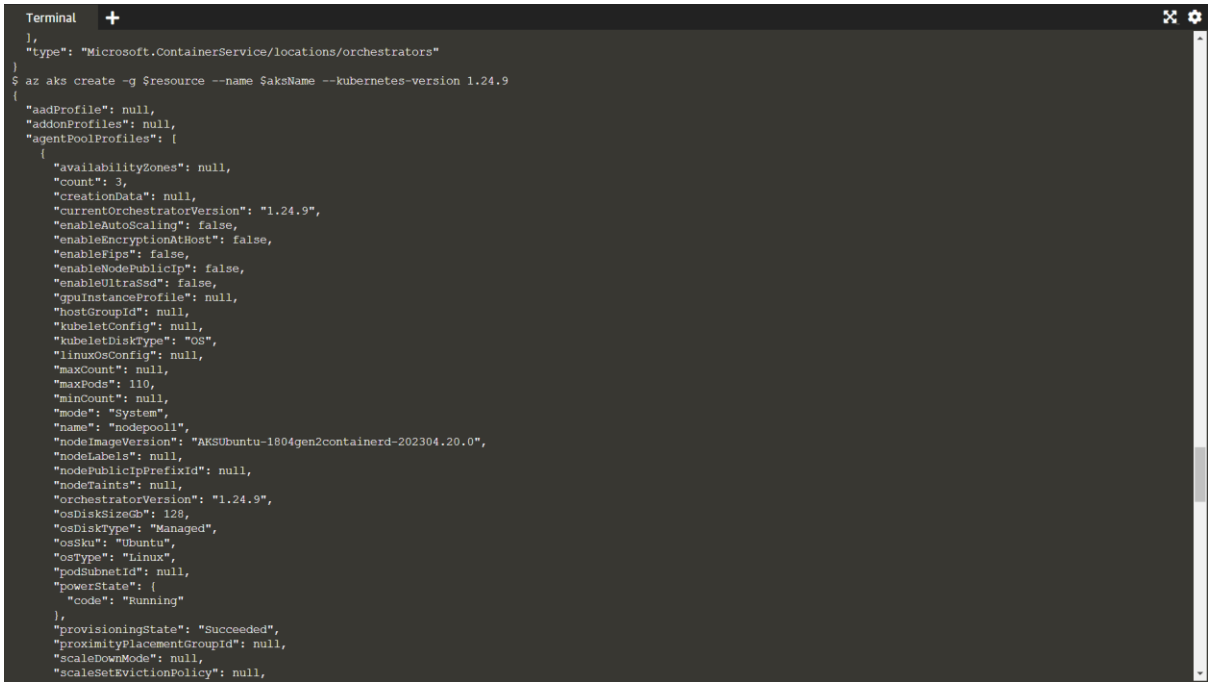
In this lab, you will learn how to deploy a simple application to your AKS cluster.

Learning Objectives
In this lab, you will learn how to do the following:

Log in to the Azure CLI
Understand application deployment to AKS
Deploy a simple application to AKS
Confirm that the application is running in AKS
Clean up the AKS cluster

Use the following credentials to log in to the Azure CLI:
az login -u $username -p $password

Create aks cluster :
az aks get-versions –location eastus
az aks create -g $resource –name $aksname –kubernetes-version 1.24.9



```
Terminal
  ],
  "type": "Microsoft.ContainerService/locations/orchestrators"
}
$ az aks create -g $resource --name $aksName --kubernetes-version 1.24.9
{
  "aadProfile": null,
  "addonProfiles": null,
  "agentPoolProfiles": [
    {
      "availabilityZones": null,
      "count": 3,
      "creationData": null,
      "currentOrchestratorVersion": "1.24.9",
      "enableAutoScaling": false,
      "enableEncryptionAtHost": false,
      "enableFips": false,
      "enableNodePublicIp": false,
      "enableUltraSsd": false,
      "gpuInstanceProfile": null,
      "hostGroupId": null,
      "kubeletConfig": null,
      "kubeletDiskType": "OS",
      "linuxOsConfig": null,
      "maxCount": null,
      "maxPods": 110,
      "minCount": null,
      "mode": "System",
      "name": "nodepool1",
      "nodeImageVersion": "AKSUbuntu-1804gen2containerd-202304.20.0",
      "nodeLabels": null,
      "nodePublicIpPrefixId": null,
      "nodeTaints": null,
      "orchestratorVersion": "1.24.9",
      "osDiskSizeGb": 128,
      "osDiskType": "Managed",
      "osSku": "Ubuntu",
      "osType": "Linux",
      "podSubnetId": null,
      "powerState": {
        "code": "Running"
      },
      "provisioningState": "Succeeded",
      "proximityPlacementGroupId": null,
      "scaleDownMode": null,
      "scaleSetEvictionPolicy": null,
```


