

Universitat Oberta de Catalunya

**Proyecto de desarrollo del software**

Conceptualización – PRAC 2

Grado de Ingeniería Informática

por

The Sprinters

Diciembre de 2022

Equipo:

Alberto Pinto Romero

Enrique Puga Hernández

Miguel Elías Villanúa

Moisés Pernas Concepción

Òscar Queraltó Aguilera

Pablo Diaz Muñiz

# Indice

1. Modelo de versionado y branching.	3
2. Arquitectura del proyecto (walking skeleton)	5
Tecnologías backend	5
Tecnología frontend	5
Infraestructura de despliegue	6
Aplicación en producción	8
Ejecución en local	8
3. Sistema de distribución e integración continuo	9
Estructura y definición	9
Detalle de implementación	10
4. Sistemas de monitorización	13
<b>Bibliography</b>	<b>16</b>

## 1. Modelo de versionado y branching.

La definición del modelo de versionado y branching es una tarea crítica para el equipo, pues de ello dependerá la administración de las distintas versiones de cada producto desarrollado, así como las posibles especializaciones realizadas. Para el análisis se tendrá en cuenta que la prioridad principal es la de tener cambios acotados, fáciles de revisar, que permita integrarlo en producción lo antes posible y que se ajusten al flujo de trabajo del equipo *The Sprinters*. En este sentido el equipo analiza 6 de los principales flujos de trabajo existentes, los cuales detallamos a continuación:

**Git Flow** [1], [2] Considerado un poco complicado y avanzado para muchos de los proyectos actuales, Git Flow permite el desarrollo paralelo donde los desarrolladores pueden trabajar por separado de la rama *main*. Posteriormente, cuando se completan los cambios, el desarrollador fusiona estos cambios nuevamente en la rama *main* para su publicación.

**GitHub Flow** [3], [4] alternativa más simple a GitFlow, ideal para equipos pequeños, ya que no necesitan administrar múltiples versiones, este modelo no tiene ramas de lanzamiento. Comienza con la rama principal, luego los desarrolladores crean ramas, que se derivan directamente de *main*, para aislar su trabajo, que luego se fusionan nuevamente en la principal. A continuación, se **elimina la rama**.

**GitLab Flow** [5] combina el desarrollo basado en funciones y la bifurcación de funciones con el seguimiento de problemas. La mayor diferencia con GitHub Flow son las ramas de entornos de GitLab Flow (*staging and production*). Es adecuado cuando se desea mantener un entorno de prueba separado del entorno de producción, no se controla los lanzamiento (aplicaciones que deben ser validadas por la App Store), define como hacer la CI/CD y se tiene un historial limpio y legible.

**Release Flow**, desarrollado por el equipo DevOps de Microsoft Azure, ver [6], se basa en un uso intensivo de la rama *master/main* por lo que siguen una aproximación basado en *Trunk-Based*, pero **no realizan** un *deploy* continuo de *master* a producción. En su lugar, lanzan la rama *master* en cada sprint creando una rama (*release*) para cada lanzamiento y los *hotfixes* se llevan a producción mediante *cherry-pick*.

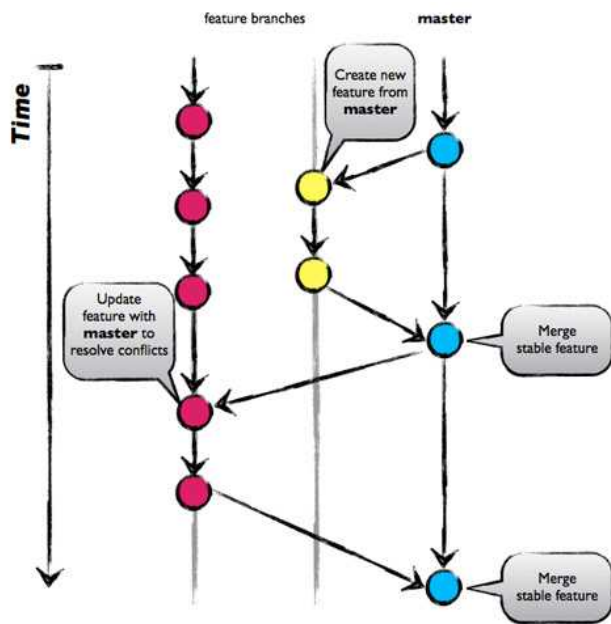
**Trunk-based Development** [7] es una estrategia de bifurcación, en la que los desarrolladores integren los cambios periódicamente (una vez al día) en un troncal compartido, este troncal debe estar **listo para ser liberado en cualquier momento**.

**Master Only Flow**, en este caso se trabaja únicamente con la rama *master* [8, Cap. 14], esto fuerza a que el código se integre continuamente, garantiza que los desarrolladores reciban los cambios y evita la problemática de las fusiones e integraciones al final del proyecto. En un primer momento, parece que se llega al punto de partida en el cual se tiene los conflictos que intentan solventar las ramificaciones. Para evitar esta problemática, este flujo de trabajo niega la mayor, es decir, que no se necesita homogeneizar el conocimiento y los consensos de equipo a posteriori si ya lo realizamos a priori o durante el desarrollo.

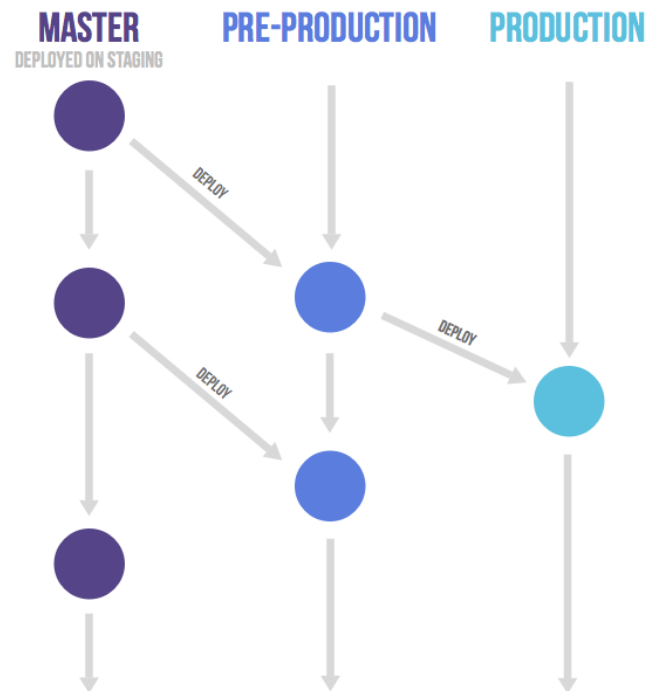
Por tanto, se desarrolla la siguiente tabla, que recoge las características principales de cada opción:

