Certified Tech Developer
The Ultimate Degree

Infraestructura III

Automatizamos el escalamiento de pods con HPA

- Actividad individual
- Nivel de complejidad: medio 🔥🔥

Consigna

Consigna: habilitar Horizontal Pod Autoscaler para administrar automáticamente el escalamiento para una aplicación web de ejemplo. Esta carga de trabajo de ejemplo es un nginx respondiendo solicitudes http.

Antes de empezar...

Dado que vamos a escalar los pods basados en las métricas de los mismos, es necesario habilitar en Minikube el addon metrics-server para que kubernetes pueda leer las métricas. Se puede instalar fácilmente con este comando:

Si están en minikube (EC2):

```
minikube addons enable metrics-server
```

Si están en docker desktop:

```
kubectl  apply -f https://raw.githubusercontent.com/damiancolaneri/server-metrics/main/components.yaml
```

¡Comenzamos!

Primero vamos a crear un archivo de nombre php.yaml con este contenido:

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: php-apache
spec:
  selector:
    matchLabels:
      run: php-apache
  template:
    metadata:
      labels:
        run: php-apache
    spec:
      containers:
        - name: php-apache
          image: registry.k8s.io/hpa-example
          ports:
            - containerPort: 80
          resources:
            limits:
              cpu: 500m
            requests:
              cpu: 200m
---
apiVersion: v1
```

```
kind: Service
metadata:
  name: php-apache
  labels:
    run: php-apache
spec:
  ports:
    - port: 80
  selector:
    run: php-apache
```

Este deployment usa una imagen de apache corriendo PHP generando procesos. Observemos en el campo réplicas que sólo levantará 1 pod.

guardamos este archivo, lo implementamos con el siguiente comando:

```
kubectl apply -f php.yaml
```

Debe devolver este mensaje:

```
deployment.apps/php created
service/php created
```

Crear el Horizontal Pod Autoscaler

Ahora que el servidor se está ejecutando, creamos el escalador automático usando kubectl. Hay un kubectl autoscale subcomando, parte de kubectl, que nos ayuda a hacer esto:

```
kubectl autoscale deployment php-apache --cpu-percent=50 --min=1
--max=10
```

Debe devolver este mensaje:

```
horizontalpodautoscaler.autoscaling/php autoscaled
```

Podemos verificar el estado actual del Horizontal Pod Autoscaler recién creado, ejecutando:

```
kubectl get hpa
```

La salida es similar a:

```
NAME         REFERENCE                TARGET    MINPODS   MAXPODS   REPLICAS   AGE
php   Deployment/php/scale   0% / 50%   1         10        1          18s
```

Aumentamos la carga

A continuación, veamos cómo reacciona el escalador automático al aumento de la carga. Para hacer esto, iniciará un Pod diferente para que actúe como cliente. El contenedor dentro del Pod del cliente se ejecuta en un bucle infinito y envía consultas al servicio php.

```
kubectl run -i --tty load-generator --rm --image=busybox:1.28 --restart=Never --
/bin/sh -c "while sleep 0.01; do wget -q -O- http://php-apache; done"
```

Ahora ejecutamos:

```
kubectl get hpa php-apache --watch
```

En aproximadamente un minuto, deberíamos ver una mayor carga de CPU; por ejemplo:

```
NAME         REFERENCE                TARGET      MINPODS   MAXPODS   REPLICAS   AGE
php-apache   Deployment/php/scale   80% / 50%   1         10        1          3m
```

Y luego, más réplicas. Por ejemplo:

```
NAME         REFERENCE                TARGET      MINPODS   MAXPODS   REPLICAS   AGE
```

```
php-apache   Deployment/php/scale        305% / 50%  1         10        7         3m
```

Aquí, el consumo de CPU aumentó al 305% de la solicitud. Como resultado, la implementación se redimensionó a 7 réplicas:

```
kubectl get deployment php-apache
```

Deberíamos ver el recuento de réplicas que coincide con la cifra del Horizontal Pod Autoscaler.

```
NAME         READY   UP-TO-DATE   AVAILABLE   AGE
php          7/7      7           7           19m
```

Dejamos de generar carga

Para finalizar el ejercicio, dejamos de enviar la carga.

En la terminal donde creamos el pod en el que estamos ejecutando la generación de carga, presionamos <Ctrl> + C para detener la tarea.

Luego verificamos el estado del resultado (después de unos minutos más o menos):

```
kubectl get hpa php-apache --watch
```

La salida es similar a:

```
NAME         REFERENCE                     TARGET       MINPODS   MAXPODS   REPLICAS   AGE
php   Deployment/php/scale                 0% / 50%     1         10        1          11m
```

Y la implementación también muestra que se ha reducido:

```
kubectl get deployment php-apache
```

```
NAME          READY   UP-TO-DATE   AVAILABLE   AGE
php           1/1     1            1           27m
```

Una vez que la utilización de la CPU se redujo a 0, HPA reducirá automáticamente la cantidad de réplicas a 1 pasados unos minutos. ¡Con esto vimos cómo autoescalar aplicaciones en kubernetes!

Extra!

Si quieren crear el HPA desde un manifiesto, pueden hacerlo con este código:

```yaml
apiVersion: autoscaling/v1
kind: HorizontalPodAutoscaler
metadata:
  name: web
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: php-apache
  minReplicas: 1
  maxReplicas: 10
  targetCPUUtilizationPercentage: 50
```