Contenedores en Kubernetes: Pods

Contenedores en Kubernetes: Pods

La unidad más pequeña que puede utilizar Kubernetes es el *Pod*, en inglés Pod significa "vaina", y podemos entender un Pod como una envoltura que contiene uno o varios contenedores (en la mayoría de los casos un solo contenedor). De forma genérica, un Pod representa un conjunto de contenedores que comparten almacenamiento y una única IP.

Un aspecto muy importante que hay que ir asumiendo es que los Pods son efímeros, se lanzan y en determinadas circunstancias se paran o se destruyen, creando en muchos casos nuevos Pods que sustituyan a los anteriores. Esto tiene importantes ventajas a la hora de realizar modificaciones en los despliegues en producción, pero tiene una consecuencia directa sobre la información que pueda tener almacenada el Pod, por lo que tendremos que utilizar algún mecanismo adicional cuando necesitemos que la información sobreviva a un Pod. Por lo tanto, aunque Kubernetes es un orquestador de contenedores, **la unidad mínima de ejecución es el Pod**, que contendrá uno a más contenedores según las necesidades:

- En la mayoría de los casos y siguiendo el principio de un proceso por contenedor, evitamos tener sistemas (como máquinas virtuales) ejecutando docenas de procesos, por lo que lo más habitual será tener un Pod en cuyo interior se define un contenedor que ejecuta un solo proceso.
- En determinadas circunstancias será necesario ejecutar más de un proceso en el mismo "sistema", como en los casos de procesos fuertemente acoplados, en esos casos, tendremos más de un contenedor dentro del Pod. Cada uno de los contenedores ejecutando un solo proceso, pero pudiendo compartir almacenamiento y una misma dirección IP como si se tratase de un sistema ejecutando múltiples procesos.

Existen además algunas razones que hacen que sea conveniente tener esta capa adicional por encima de la definición de contenedor:

- Kubernetes puede trabajar con distintos sistemas de gestión de contenedores (docker, containerd, rocket, cri-o, etc) por lo que es muy conveniente añadir una capa de

abstracción que permita utilizar Kubernetes de una forma homogénea e independiente del sistema de contenedores interno asociado.
- Esta capa de abstracción añade información adicional necesaria en Kubernetes como por ejemplo, políticas de reinicio, comprobaciones de que la aplicación esté inicializada (readiness probe) o comprobaciones de que la aplicación haya realizado alguna acción especificada (liveness probe).

Pod con un solo contenedor

En la situación más habitual, se definirá un Pod con un contenedor en su interior para ejecutar un solo proceso y este Pod estará ejecutándose mientras lo haga el correspondiente proceso dentro del contenedor. Algunos ejemplos pueden ser: ejecución en modo demonio de un servidor web, ejecución de un servidor de aplicaciones Java, ejecución de una tarea programada, ejecución en modo demonio de un servidor DNS, etc.

Pod multicontenedor

En algunos casos la ejecución de un solo proceso por contenedor no es la solución ideal, ya que existen procesos fuertemente acoplados que no pueden comunicarse entre sí fácilmente si se ejecutan en diferentes sistemas, por lo que la solución planteada en esos casos es definir un Pod multicontenedor y ejecutar cada proceso en un contenedor, pero que puedan comunicarse entre sí como si lo estuvieran haciendo en el mismo sistema, utilizando un dispositivo de almacenamiento compartido si hiciese falta (para leer, escribir ficheros entre ellos) y compartiendo externamente una misma dirección IP. Un ejemplo típico de un Pod multicontenedor es un servidor web nginx con un servidor de aplicaciones PHP-FPM, que se implementaría mediante un solo Pod, pero ejecutando un proceso de nginx en un contenedor y otro proceso de php-fpm en otro contenedor.

Al tratarse este curso de un curso de introducción a Kubernetes no vamos a poder ver todas las cargas de trabajo, ni la ejecución y despliegue de todo tipo de aplicaciones, por lo que consideramos más razonable no utilizar ejemplos de Pods multicontenedor y centrarnos en la comprensión de las características principales de Kubernetes mediante ejemplos sencillos, comunes y muy apropiados para ejecutarse en Kubernetes, mediante en uso de Pods con un solo contenedor.

Describiendo un Pod

Es posible crear un Pod directamente (lo que se denomina utilización imperativa) mediante `kubectl`:

```
kubectl run pod-nginx --image=nginx
```

De esta forma se crea un Pod con un contenedor que utiliza la imagen `nginx:latest` (no hemos especificado una versión) del registro que esté definido por defecto en el cluster de Kubernetes, se asigna una dirección IP y se lanza en uno de los nodos del cluster.

Un Pod tiene otros muchos parámetros asociados, que en este caso quedarán sin definir o Kubernetes asumirá los valores por defecto. Sin embargo es mucho más habitual trabajar con los objetos de Kubernetes de manera declarativa, definiendo los objetos de forma detallada a través de un fichero en formato YAML. De esta forma tenemos un fichero con la definición del objeto que hemos lanzado y podemos utilizar en otro momento exactamente la misma definición o podemos ir modificándola y aplicando los cambios cuando sea conveniente.

