Máster en Cloud Apps

Desarrollo y despliegue de aplicaciones en la nube

Curso académico 2021/2022

Trabajo de Fin de Máster

Gestión de expedientes con microservicios y CI/CD en cluster k8s en la nube

Autor: Antonio Gat Alba

Tutor: Micael Gallego Carrillo

"*Gestión de expedientes con microservicios y CI/CD en cluster k8s en la nube*" por *Antonio Gat Alba* tiene licencia Creative Commons Attribution Share-Alike

Contenido

1. Resumen

En el presente documento se plasmará el trabajo realizado durante la realización del TFM del máster cuya finalidad ha sido poner en práctica las siguientes técnicas y tecnologías, adquiridas en el curso:

- Patrones de diseño / arquitectura hexagonal
- DDD / CQRS
- Tecnologías de comunicación: REST, colas.
- Testing unitario, de integración, de componente, end to end.
- CI / CD
- Docker y k8s

La aplicación desarrollada corresponde con un gestor de expedientes que dispone de documentación asociada a estos y que está sujeta a un flujo de tramitación (BPM). Este flujo de tramitación es una herramienta horizontal a una organización en la que pueden integrarse distintas aplicaciones, que pueden tramitar, en un determinado momento, el flujo correspondiente. De esta manera, se puede orquestar un proceso completo en la gestión de un expediente en el que pueden verse involucradas varias aplicaciones o servicios. Funcionalmente, la aplicación dispone de los siguientes casos de uso más importantes:

- Los usuarios pueden autenticarse en la aplicación.
- La aplicación permite la visualización de todos los expedientes creados.
- Los usuarios pueden crear expedientes.
- Los usuarios pueden ver el detalle de un expediente.
- La aplicación permite editar un expediente concreto.
- La aplicación permite tramitar un expediente a su siguiente fase.

Para la elaboración de esta memoria académica, se ha estructurado esta en los siguientes apartados:

- Introducción y objetivos: descripción de las metas a conseguir en el desarrollo del TFM.
- Desarrollo del TFM: se describen las tecnologías utilizadas, así como los métodos y procesos que han sido implementados.
- Conclusiones y trabajos a futuro.
- Bibliografía.
- Anexos.

2. Introducción y objetivos

El objetivo principal del desarrollo de la aplicación para la gestión de expedientes, en adelante, **Filesmanagement**, es la de consolidar los conocimientos adquiridos en algunas tecnologías, patrones y procedimientos realizados durante el máster. Estas tecnologías o patrones utilizados son los siguientes:

- Patrones de diseño: como pueden ser AbstractFactory, Facade, MVC, DTO, DAO, SAGA, etc.
- Arquitectura hexagonal: enfoque de arquitectura basada en el aislamiento de la capa de negocio de cualquier tecnología o framework.
- DDD: Domain Driven Design.
- REST: protocolo de comunicación utilizado para la interconexión de los microservicios implementados.
- CQRS: patrón por el que se produce una segregación de la información cuando se actualiza y cuando esta es consultada.
- Docker[1]: tecnología utilizada para la creación de aplicaciones que se distribuyen dentro de un contenedor. En el desarrollo del TFM se utiliza Docker Hub como almacén de imágenes generadas.

- Docker-compose: herramienta que permite desplegar multitud de contenedores interconectados unos con otros, en un servidor o equipo local. Ideal para el desarrollo de un microservicio que tiene cierta dependencia con otros.
- Kubernetes[2]: el despliegue de la aplicación y, por tanto, de los microservicios que la componen son desplegados en Kubernetes en la nube mediante la plataforma Okteto.
- Helm[3]: componente que permite el empaquetado de aplicaciones mediante Docker.
- Mysql[4]: base de datos utilizada para todos aquellos servicios que persistan datos de forma relacional.
- Mongodb[5]: base de datos NoSql utilizada para implementar lecturas rápidas, de forma que se optimizan los tiempos de respuesta.
- Eventuate[6]: framework utilizado para la implementación del patrón SAGA.
- OAuth2: servicio de autenticación y autorización basado en tokens JWT.
- Spring Cloud Gateway[7]: componente utilizado para implementar el servicio principal de entrada a la aplicación
- Flyway[8]: framework que permite el versionado de las bases de datos utilizadas, ejecutando los scripts SQL correspondientes.
- Git: repositorio de código seleccionado. Se ha utilizado, además, gitflow como estructura de repositorio.
- Jenkins[9]: servicio para automatización de tareas relacionadas con CI/CD, permitiendo, entre otros, el despliegue automático de aplicaciones.
- Artifactory[10]: repositorio de artefactos y dependencias.
- PMD[11]: analizador estático de código fuente.
- Checkstyle[12]: control del estilo de codificación.
- Jacoco-plugin: plugin de Maven para el cálculo de cobertura de código.

Además del uso de estas tecnologías, los objetivos de este TFM también son:

- Implementación de la aplicación mediante una arquitectura de **microservicios**.
- Que la aplicación sea desplegable en cualquier clúster **k8s**.
- Que disponga de una batería de tests y documentación.
- Automatización del proceso de desarrollo y despliegue a través de herramientas de CI/CD.
- Integración con bases de datos relacionales (mysql) y no relacionales (mongodb).
- Integración con colas de eventos mediante Kafka[17].

