Implantación de sistemas seguros de despliegue de software

Tema 3. Laboratorio de ejercicios de Kubernetes

Índice general

Capítulo 1

Kubernetes

En este taller vamos a realizar el despliegue de las aplicaciones del taller de Docker en el clúster de Minikube. Por lo general, al no tratarse de un entorno de desarrollo o producción, lo normal es que las aplicaciones las acabéis utilizando en Docker. Sin embargo, es interesante ver como podríamos realizar el despliegue con Kubernetes.

1.1. Deployments

En primer lugar, vamos a definir la estructura que queremos que tenga nuestro despliegue. Puesto que las tres aplicaciones anteriormente vistas tienen una estructura similar, es decir, un servidor web y un servidor de base de datos, vamos a utilizar el mismo servidor de base de datos para las tres aplicaciones. Por tanto, una vez descargados los proyectos desde GitHub, lo primero que haremos será modificar los datos de conexión al servidor MySQL.

Primero, cambiamos los datos para la aplicación bWAPP. Para ello nos dirigimos a la carpeta bWAPP/admin y editamos el archivo settings.php modificando servidor, usuario y contraseña.

Lo siguiente que haremos, será modificar los datos de acceso a la aplicación DVWA. Para ello, nos vamos a la carpeta DVWA/config y modificamos el fichero config.inc.php utilizando como servidor, usuario y contraseña los mismos datos utilizados en la aplicación bWAPP.

El nombre del servidor será el hostname de la máquina MySQL. Como usuario y contraseña podéis utilizar la que prefiráis.

Por último, modificamos los datos de acceso de la aplicación Mutillidae II. Para ello, nos dirigimos a la carpeta mutillidae/includes y modificamos el archivo database-config.inc cambiando la variable host, el usuario y la contraseña (mismos valores utilizados en las aplicaciones anteriores). **En el taller de Docker utilizamos un repositorio secundario del proyecto que contenía exclusivamente los archivos Dockerfile y docker-compose.yml para el despliegue. Para configurar Mutillidae II, podemos descargar el proyecto desde el repositorio principal (https://github.com/webpwnized/mutillidae) y utilizar el Dockerfi-**

le de la carpeta 'www' del repositorio secundario. Tened en cuenta que es necesario editar y adaptar el Dockerfile que descargamos.

Una vez hemos modificado los parámetros de acceso al servidor de base de datos, es momento de empezar a montar el despliegue del servidor de base de datos. En primer lugar, es necesario que iniciemos el clúster de Minikube, en caso de que no esté ejecutándose en vuestra máquina local. Para ello, escribimos en el terminal el siguiente comando:

```
$ minikube start
```

Veréis en la consola como empieza a configurarse y desplegarse el clúster de Minikube. Una vez esté activo el clúster, es necesario crear el despliegue de MySQL. Para ello, nos creamos un archivo .yml en el que indicaremos las distintas opciones del servidor MySQL, así como las variables de entorno que se utilizaran para iniciar sesión. El Listing 1.1 muestra un posible ejemplo para realizar el despliegue. En nuestro caso, hemos utilizado como usuario de la base de datos 'root', la contraseña '1234' y como nombre del host 'mysqlserver'.

Listing 1.1: Archivo YAML para crear el objeto Deployment de MySQL

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: mysqlserver
  labels:
    app: mysqlserver
spec:
  replicas: 1
  selector:
    matchLabels:
      app: mysqlserver
  strategy:
    type: Recreate
  template:
    metadata:
      labels:
        app: mysqlserver
    spec:
      containers:
        - args:
            - --default-authentication-plugin=mysql_native_password
          env:
            - name: MYSQL_ROOT_PASSWORD
              value: "1234"
          image: mysql
          name: mysqlserver
          ports:
           - containerPort: 3306
```

```
    hostname: mysqlserver
    restartPolicy: Always
```

El siguiente paso es realizar el despliegue de nuestro servidor MySQL. Para ello, ejecutamos el siguiente comando en el terminal:

