Tutorial azure para crear contenedores en kubernetes con docker

Juan Casado Ballesteros

Proporcionar una IP estática al cluster

Para poderlo hacer deberemos primero obtener el nombre del grupo de recursos.

Obtener la IP

Usaremos ese nombre para crear nuestra IP a la cual la deberemos nombrer también.

```
az aks show --resource-group ${resourceGroup} \
--name ${clusterName} --query nodeResourceGroup -o tsv
az network public-ip create \
--resource-group ${resourceGroupName} \
--name ${ipName} \
--allocation-method static
```

En el campo ipAdrees de la respuesta tendremos nuestra IP estática que podremos volver a mostrar con el siguiente comado.

```
az network public-ip show --resource-group ${resourceGroupName} \
--name ${ipName} --query ipAddress --output tsv
```

Actualizar el manifiesto

Actualizaremos nuestro .yml con la IP estática la cual aplicaremos sobre nuestro balancedador de carga. Le asignaremos una nueva propiedad en el apartado spec

```
apiVersion: v1
kind: Service
metadata:
  name: azure-vote-front
spec:
  loadBalancerIP: ${ipAddress}
```

Aplicar la IP

Lo haremos mediante el sigiente comando aplicado sobre nuestro archivo .yml

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

Comprobar los cambios

Podremos verificar los cambios realizados al ver nuestra IP como punto de acdeso al front end de nuestra aplicación.

```
kubectl describe service azure-vote-front
```

Azure dev Space

Iniciar resursos necesarios

Para realizar las pruebas necesitaremos haber creado un grupo de recursos y un cluster de kubernetes

```
az group create --name ${resourceGroup} --location eastus
az aks create -g ${resourceGroup} -n ${clusterName} \
--location eastus --node-vm-size Standard_DS2_v2 \
--node-count 1 --disable-rbac --generate-ssh-keys
```

Habilitar el Dev space

```
az aks use-dev-spaces -g ${resourceGroup} -n ${clusterName}
```

Crear Dockerfile

Podremos crearlo manualmente, en nuestro caso optaremos por esta opción ya que utilizaremos python como backend el cual no es soportado por azure para esta tarea concreta.

En caso de utilizar .NET o node quye si son soportados se podría hacer de forma automática

```
azds prep --public
```

Iniciar el servicio

Iniciaremos el servicio en base a nuestro Dockerfile

```
azds up
```

Desde VS code debemos abrir el comand palet con Shift+Command+P y seleccionar la opción *Azure Dev Spaces: Prepare configuration files for Azure Dev Spaces* para habilitarla.

Ahora podremos editar nuestro código y subirlo a partir del Dockerfile así como hacer debugg sobre él.

Habilitar la monitorización de los contenedores creados

Por defecto esta opción se encuentra desabilitada y la tendremos que habilitar manualmente. En caso de tener un workspace ya creado podremos añadir nuestros contenedores a él de modo que la monitorización se activará.

```
#PARA CREAR EL WORKSPACE
az aks enable-addons -a monitoring -n ${clusterName} \
-g ${resourceGroup}
#PARA AÑADIR A UN WORSPACE EXISTENTE
az aks enable-addons -a monitoring -n ${clusterName} \
-g ${resourceGroup} --workspace-resource-id ${workspaceId}
```

Una vez realizado esto podremos ver toda la información en directo sobre el estado de nuestros contenedores y de todo el sistema al completo de kubernetes. Podremos configurar para recibir notificaciones en caso de fallo o se saturación de los recursos.

Test de rendimiento del balanceador de carga

Intentaremos ver como responde el balanceador de carga a un aumento de peticiones que generaremos de forma artificial. Podremos monitorizar el estado de nuestro pod mediante los siguientes comandos.

```
#Distintas niveles de detalle para ver los recursos
kubectl get all
kubectl get pods
kubectl get nodes
kubectl get services
kubectl get deployment
kubectl get hpa #Nos informa del uso de recursos
```

```
#Recursos concretos de un pod
kubectl describe service azure-vote-back
kubectl describe service azure-vote-front
```

Podremos configurar nuestros pods para que respondan a la carga de forma automática mediante.

```
#PRIMERO DEBEMOS ELIMINAR LA CONFIGURACIÓN ANTERIOR
kubectl delete hpa azure-vote-front
kubectl autoscale deployment ${deployment name} \
--cpu-percent=${autoescaleTrigger} \
--min=${minimumPods} \
--max=${maximumPods}
```

Adicionamente desde azure podremos configurar la cantidad de recursos que tiene cada pod.

```
az aks scale --resource-group ${resourceGroup} \
--name ${clusterName} --node-count ${nodeCount}
```

Ejemplo de respuesta del load balancer

Inicialmente hemos configurado nuestro load balancer mediante:

```
az aks scale --resource-group ${resourceGroup} \
--name ${clusterName} --node-count 2
kubectl autoscale deployment ${deployment name} \
--cpu-percent=4 --min=2 --max=5
```

De modo que nuestros pod deberían ser relativamente sencillos de saturar y oscilando su número entre 2 y 5. Podemos ver mediante *kubectl get pods* que inicialmente hay dos de ellos en front end y uno en backend. El test se realizará solo sobre el front end y no sobre el backend que no se replicará.

Para saturar las carga haremos peticiones de forma constante desde varias terminales com el objetivo de saturar al servidor.

```
while true; do curl http://${ipAddress}; done
```

Pudimos ir viendo como al aumentar el número de peticiones aumentó también el de pods que se crearon llegando hasta tener activos cuatreo de los 5 que eran el máximo. Al desactivar la carga los pods se fueron deshabilitando progresivamente hasta volver a quedar los dos iniciales.

Este ejemplo nos ha permitido observer uno de los funcionamientos clave de kubernetes, no se indica qué hacer si no los resultados que queremos obtener de modo que estos son conseguidos de forma automática a partir de realizar las acciones que sean pertinentes.