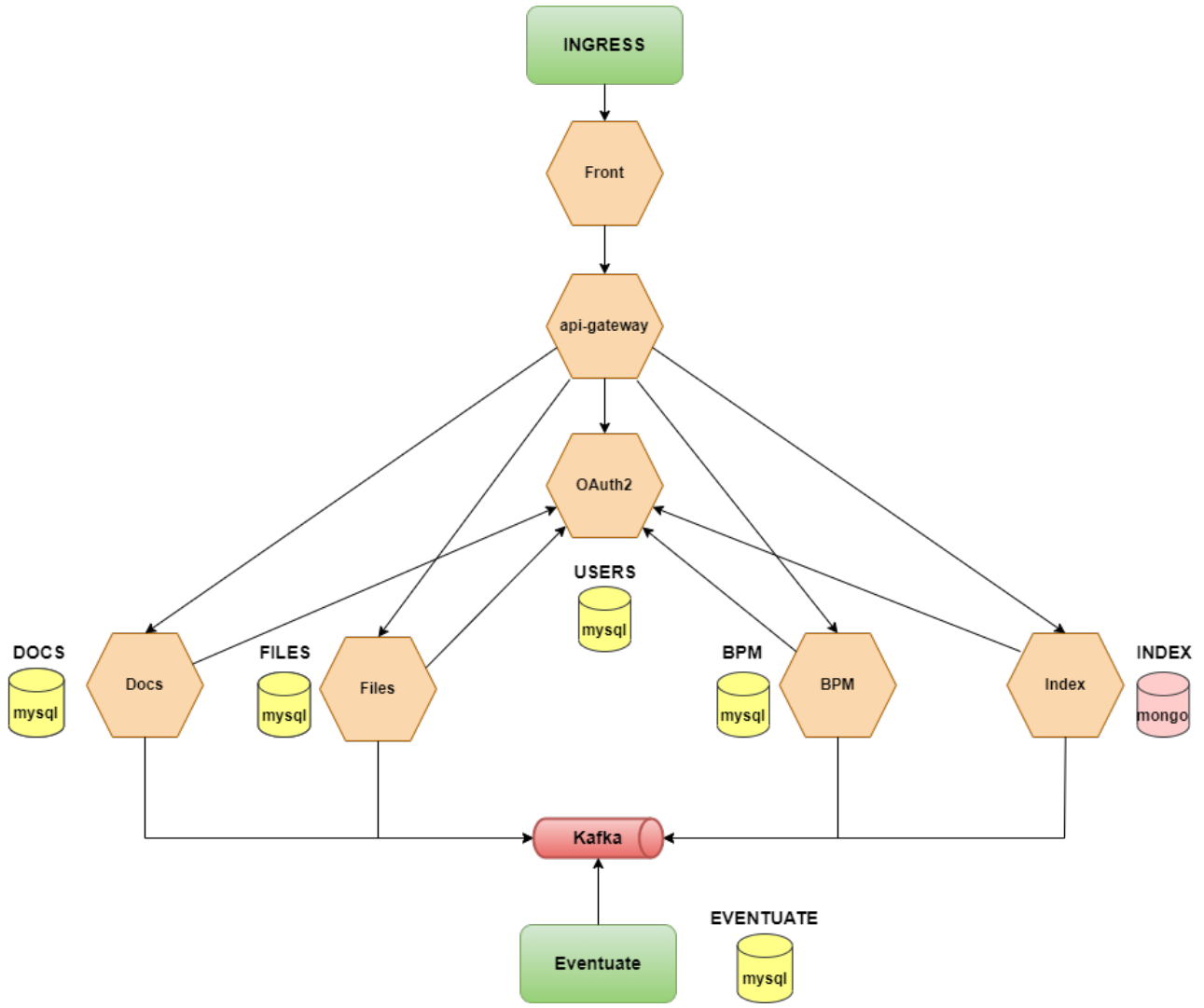
3. Desarrollo del TFM

La aplicación descrita con anterioridad ha sido implementada usando una arquitectura basada en microservicios y un diseño basado en eventos. Son 7 los microservicios desarrollados para la aplicación Filemanagement que se detallarán en los siguientes apartados, pero que se mencionan a continuación a modo de resumen:

- **OAuth2**: microservicio horizontal al resto cuya finalidad es la autenticación y autorización mediante tokens.
- **Docs**: microservicio que gestiona documentación, generando metadatos sobre estos.
- **BPM**: este microservicio contiene la información de flujos y tramitación.
- **Files**: microservicio principal que contiene la información de expedientes y referencias a su documentación y a su tramitación.
- **Index**: microservicio especializado en optimización de consultas, formando parte del patrón CQRS. Será llamado para consultas complejas y pesadas como son las búsquedas mediante filtros.
- **Gateway**: acceso unificado a la aplicación (patrón facade), que se encarga de redirigir hacia los microservicios pertinentes y de componer, en algunos casos mediante tecnologías reactivas, objetos cuya información está distribuida.
- **Front**: microservicio que contiene el frontal o interfaz de usuario.

A. Microservicios

La siguiente imagen muestra un diagrama que contiene tanto los microservicios, como los servicios necesarios, así como la interacción entre ellos. Todos estos componentes serán desarrollados en su apartado correspondiente.



1. Características comunes

De forma genérica, se puede indicar que todos estos microservicios implementados tienen las siguientes características comunes:

- Java 17: versión utilizada a lo largo del máster y aplicada en este TFM.
- Maven[12]: utilizada para la compilación, generación de artefactos, testing, checking de cobertura, de estilo, etc.
- Dockerización realizada mediante JIB
- Git: utilización de repositorios individuales en Github
- Flyway: en aquellos microservicios que utilicen acceso a base de datos.
- Spring-boot 2.6.0

- Chekstyle: control del estilo de codificación.
- PMD: analizador estático de código fuente.
- Manifiestos k8s: descriptores para su despliegue en Kubernetes.
- Manifiestos Helm: descriptores Helm para su despliegue en Kubernetes.

También y casi de forma genérica, casi todos los microservicios están implementados utilizando una arquitectura hexagonal y en muchos de ellos DDD. A continuación, se detalla la información de cada uno de los microservicios desarrollados.

2. Microservicio OAuth2

Este microservicio es el encargado de la autenticación en toda la aplicación. El resto de los microservicios se conectan para la validación de tokens provenientes de sus respectivos API REST.
El microservicio de OAuth2 está implementado siguiendo una **arquitectura hexagonal** y contiene unos controladores que publican un API REST con los siguientes endpoints:

- GET /api/users/{username} devuelve la información de un usuario a partir de su identificador.

A su vez, este microservicio cuenta con un esquema propio en una base de datos **MySql**, denominado “USERS” que contiene una sola tabla con la información de los usuarios de la aplicación.
No tiene dependencia con ningún otro microservicio o servicio (a excepción de la base de datos de MySql), por lo que puede considerarse como un servicio horizontal en la aplicación.

3. Microservicio Docs

Este microservicio está dedicado a la gestión de documentación, obteniendo metainformación de los documentos obtenidos, así como el almacenamiento en soporte físico. Está implementado mediante **arquitectura hexagonal** y publica los siguientes endpoints a través de su API REST:

- GET /api/docs/{documentCode} devuelve toda la información de un documento a partir de su identificador.

Cuenta con un esquema propio en una base de datos **MySql** denominada “DOCS” que contiene una sola tabla con la metainformación de los documentos gestionados.

Implementa, además, métodos que forma parte de **SAGA**, concretamente, para el siguiente caso de uso:

- Creación de expediente: la creación de un expediente conlleva la obligatoriedad de adjuntar un documento. En este caso, el microservicio Docs, generará el documento correspondiente o lo eliminará en caso de rollback en la SAGA.

