Memoria del Trabajo Final:

Parte de CN

Estudiante: Pablo Albert Puchol

Grado: Máster en Tecnologías Web, Computación en la Nube y Aplicaciones Móviles

Asignatura: Computación en la nube

Fecha: 31/ 05/ 2024

Índice

Introducción

En esta memoria, se presentará un resumen detallado de la arquitectura de software empleada en los servicios del proyecto, así como la documentación de las APIs implementadas. Las APIs han sido desarrolladas utilizando el ecosistema de Spring Boot, una elección que facilita la creación de aplicaciones Java robustas y escalables.

El proyecto se ha diseñado con perfiles de Spring Boot, permitiendo su ejecución independiente (standalone) sin la necesidad de un entorno Kubernetes a partir del uso de perfiles (“local” para realizarlo de forma local y “cloud” para kubernetes). Estos perfiles están documentados en la memoria, proporcionando instrucciones claras sobre cómo lanzar y configurar la aplicación en diferentes entornos.

La estructura de la memoria se divide en varios apartados:

1. **Configuración y Lanzamiento del Proyecto**: En esta sección, se describen los pasos necesarios para iniciar la parte del proyecto correspondiente a esta asignatura, detallando cómo configurar y lanzar la aplicación de manera independiente.
2. **Estructuración del Proyecto:** En esta parte, se mostrará la estructura que se creó con Spring y la causa por la que se pensó de esta forma.
3. **Documentación de APIs**: Casi toda la información específica sobre las APIs desarrolladas, así como los repositorios asociados, se encuentra recopilada en la carpeta "PDFs_rutas". Esta documentación incluye detalles sobre las rutas, métodos HTTP, modelos de datos y ejemplos de solicitudes y respuestas, proporcionando una guía completa para desarrolladores y usuarios del sistema.

4. **Traslado del proyecto a Kubernetes**: Se explicará la forma en la que se pasaron los proyectos de Spring a la nube a partir de ficheros.yml y el orden de los comandos para que la aplicación funcione correctamente.
5. **Conclusiones**

Configuración y Lanzamiento del Proyecto

Primero, para poner en funcionamiento las bases de datos, debes ir a la carpeta "docker" y ejecutar el comando `docker-compose up -d`. Esto creará las tablas e insertará los datos en las tablas o colecciones de las bases de datos.

Después, para ver el despliegue de las aplicaciones de Spring, hay dos formas principales de hacerlo:

- **Importar los proyectos en Spring Tool Suite 4**:
  - Importa las carpetas de los proyectos usando la opción "Open Projects from File System".
  - Ejecuta todas las aplicaciones y verifica en el "Boot Dashboard" el puerto al que están conectadas.
  - Si las aplicaciones no funcionan, realiza un "Maven Update" o especifica el perfil utilizado en las aplicaciones. Para esto, haz clic derecho en la aplicación en el "Dashboard", selecciona "Open Config" y luego elige el perfil "local" en la sección de perfiles.
- **Usar la línea de comandos**:
  - En cada una de las carpetas de los proyectos abre una consola.
  - Ejecuta el comando `mvn clean package spring-boot:repackage -DskipTests=true` para crear un archivo .jar dentro de la carpeta "target".
  - Luego, ejecuta el comando `java -jar ./target/yeagapalpu-0.0.1-SNAPSHOT.jar --spring.profiles.active=local` para iniciar la aplicación con el perfil local.

Estructuración del Proyecto

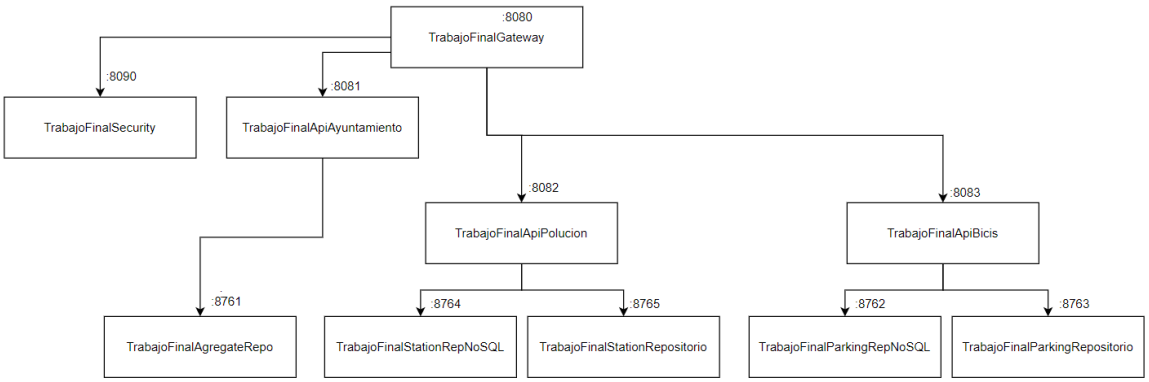

Imagen 1. Estructura del proyecto durante el despliegue local

Gateway:

