Creación de un clúster de Azure Kubernetes Service

En este ejercicio, va a crear un clúster de AKS en el que se usan varios nodos para satisfacer la demanda de muchos clientes que usan el servicio. Decide usar el plano de *control único y la arquitectura de varios nodos* , ya que proporciona la mejor manera de crear y administrar recursos de carga de trabajo.

AKS admite grupos de nodos de Linux y Windows a través del Portal o la CLI de Azure. Sin embargo, si va a usar grupos de nodos de Windows, el clúster debe crearse con requisitos previos adicionales y comandos. Seleccione un sistema operativo en función del tipo de grupos de nodos que quiera agregar.

Importante

Necesita su propia suscripción de Azure para completar este ejercicio y podría incurrir en cargos. Si aún no tiene una suscripción de Azure, cree una **cuenta gratuita** antes de empezar.

1. Inicie sesión en Azure Cloud Shell con la cuenta en la que quiera implementar los recursos.
2. En la ventana de Cloud Shell, seleccione **Configuración**>**Ir a la versión clásica**.
3. Cree variables para los valores de configuración que reutilizará en los ejercicios.

```
export RESOURCE_GROUP=rg-contoso-video

export CLUSTER_NAME=aks-contoso-video

export LOCATION=eastus
```

Actualice la variable LOCATION con la región más cercana. En este ejemplo se usa eastus.

4. Ejecute el comando az group create para crear un grupo de recursos. Implemente todos los recursos en este nuevo grupo de recursos.

```
az group create --name=$RESOURCE_GROUP --location=$LOCATION
```

5. Ejecute el comando az aks create para crear un clúster de AKS.

```
az aks create \

  --resource-group $RESOURCE_GROUP \

  --name $CLUSTER_NAME \

  --node-count 2 \

  --generate-ssh-keys \

  --node-vm-size Standard_B2s \

  --network-plugin azure
```

El comando crea un clúster de AKS denominado aks-contoso-video dentro del grupo de recursos rg-contoso-video. El clúster tiene dos nodos definidos por el parámetro --node-count. En este ejercicio solo se usan dos nodos para reducir costos. El parámetro --node-

vm-size configura los nodos de clúster como máquinas virtuales de tamaño Standard_B2s. Estos nodos forman parte del **modo sistema**.

6. Ejecute el comando az aks nodepool add para agregar otro grupo de nodos que use el sistema operativo Linux predeterminado.

```
az aks nodepool add \
  --resource-group $RESOURCE_GROUP \
  --cluster-name $CLUSTER_NAME \
  --name userpool \
  --node-count 2 \
  --node-vm-size Standard_B2s
```

El comando agrega un nuevo grupo de nodos (**modo de usuario**) al clúster de AKS existente (creado en el comando anterior). **Este grupo** de nodos de usuario se usa para hospedar aplicaciones y cargas de trabajo, a diferencia del grupo **de nodos del sistema**.

Vinculación con kubectl

1. Vincule el clúster de Kubernetes con kubectl mediante la ejecución del comando siguiente en Cloud Shell.

```
az aks get-credentials --name $CLUSTER_NAME --resource-group $RESOURCE_GROUP
```

Este comando agrega una entrada al archivo ~/.kube/config, que contiene toda la información para acceder a los clústeres. Kubectl permite administrar varios clústeres desde una única interfaz de la línea de comandos.

2. Ejecute el comando kubectl get nodes para comprobar que se puede conectar al clúster y confirme su configuración.

```
kubectl get nodes
```

En la salida se deben enumerar cuatro nodos disponibles para dos grupos de nodos.

```
NAME                                STATUS   ROLES   AGE    VERSION
aks-nodepool1-21895026-vmss000000   Ready    agent   245s   v1.23.12
aks-nodepool1-21895026-vmss000001   Ready    agent   245s   v1.23.12
aks-userpool-21895026-vmss000000    Ready    agent   105s   v1.23.12
aks-userpool-21895026-vmss000001    Ready    agent   105s   v1.23.12
```

Implementación de una aplicación en un clúster de Azure Kubernetes Service

En este ejercicio, implementará el sitio web de la empresa como una aplicación de prueba en Azure Kubernetes Service (AKS). Es un sitio web estático con una pila tecnológica subyacente de HTML, CSS y JavaScript. No recibe tantas solicitudes como

otros servicios y proporciona una manera segura de probar las opciones de implementación.

Nota

El código de la aplicación web está disponible en este **repositorio de GitHub** si quiere explorarlo con profundidad. Además, esta aplicación de muestra solo se implementará en un grupo de nodos de Linux.

Creación de un manifiesto de implementación

Cree un archivo de manifiesto de implementación para implementar la aplicación. El archivo de manifiesto permite definir el tipo de recurso que quiere implementar y todos los detalles asociados a la carga de trabajo.

Kubernetes agrupa los contenedores en estructuras lógicas denominadas pods, que no tienen inteligencia. Las implementaciones agregan la inteligencia que falta para crear la aplicación. Vamos a crear un archivo de implementación.

1. Inicie sesión en Azure Cloud Shell.
2. En Cloud Shell, cree un archivo de manifiesto para la implementación de Kubernetes denominado deployment.yaml mediante el editor integrado.

touch deployment.yaml

3. Abra el editor integrado en Cloud Shell escribiendo code ..
4. Abra el archivo deployment.yaml y agregue la siguiente sección de código de YAML.

```
# deployment.yaml

apiVersion: apps/v1 # The API resource where this workload resides

kind: Deployment # The kind of workload we're creating

metadata:

  name: contoso-website # This will be the name of the deployment
```

En este código, ha agregado las dos primeras claves para indicar a Kubernetes los elementos apiVersion y kind del manifiesto que va a crear. name es el nombre de la implementación. Lo utilizará para identificar y consultar la información de la implementación cuando use kubectl.