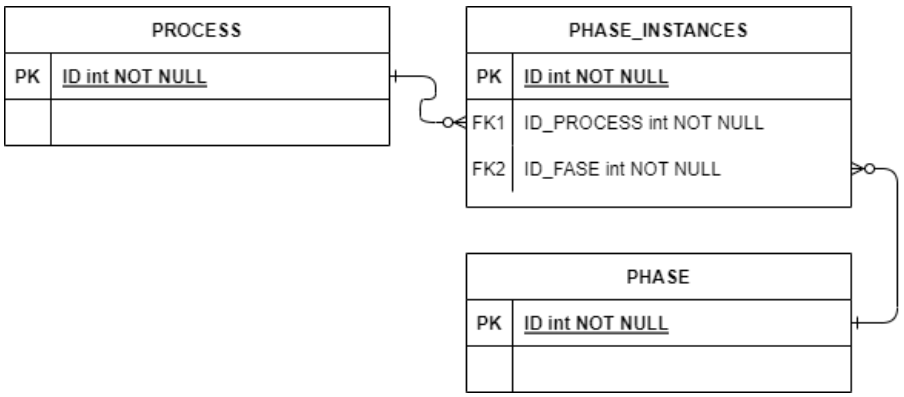
Para que este microservicio forme parte de una SAGA, este microservicio se integra con **Eventuate** (se detallará acerca de este servicio en posteriores apartados), que es un servicio de orquestación de SAGAs. Internamente, Eventuate publica mensajes en una cola de Kafka y también consume mensajes de esta, realizando las llamadas pertinentes en los microservicios afectados. Por tanto, este microservicio, a través del framework Eventuate, consume y produce mensajes en este broker de mensajería.

4. Microservicio BPM

El microservicio de BPM es un tramitador de fases implementado mediante una **arquitectura hexagonal**. Su principal cometido es la de gestionar procesos y sus estados, informando de cada cambio en una cola de mensajes de Kafka. Este comportamiento permite el uso de **CQRS**, que servirá para alimentar otra base de datos NoSql (mongodb) que explota el microservicio de “Index”, tal y como se tratará en apartados posteriores. A continuación, se muestran los endpoints que publica:

- POST /api/bpm: dado un código de proceso y el código de fase a la que tramitar, este endpoint tramita el expediente hacia esa fase.
- GET /api/bpm/{processCode}: devuelve el detalle de un proceso dado su identificador.
- GET /api/bpm/availablePhases/{phaseCode}: devuelve las fases disponibles a partir de una determinada.

Este microservicio, BPM, está desarrollado mediante **DDD**, con la creación de un agregado denominado “ProcessAggregate” que contiene la lógica de este objeto de negocio. De igual manera que ocurre con los anteriores microservicios descritos, BPM dispone de un esquema propio de base de datos MySql denominado “BPM” que contiene tres tablas:



- PROCESS: tabla que mantiene la información de las instancias de cada uno de los procesos creados.
- PHASES: tabla que contiene la información maestra de los tipos de fases.
- PHASE_INSTANCES: tabla que contiene la información relativa a las instancias de cada una de las fases por las que va pasando un proceso determinado. Se informa en esta tabla acerca de fechas de creación y actualización, así como de los usuarios creadores y actualizadores, entre otros.

De la misma manera que ocurría con el microservicio “Docs”, BPM forma parte de la **SAGA** generada en la creación de un expediente. Por tanto, se integra con el framework Eventuate que realizará las siguientes operaciones:

- Creación de un proceso de BPM: se creará un nuevo proceso al recibir el evento correspondiente.
- Eliminación de un proceso BPM: si la SAGA realiza un rollback, entonces el proceso previamente creado, debe ser eliminado.

Además, los cambios o modificaciones en este microservicio son lanzados a una cola de mensajería de Kafka cuyo objetivo es **CQRS**. En esta cola estarán suscritos los microservicios de “Index” y “Files” como se podrá detallar posteriormente.

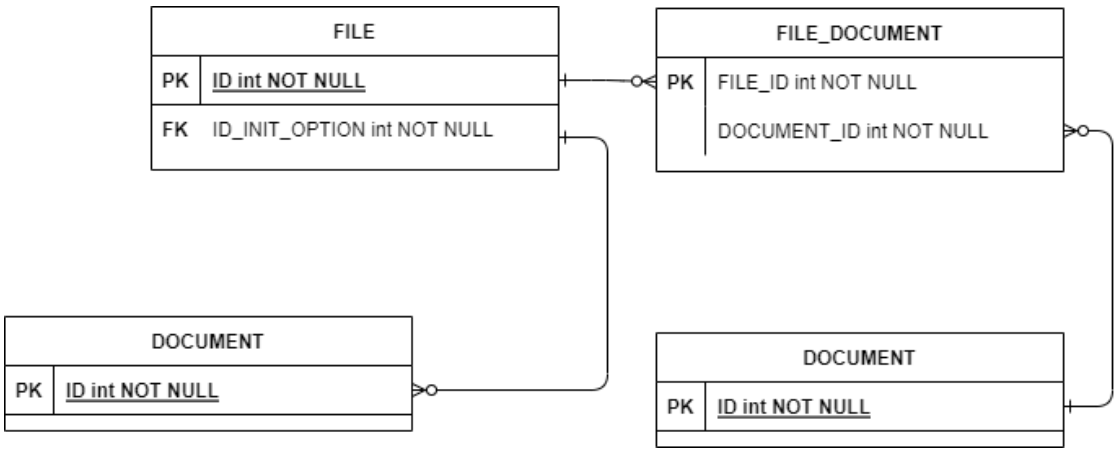
5. Microservicio Files

Este microservicio es el componente **principal** de la aplicación Filemanagement, que contiene la información relativa a los expedientes, así como las referencias tanto a su documentación, como a su proceso de tramitación gestionado por BPM. Como ocurre con los anteriores microservicios, este está implementado siguiendo una arquitectura hexagonal y publica un API REST con los siguientes endpoints:

- POST /api/files: a través de este endpoint se envía el documento y los datos de inicio correspondientes a la creación del expediente.
- PUT /api/files: actualiza un expediente. Se pasa en el body los datos a actualizar.
- GET /api/files/{filecode}: devuelve el detalle de un expediente.
- GET /api/files/initialoptions/all: devuelve los datos maestros de las opciones de inicio de un expediente.

Además, los cambios ocurridos en este microservicio son publicados en una cola de mensajería de Kafka para permitir CQRS. El suscriptor principal de estos eventos es el microservicio Index, que permite una mejora en el rendimiento de consultas. Este microservicio, Files, está desarrollado mediante **DDD**, con la creación de un agregado denominado "FileAggregate" que contiene la lógica de este objeto de negocio. Tiene asociado un esquema de base de datos de MySql compuesto por 4 tablas:

- FILE: contiene la información principal de cada expediente. En esta tabla se guarda la referencia al identificador de proceso generado en BPM.
- DOCUMENT: información relativa a los identificadores de documentos generados en Docs.
- FILE_DOCUMENT: tabla para asociar N documentos a 1 expediente.
- INIT_OPTION: maestra con información de las opciones de inicio de un expediente.



Desde este microservicio parte la generación de la **SAGA** en la creación de un expediente. Tal y como se ha comentado con anterioridad, la creación de un expediente involucra la creación de un proceso en BPM y la generación de un documento asociado.

Mediante Eventuate se consigue la orquestación de la transacción necesaria entre estos microservicios, de tal manera que, si alguno de los microservicios no puede generar la información necesaria, entonces se realizará un rollback en el resto de los microservicios involucrados.