	Git Flow	GitHub Flow	GitLab Flow	Release Flow	Trunk-based	Master Only
<b>Para equipos</b>	Medianos	Pequeños	Pequeños	Grandes	Medianos	Pequeños
<b>Dificultad</b>	Media	Baja	Media	Alta	Media	Alta
<b>Experiencia</b>	Media	Baja	Baja	Alta	Alta	Alta

Tras discutir las diferentes ventajas de todos los modelos, y dado que somos un equipo novato, pequeño, y que se va a trabajar en un proyecto con muchos paquetes que se desarrollaran de manera bastante ordenada, se considera usar GitHub Flow por su simplicidad. A continuación mostramos una representación gráfica a modo comparativo entre los 2 candidatos finalistas.



GitHub Flow



GitLab Flow

Para la nomenclatura del versionado, los tags tendrán la siguiente estructura:

- v0.X\_dd/mm/yyyy

El *tag* muestra la versión de la aplicación, secuencial sobre la anterior y la fecha de creación. Se generará al hacer *merge* con la rama *main*, y añadiremos un pequeño comentario y/o resumen de los cambios realizados sobre el proyecto principal para que, en el momento que se decida unificar (*merge*) la nueva versión con la rama *main*, el equipo tenga claro que se ha hecho y cuales son las mejores implementadas.

## 2. Arquitectura del proyecto (walking skeleton)

### Tecnologías backend

Se decide partir del proyecto del año pasado, por lo que ya sabemos gran parte de las tecnologías a utilizar, estas serán:

- **Spring Framework:** *framework Open Source* que facilita el desarrollo de aplicaciones mediante la inyección de dependencias, lo que permite que los distintos componentes dependan únicamente de interfaces, los que se traduce, en un código menos acoplado. Además, integra el servidor de aplicaciones en el propio .jar, lo que facilita configurar el servidor junto con la aplicación.
- **Postgres:** base de datos relacional multiplataforma de código abierto, que permite las consultas tanto relacionales como no relacionales.
- **Apache Kafka:** sistema de mensajería y de procesamiento de datos en tiempo real que nos proporciona la capacidad de publicar y procesar flujos de eventos de forma escalable y tolerante a fallos.
- **ZooKeeper:** servidor de código abierto que coordina procesos distribuidos de forma fiable.
- **Prometheus:** solución de monitoreo de código abierto para recopilar y agregar métricas como datos de series de tiempo.
- **Grafana:** herramienta de código abierto para el análisis y visualización de métricas.

### Tecnología frontend

En cuanto a la elección del front, se estudian las principales opciones del mercado, los cuales resumimos aquí en adelante:

**Angular**, uno de los más utilizados con muchísima documentación. La estructura del proyecto está ligada directamente al patrón de diseño MVC (Modelo Vista Controlador), que permite afrontar un proyecto de forma solvente y altamente escalable. Se puede integrar fácilmente con muchas otras tecnologías como por ejemplo bootstrap para hacer la web *responsive*.

**React**, biblioteca Javascript de código abierto diseñada para crear interfaces de usuario con el objetivo de facilitar el desarrollo de aplicaciones en una sola página. React es minimalista: sin inyección de dependencias, sin plantillas clásicas, sin funciones demasiado complicadas. La librería será bastante sencilla de entender si ya conoces bien JavaScript.

**Spring Boot**, el cual usamos para la parte *back*, y que también permite diseñar aplicaciones web.

**Bootstrap**, es un conjunto de herramientas de diseño basado en HTML y CSS. A diferencia de muchos *frameworks* web, solo se ocupa del desarrollo front-end. Por lo tanto, hace falta integrarlo con otras tecnologías para acceder a la base de datos.

Con toda la información, y tras discutir sus pros y contras que enmarcamos en tabla que mostramos a continuación, se decide utilizar Angular dada su alta capacidad de adaptación e implementación del MVC que nos puede ayudar a estructurar el frontend, así como nos facilita el mantenimiento del mismo y las futuras ampliaciones.

	Ventajas	Desventajas
<b>Angular</b>	Ampliamente utilizado, documentación detallada, MVC, fácilmente integrable.	Alta curva de aprendizaje, necesidad de aprender TypeScript.
<b>React</b>	Ampliamente utilizado, fácil de aprender si se tienen conocimientos previos de JavaScript.	Solo es una biblioteca de JavaScript para el desarrollo de UI, necesita otras librerías adicionales.
<b>SpringBoot</b>	Simplifica la implementación y configuración porque también se usa en el <i>backend</i> .	No es tan utilizado ni potente como Angular o React, necesita otras librerías adicionales.
<b>Bootstrap</b>	Ampliamente utilizado, <i>mobile-first</i> , <i>responsive</i> , estándar visual.	Se trata únicamente de un framework CSS, necesita otras librerías adicionales.

## Infraestructura de despliegue

Para poner en funcionamiento nuestro proyecto, primero decidimos tener una versión funcional en local y que sea fácil de descargar y probar, y que sea fácilmente trasladable a los servidores en producción y automatizable como veremos más adelante. Por ello, se decide utilizar un sistema basado en imágenes docker.

