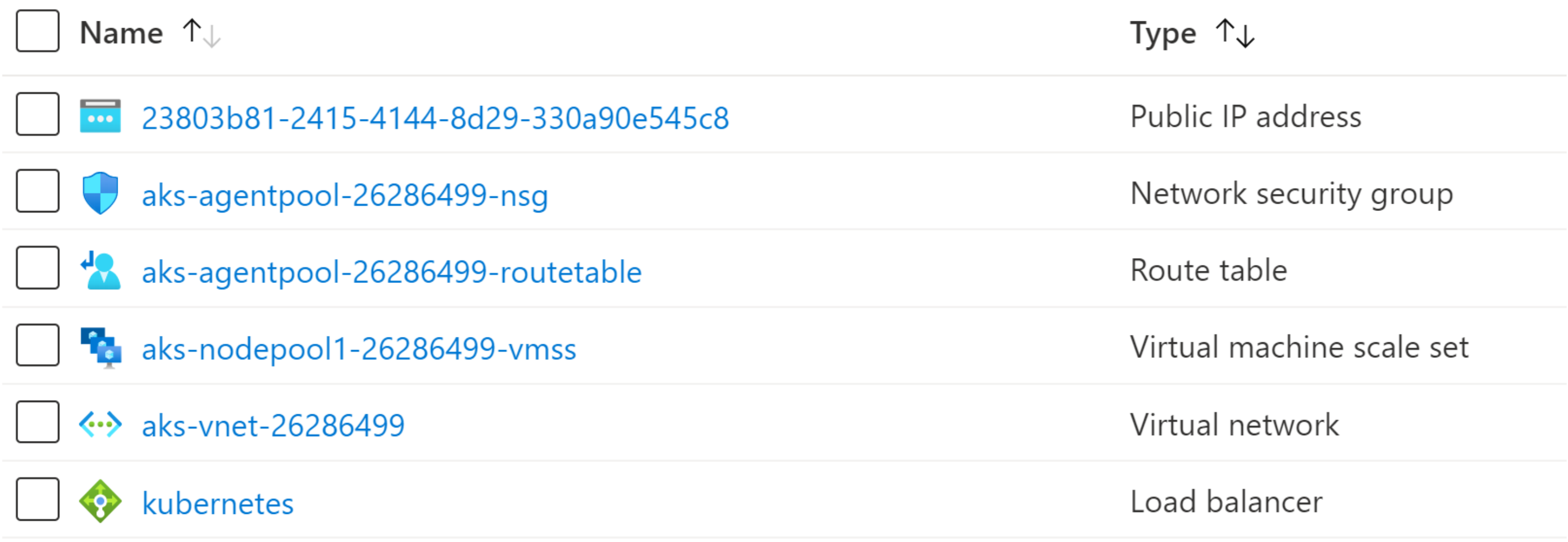
Why are there 2 Resource Groups?

gscaksdemo



□	Name ↑↓	Type ↑↓
□	gscacrdemo	Container registry
□	gscmyCluster	Kubernetes service

MC_gscaksdemo_gscmyCluster_eastus



□	Name ↑↓	Type ↑↓
□	23803b81-2415-4144-8d29-330a90e545c8	Public IP address
□	aks-agentpool-26286499-nsg	Network security group
□	aks-agentpool-26286499-routetable	Route table
□	aks-nodepool1-26286499-vmss	Virtual machine scale set
□	aks-vnet-26286499	Virtual network
□	kubernetes	Load balancer