Por otro lado, Files genera eventos en Kafka en los comandos de actualización y creación de expedientes, formando parte del patrón **CQRS**. Estas modificaciones son consumidas por el microservicio Index, que transforma los eventos en registros de una colección de Mongodb, para la optimización en las operaciones de lectura.

6. **Microservicio Gateway**

Se ha desarrollado un API Gateway para centralizar en un punto el acceso a toda la información disponible. Algunos de los endpoints realizan un simple “pass through” hacia otros endpoints de los microservicios disponibles, pero otros endpoints componen un DTO de salida con los datos proporcionados por el resto de los microservicios mediante la utilización de **programación reactiva**. De esta forma se puede ir componiendo la salida de forma asíncrona y paralela a medida que la información solicitada a los microservicios vaya entrando. Para la implementación de este microservicio se ha realizado mediante **Spring Cloud Gateway**, debido a todas las features que incluye y su fácil utilización.

7. **Microservicio Front**

Se trata del frontal de la aplicación Filemanagement. Es un microservicio desarrollado mediante **Spring MVC** y Thymeleaf[13] (framework de generación de componentes). Se basa en la implementación de los controladores, el modelo y las vistas para la determinación de la interfaz de usuario. Todas las acciones de usuario son canalizadas hacia el “Gateway” y hacia el microservicio de OAuth2 para la autenticación.

8. **Microservicio Index**

Este microservicio, implementado con **arquitectura hexagonal**, es el responsable de la ejecución de las consultas sensibles en cuanto a rendimiento se refiere, formando parte del patrón **CQRS** descrito con anterioridad. Por tanto, este microservicio consume eventos de creación y modificación que son proporcionados por el resto de los microservicios a través de Kafka. Los datos que provienen de los eventos consumidos son persistidos en una base de datos **Mongodb** cuya estructura basada en “collections” y “document” permiten, por su naturaleza NoSql y por tanto no relacional, lecturas de datos mucho más rápidas que en bases de datos relacionales. Este microservicio es utilizado para la generación del listado de expedientes y su filtrado.

También publica un API REST con los siguientes endpoints:

- GET /api/index: devuelve un listado con todos los expedientes.

B. Infraestructura

En este apartado se tratará acerca de los servicios y de la infraestructura necesaria para el despliegue de la aplicación Filemanagement. Este proyecto dispone de su propio repositorio de Github y en él se encuentran ubicados los distintos manifiestos y ficheros de configuración de las siguientes tecnologías:

1. Docker

Esta es la tecnología utilizada para "contenerizar" los distintos microservicios y servicios. La generación de las distintas imágenes de los microservicios es llevada a cabo mediante **JIB**[14], que es una herramienta opensource de Google que dispone de un plugin de Maven. La ejecución de la fase "build" de este plugin permite la creación y publicación de la imagen en Dockerhub.

Por otro lado, en la carpeta /docker del proyecto de infraestructura, se encuentran manifiestos para el despliegue de toda la aplicación mediante **docker-compose**:

- docker-compose.yml: manifiesto para el despliegue de todos los microservicios mediante docker-compose. Permite el desarrollo en un entorno local cuando hay que integrarse con otros microservicios.
- docker-compose-services.yml: crea y levanta los contenedores relativos a los servicios de:
  - mysql
  - mongodb
  - kafka
  - zookeeper
  - eventuate

  Permiten, como en el anterior caso, el desarrollo en local y la integración con todos los microservicios.
- docker-compose-ci.yml: manifiesto de docker-compose para la creación del contenedor de Jenkins.

2. **Mysql**

Base de datos relacional utilizada para albergar los esquemas de datos de los microservicios implementados. Se despliega a partir de una imagen Docker en todos los entornos. Cuando se despliega en Kubernetes, se le asigna un “volume” y se vincula con un “secret” para el establecimiento de las credenciales. El manifiesto k8s de este servicio se encuentra dentro del proyecto de “infraestructure” en la siguiente ubicación:

- /infraestructura/k8s/svc/mysql.yml: contiene la descripción del “service”, del “deployment” y del “PersistenceVolumeClaim”.

Los distintos microservicios desarrollados cuentan con **Flyway** que mantiene actualizado en todo momento cada uno de los esquemas correspondientes.

3. **Mongodb**

Base de datos NoSql seleccionada para la mejora del rendimiento en las consultas realizadas. Es utilizada, exclusivamente, por el microservicio de Index, que forma parte del patrón CQRS en la realización de consultas. Se despliega en todos los entornos mediante una imagen Docker, y al igual que ocurre con Mysql, tiene asignado un “volume” y un “secret” cuando es desplegado en Kubernetes. El manifiesto k8s de este servicio se encuentra dentro del proyecto de “infraestructure” en la siguiente ubicación:

- /infraestructura/k8s/svc/mongo.yml: contiene la descripción del “service”, del “deployment” y del “PersistenceVolumeClaim”.

4. **Kafka**

Es el “broker” de mensajería seleccionado para el consumo y envío de mensajes asíncronos, siendo pieza fundamental para DDD y para las SAGAs implementadas mediante la utilización de Eventuate. La creación de tópicos y de toda la infraestructura necesaria es creada automáticamente por el framework Eventuate, que también dispone de clases que abstraen en cada uno de los microservicios de la creación de los “consumer” y “producer” correspondientes. Como ocurre con el resto de los servicios descritos, Kafka se despliega mediante Docker y tiene asignados “volume” y “secret” en su despliegue en Kubernetes. El manifiesto k8s de este servicio se encuentra dentro del proyecto de “infraestructure” en la siguiente ubicación:

- /infraestructura/k8s/kafka.yml: contiene la descripción del “service”, del “deployment” y del “PersistenceVolumeClaim”.

5. **Eventuate**

Es una plataforma y un framework que está diseñado para solucionar los problemas inherentes en una arquitectura de microservicios tales como la descentralización de los datos, así como la creación de transacciones distribuidas. Además, incluye soporte para DDD, que es utilizado en los microservicios.

Esta plataforma utiliza su propio esquema de base de datos en Mysql, denominado "eventuate", y realiza operaciones de publicación y lectura de tópicos en Kafka. Se despliega como un contenedor Docker a partir de la imagen disponible en Dockerhub. El manifiesto k8s de este servicio se encuentra dentro del proyecto de "infraestructure" en la siguiente ubicación:

- /infraestructura/k8s/cdc-service.yml: contiene la descripción del "service" y del "deployment".