Para comenzar, se decide subir las imágenes a un repositorio gratuito de docker, esto será así para poder acceder con facilidad a las imágenes y poder levantar el entorno con un único docker-compose. Esto aísla el proyecto de ningún tipo de dependencia en local más allá de docker. Posteriormente, se empieza a trabajar en la elección del entorno de producción utilizado. En slack los profesores nos dan dos alternativas, las cuales se estudian:

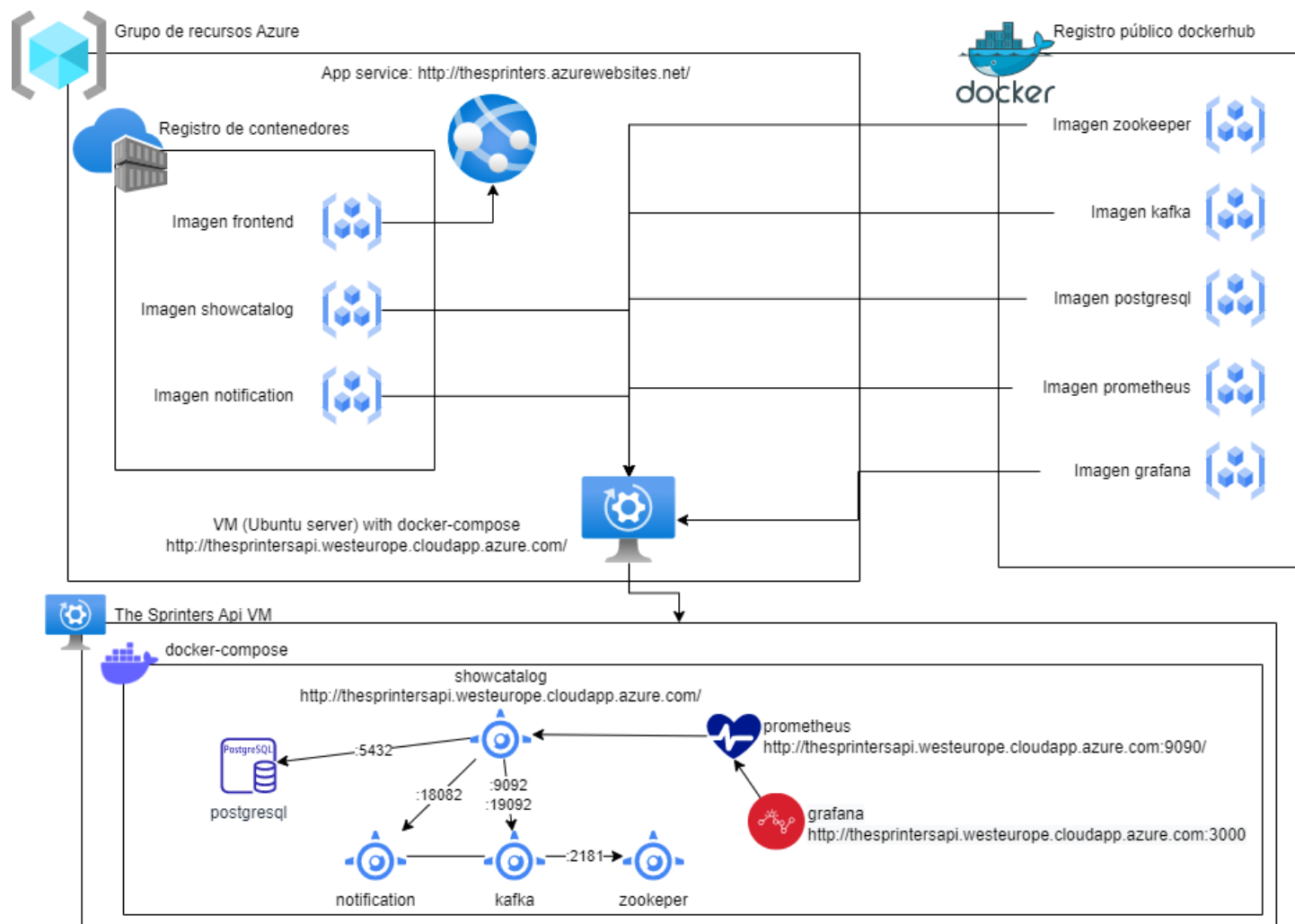
**AWS**, servidor de Amazon, con servicio propio de la UOC. Se inician las pruebas, se comprueba que es sencillo cargar las imágenes en el repositorio e incluso acceder a las de dockerhub. Se intenta generar una instancia de contenedores pero da problemas. Posteriormente se está varios días sin acceso, y se decide seguir probando con azure.

**Azure**, sistema de Microsoft, acceso con suscripción de estudiante gracias a pertenecer a la UOC. Al igual que con AWS, se comprueba que es sencillo cargar las imágenes en azure y automatizar este proceso. En este caso, levantar una instancia parece mucho más sencillo, por lo que se opta por esta plataforma dados los resultados y los problemas de la anterior.

Finalmente, dado lo anterior se opta por Azure, y se comienzan los trabajos de configuración e instalación.

En primer momento se intentó usar las instancias de contenedor, pero resultaron consumir demasiados créditos y no iba a ser posible terminar la práctica antes de que se acabasen, por ello se exploró otra alternativa con opción gratuita, los App service. En este caso, el frontend funcionó sin problema y quedó funcionando, pero el resto de contenedores se cerraban al no tener una web pública en el puerto 80.

Finalmente se optó por probar una máquina virtual con recursos algo limitados, suponiendo un coste que permite terminar la práctica antes de que se agoten, y se continuó con esta opción. En esta máquina se instaló docker compose y se habilitó un acceso SSH para poder conectarse desde fuera. Tras la configuración anterior, queda un diagrama similar al siguiente:



## Aplicación en producción




Gracias a todo lo anterior, en este punto tenemos el servidor en producción, accedemos a nuestra URL <http://thesprinters.azurewebsites.net/> y ya podemos comprobar cómo accedemos sin problemas y podemos listar y crear shows:

No es seguro | thesprinters.azurewebsites.net/shows

Fuertafit Estudios para la casa Motos Excel TuMangaOnline Suelos Gimnasios. P...

### CultureInDaHouse

LIST SHOWS CREATE SHOW CATEGORIES ABOUT CONTACT

Id	Name	Description	category	Image	Price	Capacity	Duration	OnSaleDate	Status
1	Cats	Musical Cats	Teatro		20	300	120	2022,5,1	CREATED
2	Macbeth	MacBeth by Shakespeare	Teatro		30	350	180	2022,6,15	CREATED
3	Concierto de año nuevo	Concierto de música clásica con ocasión del año nuevo 2023	Concierto		15	250	120	2022,11,1	CREATED

También podemos lanzar una llamada a nuestra API y observar como recibimos los datos de vuelta sin ningún problema <http://thesprintersapi.westeurope.cloudapp.azure.com/catalog/shows>:

