Un poco de Java y +

Otra forma de hablar de nuestro día a día…

23 DICIEMBRE 2020
LUISMI GRACIA

Un poco de Kubernetes: Conceptos principales

INTRODUCCIÓN

A estas alturas nadie duda de que los contenedores son una gran tecnología para empaquetar, desplegar y gestionar aplicaciones.

Pero además para construir una aplicación en forma de contenedor también necesitamos un lugar donde alojar los contenedores, poder actualizarlos y proporcionarles una red y almacenamiento. En la actualidad Kubernetes se ha convertido en la tecnología de orquestación de contenedores más usada y popular, casi el estándar de facto.

¿QUÉ ES KUBERNETES?

Kubernetes (también conocido por su abreviatura "k8s") es un sistema de orquestación de contenedores open-source. K8s fue creado por Google y en 2016 fue donado a la Cloud Native Computing Foundation (CNCF).

K8s es capaz de gestiona un cluster de múltiples hosts en el que desplegar y monitorizar contenedores, permitiendo de forma declarativa crear y actualizar "recursos" que describen qué contenedores ejecutar, cómo configurarlos y cómo enrutar el tráfico de la red hacia ellos.

Kubernetes actualiza y supervisa continuamente el clúster para asegurarse de que coincide con el estado deseado, incluyendo el reinicio automático, la reprogramación y la replicación para garantizar que las aplicaciones se inicien y permanezcan en funcionamiento.

Kubernetes está disponible como servicio en la mayoría de proveedores cloud además de existir diversas distribuciones como Red Hat OpenShift, Rancher Kubernetes o VMWare Tanzu.

CONCEPTOS CLAVE DE KUBERNETES

Al ser Kubernetes declarativo, iniciarse en Kubernetes significa sobre todo comprender qué recursos podemos crear y cómo se utilizan para desplegar y configurar contenedores en el cluster.

Para definir los recursos, se usa el formato YAML. Los recursos disponibles y los campos de cada recurso pueden cambiar con las nuevas versiones de Kubernetes, por lo que es importante usar el API de su versión.

POD

Un POD es un grupo de uno o más contenedores. K8s lanza todos los contenedores de un pod en el mismo host, con el mismo espacio de nombres de red, para que todos tengan la misma dirección IP y puedan acceder entre sí utilizando el localhost. Los contenedores de un pod también pueden compartir volúmenes de almacenamiento.

Normalmente no se crean recursos de pod directamente. En lugar de eso, hacemos que Kubernetes los administre a través de un **Deployment,** así garantizamos la tolerancia a fallos, escalabilidad y actualizaciones continuas.

DEPLOYMENT

Un deployment maneja una o más instancias idénticas de PODs. Kubernetes se asegura de que se esté ejecutando el número especificado de pods y, en una actualización sustituirá las instancias de pod uno a uno, lo que permitirá que las actualizaciones de la aplicación no tengan ningún tiempo de inactividad.

En la imagen vemos un ejemplo de despliegue:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  labels:
    app: nginx
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.19.3-alpine
        volumeMounts:
        - mountPath: /usr/share/nginx
          name: www-data
          readOnly: true
        ports:
        - containerPort: 80
      initContainers:
      - name: git-clone
        image: alpine/git
        args: ["clone", "https://github.com/AlanHohn/
hello-world-static.git", "/www/html"]
        volumeMounts:
        - mountPath: /www
          name: www-data
      volumes:
      - name: www-data
        emptyDir: {}
```

(https://unpocodejava.files.wordpress.com/2020/12/image002.jpg)

La sección **Template del Deployment** especifica exactamente cómo deben ser los PODs creados. K8s creará automáticamente el número necesario de pods a partir de la plantilla.

Cuando se crea un pod, Kubernetes lo supervisa y lo reiniciará automáticamente si el contenedor se termina. Kubernetes pueden configurarse para monitorizar determinar si el pod está listo (readinessProbe) y todavía vivo (livenessProbe).

El ejemplo de arriba define un contenedor en el **Pod** y también define un **Init Container.**

Init Container

El Init Container se ejecuta antes de que comience el contenedor del POD principal.

En el ejemplo usa Git para descargar los archivos que el servidor web NGINX servirá. Usamos un "init container" porque así Git se ejecuta una vez y luego acaba sin que K8s piense que es un error.

Los deployments identifican los pods que deben gestionar usando el campo **matchLabels**. Este campo debe tener siempre los mismos datos que el campo **metadata.labels** dentro de la plantilla.

El despliegue gestionará cualquier pod que se esté ejecutando y que coincida con el selector de matchLabels, incluso si se crearon por separado, por lo que estos nombres deben ser únicos

SERVICE

Un servicio proporciona balanceo de carga a un grupo de pods. Cada vez que Kubernetes crea un pod, se le asigna una dirección IP única. Cuando un pod es reemplazado, el nuevo pod típicamente recibe una nueva IP.

Al declarar un Service, podemos proporcionar un único punto de entrada para todos los pods en un despliegue. Este único punto de entrada (nombre de host y dirección IP) sigue siendo válido ya que los pods van y vienen, y el clúster de Kubernetes incluso proporciona un servidor DNS para que podamos usar los nombres de servicio como nombres de host.