- **Función principal:** Punto de entrada único para todas las solicitudes externas a la aplicación.
- **Responsabilidades:**
  - **Autenticación:** Verificar la identidad de los usuarios que intentan acceder a la aplicación.
  - **Autorización:** Determinar si los usuarios autenticados tienen los permisos necesarios para acceder a los recursos específicos solicitados.
  - **Enrutamiento:** Dirigir las solicitudes a los microservicios de API correspondientes en función del punto de acceso y la información de la solicitud.
- **Beneficios:**
  - **Centralización de la seguridad:** Simplifica la gestión de la seguridad al tener un único punto de control para la autenticación y autorización.
  - **Mejora la escalabilidad:** Permite escalar horizontalmente el gateway para manejar un mayor volumen de tráfico.
  - **Protección contra ataques:** Actúa como primera línea de defensa contra ataques maliciosos como ataques de inyección de código o denegación de servicio.

2. TrabajoFinal Security:

- **Función principal:** Microservicio dedicado a la autenticación y autorización de usuarios.
- **Responsabilidades:**
  - **Generación de tokens de acceso:** Emitir tokens de acceso únicos y válidos para los usuarios autenticados.
  - **Validación de tokens de acceso:** Verificar la validez y autenticidad de los tokens de acceso proporcionados en las solicitudes.

    - **Gestión de usuarios y roles:** Almacenar y administrar información sobre usuarios, roles y permisos de acceso.
- **Beneficios:**
    - **Mejora la seguridad:** Aísla la lógica de autenticación y autorización del resto de la aplicación, reduciendo el riesgo de vulnerabilidades.
    - **Flexibilidad:** Permite implementar diferentes mecanismos de autenticación y autorización, como autenticación básica, OAuth 2.0 o autenticación basada en tokens JWT.
    - **Escalabilidad:** Se puede escalar horizontalmente el microservicio de seguridad para manejar un gran número de usuarios y solicitudes.

3. Microservicios de API:

- **Función principal:** Proporcionar funcionalidades específicas a la aplicación.
- **Responsabilidades:**
    - Implementar la lógica de negocio específica para cada microservicio.
    - Procesar las solicitudes recibidas del gateway y devolver las respuestas correspondientes.
    - Interactuar con los repositorios de datos para acceder y modificar la información persistente.
- **Beneficios:**
    - **Modularidad:** Promueve la modularidad del código, facilitando el desarrollo, mantenimiento y escalado de la aplicación.
    - **Reutilización de código:** Permite reutilizar componentes y código entre diferentes microservicios.
    - **Despliegue independiente:** Cada microservicio se puede implementar y escalar de forma independiente, lo que aumenta la flexibilidad y la disponibilidad de la aplicación.

4. Repositorios de datos:

- **Función principal:** Gestionar los datos persistentes de la aplicación.
- **Tipos de repositorios:**
    - **Repositorios SQL:** Almacenan datos estructurados en bases de datos relacionales.
    - **Repositorios NoSQL:** Almacenan datos no estructurados o semiestructurados en bases de datos NoSQL como MongoDB o Cassandra.
- **Responsabilidades:**

- Proporcionar una interfaz para que los microservicios de API accedan, modifiquen y consulten los datos.
- Garantizar la integridad, consistencia y disponibilidad de los datos.**En resumen:**

Cabe destacar que la API de seguridad se utiliza únicamente en el Gateway, ya que en Kubernetes solo este proyecto se conectará al exterior. La API actúa como un filtro para permitir el acceso a ciertas peticiones públicas sin necesidad de pasar por el proceso completo de autenticación y autorización. Esto se logra mediante la implementación de reglas de seguridad en el Gateway, que identifican las solicitudes que no requieren validación de tokens de acceso.

Ejemplo:

- Imaginemos que la aplicación tiene un endpoint público que proporciona información general sobre la aplicación, como la obtención de la información general de las estaciones y aparcamientos. Estos endpoints no requieren autenticación de usuario, por lo que se puede configurar en el Gateway para que pase directamente al microservicio correspondiente sin necesidad de involucrar al API de seguridad.