6. **Ingress**

La aplicación Filemanagement no es accesible desde el exterior del clúster de Kubernetes debido a que los servicios están configurados como "ClusterIp". Para este cometido se ha configurado el Ingress[15] que provee Okteto, de esta manera, se establece un único punto de acceso desde fuera. Además, pueden crearse reglas de acceso, por lo que le confiere un extra de seguridad. El manifiesto k8s de este servicio se encuentra dentro del proyecto de "infraestructure" en la siguiente ubicación:

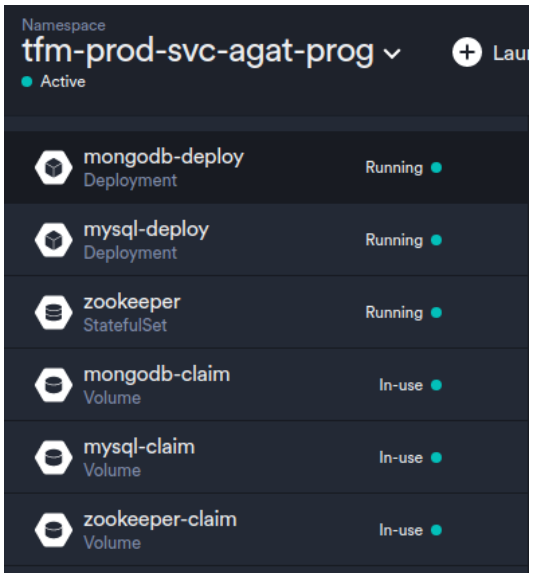
- /infraestructura/k8s/ingress.yml

7. **Okteto**

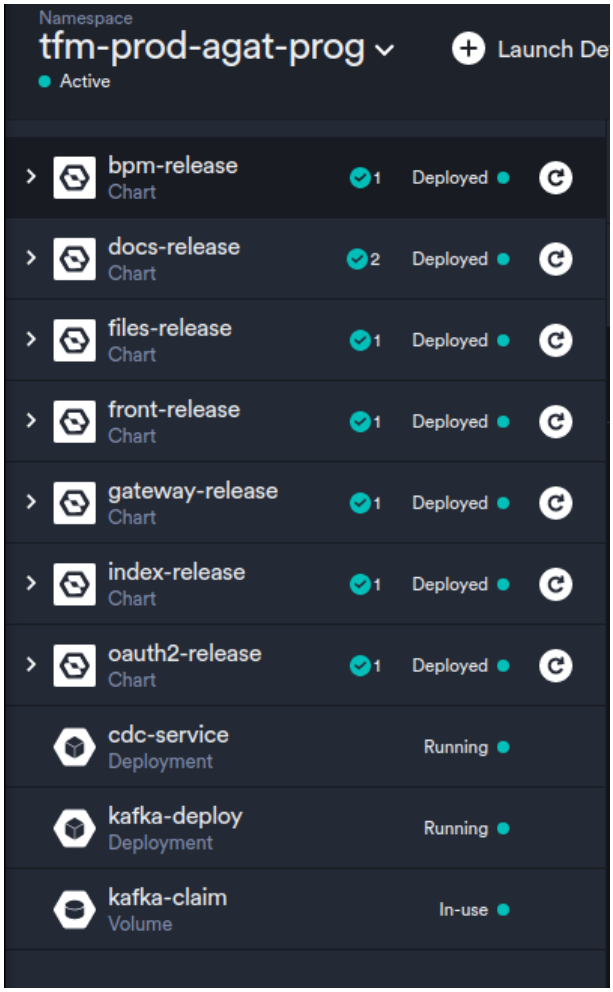
Okteto[16] es un proyecto de código abierto que ofrece una distribución de Kubernetes en cloud. En su capa gratuita permite el uso de 5 namespaces en los que se pueden desplegar un máximo de 10 pods por cada uno de ellos. Dispone de 5 Gb de almacenamiento y dota a cada pod con 1 Gb de RAM y 1 CPU, suficientes para el despliegue de toda la aplicación. No obstante, se ha definido, para cada entorno, dos namespaces distintos, **uno de ellos alberga los microservicios desarrollados y el otro namespace contiene los servicios necesarios** (Mysql, mongodb, etc). Con esta división, no solo se consigue no exceder el número máximo permitido de pods, sino que se persigue crear una división lógica entre aplicación y servicios, arquitectura adoptada en la mayoría de las organizaciones. La capa gratuita también permite el almacén de "secrets", así como el acceso a la plataforma desde sistemas CI o repositorios como

Github. Otra de las características de Okteto, y que se ha explotado en este proyecto, es la utilización de Helm como elemento de despliegue.

La siguiente imagen muestra una vista desde Okteto tanto de los servicios:



como de los microservicios desplegados:



8. Helm

Todos los microservicios implementados están desplegados en Okteto mediante Helm. La configuración de los “Charts” están ubicados en la carpeta “helm” de cada uno de los microservicios. Con estos “charts” es posible realizar el despliegue en diferentes entornos (namespaces) con diferentes configuraciones de forma externa a los manifiestos:

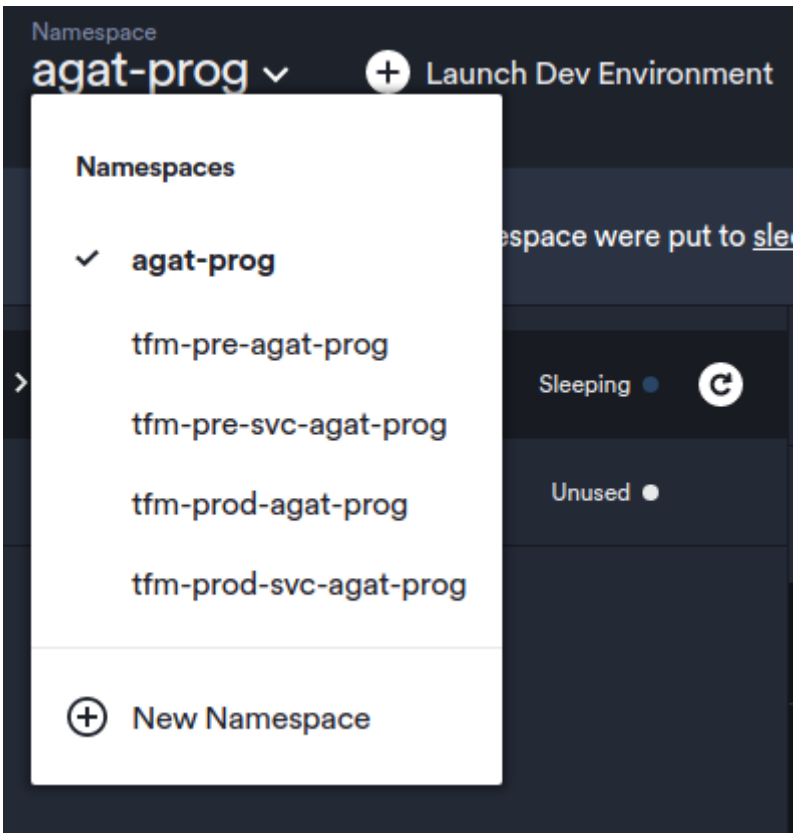
- namespace: se configura el namespace donde será instalado o actualizado.
- host/port de los servicios con los que se integrarán.
- imagen Docker utilizada para los pods. Lo que permite el despliegue de versiones snapshot o releases, dependiendo del entorno.

