Despliegues: Deployments

Despliegues: Deployments

El despliegue o Deployment es la unidad de más alto nivel que podemos gestionar en Kubernetes.

En los módulos anteriores hemos estudiado los Pods y los ReplicaSet, sin embargo, cuando queramos desplegar una aplicación en Kubernetes no vamos a gestionar éstos directamente, sino que vamos a crear un recurso de tipo Deployment. ¿Qué ocurre cuando creamos un nuevo recurso Deployment?

- La creación de un Deployment conlleva la creación de un ReplicaSet que controlará un conjunto de Pods creados a partir de la versión de la imagen que se ha indicado.
- Si hemos desarrollado una nueva versión de la aplicación y hemos creado una nueva imagen con la nueva versión, podemos modificar el Deployment indicando la nueva versión de la imagen. En ese momento se creará un nuevo ReplicaSet que controlará un nuevo conjunto de Pods creados a partir de la nueva versión de la imagen (habremos desplegado una nueva versión de la aplicación).
- Por lo tanto podemos decir que un Deployment va guardando un historial con los ReplicaSet que se van creando al ir cambiado la versión de la imagen. El ReplicaSet que esté activo en un determinado momento será el responsable de crear los Pods con la versión actual de la aplicación.
- Si tenemos un historial de ReplicaSet según las distintas versiones de la imagen que estamos utilizando, podemos, de una manera sencilla, volver a una versión anterior de la aplicación (**Rollback**).

Por la manera de trabajar de un Deployment, podemos indicar las funciones que nos aporta:

- Control de réplicas
- Escalabilidad de pods
- Actualizaciones continuas
- Despliegues automáticos
- Rollback a versiones anteriores

Arquitectura de nuestras aplicaciones

Tenemos dos clases de aplicaciones que podemos desplegar en un cluster de Kubernetes:

1. Aplicaciones que necesitan varios servicios para ejecutarse: por ejemplo una aplicación escrita en PHP y servida por un servidor web que necesita un servidor de base de datos para guardar la información. En este caso crearemos dos recursos Deployment: uno para desplegar la aplicación PHP y otro para desplegar la base de datos. Por cada servicio que necesite nuestra aplicación crearemos un Deployment para desplegarlo.
2. Aplicaciones construidas con microservicios: cada microservicio se puede desplegar de manera autónoma. Por cada microservicio que forma parte de la aplicación crearemos un recurso Deployment. Por ejemplo, una aplicación que tenga un frontend para ofrecer la información y que haga llamadas a un backend que ofrece un servicio web por medio de una API RESTful.

Describiendo un Deployment

Podemos crear un Deployment de forma imperativa utilizando un comando como el siguiente (se podrían indicar muchos más parámetros de configuración que podemos consultar en la documentación):

```
kubectl create deployment nginx --image nginx
```

Nosotros, sin embargo, vamos a seguir describiendo los recursos en un fichero yaml. En este caso para describir un Deployment de nginx podemos escribir un fichero `nginx-deployment.yaml`:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: deployment-nginx
  labels:
    app: nginx
spec:
  revisionHistoryLimit: 2
  strategy:
    type: RollingUpdate
  replicas: 2
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - image: nginx
        name: contendor-nginx
        ports:
        - name: http
          containerPort: 80
```

La creación de un Deployment crea un ReplicaSet y los Pods correspondientes. Por lo tanto en la definición de un Deployment se define también el ReplicaSet asociado (los parámetros

`replicas`, `selector` y `template`). Los atributos relacionados con el Deployment que hemos indicado en la definición son:

- `revisionHistoryLimit`: Indicamos cuántos ReplicaSets antiguos deseamos conservar, para poder realizar rollback a estados anteriores. Por defecto, es 10.
- `strategy`: Indica el modo en que se realiza una actualización del Deployment. Es decir, cuando modificamos la versión de la imagen del Deployment, se crea un ReplicaSet nuevo y ¿qué hacemos con los pods?:
  - `Recreate`: elimina los Pods antiguos y crea los nuevos.
  - `RollingUpdate`: va creando los nuevos Pods, comprueba que funcionan y se eliminan los antiguos; es la opción por defecto.

Además, hemos introducido un nuevo parámetro al definir el contenedor del pod: con el parámetro `ports` hemos indicado el puerto que expone el contenedor (`containerPort`) y le hemos asignado un nombre (`name`).

Para seguir aprendiendo

- Para más información acerca de los Deployment puedes leer: la documentación de la API y la guía de usuario.

Vídeo: Describiendo un Deployment

https://www.youtube.com/embed/NBsc_fILt8g

Vídeo: Describiendo un Deployment

Gestión básica de un Deployment

Ten en cuenta...

En esta unidad vamos a trabajar con el recurso Deployment, vamos a crear un Deployment de un servidor nginx, usando el fichero yaml que hemos visto en la unidad anterior: `nginx-deployment.yaml`.

Creación del Deployment

Cuando creamos un Deployment, se creará un ReplicaSet asociado, que creará y controlará los Pods que hayamos indicado.