Y en el caso de diferenciar la autentificación de un token de otro, se pasa un header a la Apis donde se identifica el nombre de usuario del token el cual muestra los permisos de la aplicación o si esta petición la está realizando el usuario al aparcamiento o estación que le correspondía.

Documentación de APIs

Como se comentó anteriormente, la gran mayor parte de la documentación se encuentra en la carpeta “PDFs_rutas”, aunque hay 2 proyectos que no hemos querido incluir dentro de estos PDF creados con Swagger.

TrabajoFinalGateway

Overview

Simple aplicación que controla el flujo de las peticiones y que controla que ruta han de pasar por el filtro de seguridad y cuáles no.

Sobre este se ha creado un vector de rutas que descarta al pasar por el filtro de Seguridad y que se envían directamente a las Apis pertinentes.

```
auth.excluded-routes=
/aparcamientos/aparcamientos,/aparcamientos/aparcamiento/*/status,/estaciones/estaciones,/estaciones/estacion/*/status,/ayuntamiento/aparcamientoCercano,/ayuntamiento/aggregatedData
```

TrabajoFinalSecurity

Overview

Simple aplicación que hace de filtro de autentificación sobre las rutas que se le pasen.

Paths

GET /auth /authorize Servicio para validar el token del usuario

Servicio para validar el token del usuario

Responses

Code	Description	Links
200	Token validado *Content* *application/json*	No links
401	No estás autorizado	No links

POST /auth /authenticate Servicio para obtener el token del usuario

Servicio para obtener el token del usuario

Responses

Code	Description	Links
200	Identificación satisfactoria *Content* *application/json*	No links
401	Identificación incorrecta	No links

POST /auth /refresh Servicio para refrescar el token del usuario

Servicio para refrescar el token del usuario pasándole el token por el header

Responses

Code	Description	Links
200	Token refrescado satisfactoriamente *Content* *application/json*	No links
401	Identificación incorrecta	No links

Components (Schemas)

AuthenticationRequest

Properties

Name	Description	Schema
username	Es el username con el que se identifiza el cliente en la aplicación (Solo se permite ADMIN, SERVICIO,PN o SN)	string
password	No es de gran importancia para la seguridad, ni se persiste	string