No es seguro | thesprintersapi.westeurope.cloudapp.azure.com/catalog/shows

Anime Tabs Fuertafit Estudios para la casa Motos Excel TuMangaOnline Suelos Gimnasios. P... Otros marcadores

```
[{"id":1,"name":"Cats","description":"Musical Cats","image":"https://picsum.photos/id/1/200","price":20.0,"duration":120.0,"capacity":300,"onSaleDate":2022,5,1,"status":"CREATED","category":{"id":1,"name":"Teatro","description":"Obras de teatro, musicales, drama, comedia..."}}, {"id":2,"name":"Macbeth","description":"Macbeth by Shakespeare","image":"https://picsum.photos/id/2/200","price":30.0,"duration":180.0,"capacity":350,"onSaleDate":2022,6,15,"status":"CREATED","category":{"id":1,"name":"Teatro","description":"Obras de teatro, musicales, drama, comedia..."}}, {"id":3,"name":"Concierto de año nuevo","description":"Concierto de música clásica con ocasión del año nuevo 2023","image":"https://picsum.photos/id/3/200","price":15.0,"duration":120.0,"capacity":250,"onSaleDate":2022,11,1,"status":"CREATED","category":{"id":2,"name":"Concierto","description":"Conciertos de música clásica, pop, rock, etc."}}, {"id":4,"name":"Las 4 estaciones","description":"Las 4 estaciones de Vivaldi","image":"https://picsum.photos/id/4/200","price":17.0,"duration":90.0,"capacity":200,"onSaleDate":2022,11,1,"status":"CREATED","category":{"id":2,"name":"Concierto","description":"Conciertos de música clásica, pop, rock, etc."}}, {"id":5,"name":"El lago de los cisnes","description":"Interpretado por la compañía de danza de San Petersburgo","image":"https://picsum.photos/id/5/200","price":40.0,"duration":180.0,"capacity":325,"onSaleDate":2022,12,5,"status":"CREATED","category":{"id":3,"name":"Danza","description":"Representaciones de danza clásica, contemporánea..."}}, {"id":6,"name":"El cascanueces","description":"Interpretado por la compañía de danza de Sebastopol","image":"https://picsum.photos/id/6/200","price":38.0,"duration":120.0,"capacity":300,"onSaleDate":2022,9,18,"status":"CREATED","category":{"id":3,"name":"Danza","description":"Representaciones de danza clásica, contemporánea..."}}, {"id":7,"name":"Così fan tutte","description":"Ciclo Mozart en el Palau de la Musica","image":"https://picsum.photos/id/7/200","price":24.0,"duration":110.0,"capacity":225,"onSaleDate":2023,1,12,"status":"CREATED","category":{"id":4,"name":"Ópera","description":"Ópera"}}, {"id":8,"name":"Rigoletto","description":"Verdi al Liceu","image":"https://picsum.photos/id/8/200","price":42.0,"duration":100.0,"capacity":150,"onSaleDate":2022,8,1,"status":"CREATED","category":{"id":4,"name":"Ópera","description":"Ópera"}}, {"id":9,"name":"Circo Raluy","description":"El Circo Raluy visita Barcelona","image":"https://picsum.photos/id/9/200","price":10.0,"duration":120.0,"capacity":400,"onSaleDate":2022,6,1,"status":"CREATED","category":{"id":5,"name":"Circo","description":"Actuaciones de circo"}}, {"id":10,"name":"Circo Mundial","description":"Gira de verano del circo Mundial - Ahora SIN leones!","image":"https://picsum.photos/id/10/200","price":12.0,"duration":100.0,"capacity":300,"onSaleDate":2022,8,10,"status":"CREATED","category":{"id":5,"name":"Circo","description":"Actuaciones de circo"}}, {"id":11,"name":"Vermundólogos","description":"Diferentes artistas cada día!","image":"https://picsum.photos/id/11/200","price":15.0,"duration":145.0,"capacity":100,"onSaleDate":2022,6,15,"status":"CREATED","category":{"id":6,"name":"Monólogo","description":"Monólogos, stand-up comedy, improvisación, etc."}}, {"id":12,"name":"Tinder sorpresa","description":"El nuevo espectáculo de Toni Hoog","image":"https://picsum.photos/id/12/200","price":15.0,"duration":90.0,"capacity":200,"onSaleDate":2022,7,20,"status":"CREATED","category":{"id":6,"name":"Monólogo","description":"Monólogos, stand-up comedy, improvisación, etc."}}, {"id":13,"name":"El club de la comedia","description":"Actuación en vivo del aclamado programa de humor","image":"https://picsum.photos/id/13/200","price":20.0,"duration":90.0,"capacity":300,"onSaleDate":2022,10,5,"status":"CREATED","category":{"id":6,"name":"Monólogo","description":"Monólogos, stand-up comedy, improvisación, etc."}}]
```

## Ejecución en local

Para poder levantar todo el entorno **en local** y jugar con él, se ha preparado [el siguiente paquete con un docker compose](#). Únicamente será necesario extraerlo en una carpeta y dentro de esta ejecutar **“docker compose up -d”** o **“docker-compose up -d”** dependiendo del SO. Si da conflicto de nombre algún contenedor, será necesario ejecutar **“docker rm “nombrecontenedor”**. La ruta una vez desplegado será: <http://localhost:8080/> y <http://localhost:18081/> para la api.



### 3. Sistema de distribución e integración continuo

#### Estructura y definición

La automatización de tareas es fundamental en equipos interesados en adoptar metodologías ágiles y DevOps, ya que las tareas manuales y repetitivas van en contra de la productividad. Con esta idea en mente, en este proyecto se ha tratado de automatizar las tareas de construcción, test y despliegue de software, lo que se conoce como CI/CD (*Continuous Integration/Continuous Deployment-Delivery*). Para ello, se han creado diferentes *pipelines* en un entorno GitLab, donde también reside todo el código del proyecto.

De forma resumida, se puede pensar en una *pipeline* como algo similar a una cadena de ensamblaje. En este caso, la entrada de la cadena sería el código, que se extraerá de los repositorios, y al que se hará pasar por diferentes etapas o *stages* en las que se realizarán diferentes operaciones o *jobs*. En cada etapa, las diferentes operaciones se podrán realizar de forma paralela y el comienzo de cada etapa estará supeditado a la finalización de la anterior de forma exitosa.

Además, se puede decidir en qué situaciones se desea que se ejecuten las operaciones, por ejemplo, cuando haya cambios (*commit*) en una determinada rama del repositorio. Finalmente, la salida de la cadena será el producto en funcionamiento (tras su despliegue en el entorno de producción), listo para que el cliente pueda utilizarlo de forma satisfactoria.