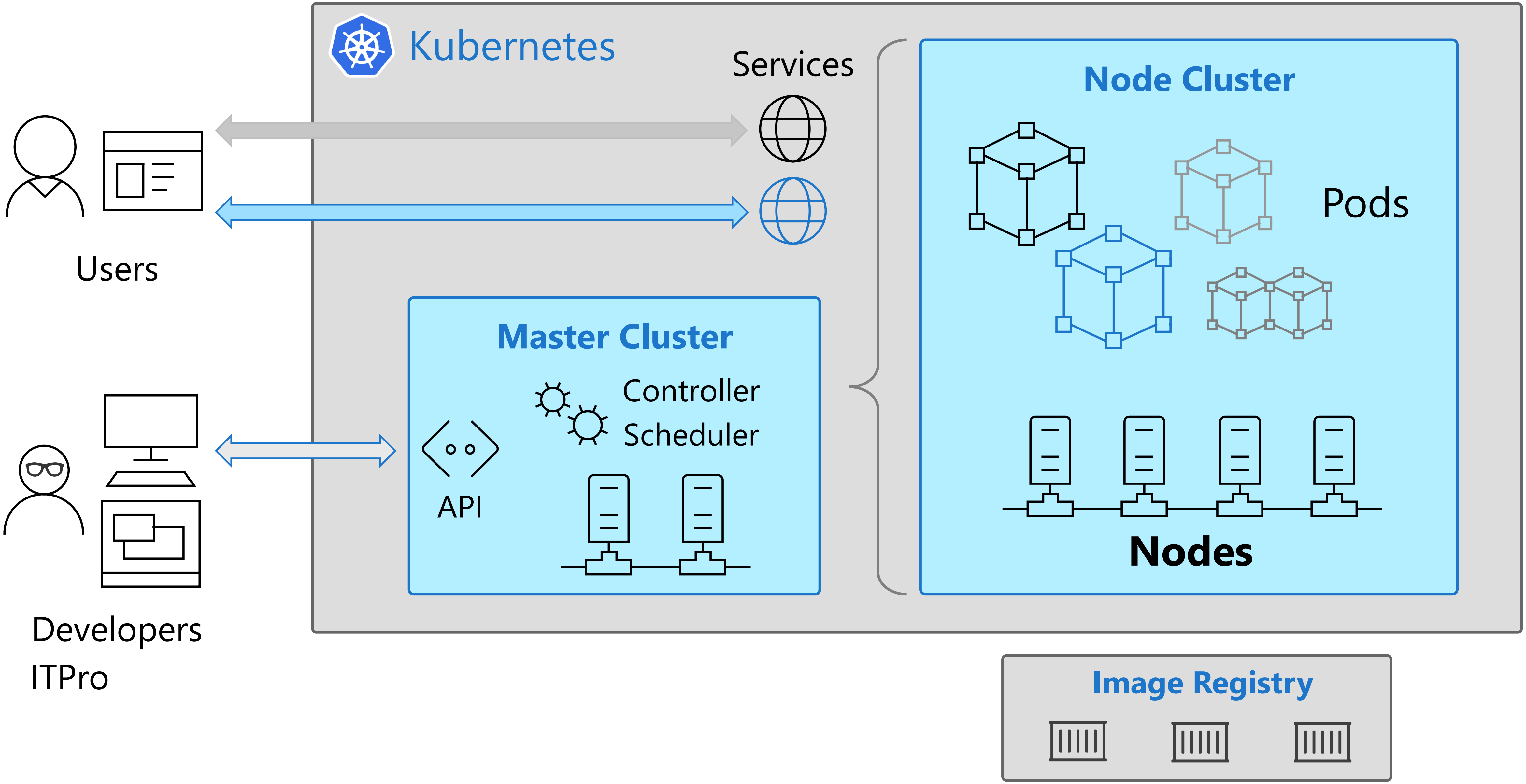
Connecting to an AKS Cluster

Azure CLI

```
az aks get-credentials -g gscaksdemo -n gscmyCluster
```

- Get the access credentials and configuration information to connect to the AKS cluster using **kubectl** and merges them into the **kubeconfig** file.
- RBAC is used to limit who can access the Kubernetes configuration information.
- Multiple clusters can be defined in this kubeconfig file. You switch between clusters using the **kubectl config use-context** command.

Kubernetes Simplified Architecture



Deployments

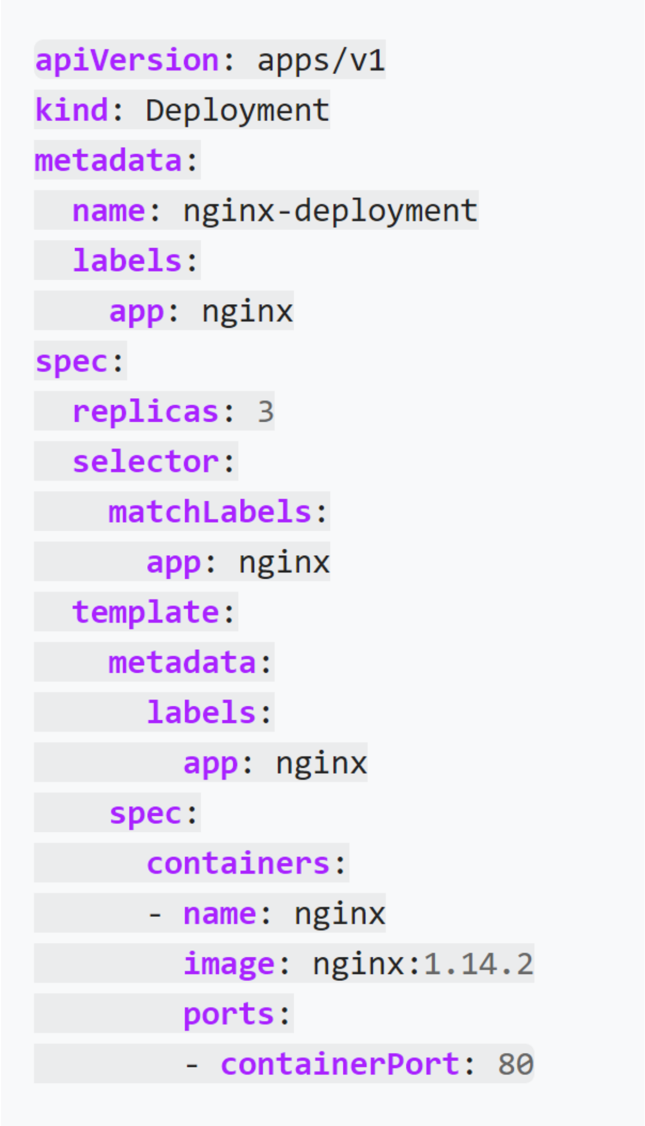
provide declarative updates to Pods and ReplicaSets

A **Deployment** represents **one or more identical pods or replicas** to create, managed by the Kubernetes Deployment Controller.