5. Una implementación encapsula un pod. Use una definición de plantilla para definir la información del pod dentro del archivo de manifiesto. La plantilla se coloca en el archivo de manifiesto, bajo la sección de especificación de la implementación.

Actualice el archivo deployment.yaml para que coincida con el código YAML siguiente.

```
# deployment.yaml

apiVersion: apps/v1
```

```
kind: Deployment
metadata:
  name: contoso-website
spec:
  template: # This is the template of the pod inside the deployment
    metadata: # Metadata for the pod
      labels:
        app: contoso-website
```

Los pods no usan los mismos nombres que las implementaciones. El nombre del pod será una combinación del nombre de la implementación con un identificador aleatorio agregado al final.

Fíjese en el uso de la clave labels. La clave labels se agrega para permitir que las implementaciones busquen y agrupen los pods.

6. Un pod encapsula uno o varios contenedores. Todos los pods tienen una sección de especificación que le permite definir los contenedores dentro de ese pod.

Actualice el archivo deployment.yaml para que coincida con el código YAML siguiente.

```
# deployment.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: contoso-website
spec:
  template: # This is the template of the pod inside the deployment
    metadata:
      labels:
        app: contoso-website
    spec:
      containers: # Here we define all containers
        - name: contoso-website
```

La clave containers es una matriz de especificaciones de contenedor porque un pod puede tener uno o más contenedores. La especificación define los elementos image, name, resources, ports y otra información importante sobre el contenedor.

Todos los pods en ejecución seguirán el nombre contoso-website-<UUID>, donde UUID es un identificador generado para identificar todos los recursos de forma única.

7. Se recomienda definir una cantidad mínima y máxima de los recursos que la aplicación puede usar desde el clúster. La clave resources se usa para especificar esta información.

Actualice el archivo deployment.yaml para que coincida con el código YAML siguiente.

```
# deployment.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
 name: contoso-website
spec:
 template: # This is the template of the pod inside the deployment
  metadata:
   labels:
    app: contoso-website
  spec:
   containers:
    - image: mcr.microsoft.com/mslearn/samples/contoso-website
      name: contoso-website
      resources:
       requests: # Minimum amount of resources requested
        cpu: 100m
        memory: 128Mi
       limits: # Maximum amount of resources requested
        cpu: 250m
        memory: 256Mi
```

Observe cómo la sección de recursos le permite especificar la cantidad mínima de recursos como una solicitud y la cantidad máxima de recursos como un límite.

8. El último paso consiste en definir los puertos que este contenedor expone externamente a través de la clave ports. La clave ports es una matriz de objetos, lo que significa que un contenedor de un pod puede exponer varios puertos con varios nombres.

Actualice el archivo deployment.yaml para que coincida con el código YAML siguiente.

```
# deployment.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: contoso-website
spec:
  template: # This is the template of the pod inside the deployment
    metadata:
      labels:
        app: contoso-website
    spec:
      nodeSelector:
        kubernetes.io/os: linux
      containers:
        - image: mcr.microsoft.com/mslearn/samples/contoso-website
          name: contoso-website
          resources:
            requests:
              cpu: 100m
              memory: 128Mi
            limits:
              cpu: 250m
              memory: 256Mi
          ports:
            - containerPort: 80 # This container exposes port 80
              name: http # We named that port "http" so we can refer to it later
```

Observe cómo se asigna un nombre al puerto con la clave name. La asignación de nombres a puertos permite cambiar el puerto expuesto sin modificar los archivos que le hacen referencia.

9. Por último, agregue una sección de selector para definir las cargas de trabajo que administra la implementación. La clave selector se coloca dentro de la sección de especificación de implementación del archivo de manifiesto. Use la

clave matchLabels para mostrar las etiquetas de todos los pods administrados por la implementación.

Actualice el archivo deployment.yaml para que coincida con el código YAML siguiente.

```
# deployment.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
 name: contoso-website
spec:
 selector: # Define the wrapping strategy
  matchLabels: # Match all pods with the defined labels
   app: contoso-website # Labels follow the `name: value` template
 template: # This is the template of the pod inside the deployment
  metadata:
   labels:
    app: contoso-website
  spec:
   nodeSelector:
    kubernetes.io/os: linux
   containers:
    - image: mcr.microsoft.com/mslearn/samples/contoso-website
      name: contoso-website
      resources:
       requests:
        cpu: 100m
        memory: 128Mi
       limits:
        cpu: 250m
        memory: 256Mi
      ports:
       - containerPort: 80
```

```
      name: http
```

Los nodos de Linux no pueden ejecutar contenedores de Windows, y viceversa.

10. Guarde el archivo de manifiesto y cierre el editor.

Aplicación del manifiesto

1. En Cloud Shell, ejecute el comando kubectl apply para enviar el manifiesto de implementación al clúster.

kubectl apply -f ./deployment.yaml

Este comando debe generar una salida similar a la del ejemplo siguiente.

deployment.apps/contoso-website created

2. Ejecute el comando kubectl get deploy para comprobar si la implementación se ha realizado correctamente.

kubectl get deploy contoso-website

El comando debe generar una tabla similar a la del ejemplo siguiente.

NAME READY UP-TO-DATE AVAILABLE AGE

contoso-website 0/1 1 0 16s

3. Ejecute el comando kubectl get pods para comprobar si se está ejecutando el pod.

kubectl get pods

El comando debe generar una tabla similar a la del ejemplo siguiente.

ResultadosCopiar

NAME READY STATUS RESTARTS AGE

contoso-website-7c58c5f699-r79mv 1/1 Running 0 63s