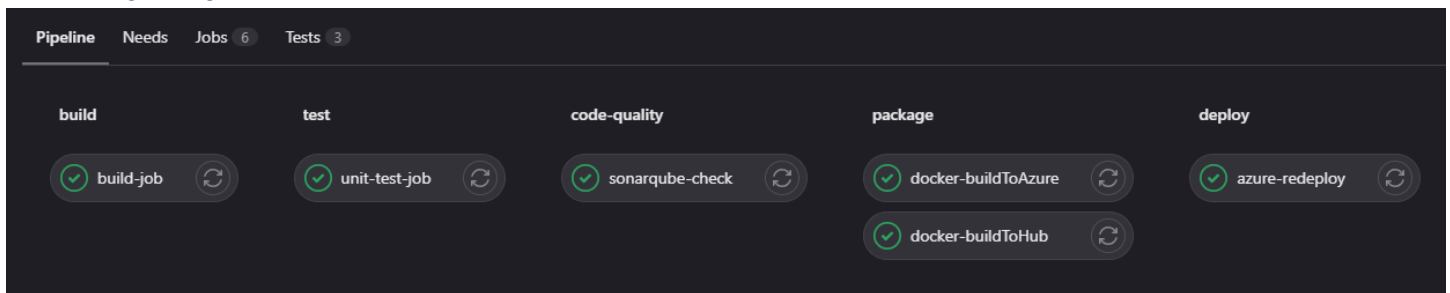
Las operaciones sobre el código se realizan en entornos generados por imágenes *Docker*, que contienen todas las herramientas necesarias para su ejecución. Además, también se han utilizado variables (globales, específicas de una operación, configuradas en las *settings* del proyecto en GitLab), *cachés* (por ejemplo, para las dependencias *maven*) y persistencia de resultados (*artifacts*) de cada operación para su uso posterior, con el fin de mejorar la eficiencia, la legibilidad y la seguridad del resultado final.

Para la presente práctica se han desarrollado tantas *pipelines* como proyectos, y se ha definido una plantilla para conseguir una mayor homogeneidad, consiguiendo la siguiente estructura de etapas:

- **build:** En esta etapa se realizan las operaciones de construcción (compilación/traducción) del código. Para ello se ha utilizado imágenes *Docker* de *Maven*, en el caso de los proyectos *Spring Boot*, y de *Node* en el caso del *frontend*, que se ha creado con *Angular*.
- **test:** En esta etapa se realizan las operaciones relacionadas con la ejecución de los test que se han creado. Además de las herramientas comentadas en la etapa anterior, para el *frontend* también se ha necesitado una instancia de *Chrome* (*headless*, preparado para ejecutarse sin interfaz gráfica) que se ha instalado sobre la imagen *Docker* de *Node*.
- **code-quality:** En este apartado o estado (*stage*) es donde se monitoriza el estado del código que se ha implementado en cada uno de los servicios que disponemos. Este control del código se realiza a través de sonarqube, donde podemos ver diversos factores de la calidad de nuestra implementación. En la mayoría de los servicios se ha usado una imagen maven (una versión anterior ya que en nuestro caso la más reciente nos fallaba), excepto en el proyecto del front-end, donde se ha usado una imagen propia de sonarqube.

- **package:** En este estado se construye el contenedor empleando *dind* y se sube al repositorio de Dockerhub para que los contenedores queden disponibles y públicos, como requisito previo es necesarios realizar el login antes de realizar un push al repositorio. También se incluyen en el repositorio de Azure, para su posterior despliegue. Además, se hace uso de las variables de entorno de GitLab para taguear cada uno de los contenedores de acuerdo a la rama en la que se ejecute, en este sentido, el tag de la rama principal será el latest y el que se lanzará a producción.
- **deploy:** En esta etapa se ejecutan los scripts necesarios para la puesta en producción de los paquetes desplegados. En el caso del *front* se despliega automáticamente al recibir el push del paquete, pero para el resto de elementos se establece una conexión ssh con el servidor que descarga los contenedores actualizados y redespiega docker.
- **autotag:** Este estado está en desarrollo actualmente y se terminará de implementar durante el segundo sprint. Se trata de implementar el sistema de auto versionado que hemos propuesto como un estado en los ficheros gitlab-ci.yml. Un nuevo tag se creará en el momento de realizar un merge en la rama main. De esta manera, tendremos ordenadas automáticamente todas las versiones.

En la imagen siguiente se puede apreciar el proceso.



En el ejemplo que encontraréis [aquí](#), podemos ver un fichero gitlab-ci.yml completo que incluye todos los pasos mencionados anteriormente, y los detalles que comentaremos a continuación.

## Detalle de implementación

Durante el desarrollo de los pipelines, se han encontrado algunos pasos especialmente conflictivos de implementar y hacer funcionar, por lo que se detallan en este apartado. Entre estas, nos encontramos el envío de las imágenes a los diferentes repositorios, los test de sonarqube y el despliegue en los servidores de producción.

Para el **envío a dockerhub y Azure**, se comenzó originalmente intentando hacer una conexión basada en certificados que funcionaba en unas ocasiones sí, y en otras no. Posteriormente se decidió simplificar la conexión usando usuario y contraseña con un *docker login*, y almacenar estas en las variables de configuración de CI/CD de gitlab, y enmascararlas. Esto además ayuda en poder cambiar parámetros de manera futura en los *pipeline* sin tener que hacer un *commit* al proyecto. Por último, se ejecuta un *docker push* que envía la imagen al servidor.

Para el **despliegue en Azure**, existen varias posibilidades. En primer lugar se estudia la posibilidad de hacer una conexión con Azure y usar los comandos “az”, pero no se consigue una manera segura de hacer login sin

exponer ninguna de las claves de los usuarios de la UOC a los compañeros, por lo que se descarta. Para la parte del frontend, se deja configurado usando los *webhooks* de azure, que se configuran en 2 pasos:

1. Primero se activa la integración continua en el AppService del front, y se copia la URL que proporciona el servicio. Cada vez que se invoque esta URL, se recuperará la última versión de la imagen y se desplegará.
2. Después se crea un *webhook*, asociado a la imagen frontend:latest con la orden push. Esto quiere decir que cada vez que se haga un push sobre esa imagen con esa etiqueta, se ejecutará la URL indicada. En este caso, ponemos la que nos proporciona el AppService.

