MIGRACIÓN DE UNA APLICACIÓN A KUBERNETES

Experto en Sistemas Software Distribuidos

Práctica

Abraham Santana Cebrián
abraham.santana.cebrian@alumnos.upm.es

Tabla de contenido

Enunciado

Objetivo

El objetivo de la práctica es que el alumno migre una aplicación ya existente de cierta complejidad a un entorno Kubernetes, desarrollando los specs o helms necesarios para su correcto despliegue y funcionamiento, poniendo en práctica los conceptos vistos en el módulo de Computación en la Nube del Curso de Experto en Sistemas Software Distribuidos.

Proyecto Software

Los specs o helms desarrollados se usarán para gestionar el despliegue de un proyecto software existente. El software que deberá usarse es alguno de los proyectos incluidos dentro de la carpeta 2018 del repositorio de GitHub https://github.com/codeurjc/desarrollo-aplicaciones-distribuidas/.

Estos proyectos están basados en SpringBoot, y consisten en dos aplicaciones, una de las cuales presenta el frontal web, la otra sirve de apoyo a la primera para alguna tarea concreta, una base de datos MySQL, y un balanceador de carga.

Especificaciones para el despliegue en Kubernetes

El despliegue en Kubernetes deberá tener las siguientes características:

- Se usará un Kubernetes con Minikube. Esto permitirá al profesor poder lanzar la aplicación en su propio Minikube.
- Se clonará el repositorio con el proyecto software original en una cuenta GitHub del alumno, y dentro de este nuevo repositorio se introducirán los ficheros necesarios para el despliegue en Kubernetes.
- El despliegue se puede describir con Specs o Helm, pero se valorará el uso de Helm.
    - Las etiquetas usadas para las aplicaciones deben coincidir con la primera parte de la dirección de correo del alumno. Si su dirección de correo es pepe.sanchez@miempresa.com la etiqueta usada para la aplicación debería ser pepesanchez. De esta forma se evitan conflictos al desplegar aplicaciones con el mismo nombre.
- La base de datos debe usar un PersistentVolume en una carpeta local del nodo Minikube y no debe ser expuesta al exterior.
- La aplicación que hace de frontal web debe desplegarse con dos réplicas y estar expuesta al exterior. Nota: es posible que la aplicación realice inicializaciones en la bbdd, que podrían no funcionar al arrancar dos réplicas. Si esto es así, se considerará una sola réplica para el frontal web.
- La aplicación interna debe desplegarse con dos réplicas, pero no debe ser expuesta al exterior.
- Se valorará el uso de Ingress para el acceso al frontal web.

Ejecución de la Práctica

Repositorio GitHub

El proyecto se encuentra en el repositorio de código GitHub en la siguiente URL:

https://github.com/Abrin09/FoodAtHome

Servicio Interno

Según los requisitos:

- Para poder tener múltiples instancias controladas por Kubernetes se ha dispuesto un Deployments con dos replicas, puesto que el servicio interno debe desplegarse con dos replicas.
- Para poder exponer al exterior el frontal web se va a utilizar un Ingress Controller. Para poder exponer el servicio al exterior se ha optado por utilizar un servicio de tipo ClusterIP.
- No necesitamos ningún tipo de elemento adicional para balancear la carga entre las dos instancias del frontal web puesto que esta característica nos la ofrece Kubernetes.

Imagen Docker

Para poder desplegar el frontal web en Kubernetes necesitamos una imagen Docker. Para construirla a partir del proyecto actual utilizamos el siguiente fichero Dockerfile, que puede encontrarse en ./ServicioInterno/Dockerfile

Para la construcción de esta imagen Docker se ha utilizado:

- Un Dockerfile multistage para no tener instaladas en la imagen final herramientas que solo son necesarias en la construcción del proyecto.
- Una capa independiente para la descarga de las dependencias del proyecto Maven. Esta capa actúa de caché de dependencias. Mientras no cambie el fichero pom.xml no se descargarán nuevamente.

```
RUN mvn dependency:go-offline
```

El despliegue de la base de datos de realizará mediante el esquema YAML para Kubernetes del fichero `servicio-interno.yaml`. Se puede ver el contenido de este fichero en el Anexo 3: servicio-interno.yaml.

Web Front End

Según los requisitos:

- Para poder tener múltiples instancias controladas por Kubernetes se ha dispuesto un Deployments con dos replicas, puesto que el frontal web debe desplegarse con dos replicas.
- Para poder exponer al exterior el frontal web se va a utilizar un Ingress Controller. Para poder exponer el servicio al exterior se ha optado por utilizar un servicio de tipo ClusterIP.
- No necesitamos ningún tipo de elemento adicional para balancear la carga entre las dos instancias del frontal web puesto que esta característica nos la ofrece Kubernetes.

Imagen Docker

Para poder desplegar el frontal web en Kubernetes necesitamos una imagen Docker. Para construirla a partir del proyecto actual utilizamos el siguiente fichero Dockerfile, que puede encontrarse en ./EjemploparaDAD/Dockerfile

Para la construcción de esta imagen Docker se ha utilizado:

- Un Dockerfile multistage para no tener instaladas en la imagen final herramientas que solo son necesarias en la construcción del proyecto.
- Una capa independiente para la descarga de las dependencias del proyecto Maven. Esta capa actúa de caché de dependencias. Mientras no cambie el fichero pom.xml no se descargarán nuevamente.

```
RUN mvn dependency:go-offline
```

El despliegue de la base de datos se realizará mediante el esquema YAML para Kubernetes del fichero `front-web.yaml`. Se puede ver el contenido del fichero en el Anexo 4: front-web.yaml.