Un ejemplo podría ser el contenido del fichero `pod.yaml`:

```
apiVersion: v1 # required
kind: Pod # required
metadata: # required
 name: pod-nginx # required
 labels:
   app: nginx
   service: web
spec: # required
 containers:
   - image: nginx:1.16
     name: contenedor-nginx
     imagePullPolicy: Always
```

Veamos cada uno de los parámetros que hemos definido:

- `apiVersion: v1`: La versión de la API que vamos a usar.
- `kind: Pod`: La clase de recurso que estamos definiendo.
- `metadata`: Información que nos permite identificar unívocamente el recurso:
    - `name`: Nombre del pod

    - `labels`: Las Labels nos permiten etiquetar los recursos de Kubernetes (por ejemplo un pod) con información del tipo clave/valor.
- `spec`: Definimos las características del recurso. En el caso de un Pod indicamos los contenedores que van a formar el Pod (sección `containers`), en este caso sólo uno.
    - `image`: La imagen desde la que se va a crear el contenedor
    - `name`: Nombre del contenedor.
    - `imagePullPolicy`: Las imágenes se guardan en un registro interno. Se pueden utilizar registros públicos (google o docker hub son los más usados) y registros privados. La política por defecto es `IfNotPresent`, que se baja la imagen si no está en el registro interno. Si queremos forzar la descarga desde el repositorio externo, tendremos que indicar `imagePullPolicy:Always`.

Para seguir aprendiendo

- Para más información acerca de los Pod puedes leer: la documentación de la API y la guía de usuario.
- Para más información acerca de la estructura de la definición de los objetos de Kubernetes: Understanding Kubernetes Objects.

Vídeo: Describiendo un Pod

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

Vídeo: Describiendo un Pod

Gestionando los Pods

Tenemos un fichero pod.yaml, donde hemos definido un Pod de la siguiente manera:

```
apiVersion: v1
kind: Pod
metadata:
 name: pod-nginx
 labels:
   app: nginx
   service: web
spec:
 containers:
   - image: nginx:1.16
     name: contenedor-nginx
     imagePullPolicy: Always
```

Podemos crear directamente el Pod desde el fichero yaml:

```
kubectl create -f pod.yaml
```

Y podemos ver el estado en el que se encuentra y si está o no listo:

```
kubectl get pods
```

(Sería equivalente usar po, pod o pods).

Si queremos ver más información sobre los Pods, como por ejemplo, saber en qué nodo del cluster se está ejecutando:

```
kubectl get pod -o wide
```

Para obtener información más detallada del Pod (equivalente al inspect de docker):

```
kubectl describe pod pod-nginx
```

Podríamos editar el Pod y ver todos los atributos que definen el objeto, la mayoría de ellos con valores asignados automáticamente por el propio Kubernetes y podremos actualizar ciertos valores:

```
kubectl edit pod pod-nginx
```

Sin embargo, es una opción compleja para utilizarla a estas alturas del curso y hay que comprender mejor cómo funcionan los objetos de Kubernetes para poder hacer modificaciones de forma apropiada, y además, veremos más adelante otra manera más correcta de actualizar un objeto de Kubernetes.

Normalmente no se interactúa directamente con el Pod a través de una shell, pero sí se obtienen directamente los logs al igual que se hace en docker:

```
kubectl logs pod-nginx
```

En el caso poco habitual de que queramos ejecutar alguna orden adicional en el Pod, podemos utilizar el comando `exec`, por ejemplo, en el caso particular de que queremos abrir una shell de forma interactiva:

```
kubectl exec -it pod-nginx -- /bin/bash
```

Podemos acceder a la aplicación, redirigiendo un puerto de localhost al puerto de la aplicación:

```
kubectl port-forward pod-nginx 8080:80
```

Y accedemos al servidor web en la url http://localhost:8080.

NOTA: Esta no es la forma con la que accedemos a las aplicaciones en Kubernetes. Para el acceso a las aplicaciones usaremos un recurso llamado Service. Con la anterior instrucción lo que estamos haciendo es una redirección desde localhost el puerto 8080 al puerto 80 del Pod y es útil para pequeñas pruebas de funcionamiento, nunca para acceso real a un servicio.
NOTA2: El `port-forward` no es igual a la redirección de puertos de docker, ya que en este caso la redirección de puertos se hace en el equipo que ejecuta `kubectl`, no en el equipo que ejecuta los Pods o los contenedores.

Para obtener las etiquetas de los Pods que hemos creado:

```
kubectl get pods --show-labels
```

Las etiquetas las hemos definido en la sección metadata del fichero yaml, pero también podemos añadirlos a los Pods ya creados:

```
kubectl label pods pod-nginx service=web --overwrite=true
```

Las etiquetas son muy útiles, ya que permiten seleccionar un recurso determinado (en un cluster de Kubernetes puede haber cientos o miles de objetos).Por ejemplo para visualizar los Pods que tienen una etiqueta con un determinado valor:

```
kubectl get pods -l service=web
```

También podemos visualizar los valores de las etiquetas como una nueva columna:

```
kubectl get pods -Lservice
```

Y por último, eliminamos el Pod mediante:

```
kubectl delete pod pod-nginx
```

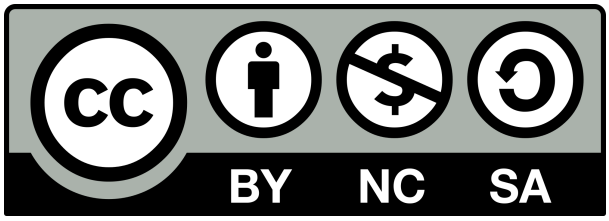
Vídeo: Gestionando los Pods

https://www.youtube.com/embed/OA0OheCtrXo

Vídeo: Gestionando los Pods

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