Traslado del proyecto a Kubernetes

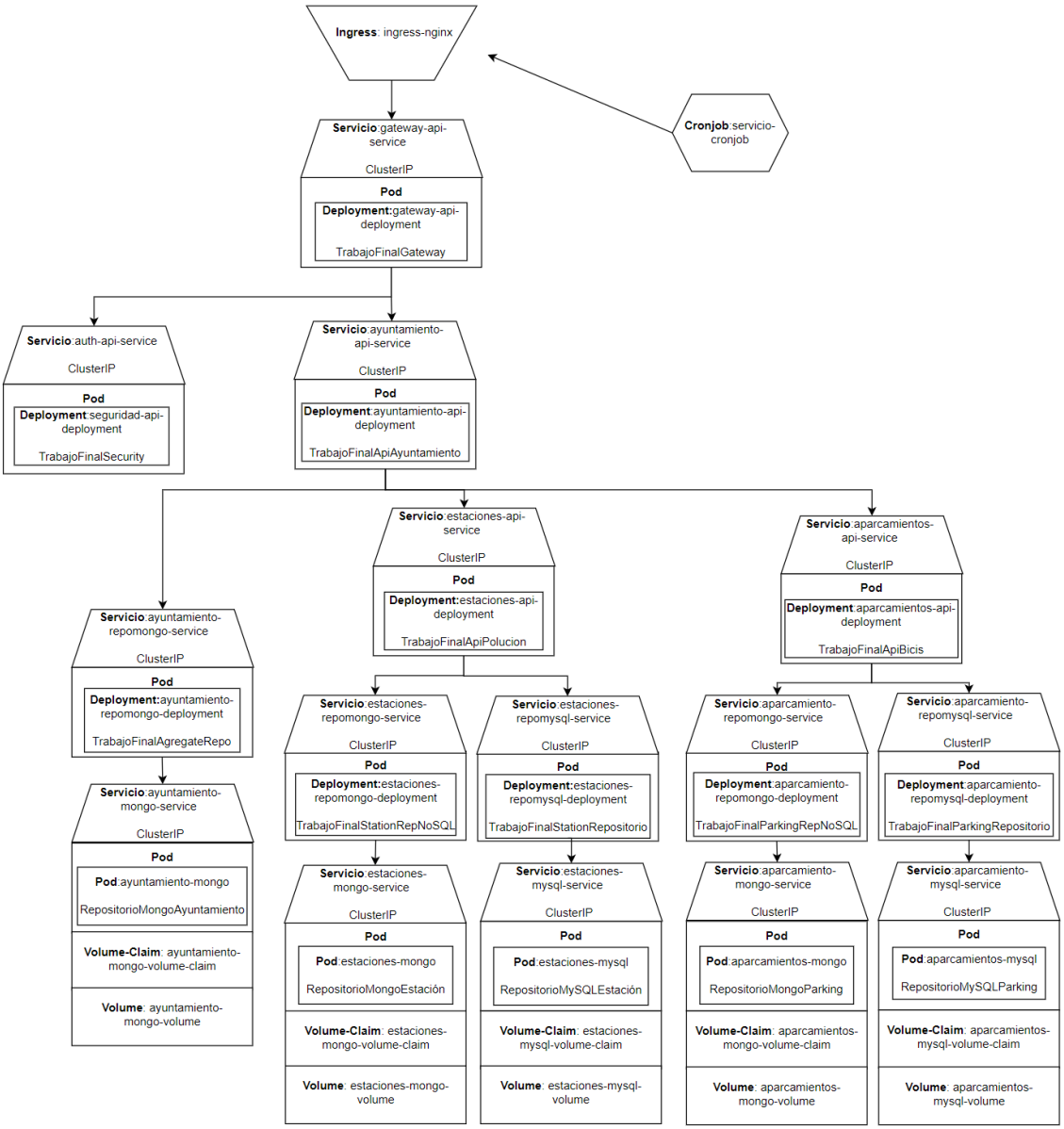

Imagen 2. Estructura del proyecto durante el despliegue en Kubernetes

En esta parte, explicaré los cinco tipos de conjuntos de recursos que hemos utilizado para este proyecto.

- **Estructuras necesarias para bases de datos:**
  Todas las bases de datos dentro de nuestro proyecto necesitan un "Pod" diferente para cada una, donde se utilice una imagen de Docker con MySQL (en nuestro caso, "mysql/mysql-server:8.0.32") o MongoDB (en nuestro caso, "mongo:4.4"). Esto permite la utilización de estas bases de datos. Para inicializar estos datos, hemos utilizado "ConfigMaps" que configuran las tablas o colecciones al iniciar el archivo .yml principal.

Además, para mantener la persistencia de datos en cada una, hemos creado un "PersistentVolume" que almacena hasta 1 GB y un "PersistentVolumeClaim" que permite reclamar ese volumen a nuestro "Pod" de base de datos. Finalmente, para que cualquier "Pod" de nuestro nodo se pueda conectar a las bases de datos, se crearon servicios ClusterIP que se conectan al puerto 27017 en MongoDB y al puerto 3306 en MySQL.

- **Estructuras necesarias para los proyectos para gestionar los repositorios:**
  Estos también están dentro de un "Pod" al cual se ha formado al desplegar un "Deployment" al que se le aplicó una imagen de Docker con nuestro proyecto.jar correspondiente y que está usando el perfil de "cloud" con "SPRING_PROFILES_ACTIVE" para que funcione sobre el puerto 80. Finalmente, para que cualquier "Pod" de nuestro nodo se pueda conectar a los repositorios, se crearon servicios ClusterIP que se conectan a ese puerto 80.

- **Estructuras necesarias para los proyectos para gestionar las APIs:**
  Estos presentan el mismo despliegue que los repositorios, pero al usar el perfil de "cloud", se muestra que estos dependen de cada vez menos puertos para poder ejecutar sus servicios. Al igual que en el punto anterior, sus "Pods" se encuentran conectados con un servicio ClusterIP para poder recibir peticiones en el puerto 80.

- **Ingress:**
  Si en un futuro quisiéramos elegir cuáles rutas desactivar, con este componente sería tan fácil como desviarlas a otra ruta distinta o incluso bloquearlas. Por ello, todo este despliegue se hizo pensando en este componente de Kubernetes. En este proyecto, permite todo el tráfico de peticiones, pero solo reenvía todo a la API de Gateway, donde se verificará si se tiene permiso para la petición o no.

- **CronJob para el cliente "Service":**
  En este componente específico, se hace uso de un CronJob que se ejecuta por defecto cada minuto para hacer un curl de curl -H "Authorization: $AUTH_TOKEN" http://192.168.56.3:$INGRESS_PORT/ayuntamiento/aggregateData para generar automáticamente recursos agregados en la API del ayuntamiento.