A new AKS cluster is automatically created for you. Use the following command to list all the AKS clusters in your temporary Azure subscription:
az aks list --query "[].{Name: name}"

```
$ az aks list --query "[].{Name: name}"
[
  {
    "Name": "aks421990664"
  }
]
$ az aks list
[
  {
    "aadProfile": null,
    "addonProfiles": null,
    "agentPoolProfiles": [
      {
        "availabilityZones": null,
        "count": 3,
        "creationData": null,
        "currentOrchestratorVersion": "1.24.9",
        "enableAutoScaling": false,
        "enableEncryptionAtHost": false,
        "enableFips": false,
        "enableNodePublicIp": false,
        "enableUltraSsd": false,
        "gpuInstanceProfile": null,
        "hostGroupId": null,
        "kubeletConfig": null,
        "kubeletDiskType": "OS",
        "linuxOsConfig": null,
        "maxCount": null,
        "maxPods": 110,
        "minCount": null,
        "mode": "System",
        "name": "nodepool1",
        "nodeImageVersion": "AKSUbuntu-1804gen2co
        "nodeLabels": null,
        "nodePublicIpPrefixId": null,
        "nodeTaints": null,
        "orchestratorVersion": "1.24.9",
        "osDiskSizeGb": 128,
        "osDiskType": "Managed",
        "osSku": "Ubuntu",
```

Understanding Application Deployment to AKS
Microsoft Azure has a few services which allow you to host Docker-based containers. These services include:

Azure Container Instances (ACI)
Azure Kubernetes Services (AKS)
Azure Functions
Azure App Services

All these services have something in common. They need to first pull the container images from a Container Registry such as Azure Container Registry or Docker Hub.

In this lab, we will deploy a simple web application to AKS. This application uses Redis cache to keep track of simple votes. Redis is also deployed as a container as an application dependency.

The application container images should be hosted in a container registry service. For simplicity, we will deploy container images already hosted by Microsoft Azure so we can focus on the AKS deployment steps. See this document for more details.

Note: There might be more than one container, so multiple images need to be hosted in the container registry.

Now let's deploy our first application to AKS in the next step.

Deploying a Simple Application to AKS
Assuming that we have the application container images hosted in a container registry, we need to take the following steps to deploy the application to an AKS cluster:

Create a YAML manifest file. This file specifies the desired configuration for our AKS cluster. This includes the applications to install and the list of required container images.
Run the kubectl apply command to deploy the YAML file.

Note: You can also use other technologies such as Helm and Dapr to develop and deploy AKS applications.

We already created the required YAML manifest for you. To see the content, run the following command:

```
$ cat azurevoteapp.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: azure-vote-back
spec:
  replicas: 1
  selector:
    matchLabels:
      app: azure-vote-back
  template:
    metadata:
      labels:
        app: azure-vote-back
    spec:
      nodeSelector:
        "kubernetes.io/os": linux
      containers:
      - name: azure-vote-back
        image: mcr.microsoft.com/oss/bitnami/redis:6.0.8
        env:
        - name: ALLOW_EMPTY_PASSWORD
          value: "yes"
        resources:
```

```
          requests:
            cpu: 100m
            memory: 128Mi
          limits:
            cpu: 250m
            memory: 256Mi
        ports:
        - containerPort: 6379
          name: redis
---
apiVersion: v1
kind: Service
metadata:
  name: azure-vote-back
spec:
  ports:
  - port: 6379
  selector:
    app: azure-vote-back
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: azure-vote-front
spec:
  replicas: 1
  selector:
    matchLabels:
      app: azure-vote-front
  template:
    metadata:
      labels:
        app: azure-vote-front
    spec:
      nodeSelector:
        "kubernetes.io/os": linux
      containers:
      - name: azure-vote-front
        image: mcr.microsoft.com/azuredocs/azure-vote-front:v1
        resources:
          requests:
            cpu: 100m
            memory: 128Mi
          limits:
            cpu: 250m
            memory: 256Mi
        ports:
        - containerPort: 80
        env:
        - name: REDIS
          value: "azure-vote-back"
---
apiVersion: v1
kind: Service
metadata:
  name: azure-vote-front
spec:
```

```
  type: LoadBalancer
  ports:
  - port: 80
  selector:
    app: azure-vote-front
```

This manifest file contains the list of images and their container registry URL so that AKS can pull the images and run them.