```
$ kubectl create -f file.yml
```

En estos momentos, debe haberse creado el despliegue de MySQL y el número de réplicas (Pods) indicado en nuestro archivo de configuración. Podemos comprobar que se ha desplegado correctamente ejecutando los siguientes comandos:

```
$ kubectl get Deployments
$ kubectl get pods
```

Si todo ha funcionado correctamente os debería aparecer que se encuentra ejecutándose la imagen de MySQL. Sin embargo, para que pueda ser accesible es necesario crear un servicio para que las peticiones sean redirigidas al Pod. Aunque primero, vamos a realizar el despliegue de las dos aplicaciones. El Listing 1.2 muestra un ejemplo de despliegue para la aplicación bWAPP. Para las otras aplicaciones, el archivo es muy similar y solo basta modificar las etiquetas, el nombre del Deployment y del contenedor, y la imagen que utilizará el contenedor.

Listing 1.2: Archivo YAML para crear el objeto Deployment de la aplicación bWAPP

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: bwapp
  labels:
    app: bwapp
spec:
  replicas: 1
  selector:
    matchLabels:
      app: bwapp
  strategy:
    type: Recreate
  template:
    metadata:
      labels:
        app: bwapp
    spec:
      containers:
```

```
    - image: bwapp
      name: bwapp
      ports:
        - containerPort: 80
      imagePullPolicy: Never
  restartPolicy: Always
```

Una vez generados los archivos de despliegue podemos generar los objetos de manera similar a como lo realizamos con MySQL, utilizando el comando 'kubectl create -f fichero.yml'. Una vez creados los objetos de despliegue comprobamos si se han desplegado de manera correcta los Pods con el comando 'kubectl get pods'. **¿Por qué el Pod no se está ejecutando correctamente? ¿Cómo lo solucionamos?**

Como habréis imaginado, se están buscando las imágenes de Docker en el entorno local y estas no se encuentran disponibles en el entorno de Minikube. Por tanto, debemos conectarnos al entorno de Docker de Minikube y generar las imágenes (con el mismo nombre que se indica en el despliegue). Utilizamos el siguiente comando para que nuestro cliente de Docker apunte al demonio de Docker del clúster de Minikube:

```
$ eval $(minikube docker-env)
```

Ahora cuando construyamos una imagen en Docker o cuando despleguemos un contenedor se estará ejecutando en el entorno de Minikube. Por tanto, cuando creemos las imágenes de las aplicaciones se crearan en el clúster de Kubernetes y los Pods podrán ejecutarse de manera correcta.

En este punto, deberíais tener las imágenes de las aplicaciones y MySQL ejecutándose en vuestro clúster de Minikube. El siguiente paso será la creación de los servicios para que se puedan comunicar y puedan ser accesibles desde el exterior del clúster.

1.2. Servicios

En esta sección vamos a crear los servicios para las aplicaciones desplegadas anteriormente y para la imagen de MySQL. Para MySQL crearemos un servicio del tipo ClusterIP, ya que no es necesario que el servicio sea accesible desde fuera del clúster. Para las aplicaciones crearemos un servicio del tipo NodePort. Los Listings 1.3 y 1.4 muestran un ejemplo del servicio MySQL y de la aplicación bWAPP. Para las aplicaciones de DVWA y Mutillidae el archivo es muy similar al de bWAPP. Recordad utilizar las mismas etiquetas y selectores que utilizasteis en sus respectivos Deployments para que se mapee el servicio con los Pods que se están ejecutando.