You describe a ***desired state*** in a **Deployment** using YAML Manifests, such as what container images to run.

The Deployment Controller changes the actual state to the desired state using the manifests.

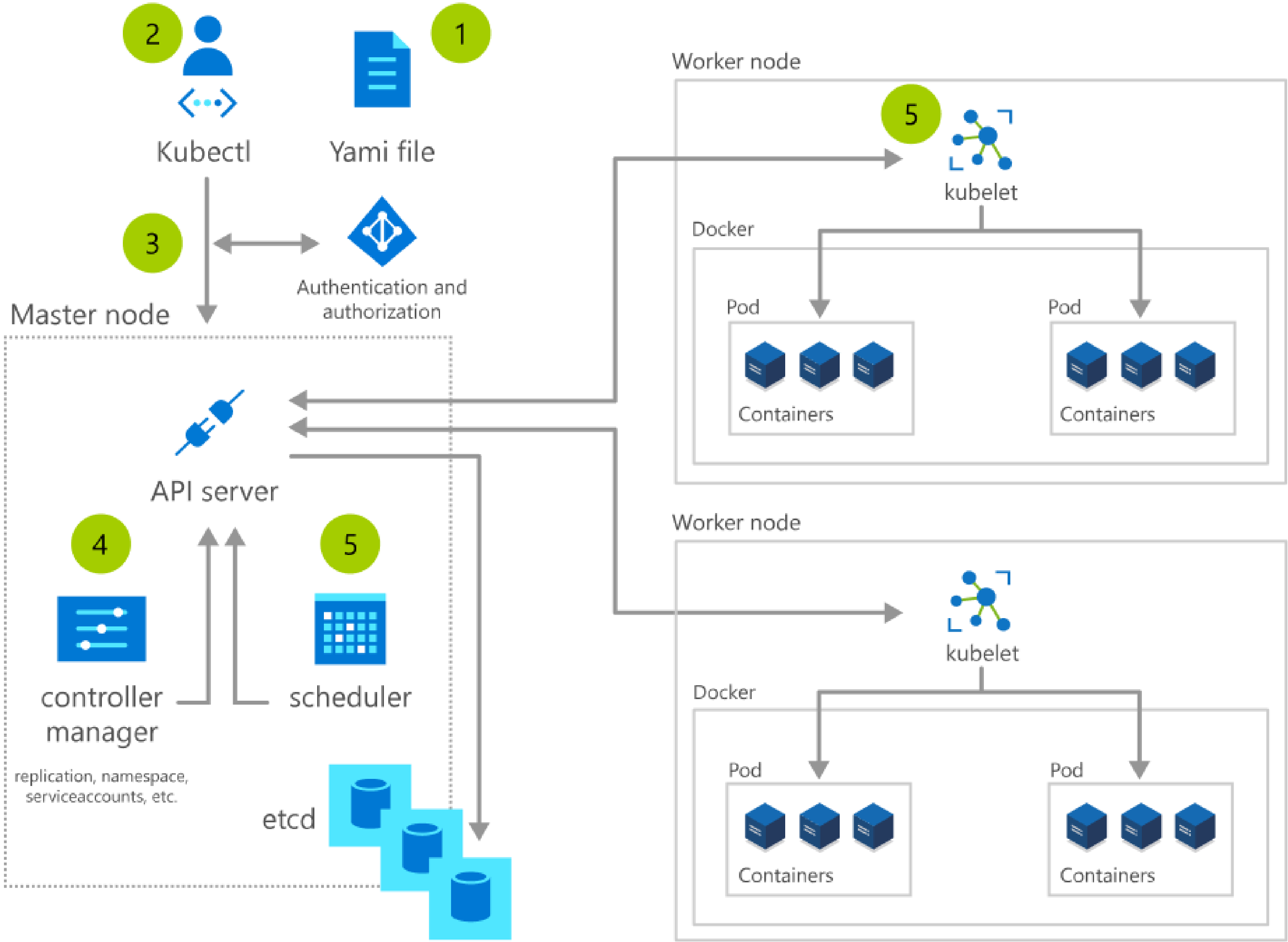
Kubernetes Scheduler ensures that if pods or nodes encounter problems, additional pods are scheduled on healthy nodes.



```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  labels:
    app: nginx
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.14.2
        ports:
        - containerPort: 80
```

An example Deployment YAML manifest to create a ReplicaSet to bring up three nginx Pods

Kubernetes Deployments



5. Create Deployment Manifests

Azure Voting App Manifest

A manifest is used to create all objects needed to run the Azure Vote application.

This manifest includes two Kubernetes deployments - one for the sample Azure Vote Python application, and the other for a Redis instance.

Two Kubernetes Services are also created - an internal service for the Redis instance, and an external service to access the Azure Vote application from the internet.