```
kubectl apply -f nginx-deployment.yaml
kubectl get deploy,rs,pod
```

Para ver los recursos que hemos creado también podemos utilizar la instrucción:

```
kubectl get all
```

Esta orden muestra los Deployments, ReplicaSets, Pods y Services que tenemos creados en el cluster. Los Services lo estudiaremos en el siguiente módulo.

Escalado de los Deployments

Como ocurría con los ReplicaSets los Deployment también se pueden escalar, aumentando o disminuyendo el número de Pods asociados. Al escalar un Deployment estamos escalando el ReplicaSet asociado en ese momento:

```
kubectl scale deployment deployment-nginx --replicas=4
```

Otras operaciones

Si queremos acceder a la aplicación, podemos utilizar la opción de `port-forward` sobre el despliegue (de nuevo recordamos que no es la forma adecuada para acceder a un servicio que se ejecuta en un Pod, pero de momento no tenemos otra). En este caso si tenemos asociados más de un Pod, la redirección de puertos se hará sobre un solo Pod (no habrá balanceo de carga):

```
kubectl port-forward deployment/deployment-nginx 8080:80
```

Si queremos ver los logs generados en los Pods de un Deployment:

```
kubectl logs deployment/deployment-nginx
```

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

```
kubectl describe deployment deployment-nginx
```

Eliminando el Deployment

Si eliminamos el Deployment se eliminarán el ReplicaSet asociado y los Pods que se estaban gestionando.

```
kubectl delete deployment deployment-nginx
```

Vídeo: Gestión básica de un Deployment

https://www.youtube.com/embed/dcKWO6gXhhs

Vídeo: Gestión básica de un Deployment

Actualización y desactualización de un Deployment

Ciclo de vida del desarrollo de aplicaciones...

El ciclo de vida del desarrollo de aplicaciones cuando trabajamos con contenedores nos facilita la labor de versionar nuestros desarrollos. Por cada nueva versión que se desarrolla de nuestra aplicación podemos crear una nueva imagen del contenedor que podemos versionar utilizando la etiqueta del nombre de la imagen.

Por lo tanto, al crear un Deployment indicaremos la imagen desde la que se van a crear los Pods. Al indicar la imagen podremos indicar la etiqueta que nos indica la versión de la aplicación que vamos a implantar.

Una vez que hemos creado un Deployment a partir de una imagen de una versión determinada, tenemos los Pods ejecutando la versión indicada de la aplicación.

¿Cómo podemos actualizar a una nueva versión de la aplicación?. Se seguirán los siguientes pasos:

1. Tendremos que modificar el valor del parámetro `image` para indicar una nueva imagen, especificando la nueva versión mediante el cambio de etiqueta.
2. En ese momento el Deployment se actualiza, es decir, crea un nuevo ReplicaSet que creará nuevos Pods de la nueva versión de la aplicación.
3. Según la estrategia de despliegue indicada, se irán borrando los antiguos Pods y se crearán lo nuevos.
4. El Deployment guardará el ReplicaSet antiguo, por si en algún momento queremos volver a la versión anterior.

Veamos este proceso con más detalles estudiando un ejemplo de despliegue:

Desplegando la aplicación mediawiki

Vamos a partir del fichero `mediawiki-deployment.yaml` para desplegar la aplicación:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: mediawiki
  labels:
    app: mediawiki
spec:
  replicas: 1
  selector:
    matchLabels:
      app: mediawiki
  template:
    metadata:
      labels:
        app: mediawiki
    spec:
      containers:
      - name: contenedor-mediawiki
        image: mediawiki:1.31
        ports:
        - containerPort: 80
```

Si nos fijamos vamos a desplegar la versión 1.31 de la aplicación mediawiki. Creamos el despliegue con la siguiente instrucción:

```
kubectl apply -f mediawiki-deployment.yaml --record
```

Con la opción `--record` vamos a registrar las instrucciones que vamos a ejecutar a continuación para ir actualizando el despliegue. De esta forma al visualizar el historial de modificaciones veremos las instrucciones que han provocado cada actualización.