Y toda esta explicación conforma la estructuración de nuestro proyecto para kubernetes.

Por la parte del despliegue de este proyecto en la nube, deberíamos seguir estos pasos:

1. **Crear y desplegar las imagenes de los proyectos de spring:** Esto se consigue al generar primero los archivos.jar con “mvn clean package spring-boot:repackage -DskipTests=true”. Luego, generamos un archivo Dockerfile con la siguiente estructura:
   FROM openjdk:17
   WORKDIR /app
   COPY yeagapalpu-0.0.1-SNAPSHOT.jar app.jar
   EXPOSE 80
   ENTRYPOINT ["java", "-jar", "app.jar"]
   A continuación, ejecutamos el siguiente comando para crear la imagen de ese proyecto “docker build -t autor/nombre-imagen:latest .” y finalmente solo tendremos que subir esa imagen a la red con “docker push autor/nombre-imagen:latest”.

2. **Crear el namespace del despliegue:** Al principio deberemos de generar el namespace de “trabajo-final” con el siguiente comando “kubectl create namespace trabajo-final”

3. **Ejecutar los pods de bases de datos:** En esta parte debemos de ejecutar los archivos yaml de las bases de datos con “kubectl apply -f nombre-del-componente.yml”. El orden de ejecución debe ser: primero los volúmenes, después el "PersistentVolumeClaim", luego el "ConfigMap" y, finalmente el archivo donde se define el pod.

4. **Ejecutar los servicios:** En esta parte debemos de ejecutar los archivos yaml de la carperta de “servicios” con “kubectl apply -f nombre-del-servicio.yml”. Aquí el orden da igual, ya que los servicios se quedarán a la espera del despliegue del pod del que tengan que conectarse sin causar fallos.

5. **Ejecutar los deployments:** En esta parte debemos de ejecutar los archivos yaml de la carperta de “deployments” con “kubectl apply -f nombre-del-deployment.yml”. Aquí si que es importante ejecutarlo en el orden como se especifica en la imagen 2, desde los repositorios hasta el

api de gateway, para evitar fallas de conexión de las imágenes de nuestros proyectos a ciertos puertos.

6. **Deplegar el ingress y el cronjob:** Como en los apartados anteriores, para poder ejecutar estos componentes te tendrás que desplazar a las carpetas “jobs” y “ingress” donde deberás de ejecutar los comandos “kubectl apply -f ingress-nginx.yml” y “kubectl apply -f servicio-cronjob.yml” para que comience a funcionar el cliente “Servicio” y ya se puedan conectar desde fuera de nuestro nodo.

Y si se quiere comprobar que el sistema funciona correctamente puedes ejecutar los siguientes comandos:

- **Comprobar los volúmenes:** kubectl get pv –n trabajo-final
- **Comprobar los reclamadores de volúmenes:** kubectl get pvc –n trabajo-final
- **Comprobar los servicios:** kubectl get service –n trabajo-final
- **Comprobar los despliegues:** kubectl get deployment –n trabajo-final
- **Comprobar los pods:** kubectl get pod–n trabajo-final
- **Comprobar el ingress:** kubectl get ingress –n trabajo-final
- **Comprobar el cronjob de “Servicio”:** kubectl get cronjob –n trabajo-final
- **Comprobar los trabajos deplegados por el cronjob:** kubectl get job –n trabajo-final

Conclusiones

A lo largo de este proyecto, se ha aprendido a integrar y desplegar tecnologías modernas para crear una solución robusta y escalable utilizando Spring Boot y Kubernetes. La implementación de una arquitectura modular basada en microservicios ha demostrado ser esencial para mejorar la mantenibilidad y seguridad de la aplicación, centralizando la autenticación y autorización en un único punto de control. El despliegue en la nube mediante Kubernetes me ha permitido familiarizarme con la gestión de Pods, Deployments, ConfigMaps y servicios ClusterIP, asegurando un entorno de ejecución altamente disponible y escalable. Además, la automatización mediante CronJobs ha facilitado la actualización periódica de los datos, mostrando la importancia de la automatización en la gestión de aplicaciones en la nube. Este proyecto ha sido fundamental para aplicar y consolidar conocimientos sobre la creación, configuración y despliegue de aplicaciones en la nube, sentando una base

sólida para futuras expansiones y mejoras en términos de rendimiento, seguridad y monitoreo.