5. Create Deployment Manifests

Azure Voting App Manifest

Containerization / K8s Process

Specifically the manifest defines:

A deployment for the Redis backend of the voting application

A deployment for the voting application front end

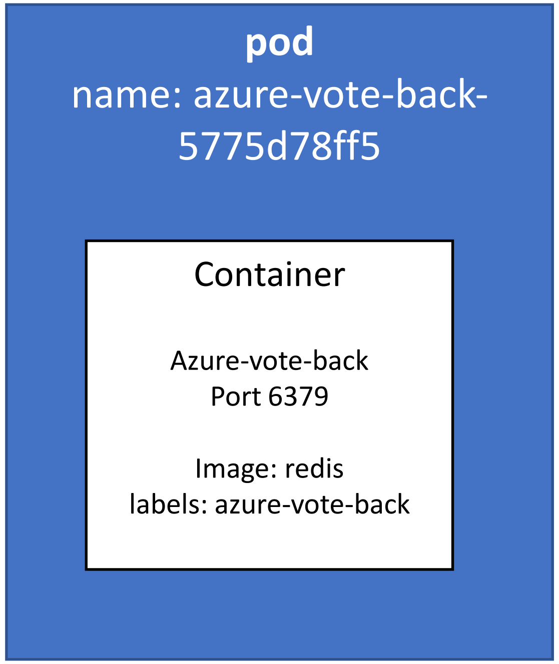
A Service exposing the Redis backend using an internal cluster IP Address and port 6379

A Service exposing the voting application front end on an external load balancer using port 80

Deployment Manifest – Azure Voting App

Defines a Redis backend of the voting application

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: azure-vote-back
spec:
  replicas: 1
  template:
    metadata:
      labels:
        app: azure-vote-back
    spec:
      containers:
      - name: azure-vote-back
        image: redis
        ports:
        - containerPort: 6379
          name: redis
```

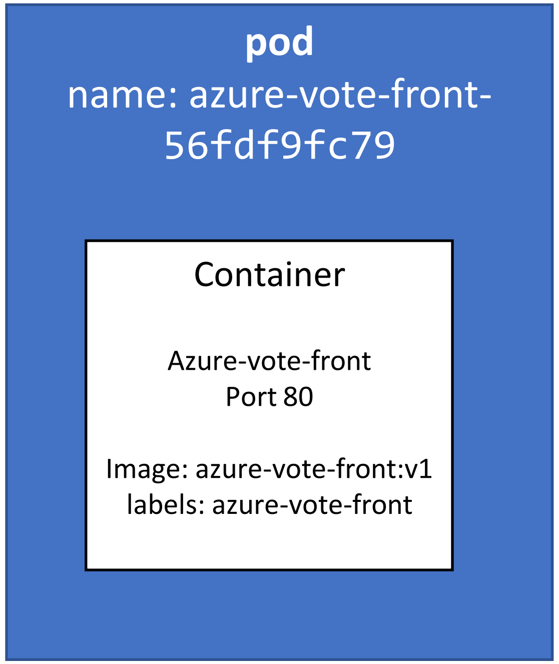


Azure-vote-all-in-one-redis.yaml

Deployment Manifest – Azure Voting App

Defines a deployment for the voting application front end

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: azure-vote-front
spec:
  replicas: 1
  template:
    metadata:
      labels:
        app: azure-vote-front
    spec:
      containers:
      - name: azure-vote-front
        image: gscacrdemo.azurecr.io/azure-vote-front:v1
        ports:
        - containerPort: 80
```



pod

name: azure-vote-front-56fdf9fc79

Container

Azure-vote-front
Port 80

Image: azure-vote-front:v1
labels: azure-vote-front

Azure-vote-all-in-one-redis.yaml

What is a Service?

- Services allow your applications (Pods) to receive traffic.

- Although each Pod has a unique IP address, those IPs are not exposed outside the cluster without a Service.

- A REST object, similar to a Pod. Like all of the REST objects, you can POST a Service definition to the API server to create a new instance.

- Services can be exposed in different ways by specifying a type in the ServiceSpec attribute:
  - ClusterIP
  - NodePort
  - LoadBalancer
  - ExternalName

Service Manifest – Azure Voting App

Defines a Redis backend service using an internal cluster IP Address and port 6379

```yaml
apiVersion: v1
kind: Service
metadata:
  name: azure-vote-back
spec:
  clusterIP: 10.0.222.76
  ports:
  - port: 6379
    protocol: TCP
    targetPort: 6379
  selector:
    app: azure-vote-back
  type: ClusterIP
```

Note! Services using **ClusterIP** are only reachable from within the cluster.