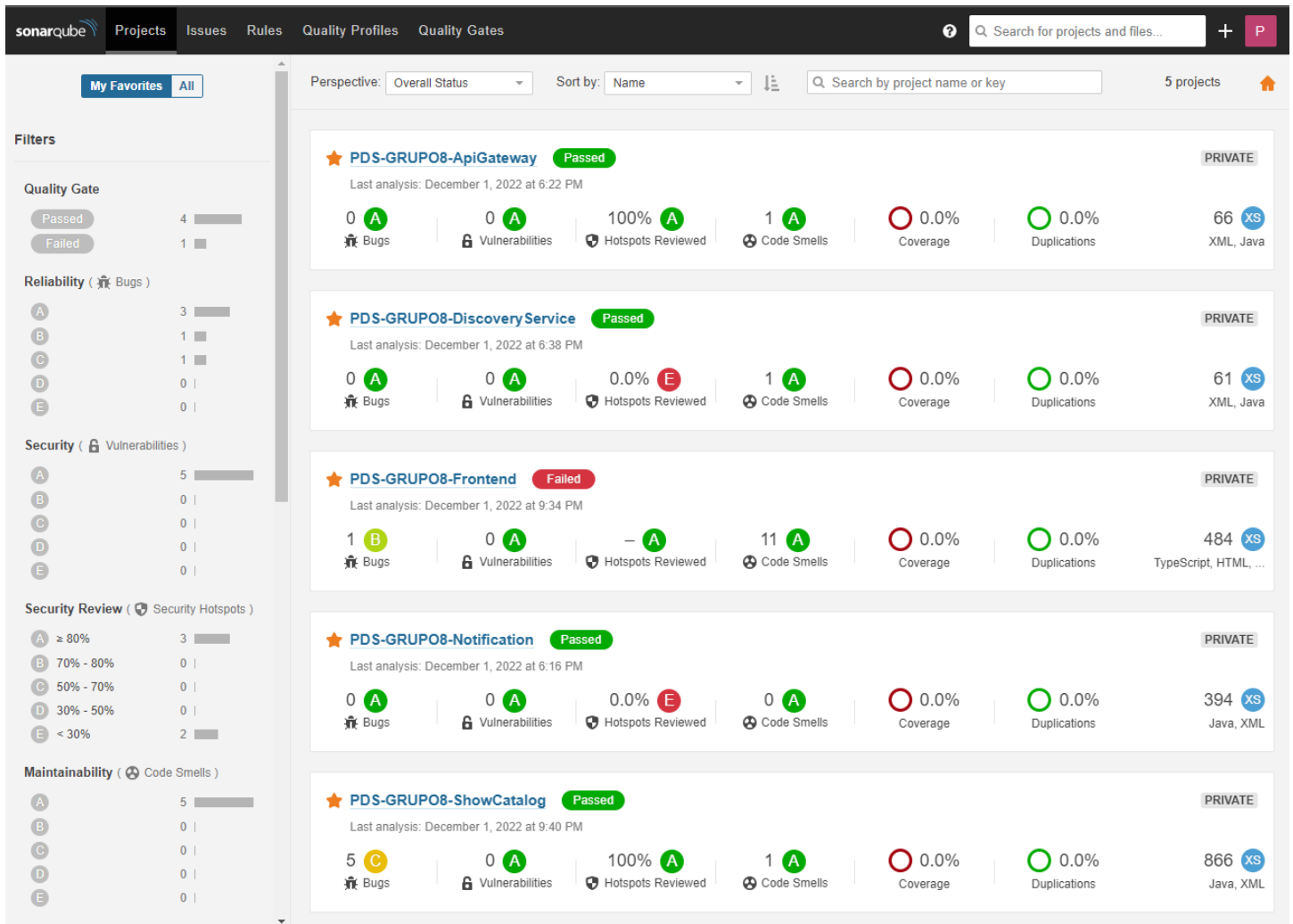
Con lo anterior ya tenemos automatizado el despliegue del front, pero no podemos hacer lo mismo con el back ya que se trata de una máquina virtual en lugar de un AppService, así que generamos un par de claves pública-privada para la conexión ssh al servidor. Esta la almacenamos en las variables de gitlab y la incluimos en el pipeline, lanzando a través de ssh un sencillo *docker pull* y *docker-compose up* sobre la imagen actualizada.

En cuanto a las pruebas, incluimos la **integración con sonarqube**, la cual resolvemos de dos maneras. En el caso del frontend, ya que usa un proyecto node, usaremos la imagen de sonar-cli. En cambio, para la parte de backend, basada en maven, usaremos la imagen maven 3.6.3-jdk-11. A pesar de esta diferencia, ambos métodos seguirán una ejecución similar.

Para la conexión con sonarqube, usaremos un token. Este lo generamos creando un proyecto desde la web de sonarqube, pero podrá ser usado en cualquier otro proyecto. El token, la URL de conexión y el nombre del proyecto sobre el que queremos que se ejecute, se guardará en variables de CI/CD de gitlab, y luego las recuperaremos como variables de sonar dentro del pipeline.

Con todo lo anterior configurado, únicamente será necesario ejecutar los comandos que ofrece sonarqube y maven para realizar el escaneado de nuestros proyectos, para dar la clave de los proyectos y para nombrar los proyecto en sonarqube y para versionar el proyecto con el tag correspondiente. En nuestro caso, usaremos la misma key y nombre para encontrarlo fácilmente dentro del repositorio compartido de la UOC, siguiendo la nomenclatura "GRUPO-8-NombreDelServicio"

A continuación tenemos un resumen de los resultados de los escaneos de cada proyecto/servicio:



Para el caso del proyecto del *front-end*, se han empezado a implementar nuevas funcionalidades y diseños para el siguiente sprint que aún no están completos, por lo que, al volver a pasar el test de sonarqube, han aparecido más code smells respecto a la versión anterior y por eso no tenemos suficiente cobertura (*coverage*) como para pasar el test. Para el siguiente sprint, se terminarán de implementar las nuevas funcionalidades y se arreglarán los bug y code smells que indica sonarqube.

## 4. Sistemas de monitorización

Con el objetivo de asegurar la fiabilidad, estabilidad y escalabilidad del sistema se implementa un sistema de monitorización que supervise los servicios críticos de la solución planteada por el equipo. En este sentido, se valora implementar el seguimiento y las recopilaciones de estadísticas a nivel de proxy, empleando Service Mesh [9, Cap. 4] y el patrón *Sidecar* mediante *Istio*, pero las limitaciones a nivel de infraestructura nos obligan a abordar la observabilidad desde otro punto de vista.