C. Entornos

Se dispone de dos entornos de despliegue en Okteto:

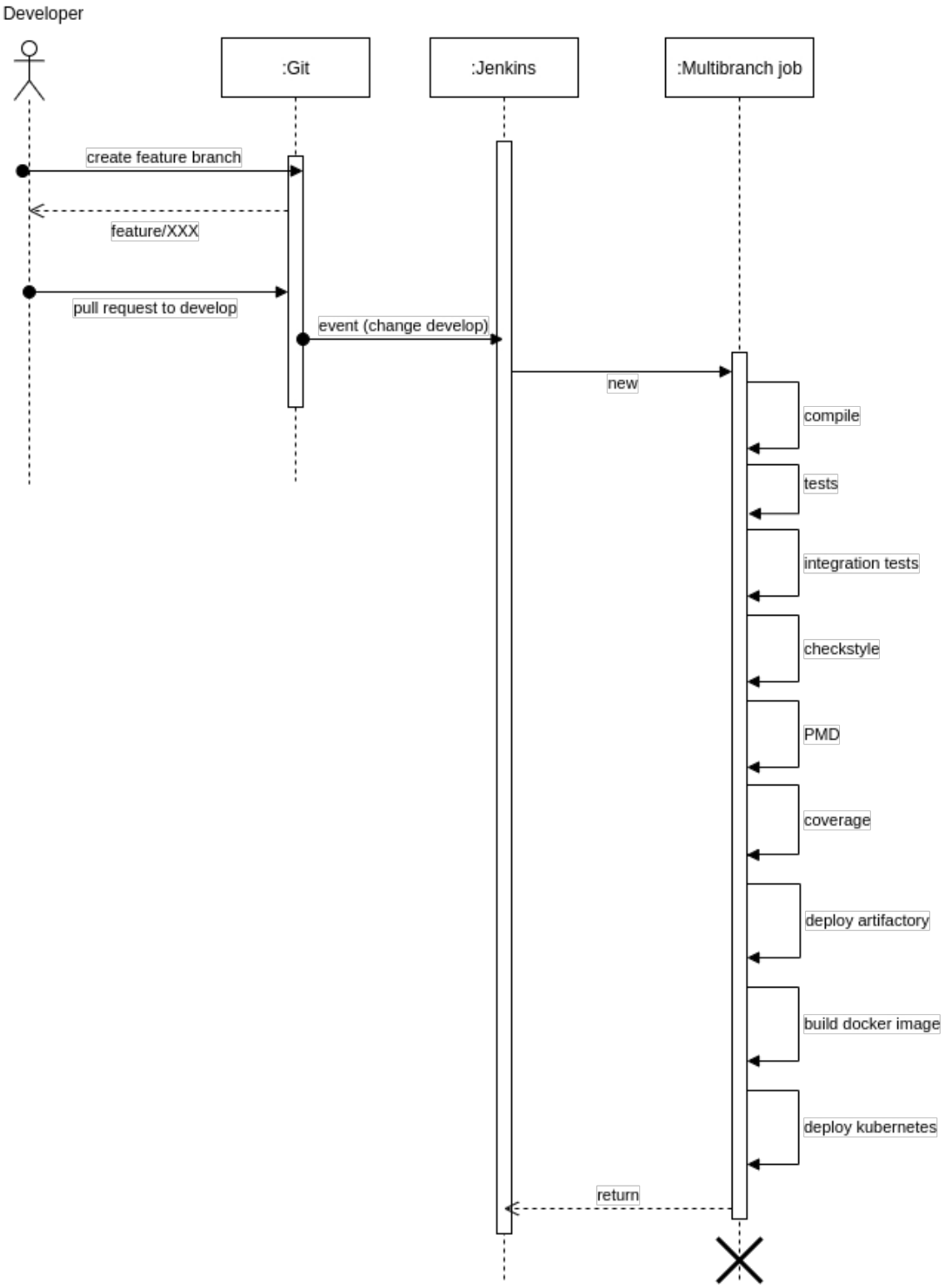
- **PRE**: entorno pre-productivo que es actualizado en cada cambio en la rama “**develop**”. Se trata de un entorno para la realización de pruebas funcionales y en las que se encuentran desplegados contenedores con versión SNAPSHOT. Está compuesto por dos namespace, uno de ellos donde se despliegan los microservicios y otro donde están desplegados los servicios.
- **PROD**: entorno productivo y actualizado en cada cambio de la rama **máster**, según se mostrará en el apartado de calidad posterior. Al igual que ocurre con PRE, está compuesto por dos namespaces.

La siguiente imagen muestra los entornos creados y sus namespaces asociados en Octeto Cloud:



D. Calidad

En cuanto a la calidad, se ha implementado un servicio de integración continua, que permite, no solamente el despliegue automático de los aplicativos desarrollados, sino de la ejecución de tareas de validación y de evaluación de la calidad del software desarrollado. Este conjunto de tareas determinan un proceso CD/CI que se detallan a continuación y que se ha esquematizado en el siguiente diagrama:



Tal y como se observa en el diagrama anterior, el flujo seguido desde que un desarrollador realiza un commit en el repositorio Git hasta el despliegue de los cambios, se realizan los siguientes pasos:

- Un desarrollador crea una rama para una nueva funcionalidad, según se establece en la metodología gitflow[18] adoptada en este TFM.

- Cuando el desarrollador ha implementado la funcionalidad realiza una “pull request” de la rama “feature” en la que ha estado trabajando, sobre la rama “develop”.
- Al realizar la “pull request” sobre la rama “develop” se desencadenan una serie de eventos sobre el sistema CD/CI. El primer evento es la ejecución de la tarea de Jenkins asociada al repositorio del microservicio que ha sido modificado.
- La tarea de Jenkins asociada realiza las siguientes subtareas:
    - Compilación
    - Ejecución de tests unitarios
    - Ejecución de tests de integración.
    - Ejecución y validación de reglas de estilo mediante el plugin de checkstyle.
    - Ejecución de plugin de PMD.
    - Ejecución y validación de cobertura de código. Se ha establecido una cobertura mínima del **90%** para validar esta métrica.
    - Generación de la imagen Docker y su despliegue en Docker Hub.
    - Se despliegan los artefactos generados en el repositorio “Artifactory”, tanto las versiones SNAPSHOT como las RELEASES, según corresponda.
    - Si todos los pasos anteriores han tenido éxito, el último paso es el despliegue de la aplicación mediante Helm en Kubernetes, disponible mediante Okteto, en el entorno correspondiente.