Azure-vote-all-in-one-redis.yaml

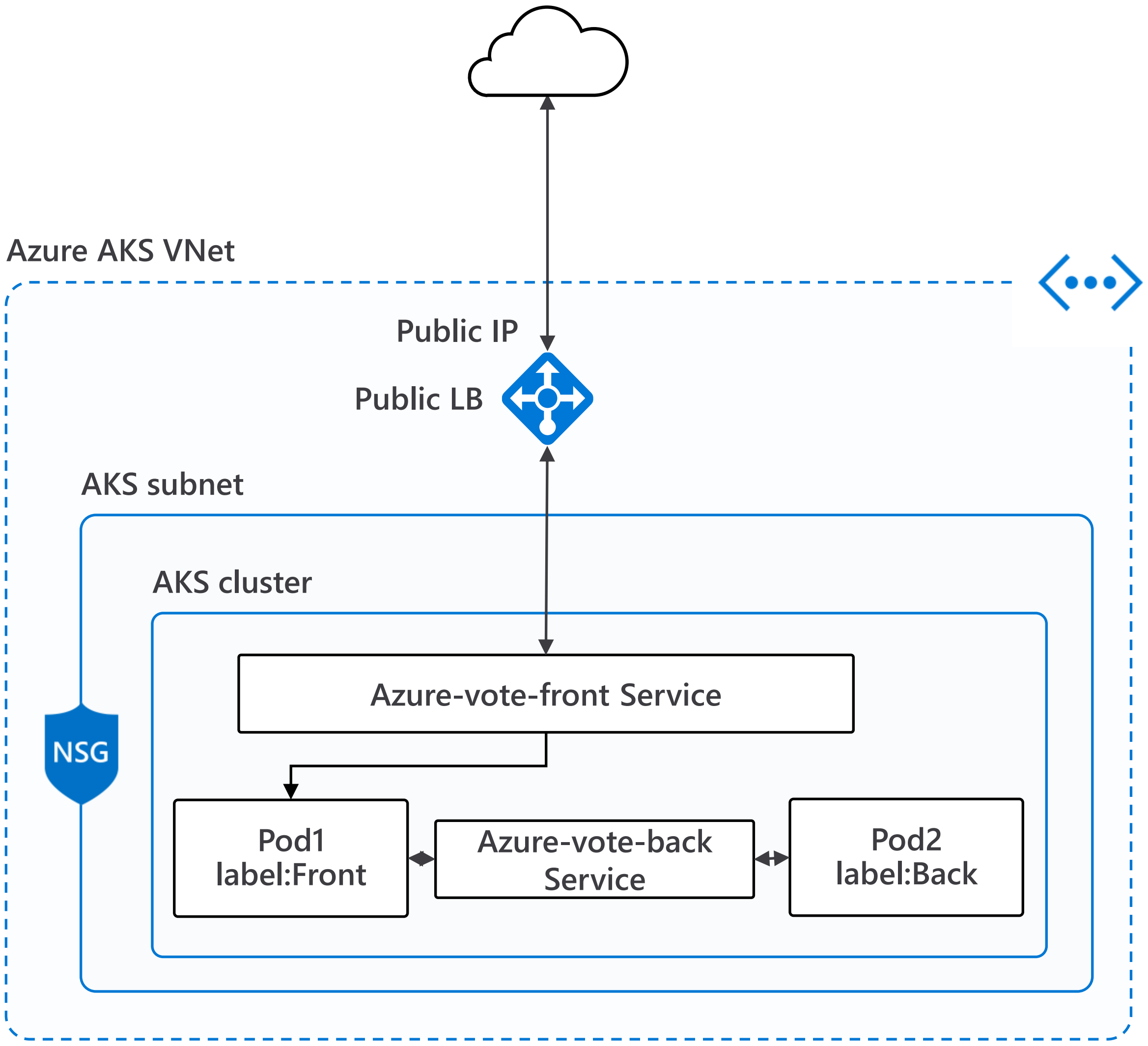
Service Manifest – Azure Voting App

Defines a voting application front end service on an external load balancer using port 80

```yaml
- apiVersion: v1
  kind: Service
  metadata:
    name: azure-vote-front
  spec:
    clusterIP: 10.0.230.169
    ports:
    - nodePort: 30076
      port: 80
      protocol: TCP
      targetPort: 80
    selector:
      app: azure-vote-front
    type: LoadBalancer
  status:
    loadBalancer:
      ingress:
      - ip: 52.188.76.107
```