Listing 1.3: Archivo YAML para crear el objeto servicio de MySQL

```yaml
apiVersion: v1
kind: Service
metadata:
  labels:
    service: mysqlserver
  name: mysqlserver
spec:
  type: ClusterIP
  ports:
    - port: 3306
  selector:
    service: mysqlserver
```

Listing 1.4: Archivo YAML para crear el objeto servicio de la aplicación bWAPP

```yaml
apiVersion: v1
kind: Service
metadata:
  name: bwapp
  labels:
    app: bwapp
spec:
  type: NodePort
  selector:
    app: bwapp
  ports:
    - port: 8080
      targetPort: 80
```

Una vez creados los servicios podemos comprobar que funcionan correctamente a través del comando 'kubectl describe service nombre_servicio'. Deberíamos ver en el apartado de 'Endpoints' como el servicio está mapeado con los distintos Pods. Para comprobar el correcto funcionamiento de las aplicaciones web podemos acceder a través de su IP 'pública' desde fuera del clúster. Para obtener la información de los servicios, ejecutamos el siguiente comando:

```
$ minikube service list
```

El comando anterior debería de arrojarnos como salida los distintos servicios creados y la URL desde la que se puede acceder desde nuestra máquina local. Si utilizamos nuestro navegador para acceder a esas direcciones web, deberíamos poder visualizar correctamente las distintas aplicaciones desplegadas.

Recordad para cada una de las aplicaciones crear/restablecer la base de datos ya que inicialmente estarán vacías.

1.3. Ingress

En esta sección vamos a configurar el objeto Ingress, que nos permite gestionar el acceso externo a servicios del clúster. Ingress expone rutas de HTTP/HTTPS a los servicios dentro del clúster basándose en las reglas definidas en el objeto. En primer lugar, es necesario activar el plugin de Ingress en el clúster de Minikube.

```
$ minikube addons enable ingress
```

Lo siguiente que realizaremos es crear el archivo YAML del objeto Ingress y definir las reglas. Si hemos seguido todos los mismos pasos hasta aquí, cuando accedemos a cada una de las aplicaciones deberíamos acceder de la siguiente manera:

- bWAPP –>http://ip:puerto/bWAPP
- DVWA –>http://ip:puerto/DVWA
- Mutillidae –>http://ip:puerto

Por tanto, podemos crear diferentes reglas como si se tratara de VirtualHost en Apache, de manera que cuando coincida la dirección con una regla establecida encamine el tráfico hacia ese servicio. El Listing 1.5 muestra el archivo de configuración del objeto Ingress.

Listing 1.5: Archivo YAML para crear el objeto Ingress

```yaml
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
   name: aplicaciones-vulnerables
spec:
   rules:
      - host: aplicaciones-vulnerables.com
        http:
          paths:
             - path: /bWAPP
               pathType: Prefix
               backend:
                 service:
                    name: bwapp
                    port:
                       number: 8080
             - path: /DVWA
               pathType: Prefix
               backend:
                 service:
                    name: dvwa
                    port:
```

```
            number: 8080
    - path: /
      pathType: Prefix
      backend:
        service:
          name: mutillidae
          port:
            number: 8080
```

Modificar el nombre y puerto de los servicios por los nombres y puertos que hayáis usado antes

Finalmente, creamos el objeto Ingress utilizando el comando 'kubectl apply -f fichero.yml'. Para listar las características del objeto Ingress podéis utilizar el comando 'kubectl get ingress nombre_objeto'. En las características del objeto mostrado, deberías poder identificar los distintos path y los servicios a los que se les enviará la petición cuando se produzca una coincidencia con la regla establecida. Para probar que todo funciona correctamente, puesto que no tenemos un dominio, tenemos que engañar a nuestro sistema añadiendo el host que hayamos especificado en el archivo '/etc/hosts'. Modificamos el fichero '/etc/hosts' e incluimos la siguiente linea:

```
192.168.49.2 aplicaciones-vulnerables.com
```

La dirección IP es la dirección que aparece en el objeto Ingress que hemos creado. Usad 'kubectl get ingress' para obtenerla

Una vez añadida esta linea, cuando se realice una petición a ese host podrá realizar la traducción del 'nombre de dominio' a una dirección IP como si de un servidor DNS se tratase. Por tanto, si abrimos el navegador y accedemos a la dirección 'http://aplicaciones-vulnerables.com' deberíamos ser redirigidos a la aplicación Mutillidae II, mientras que si le añadimos a ese host el path bWAPP deberíamos ser dirigidos hacia esa aplicación, etc.