```
kind: Service
apiVersion: v1
metadata:
  name: nginx-service
spec:
  selector:
    app: nginx
  ports:
  - protocol: TCP
    port: 80
```

(https://unpocodejava.files.wordpress.com/2020/12/image004.jpg)

Los services y deployments pueden ser creados en cualquier orden. El servicio monitoriza activamente k8s en busca de pods que coincidan con el campo de selección. En este caso, el servicio hará coincidir los pods con el contenido de **metadata.labels** de la aplicación: **nginx**, como el que se muestra en el ejemplo de despliegue anterior.

El tipo de servicio más común es el **ClusterIP**, este servicio tiene una dirección IP accesible sólo desde dentro del cluster de Kubernetes; exponer el servicio fuera del cluster requiere otro recurso como un Ingress.

Es importante saber que cuando el tráfico de la red se envía a una dirección de servicio y a un puerto, Kubernetes utiliza port-forwarding para enrutar el tráfico a un pod específico. Sólo se reenvían los protocolos y puertos declarados, por lo que otros tipos de tráfico (como el ping ICMP) no funcionarán a una dirección de servicio, incluso dentro del clúster.

INGRESS

Un **Ingress** es una de las formas para enrutar el tráfico desde fuera del cluster. Para usar Ingress, el administrador del clúster primero despliega un **Ingress Controller,** que se registra en el clúster de Kubernetes para ser notificado cuando se cree, actualice o elimine un recurso de Ingress, y se configura para dirigir el tráfico en función de los recursos de entrada.

La ventaja de este enfoque es que sólo el Ingress Controller necesita una dirección IP a la que se pueda acceder desde fuera del clúster, lo que simplifica la configuración. Aquí hay un ejemplo de Ingress:

```
apiVersion: networking.k8s.io/v1beta1
kind: Ingress
metadata:
  name: nginx-ingress
spec:
  rules:
  - http:
      paths:
      - path: /
        backend:
          serviceName: nginx-service
          servicePort: 80
```

(https://unpocodejava.files.wordpress.com/2020/12/image006.jpg)

Este ejemplo dirige todo el tráfico en el cluster a nuestro servicio NGINX. En un clúster de producción, se puede utilizar el DNS para enrutar muchos nombres de host a la misma dirección IP de entrada, y luego utilizar reglas de host para enrutar el tráfico a la aplicación correcta.

PERSISTENT VOLUME CLAIM

Kubernetes tiene diversos tipos de recursos de almacenamiento.

El deployment anterior muestra el más simple, un directorio vacío montado en múltiples contenedores en el mismo pod. Para un almacenamiento verdaderamente persistente, el enfoque más flexible es utilizar una **Persistent Volume Claim**, este solicita a Kubernetes que asigne dinámicamente el almacenamiento de una **storage class.**

Una vez que se crea la persistente volumen claim, se puede adjuntar a un pod.

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: web-static-files
spec:
  accessModes:
  - ReadWriteOnce
  resources:
    requests:
      storage: 8Gi
```

(https://unpocodejava.files.wordpress.com/2020/12/image008.jpg)