Podemos comprobar los recursos que hemos creado:

```
kubectl get all
```

Y si accedemos al Pod con un `port-forward` comprobamos que la versión actual de la mediawiki es la 1.31:

```
kubectl port-forward deployment/mediawiki 8080:80
```

Imagen de elaboración propia (CC BY-NC-SA)

Actualizar un Deployment

A continuación queremos desplegar una versión más reciente de la mediawiki. Para ello tenemos que modificar el campo `image` de nuestro Deployment, esta operación la podemos hacer de varias formas:

1. Modificando el fichero yaml y volviendo a ejecutar un `kubectl apply`.
2. Ejecutando la siguiente instrucción:

```
kubectl set image deployment/mediawiki contenedor-mediawiki=mediawiki:1.34 --r
```

Al ejecutar la actualización del Deployment podemos observar que se ha creado un nuevo ReplicaSet, que creará los nuevos Pods a partir de la versión modificada de la imagen. ¿Cómo se crean los nuevos Pods y se destruyen los antiguos? Dependerá de la estratégia de despliegue:

- Por defecto la estrategía de despliegue es `Recreate` que elimina los Pods antiguos y crea los nuevos.
- Si indicamos en el despliegue el tipo de estrategia `RollingUpdate`, se van creando los nuevos Pods, se comprueba que funcionan y se eliminan los antiguos.

Veamos los recursos que se han creado en la actualización:

```
kubectl get all
```

Kubernetes utiliza el término *rollout* para la gestión de diferentes versiones de despliegues. Podemos ver el historial de actualizaciones que hemos hecho sobre el despliegue:

```
kubectl rollout history deployment/mediawiki
```

Y volvemos a acceder a la aplicación con un `port-forward` para comprobar que realmente se ha desplegado la versión 1.34.

Imagen de elaboración propia (CC BY-NC-SA)

Rollback del Deployment

El proceso de despliegue de una nueva versión de una aplicación es una labor crítica, que tradicionalmente ha dado muchos problemas. Si estamos sirviendo una aplicación web que utilizan muchos usuarios, no nos podemos permitir que haya un corte en el servicio por un problema en el despliegue de una nueva versión.

Evidentemente, los problemas que pueden aparecer durante el despliegue de una nueva versión pueden estar causados por muchos motivos, y muchas veces es complicado tener todos los factores controlados. Si finalmente tenemos alguno, la pregunta sería: ¿Hemos diseñado un proceso que nos permita de una manera sencilla y rápida volver a la versión anterior de la aplicación que sabíamos que funcionaba bien?

A ese proceso de volver a una versión anterior de la aplicación es lo que llamamos **rollback**, o de forma concreta en k8s, "deshacer" un **rollout**. Veremos en este ejemplo que Kubernetes nos

ofrece un mecanismo sencillo de volver a versiones anteriores. Como hemos comentado, las actualizaciones de los Deployment van creando nuevos ReplicaSet, y se va guardando el historial de ReplicaSet anteriores. Deshacer un Rollout será tan sencillo como activar uno de los ReplicaSet antiguos.

Ahora vamos a desplegar una versión que nos da un error (la versión 2 de la aplicación no existe, no existe la imagen `mediawiki:2`). ¿Podremos volver al despliegue anterior?

```
kubectl set image deployment mediawiki contenedor-mediawiki=mediawiki:2 --record
```

Dependiendo de la estrategia de despliegue, esto puede provocar que la aplicación se quede en la versión anterior (`RollingUpdate`) o que no haya ningún Pod válido desplegado (`Recreate`). En cualquier caso, se puede volver a la versión anterior del despliegue mediante rollout:

```
kubectl rollout undo deployment/mediawiki
kubectl get all
```

Y terminamos comprobando el historial de actualizaciones:

```
kubectl rollout history deployment mediawiki
```

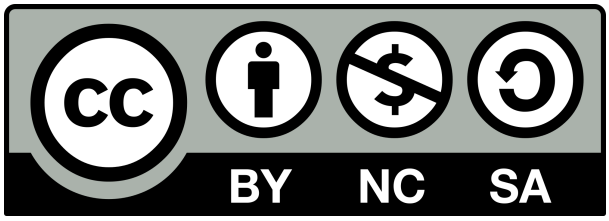
Vídeo: Actualización y desactualización de un Deployment

https://www.youtube.com/embed/6LjTwopWDFw

Vídeo: Actualización y desactualización de un Deployment

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