Azure-vote-all-in-one-redis.yaml

Oversimplified Deployment architecture

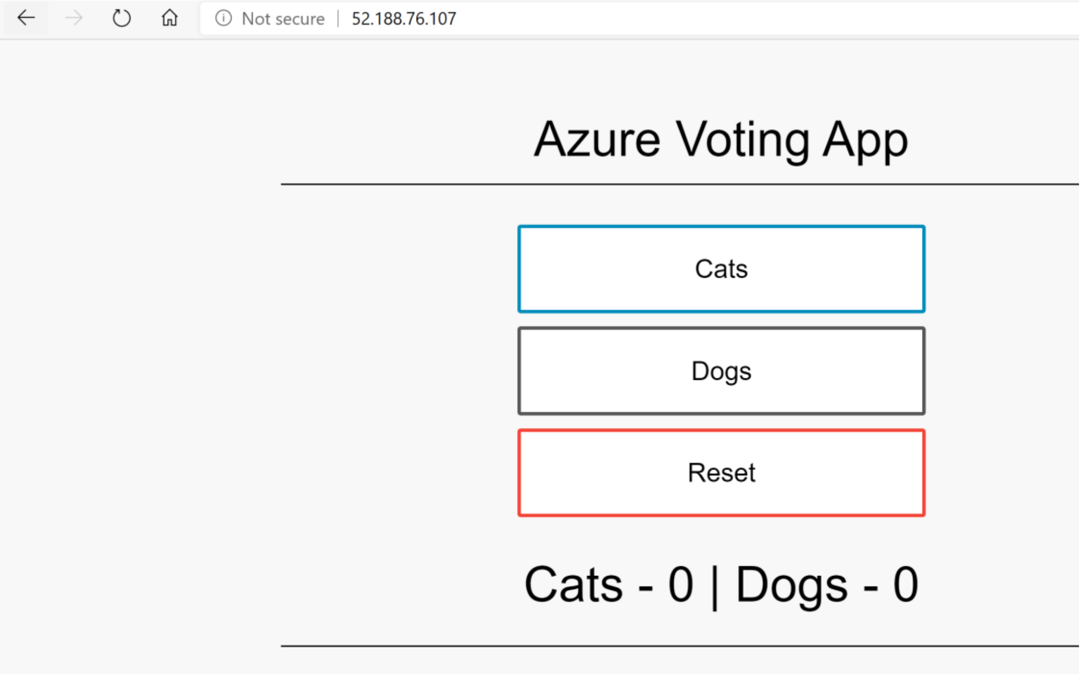


Containerization / K8s Process

6. Deploy multi-container application

```
kubectl apply -f ./azure-vote-all-in-one-redis.yaml
```

```
kubectl get service azure-vote-front
NAME              TYPE          CLUSTER-IP     EXTERNAL-IP    PORT(S)       AGE
azure-vote-front  LoadBalancer  10.0.230.169   52.188.76.107  80:30076/TCP  35s
```



Cats - 0 | Dogs - 0

Services

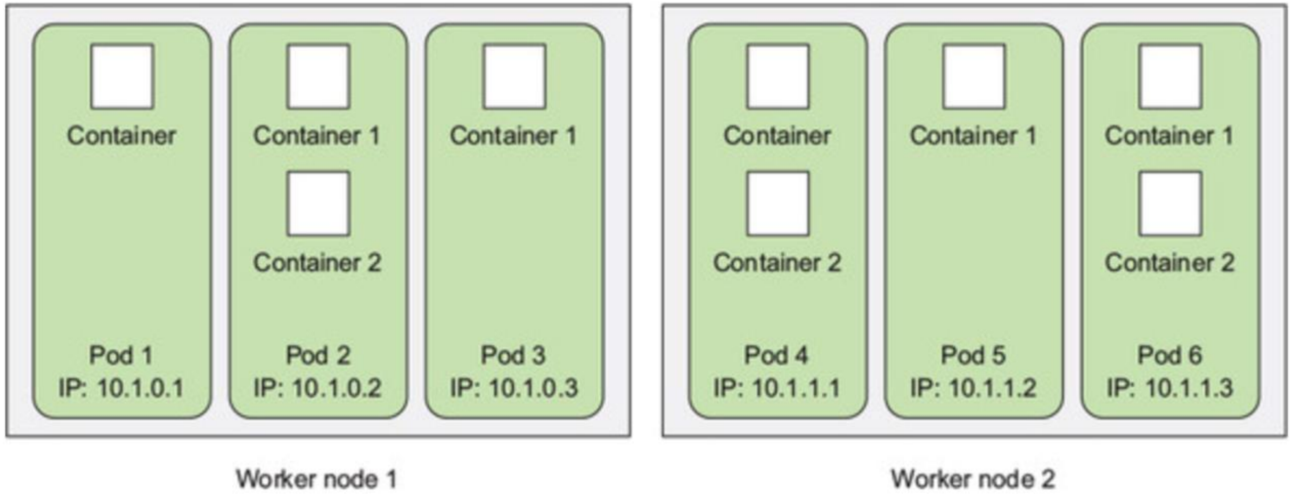
```
kubectl get svc
```

```
NAME              TYPE          CLUSTER-IP     EXTERNAL-IP    PORT(S)        AGE
azure-vote-back   ClusterIP     10.0.222.76    <none>         6379/TCP       25h
azure-vote-front  LoadBalancer  10.0.230.169   52.188.76.107  80:30076/TCP   25h
kubernetes        ClusterIP     10.0.0.1       <none>         443/TCP        26h
```

- Azure Voting App has three services running; two services pertaining to the application and one default Kubernetes service.
- The azure-vote-back service has been deployed with an internal cluster IP address of 10.0.222.76.
- The azure-vote-front service has been deployed with a load balancer that exposes the service publicly on the public IP address 52.188.76.107 and port 80.
- The Kubernetes service is created as part of the AKS cluster creation and is not part of the deployed application.