Base de Datos

Según los requisitos

- La base de datos debe utilizar un PersistentVolume en el clúster de Kubernetes para guardar los datos.
- La base de datos utilizada por el proyecto es "MySQL", de modo que utilizaremos el YAML oficial de MySQL.

Se ha utilizado la implementación oficial de MySQL para Kubernetes[1] así como la utilización de *Secret*[2] para su implementación.

El despliegue de la base de datos de realizará mediante el esquema YAML para Kubernetes de los ficheros `mysql.yaml y secret.yaml`. Se puede ver el contenido de estos ficheros en el Anexo 1: mysql.yaml y Anexo 2: secret.yaml.

[1] https://kubernetes.io/docs/tasks/run-application/run-single-instance-stateful-application/#deploy-mysql
[2] https://kubernetes.io/docs/concepts/configuration/secret/

Anexos

Anexo 1: mysql.yaml

```yaml
---
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: abrahamsc-mysql-pvc
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 2Gi
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: abrahamsc-mysql-deployment
spec:
  selector:
    matchLabels:
      app: abrahamsc-mysql
  strategy:
    type: Recreate
  template:
    metadata:
      labels:
        app: abrahamsc-mysql
    spec:
      containers:
      - image: mysql:5.6
        name: abrahamsc-mysql-container
        env:
        - name: MYSQL_ROOT_PASSWORD
          valueFrom:
            secretKeyRef:
              name: abrahamsc-secrets
              key: mysql-root-password
        - name: MYSQL_DATABASE
          value: test
        ports:
        - containerPort: 3306
          name: abrahamsc-mysql
        volumeMounts:
        - name: abrahamsc-mysql-persistent-storage
          mountPath: /var/lib/mysql
      volumes:
      - name: abrahamsc-mysql-persistent-storage
        persistentVolumeClaim:
          claimName: abrahamsc-mysql-pvc
---
apiVersion: v1
kind: Service
metadata:
  name: abrahamsc-mysql-service
spec:
  type: ClusterIP
  ports:
  - port: 3306
  selector:
    app: abrahamsc-mysql
```

Anexo 2: secret.yaml

```yaml
---
apiVersion: v1
kind: Secret
metadata:
  name: abrahamsc-secrets
type: Opaque
data:
  mysql-root-password: pass2019
```

Anexo 3: servicio-interno.yaml

```yaml
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: abrahamsc-internal-deployment
  labels:
    app: abrahamsc-internal
spec:
  selector:
    matchLabels:
      app: abrahamsc-internal
  replicas: 2
  strategy:
    type: Recreate
  template:
    metadata:
      name: abrahamsc-internal
      labels:
        app: abrahamsc-internal
    spec:
      containers:
      - name: abrahamsc-internal-container
        image: abrin09/servicio-interno:1.0
---
apiVersion: v1
kind: Service
metadata:
  name: abrahamsc-internal-service
  labels:
    app: abrahamsc-internal
spec:
  ports:
    - port: 8070
      targetPort: 8070
      protocol: TCP
      name: abrahamsc-internal
  selector:
    app: abrahamsc-internal
  type: ClusterIP
```

Anexo 4: front-web.yaml

```yaml
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: abrahamsc-front-web-deployment
  labels:
    app: abrahamsc-front-web
spec:
  selector:
    matchLabels:
      app: abrahamsc-front-web
  replicas: 2
  strategy:
    type: Recreate
  template:
    metadata:
      name: abrahamsc-front-web
      labels:
        app: abrahamsc-front-web
    spec:
      containers:
      - name: abrahamsc-front-web-container
        image: abrin09/front-web:1.0
        env:
          - name: MYSQL_ROOT_PASSWORD
            valueFrom:
              secretKeyRef:
                name: abrahamsc-secrets
                key: mysql-root-password
          - name: MYSQL_DATABASE
            value: test
---
apiVersion: v1
kind: Service
metadata:
  name: abrahamsc-front-web-service
  labels:
    app: abrahamsc-front-web
spec:
  ports:
    - port: 443
      targetPort: 8443
      protocol: TCP
      name: abrahamsc-front-web
  selector:
    app: abrahamsc-front-web
  type: LoadBalancer
```

Anexo 5: Dockerfile del Servicio Interno

```
FROM maven:3.6.0-jdk-8-alpine as builder
COPY ./pom.xml /project/pom.xml
COPY ./src /project/src
WORKDIR /project
RUN mvn dependency:go-offline
RUN mvn -DskipTests=true package

FROM openjdk:8-jre
RUN apt-get update && apt-get install -y netcat
RUN mkdir /internalService
COPY --from=builder /project/target/*.jar /internalService
WORKDIR /internalService
EXPOSE 8070
CMD java -jar ServicioInterno-0.0.1-SNAPSHOT.jar
```

Anexo 6: Dockerfile del Front Web

```
FROM maven:3.6.0-jdk-8-alpine as builder
COPY ./pom.xml /project/pom.xml
COPY ./src /project/src
WORKDIR /project
RUN mvn dependency:go-offline
RUN mvn -DskipTests=true package

FROM openjdk:8-jre
RUN apt-get update && apt-get install -y netcat
RUN mkdir /frontService
COPY --from=builder /project/target/*.jar /frontService
WORKDIR /frontService
EXPOSE 8080
CMD java -jar ejem1-0.0.1-SNAPSHOT.jar
```