First, let's get the AKS credentials using this command:

az aks get-credentials --resource-group $resource --name $aksName

Confirm that you see this command output:

```
$ az aks get-credentials --resource-group $resource --name $aksName
Merged "aks421990664" as current context in /root/.kube/config
```

Now use the following command to deploy the azurevoteapp.yaml file to your AKS cluster:

kubectl apply -f azurevoteapp.yaml

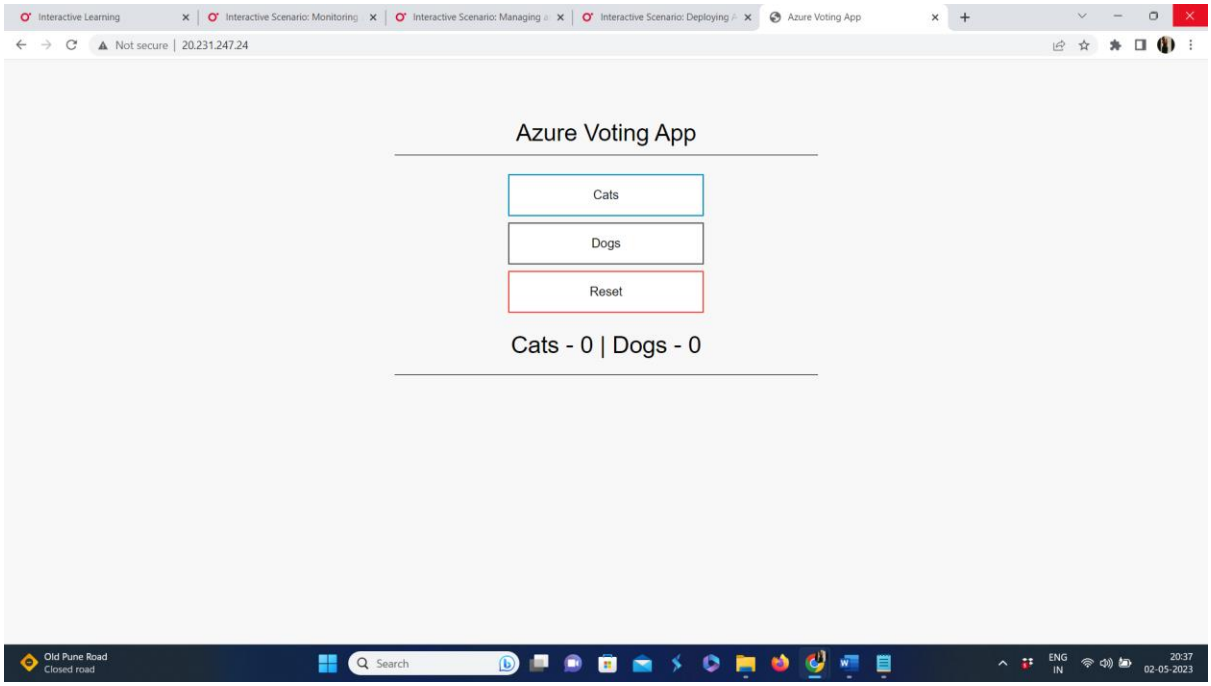
```
$ kubectl apply -f azurevoteapp.yaml
deployment.apps/azure-vote-back created
service/azure-vote-back created
deployment.apps/azure-vote-front created
service/azure-vote-front created
```

Now we can confirm the successful application deployment in the next step.

Confirming the Application Is Successfully Deployed
Our test application is a web app, so we need to get its IP address in order to test it. Use the following command to get the address:

kubectl get service azure-vote-front –watch

```
$ kubectl get service azure-vote-front --watch
NAME               TYPE           CLUSTER-IP    EXTERNAL-IP     PORT(S)        AGE
azure-vote-front   LoadBalancer   10.0.46.4     20.231.247.24   80:30686/TCP   2m41s
^C$
```



Deleting the AKS Cluster
The following command will clean up the new AKS cluster:

```
az aks delete --name $aksName --resource-group $resource
```