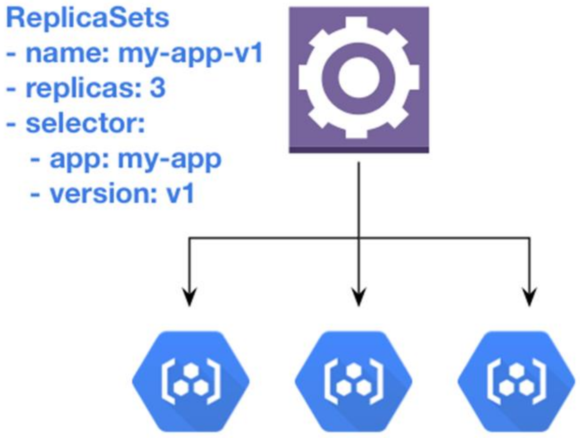
What is a pod?

- Pod is the basic building block in Kubernetes
- Pods are how containers are delivered
- Can be multiple containers (e.g. side car)
- Encapsulates container(s), storage, network IP, and options on how to run



What is a ReplicaSet?

- ReplicaSets ensure that a specific number of pods (or *replicas*) are running at any given time.
- If you want your pod to stay alive you make sure you have a Replica Set specifying at least one replica for that pod.
- Replica set then takes care of (re)scheduling your instances to ensure the desired number of pods is always up and available.



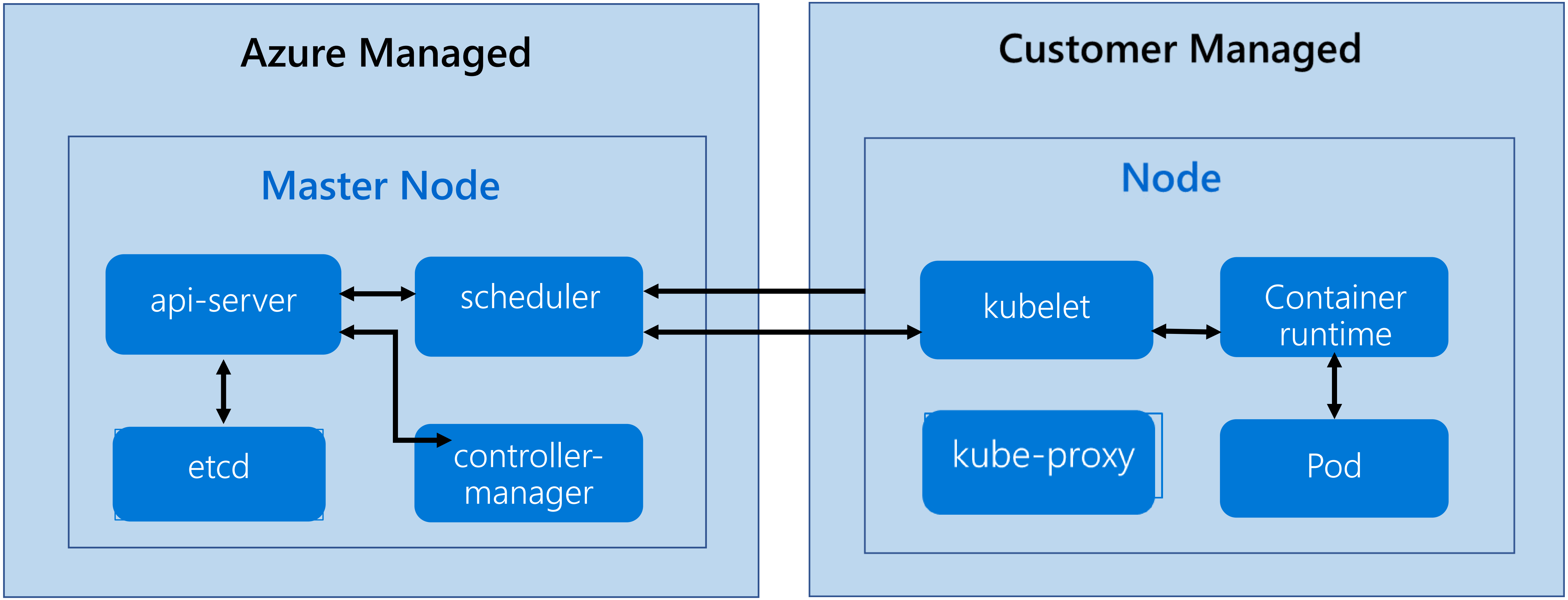
Where are my Pods and Containers?

```
kubectl get nodes
NAME                                STATUS   ROLES   AGE   VERSION
aks-nodepool1-26286499-vmss000000   Ready    agent   27h   v1.15.11
aks-nodepool1-26286499-vmss000001   Ready    agent   27h   v1.15.11
aks-nodepool1-26286499-vmss000002   Ready    agent   27h   v1.15.11
```

```
kubectl get pods
NAME                                READY   STATUS    AGE
azure-vote-back-5775d78ff5-hp68b    1/1     Running   26h
azure-vote-front-56fdf9fc79-mrxfb   1/1     Running   26h
```

```
kubectl get pods -o wide
NAME                                READY   IP           NODE
azure-vote-back-5775d78ff5-hp68b    1/1     10.244.2.2   aks-nodepool1-26286499-vmss000000
azure-vote-front-56fdf9fc79-mrxfb   1/1     10.244.0.7   aks-nodepool1-26286499-vmss000002
```

