DevOps Engineer Code Challenge

Challenge 1

1. Para evitar que los Pods se desplieguen en determinados nodos, primero etiqueto los nodos en los que sí quiero que haya Pods con el comando:

kubectl label node <node name> app=frontend

Posteriormente creo el campo ‘nodeSelector’ en el values.yaml del chart, donde incorporo la etiqueta que acabo de crear en el nodo. De la misma forma, modifico el deployment.yaml para incorporar esta etiqueta al manifiesto:

{{- with .Values.nodeSelector }}

nodeSelector:

{{- toYaml . | nindent 8 }}

{{- end }}

Gracias a esto, los Pods se van a crear en los nodos donde esté esa etiqueta, dejando una serie de nodos sin etiquetar aislados.

1. Para conseguir que los Pods no se ejecuten en un nodo que ya tiene un Pod del mismo tipo, uso podAntiAffinity:

podAntiAffinity:

requiredDuringSchedulingIgnoredDuringExecution:

- labelSelector:

matchExpressions:

- key: app.kubernetes.io/name

operator: In

values:

- ping

topologyKey: "kubernetes.io/hostname"

Esto hace que en los nodos en los que se encuentre la etiqueta “kubernetes.io/hostname” y que ya tienen asignado un Pod con la etiqueta “app.kubernetes.io/name=ping” no se despliegue ningún otro Pod que contenta esta última etiqueta.

1. Por último, para que los Pods se desplieguen en distintas zonas de disponibilidad, he usado nodeAffinity:

nodeAffinity:

requiredDuringSchedulingIgnoredDuringExecution:

nodeSelectorTerms:

- matchExpressions:

- key: kubernetes.io/zone

operator: In

values:

- blue

- red

- green

Este bloque permite que los Pods se asignen a una de esas 3 zonas de disponibilidad (“blue”, “red” o “green”) que son etiquetas que se deben poner en los nodos antes de instalar el chart.

Challenge 2

La estructura que he usado es la siguiente:

+terraform-main-module/

|

|--main.tf

+--terraform-helm/

|

|--main.tf

|--variables.tf

El fichero variables.tf contiene las variables que el módulo reusable va a necesitar. Estas variables son valores del usuario, el nombre de los charts, información sensible, etc.

El archivo main.tf (el que se encuentra dentro de terraform-helm/) contiene la lógica del módulo y se divide en dos partes: el provisionador “local-exec” que se encargará de copiar los charts del registro de referencia al registro de instancia y el recurso “install-charts” que, como su nombre indica, va a instalar los charts de Helm que se encuentran en el registro de instancia en el clúster AKS.

Por último, el main.tf que se encuentra en el directorio raíz va a comenzar la ejecución del módulo cuando se lance el terraform apply.

Challenge 3

Este workflow de Github instala el chart “ping” en el clúster AKS usando el módulo de Terraform anterior.

En el fragmento “Azure Login” utilizo “azure/login” con credenciales almacenadas en GitHub Secrets bajo “AZURE\_CREDENTIALS”:

- name: Azure Login

uses: azure/login@v1

with: creds: ${{ secrets.AZURE\_CREDENTIALS }}

En “Get AKS Credentials” se obtienen las credenciales del clúster AKS (cuyo nombre es “cluster-test”) en el grupo de recursos “resourceGroup”:

- name: Get AKS Credentials

run: az aks get-credentials --resource-group resourceGroup --name cluster-test --overwrite-existing

Por último “Deploy Ping Chart with Helm” instala el chart local “ping” con el comando “helm install”:

- name: Deploy Ping Chart with Helm

run: |

helm install ping ../challenge1/ping