Tolerancia y escalabilidad: ReplicaSets

Tolerancia y escalabilidad: ReplicaSets

ReplicaSet es un recurso de Kubernetes que asegura que siempre se ejecuta un número de réplicas concreto de un Pod determinado. Por lo tanto, nos garantiza que un conjunto de Pods siempre están funcionando y disponibles proporcionándonos las siguientes características: **Tolerancia a fallos y Escalabilidad dinámica**.

Aunque en el módulo anterior estudiamos como gestionar el ciclo de vida de los Pods, en Kubernetes no vamos a trabajar directamente con Pods. Un recurso ReplicaSet controla un conjunto de Pods y es el responsable de que estos Pods siempre estén ejecutándose (**Tolerancia a fallos**) y de aumentar o disminuir las réplicas de dicho Pod (**Escalabilidad dinámica**). Estas réplicas de los Pods se ejecutarán en nodos distintos del cluster, aunque en nuestro caso al utilizar `minikube`, un cluster de un solo nodo, no vamos a poder apreciar como se reparte la ejecución de los Pods en varios nodos, todos los Pods se ejecutarán en la misma máquina.

El ReplicaSet va a hacer todo lo posible para que el conjunto de Pods que controla siempre se estén ejecutando. Por ejemplo: si el nodo del cluster donde se están ejecutando una serie de Pods se apaga, el ReplicaSet crearía nuevos Pods en otro nodo para tener siempre ejecutando el número que hemos indicado. Si un Pod se para por cualquier problema, el ReplicaSet intentará que vuelva a ejecutarse para que siempre tengamos el número de Pods deseado.

Describiendo un ReplicaSet

En este caso también vamos a definir el recurso de ReplicaSet en un fichero `nginx-rs.yaml`, por ejemplo como este:

```
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: replicaset-nginx
spec:
  replicas: 2
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
        - image: nginx
          name: contenedor-nginx
```

Algunos de los parámetros definidos ya lo hemos estudiado en la definición del Pod. Los nuevos parámetros de este recurso son los siguientes:

- `replicas`: Indicamos el número de Pods que siempre se deben estar ejecutando.
- `selector`: Seleccionamos los Pods que va a controlar el ReplicaSet por medio de las etiquetas. Es decir este ReplicaSet controla los Pods cuya etiqueta `app` es igual a `nginx`.
- `template`: El recurso ReplicaSet contiene la definición de un Pod. Fíjate que el Pod que hemos definido en la sección `template` tiene indicado la etiqueta necesaria para que sea seleccionado por el ReplicaSet (`app: nginx`).

Para seguir aprendiendo

- Para más información acerca de los ReplicaSet puedes leer: la documentación de la API

y la guía de usuario.

Vídeo: Describiendo un ReplicaSet

https://www.youtube.com/embed/VL3J63JqV5o

Vídeo: Describiendo un ReplicaSet

Gestionando los ReplicaSet

Ten en cuenta...

En esta unidad vamos a crear un recurso ReplicaSet que controlará un conjunto de recursos Pods. Para ello vamos a utilizar el fichero que estudiamos en la unidad anterior: `nginx-rs.yaml`.

Creación del ReplicaSet

Aunque en la unidad anterior usamos `kubectl create` para crear los recursos de nuestro cluster, es recomendable usar `kubectl apply`. La diferencia es la forma en la que actuamos sobre el cluster:

- **Configuración imperativa de objetos**: La definición del objeto está guardada en un fichero Yaml y ejecutamos un comando imperativo. Posteriormente no podremos modificar el objeto, habrá que borrarlo y crearlo de nuevo. Ejemplos:

  ```
  kubectl create -f recurso.yaml
  kubectl delete -f recurso.yaml
  ```

- **Configuración declarativa de objetos**: No se definen las acciones a realizar. Cuando se aplica la configuración del objeto estamos indicando un estado deseado al que queremos llegar. Posteriormente si la definición cambia, podremos cambiar el objeto. Recomendado en producción. Ejemplo:

  ```
  kubectl apply -f recurso.yaml
  ```

Por lo tanto para crear nuestro ReplicaSet, ejecutamos:

```
kubectl apply -f nginx-rs.yaml
```

Y podemos ver los recursos que se han creado con:

```
kubectl get rs,pods
```

Observamos que queríamos crear 2 replicas del Pod, y efectivamente se han creado.

Si queremos obtener información detallada del recurso ReplicaSet que hemos creado:

```
kubectl describe rs replicaset-nginx
```

Tolerancia a fallos

Y ahora comenzamos con las funcionalidades llamativas de Kubernetes. ¿Qué pasaría si borro uno de los Pods que se han creado? Inmediatamente se creará uno nuevo para que siempre estén ejecutándose los Pods deseados, en este caso 2:

```
kubectl delete pod <nombre_del_pod>
kubectl get pods
```

Escalabilidad

Para escalar el número de pods:

```
kubectl scale rs replicaset-nginx --replicas=5
kubectl get pods
```

Otra forma de hacerlo sería cambiando el parámetro `replicas` de fichero yaml, y volviendo a ejecutar:

```
kubectl apply -f nginx-rs.yaml
```

La escalabilidad puede ser para aumentar el número de Pods o para reducirla:

```
kubectl scale rs replicaset-nginx --replicas=1
```

Eliminando el ReplicaSet

Por último, si borramos un ReplicaSet se borrarán todos los Pods asociados:

```
kubectl delete rs replicaset-nginx
```

Otra forma de borrar el recurso, es utilizar el fichero yaml:

```
kubectl delete -f nginx-rs.yaml
```

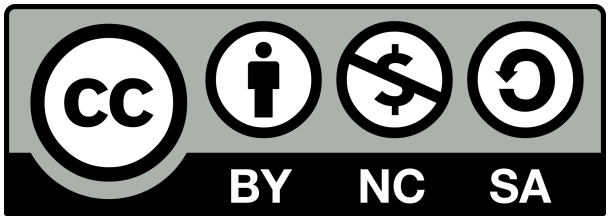
Vídeo: Gestionando los ReplicaSet

https://www.youtube.com/embed/MeCraOsxRPo

Gestionando los ReplicaSet

Créditos

Materiales desarrollados por:
Alberto Molina Coballes
José Domingo Muñoz Rodríguez
Propiedad de la Consejería de Educación y Deporte de la Junta de Andalucía



2021