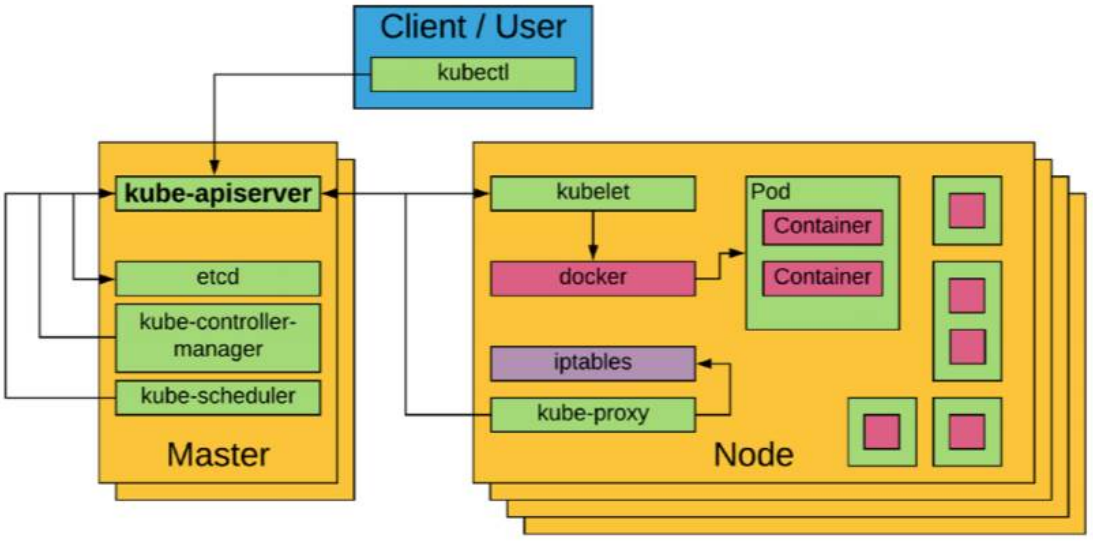
En el YAML se declara un persistent volumen claim:

```
    volumes:
     - name: www-date
       persistentVolumeClaim:
         claimName: web-static-files
```

(https://unpocodejava.files.wordpress.com/2020/12/image010.jpg)

ARQUITECTURA DE KUBERNETES

Kubernetes utiliza una arquitectura cliente-servidor, como se ve en la imagen:



(https://unpocodejava.files.wordpress.com/2020/12/image018.jpg)

Un cluster de Kubernetes es un conjunto de máquinas físicas o virtuales y otros recursos de infraestructura que se utilizan para ejecutar aplicaciones.

Las máquinas que gestionan el clúster se denominan **Masters** y las máquinas que ejecutan los contenedores se denominan **Nodes/Nodos**.

MASTER

El master ejecuta los servicios que gestionan el cluster.

El más importante es el **kube-apiserver,** que es el servicio principal que los clientes y los nodos utilizan para consultar y modificar los recursos que se ejecutan en el clúster. El API Server es asistido por **etcd**, un almacén distribuido de llaves-valor usado para registrar el estado del cluster;

kube-controllermanager es un programa de monitorización que decide qué cambios hacer cuando se añaden, cambian o eliminan recursos

kubescheduler es un programa que decide dónde ejecutar los pods en base a los nodos disponibles y su configuración.

En una instalación de Kubernetes de alta disponibilidad, habrá múltiples maestros, uno actuando como el principal y los otros como réplicas.

NODO

Un nodo es una máquina física o virtual con los servicios necesarios para ejecutar los contenedores.

Un cluster de Kubernetes debe tener tantos nodos como sean necesarios para ejecutar todos los pods requeridos.

Cada nodo tiene dos servicios de Kubernetes:

kubelet, que recibe comandos para ejecutar los contenedores y utiliza el motor de contenedores (por ejemplo, Docker) para ejecutarlos

y **kubeproxy**, que gestiona las reglas de red para que las conexiones a las direcciones IP de los servicios se enruten correctamente a los pods.

Cada nodo puede ejecutar varios pods, y cada pod puede incluir uno o más contenedores. El pod es puramente un concepto de Kubernetes y el kubelet configura el motor de contenedores para colocar múltiples contenedores en el mismo espacio de nombres de la red de manera que esos contenedores compartan una dirección IP.

Extraído de Getting Started with Kubernetes (https://dzone.com/storage/assets/14140908-refcard-233-update-getting-started-kubernetes-2020.pdf)

CLOUD **KUBERNETES** **QUÉ ES** **UN POCO DE**