La ejecución de las validaciones y plugins mencionados garantizan una buena calidad del software implementado, si bien, podría incorporarse plataformas como Sonar que realizan un análisis más exhaustivo.

A continuación, se detallan los elementos que intervienen en el proceso de CD / CI que se han explicado en este apartado.

1. Jenkins

Jenkins es un servicio de automatización basado en Jobs preparados para la implementación de tareas de CD/CI. Concretamente para este TFM, se ha creado un proyecto “**Multibranch**” para cada microservicio existente. Este tipo de proyectos realizan un sondeo de todo el repositorio configurado, de tal manera que cualquier cambio en alguna de sus ramas pueden lanzar tareas asociadas. La imagen siguiente corresponde a la vista general de todos los proyectos “Multibranch” creados para Filemanagement:

S	W	Nombre ↓	Último Éxito	Último Fallo	Última Duración
		bpm-multibranch	4.8 Seg log	N/D	1.3 Seg
		docs-multibranch	4.8 Seg log	N/D	1.5 Seg
		files-multibranch	4.8 Seg log	N/D	1.3 Seg
		front-multibranch	4.8 Seg log	N/D	1.2 Seg
		gateway-multibranch	4.8 Seg log	N/D	1.2 Seg
		index-multibranch	3.5 Seg log	N/D	1.1 Seg
		oauth2-continuous-deploy-pre	3 Mes 12 días #15	N/D	10 Seg
		oauth2-continuous-integration	3 Mes 12 días #4	3 Mes 13 días #1	11 Seg
		oauth2-continuous-release	N/D	N/D	N/D
		oauth2-multibranch	3.5 Seg log	N/D	1.2 Seg

En la configuración de cada uno de estos proyectos “Multibranch” se han seguido los siguientes pasos:

- Configuración del repositorio Git del microservicio.
- Configuración de los repositorios de “Artifactory” y de “Dockerhub” utilizados.
- Ubicación relativa del fichero “Jekinsfile” que cada repositorio contiene. Este fichero es muy importante porque contiene todos los pasos que deben ejecutarse dentro de Jenkins.
- Configuración de “poll” de escucha del repositorio referenciado.

Además, Jenkins ofrece una interfaz web que permite la visualización de los estados de las tareas de una forma gráfica. A continuación, se muestra un ejemplo de ejecución real llevada a cabo en uno de los microservicios desarrollados:

Branch develop

Full project name: files-multibranch/develop

Stage View

	Declarative: Checkout SCM	Declarative: Tool Install	Unit Test	Deploy dependencies	PMD	Coverage check	Build image	Deploy into Kubernetes
Average stage times: (Average full run time: ~55s)	1s	110ms	8s	7s	3s	7s	13s	8s
#12 Mar 23 19:54 2 commits	1s	72ms	8s	6s	3s	7s	9s	7s
#11 Mar 23 19:37 2 commits	1s	148ms	9s	8s	3s	8s	18s	8s

2. Artifactory

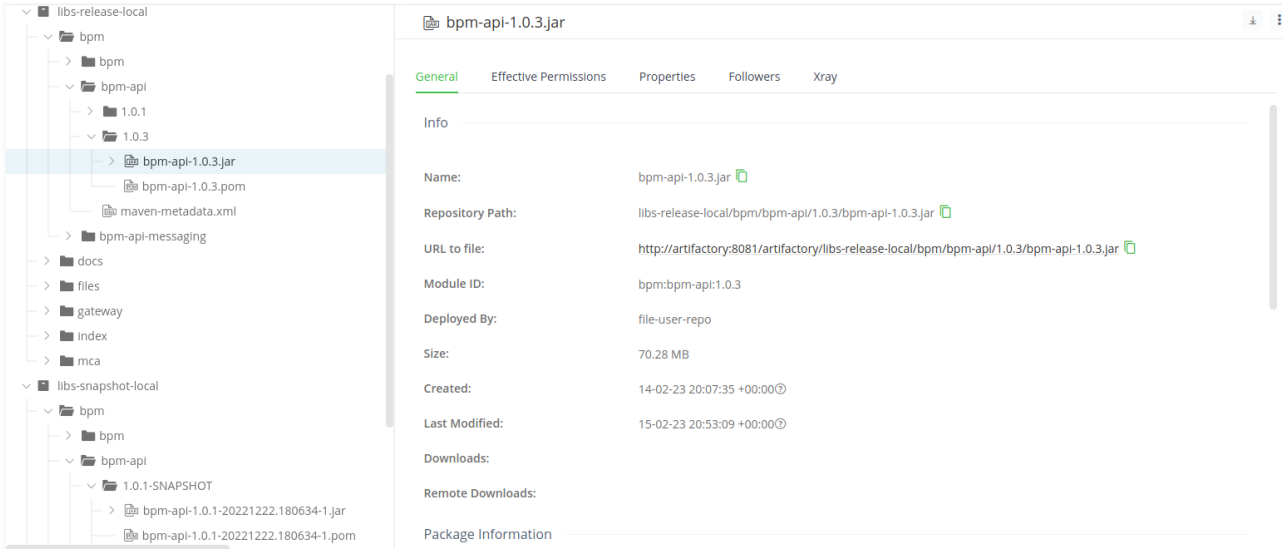
Se ha elegido “Artifactory” como repositorio Maven para los artefactos generados. Esta plataforma contiene de fábrica, dos repositorios concretos utilizados en el desarrollo de la aplicación:

- repositorio para versiones “SNAPSHOT”: almacena los artefactos relativos a versiones de desarrollo y, por tanto, no definitivas.
- repositorio para versiones “RELEASES”: almacena los artefactos finales etiquetados con versiones determinadas preparadas para su utilización en entornos productivos.

Estos repositorios son importantes en el desarrollo de los microservicios y se acceden a ellos mediante los plugins de Maven correspondientes configurados en los ficheros POM de cada microservicio.

El despliegue de “Artifactory” se ha llevado a cabo utilizando una imagen Docker disponible en “Dockerhub”.

La siguiente imagen corresponde con la visualización desde Artifactory de los repositorios “SNAPSHOTS” (libs-snapshot-local) y “RELEASES” (libs-release-local):



3. PMD

Es un analizador de código estático que incluye un conjunto de reglas que pueden ser modificadas. Se ha utilizado PMD en una de las subtareas ejecutadas en Jenkins y además se ha integrado en el IDE Eclipse para facilitar el desarrollo de las aplicaciones. Esta integración con Eclipse permite visualizar en tiempo de diseño si el código contiene algún error o warning que el desarrollador debe solventar, evitando subir código a los repositorios que contengan errores estáticos.