En primer lugar, se integran las dependencias requeridas para hacer uso de la biblioteca de instrumentación de medidas (*Micrometer*) en servicios críticos como *showcatalog* para monitorizar el uso de CPU, memoria consumo, red, logs del sistema y disponibilidad. En la figura siguiente se muestra los *endpoints* expuestos para recolectar las métricas del servicio.

```
{ "href": "http://thesprintersapi.westeurope.cloudapp.azure.com/actuator/caches", "templated": false, "caches-cache":  
{ "href": "http://thesprintersapi.westeurope.cloudapp.azure.com/actuator/caches/{cache}", "templated": true, "health-path":  
{ "href": "http://thesprintersapi.westeurope.cloudapp.azure.com/actuator/health/{*path}", "templated": true, "health":  
{ "href": "http://thesprintersapi.westeurope.cloudapp.azure.com/actuator/health", "templated": false, "info":  
{ "href": "http://thesprintersapi.westeurope.cloudapp.azure.com/actuator/info", "templated": false, "conditions":  
{ "href": "http://thesprintersapi.westeurope.cloudapp.azure.com/actuator/conditions", "templated": false, "configprops-prefix":  
{ "href": "http://thesprintersapi.westeurope.cloudapp.azure.com/actuator/configprops/{prefix}", "templated": true, "configprops":  
{ "href": "http://thesprintersapi.westeurope.cloudapp.azure.com/actuator/configprops", "templated": false, "env-toMatch":  
{ "href": "http://thesprintersapi.westeurope.cloudapp.azure.com/actuator/env/{toMatch}", "templated": true, "env":  
{ "href": "http://thesprintersapi.westeurope.cloudapp.azure.com/actuator/env", "templated": false, "loggers-name":  
{ "href": "http://thesprintersapi.westeurope.cloudapp.azure.com/actuator/loggers/{name}", "templated": true, "loggers":  
{ "href": "http://thesprintersapi.westeurope.cloudapp.azure.com/actuator/loggers", "templated": false, "heapdump":  
{ "href": "http://thesprintersapi.westeurope.cloudapp.azure.com/actuator/heapdump", "templated": false, "threaddump":  
{ "href": "http://thesprintersapi.westeurope.cloudapp.azure.com/actuator/threaddump", "templated": false, "prometheus":  
{ "href": "http://thesprintersapi.westeurope.cloudapp.azure.com/actuator/prometheus", "templated": false, "metrics":  
{ "href": "http://thesprintersapi.westeurope.cloudapp.azure.com/actuator/metrics", "templated": false, "metrics-requiredMetricName":  
{ "href": "http://thesprintersapi.westeurope.cloudapp.azure.com/actuator/metrics/{requiredMetricName}", "templated": true, "scheduledtasks":  
{ "href": "http://thesprintersapi.westeurope.cloudapp.azure.com/actuator/scheduledtasks", "templated": false, "mappings":  
{ "href": "http://thesprintersapi.westeurope.cloudapp.azure.com/actuator/mappings", "templated": false, "refresh":
```

En segundo lugar, una vez la aplicación expone las métricas, se ejecuta, mediante un contenedor, el servidor de Prometheus y se configura para que extraiga la métricas de la aplicación a intervalos de tiempos establecidos en la configuración inicial. A continuación, se muestran los endpoint y el estado de los servicios monitorizados. <http://thesprintersapi.westeurope.cloudapp.azure.com:9090/>

AllUnhealthyCollapse All

Filter by endpoint or labels

docker-metrics (1/1 up)

show less

Endpoint	State	Labels	Last Scrape	Scrape Duration	Error
<a href="http://20.16.93.42:9323/metrics">http://20.16.93.42:9323/metrics</a>	UP	instance="20.16.93.42:9323"job="docker-metrics"	2.162s ago	5.053ms	

microservice-showcatalog (1/1 up)

show less

Endpoint	State	Labels	Last Scrape	Scrape Duration	Error
<a href="http://showcatalog/actuator/prometheus">http://showcatalog/actuator/prometheus</a>	UP	instance="showcatalog:80"job="microservice-showcatalog"	3.679s ago	880.223ms	

prometheus (1/1 up)