Kubernetes Cluster Architecture



Summary

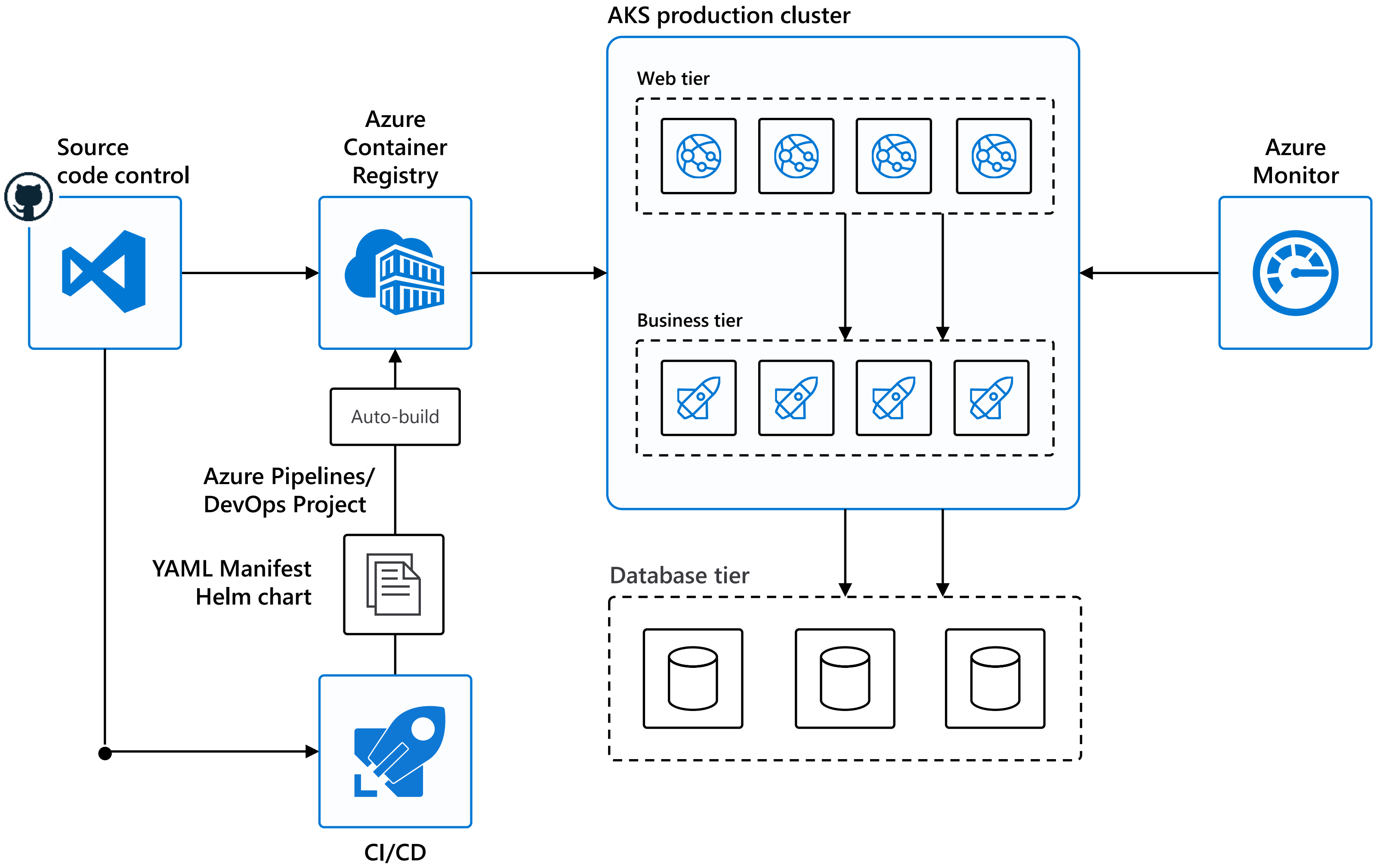
- Why Containers and why Kubernetes
- Multi-Container App containerization process
- The use of Azure CLI and Kubectl for AKS cluster provisioning
- Azure Container Registry and container image deployment
- Overview of Kubernetes Architecture and key components
- Nodes, Pods, Deployments, ReplicaSets, Services, Manifests

Lab 1

Lab 1 Scope

- Provision the Azure DevOps Team Project with a .NET Core application.
- Create an AKS cluster that will host your ASP.NET Core web application.
- Create an Azure Container Registry (ACR) instance which will host your container images in a private repository.
- Configure application and database deployment, using Continuous Deployment (CD) in Azure DevOps.
- Initiate the build to automatically deploy the application.

Integrated end-to-end Kubernetes experience
Example 3 Tier architecture



Lab 1 Prerequisites

- Microsoft Azure Account: You will need a valid and active Azure account for the Azure labs. If you do not have one, you can sign up for a free trial.
- You will need an Azure DevOps account. If you do not have one, you can sign up for free here.
- Ensure your account has the following roles:
    - The Owner built-in role for the subscription you will use.
    - A Member user in the Azure AD tenant you will use. (Guest users will not have the necessary permissions).

Microsoft Azure

Thank you. See you next week!
Stay safe!