4. Checkstyle

Es un analizador de código que busca errores de estilo según la configuración o reglas que se determinen. Todos los microservicios disponen de un archivo de configuración donde se establecen estas reglas de codificación en la siguiente ubicación relativa a cada proyecto:

- /checkstyle/checkstyle.xml

Las reglas establecidas en este archivo son ejecutadas en uno de los “stage” de Jenkins, por lo que es obligatorio que el código tenga una codificación acorde a estas reglas. Como ayuda al desarrollo, puede utilizarse el plugin de checkstyle para el IDE Eclipse o ejecutar mediante Maven el comando: mvn verify

5. Testing (Junit, Mockito)

La implementación de los test unitarios se ha realizado mediante **JUnit** 5 y con el framework **Mockito**[19]. Este framework permite la creación de objetos dobles de pruebas para la realización de tests unitarios. Mediante estas tecnologías, se ha diseñado e implementado una batería de test **que cubren un mínimo de un 90% de cobertura.** Este dato de cobertura mínima es controlada por el plugin “jacoco-maven-plugin” que también es ejecutado en otro de los “stage” de Jenkins, y que por tanto forman parte del CD/CI.

4. Conclusiones y trabajos a futuro

La realización de este TFM me ha servido para refrescar gran parte de las tecnologías impartidas a lo largo del máster. Ignoraba el mundo que había detrás de las tecnologías de microservicios, DevOps, Cloud, etc, por lo que tuve claro desde el principio que mi TFM tendría que ser aquel que me permitiera profundizar (al menos, un poco más) en cada materia. A lo largo del desarrollo de este, me he encontrado con varios obstáculos que he ido lidiando a base de leer documentación técnica de aquellas tecnologías o frameworks empleados. Uno de los escollos que más problemas me ha dado es el correspondiente a Eventuate. Por poner un ejemplo, me resultó muy complicado configurar un RabbitMQ en lugar de Kafka, por lo que desistí. Otro problema es que no conseguí que el servicio funcionara correctamente en instancias de Mysql distintas. Por otro lado, no pude configurar el servicio de kafka en los namespaces de Okteto creados para tal fin, los tuve que desplegar en el mismo en el que se despliegan los microservicios, debido a problemas de visibilidad.

El proceso de dockerización me ha resultado muy interesante y bastante práctico, así como la utilización de docker-compose para la integración entre microservicios en entornos locales. Otra sorpresa grata ha sido el uso de Okteto, a pesar de sus limitaciones en la versión gratuita, que una vez configurado en local y, mediante el uso de kubectl, he podido realizar toda la gestión y despliegues sin apenas conectarme a la aplicación. La generación de manifiestos para k8s también me ha resultado positiva, con alguna desesperación incluida al tratar de conectar servicios entre namespaces distintos.

En cuanto a la metodología de trabajo, se hace dura al principio, debido al uso de gitflow y todo el proceso CI/CD que hay por detrás, pero una vez entrado en materia es muy confortable desarrollar con unas **redes de seguridad** enormes creadas a base, no solo del mencionado proceso CI/CD, sino de la cobertura obtenida en los test de cada microservicio. En el marco de CI/CD, quise desde el principio utilizar como repositorio de artefactos Artifactory, en lugar de utilizar Github para este propósito, a pesar de haberlo usado en alguna práctica del máster. Esta opción la he seleccionado debido a su amplío uso en proyectos reales.

Este TFM ha abarcado gran parte de las tecnologías y metodologías impartidas a lo largo del máster y se ha dejado para un posterior evolutivo, o trabajos a futuro, los siguientes puntos:

- Metodología API First: metodología de definición de APIs en la que se prioriza la generación de contratos.

- Generación de documentación y acceso a los microservicios mediante OpenApi y Swagger.
- Tests de regresión mediante la ejecución de scripts de Postman.
- Test de integración mediante el uso de Testcontainer, generando pruebas e2e con persistencia de datos en bases de datos reales.
- Agregar “Network Policies” de Kubernetes para restringir las llamadas a cada microservicio.
- Añadir paginación y filtrado de datos en el listado principal de expedientes.
- Pruebas de carga con la herramienta Artillery
- Realización de despliegues mediante la técnica de “Canary” o “Blue/Green”
- Incorporación de herramientas de observabilidad:
    - Prometheus
    - Grafana
- Trazas distribuídas mediante Zipkin
- Completar el proceso de CI/CD con Sonar
- Utilización de Websocket para la notificación asíncrona desde el frontal.
- Conectar el entorno de producción con una base de datos creada en AWS, así como Mongodb.

5. Bibliografía

Docker[1]

https://hub.docker.com/

https://faun.pub/how-to-install-docker-in-jenkins-container-4c49ba40b373

Kubernetes[2]

https://kubernetes.io/es/

https://minikube.sigs.k8s.io/docs/start/

https://www.maquinasvirtuales.eu/kubernetes-servicios-clusterip-ingress-nodeport-y-loadbalancer/

Helm[3]

https://helm.sh/

Mysql[4]

https://www.mysql.com/

Mongodb[5]

https://www.mongodb.com/

Eventuate[6]

https://eventuate.io/docs/manual/eventuate-tram/latest/cdc-configuration.html

https://github.com/eventuate-foundation/eventuate-common/tree/master/mysql

https://github.com/microservices-patterns/ftgo-application/blob/master/docker-compose.yml

Spring Cloud Gateway[7]

https://spring.io/projects/spring-cloud-gateway#overview

https://spring.io/guides/gs/gateway/

Flyway[8]

https://flywaydb.org/

Jenkins[9]

https://www.jenkins.io/

Artifactory[10]

https://jfrog.com/artifactory/

PMD[11]

https://pmd.github.io/

Checkstyle[11]

https://checkstyle.sourceforge.io/

https://www.baeldung.com/checkstyle-java

Maven[12]

https://github.com/phillipuniverse/githook-maven-plugin

Thymeleaf[13]

https://www.thymeleaf.org/

Jib[14]

https://github.com/GoogleContainerTools/jib

https://github.com/GoogleContainerTools/jib/tree/master/jib-maven-plugin

Ingress[15]

https://kubernetes.io/docs/concepts/services-networking/ingress/

Okteto[16]

https://cloud.okteto.com

https://www.okteto.com/docs/cloud/ssl/

Kafka[17]

https://www.baeldung.com/spring-kafka

https://developer.confluent.io/quickstart/kafka-on-confluent-cloud/?utm_medium=sem&utm_source=google&utm_campaign=ch.sem_br.nonbrand_tp.prs_tgt.kafka_mt.xct_rgn.emea_lng.eng_dv.all_con.kafka-general&utm_term=apache+kafka&placement=&device=c&creative=&gclid=CjwKCAjw9-KTBhBcEiwAr19ig6LPZ2uCmKN-zlCLkUXtf0ffnFmxmtK_kuDLt2JexWfiR31ULoqKrBoChUwQAvD_BwE

https://kafka.js.org/docs/consumer-example

Gitflow[18]

https://www.atlassian.com/es/git/tutorials/comparing-workflows/gitflow-workflow

Mockito[19]

https://site.mockito.org/