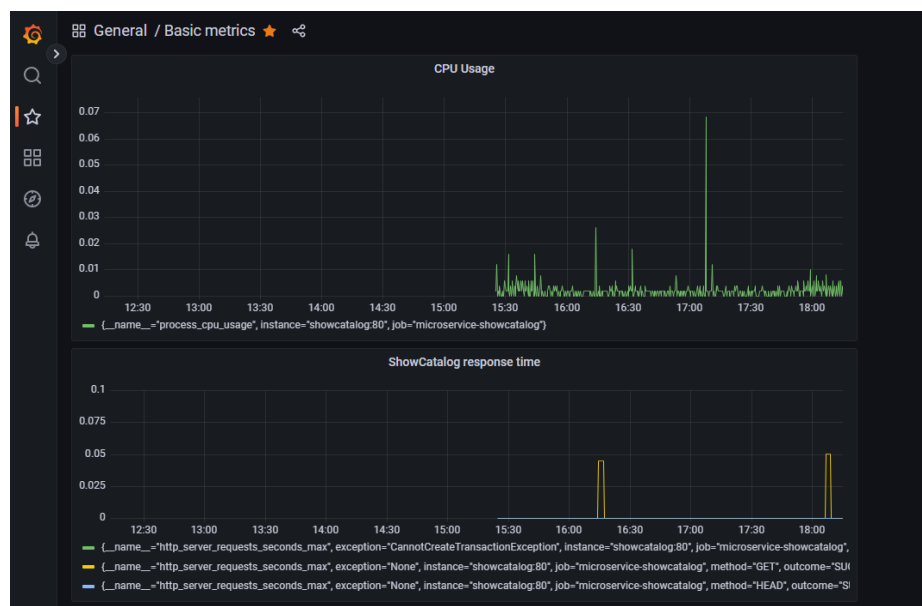
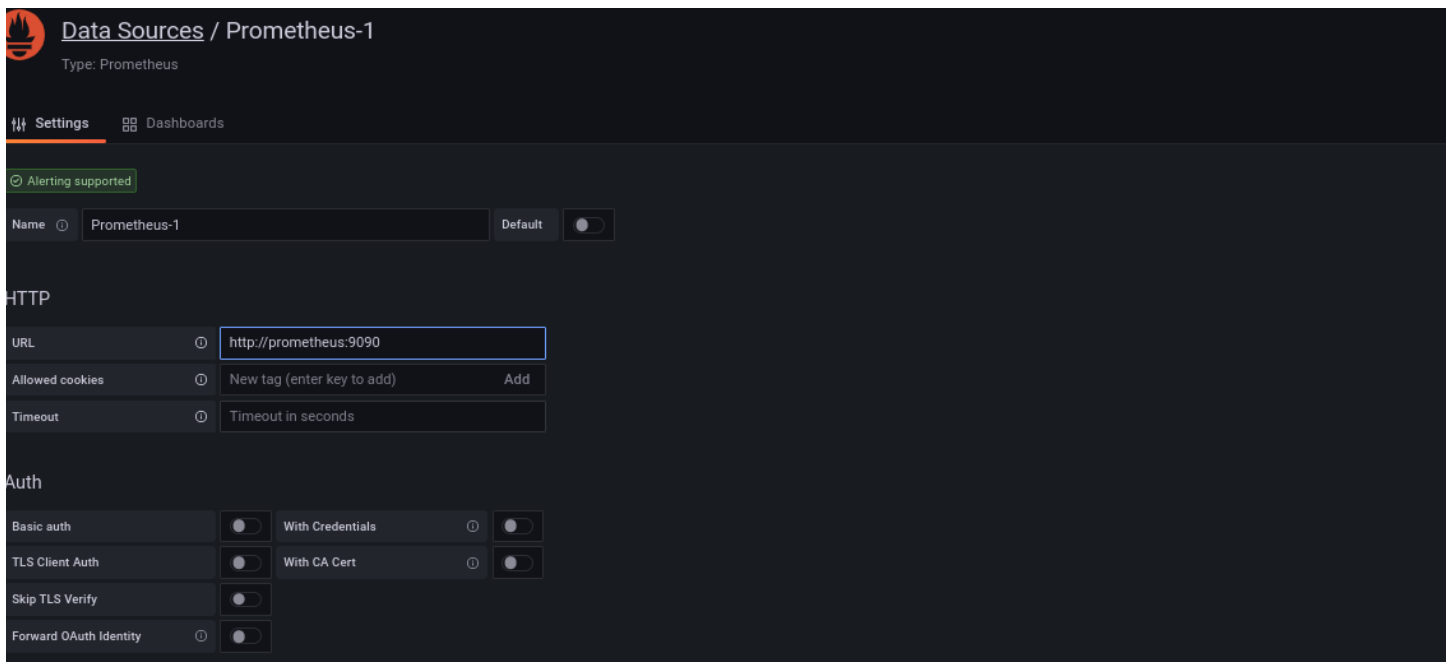
show less

Endpoint	State	Labels	Last Scrape	Scrape Duration	Error
----------	-------	--------	-------------	-----------------	-------

Finalmente, para obtener un panorama gráfico de la situación de nuestros servicios se emplea Grafana en su correspondiente contenedor. A continuación, después del login, se configura el data source y se muestra el gráfico y estadística del uso del cpu del servicio *showcatalog*, junto con el tiempo de respuesta de las llamadas http, y se guarda como el dashboard básico.

<http://thesprintersapi.westeurope.cloudapp.azure.com:3000/>

Para acceder es necesario un usuario, se proporciona uno con permisos de lectura para poder revisar los gráficos, **Usuario:** uoc, **Pass:** uocpass



## Bibliography

- [1] «A successful Git branching model», *nvie.com*. <http://nvie.com/posts/a-successful-git-branching-model/> (accedido 7 de noviembre de 2022). <https://learning.oreilly.com/library/view/continuous-delivery-reliable/9780321670250/ch14.xhtml>
- [2] «git-flow cheatsheet». <https://danielkummer.github.io/git-flow-cheatsheet/> (accedido 7 de noviembre de 2022).
- [3] «GitHub flow», *GitHub Docs*. <https://ghdocs-prod.azurewebsites.net/en/get-started/quickstart/github-flow> (accedido 6 de noviembre de 2022).
- [4] «GitHub Flow – Scott Chacon». <http://scottchacon.com/2011/08/31/github-flow.html> (accedido 6 de noviembre de 2022).
- [5] «Introduction to GitLab Flow | GitLab». [https://docs.gitlab.com/ee/topics/gitlab\\_flow.html](https://docs.gitlab.com/ee/topics/gitlab_flow.html) (accedido 6 de noviembre de 2022).
- [6] E. Thomson, «Release Flow: How We Do Branching on the VSTS Team», *Azure DevOps Blog*, 19 de abril de 2018. <https://devblogs.microsoft.com/devops/release-flow-how-we-do-branching-on-the-vsts-team/> (accedido 4 de noviembre de 2022).
- [7] paul-hammant, «Trunk Based Development». <https://trunkbaseddevelopment.com/> (accedido 6 de noviembre de 2022).
- [8] *Chapter 14. Advanced Version Control*. Accedido: 5 de noviembre de 2022. [En línea]. Disponible en: <https://learning.oreilly.com/library/view/continuous-delivery-reliable/9780321670250/ch14.xhtml>
- [9] *4. The Three Pillars of Observability*. Accedido: 30 de noviembre de 2022. [En línea]. Disponible en: <https://learning.oreilly.com/library/view/distributed-systems-observability/9781492033431/ch04.html>
- [10] «Angular + GitLab DevOps: Integración continua y entrega continua en proyectos Angular», YouTube. Accedido el 10 de noviembre de 2022. [En línea]. Disponible en: <https://www.youtube.com/watch?v=C5voC7WAEaY>
- [11] «GitLab CI/CD Tutorial for Beginners», YouTube. Accedido el 12 de noviembre de 2022. [En línea]. Disponible en: <https://www.youtube.com/watch?v=qP8kir2GUgo>
- [12] «GitLab complete tutorial for beginners», YouTube. Accedido el 12 de noviembre de 2022. [En línea]. Disponible en: <https://www.youtube.com/watch?v=8aV5AxJrHDg>
- [13] «DevOps with GitLab CI Course – Build Pipelines and Deploy to AWS) », YouTube. Accedido el 14 de noviembre de 2022. [En línea]. Disponible en: <https://www.youtube.com/watch?v=PGyhBwLyK2U>