UNIVERSIDAD NACIONAL DE LA PLATA

FACULTAD DE INFORMÁTICA

TESINA DE LICENCIATURA

Programa de Apoyo al Egreso para Alumnos con Práctica Profesional Supervisada

TÍTULO: Diseño, desarrollo y mantenimiento de Microservicios para el área de Arquitectura de Software de Seguros Rivadavia.

AUTOR/A: Godoy Francisco Manuel.

DIRECTOR/A ACADÉMICO: Marrero Luciano.

DIRECTOR/A PROFESIONAL: Godoy Roberto.

CODIRECTOR/A ACADÉMICO: -

CARRERA: Licenciatura en Sistemas UNLP.

RESUMEN

En este trabajo se abordan aspectos importantes para el desarrollo de soluciones tecnológicas en microservicios, explorando metodologías para soluciones eficientes y escalables. También se destacan los objetivos, ventajas y desventajas de la utilización de microservicios. Se estudian y analizan distintas herramientas y patrones de diseño para el desarrollo de los servicios, como por ejemplo: OpenShift, Kubernetes, Keycloak, MVC, GitLab, Oracle, entre otros.

Además, se hace hincapié en un proyecto que se encuentra actualmente en curso, destacando su enfoque a largo plazo, compromiso con la calidad del software y adopción de nuevas tecnologías. Este proyecto se enfoca en realizar integraciones en donde el equipo de desarrollo debe analizar y evaluar las distintas herramientas que sean necesarias.

Palabras Claves

Gestión de proyectos, Microservicios, Metodologías ágiles, aplicaciones web, Ingeniería de Software y Bases de Datos, Integración de proyectos y servicios en la nube

Conclusiones

Se realizó un relevamiento de los microservicios que se han desarrollado en la empresa y se exponen las tareas más importantes realizadas. Entre ellas se tienen el diseño, en donde se define la información necesaria para el negocio, la implementación del microservicio, en donde se evalúa y define la tecnología aplicada, las pruebas unitarias, el despliegue y la continua comunicación entre el usuario y el equipo de trabajo.

Trabajos Realizados

Se llevó a cabo un programa de capacitación que no solo ha abordado el uso de herramientas específicas, como OpenShift basada en Kubernetes, sino que también ha explorado la implementación y despliegue de microservicios.

Trabajos Futuros

Como trabajo futuro se pretende el desarrollo de aplicaciones móviles. Una para asegurados, otra para productores. Además, se analiza la posibilidad de desarrollar una API Rivadavia y aplicaciones de monitoreo, entre otras herramientas.

Fecha de la presentación: FEBRERO 2024

ÍNDICE

1. Exposición de lo realizado en la PPS

En el marco de la Práctica Profesional Supervisada (PPS) se realizó una capacitación que tuvo una duración aproximada de un mes. En esta capacitación se presentaron temas específicos de cómo utilizar distintas herramientas que serán necesarias para poder implementar los microservicios solicitados por la empresa Seguros Rivadavia [33]. Una de estas herramientas es OpenShift [1], la cual es una plataforma de contenedores basada en Kubernetes [2]. Esta herramienta facilita la implementación y gestión de aplicaciones en contenedores. Además, se abordaron diversos temas sobre el uso de los microservicios y cómo desplegarlos [26] [27].

Luego de la capacitación, se realizó un desarrollo de una nueva aplicación para integrar la empresa con la central telefónica. Este sector se dedica a atender los llamados de clientes para denunciar siniestros, altas de pólizas u otros tipos de consulta.

Las primeras tareas se centran en la investigación sobre la información y los datos que requiere el servicio. Se contactaron personas externas al grupo de trabajo y se plantearon un conjunto de reuniones colaborativas con todas las partes interesadas para analizar y diseñar los cambios según los requerimientos relevados.

Luego del diseño y la definición de las reglas de negocio, se continuó con la implementación de la aplicación en el lenguaje Java [3], utilizando el IDE Eclipse [4]. Se optó por Java [3] porque es un lenguaje con amplia cantidad de recursos. Dentro de sus ventajas para desarrollar microservicios se encuentran:

- **Portabilidad**: Java es conocido por su portabilidad, lo que significa que los microservicios desarrollados en Java pueden ejecutarse en cualquier plataforma que admita la máquina virtual Java (JVM), lo que facilita la implementación en diversos entornos.
- **Amplia comunidad y recursos**: Java cuenta con una comunidad de desarrolladores muy grande y activa, así como una amplia cantidad de recursos, bibliotecas y frameworks diseñados específicamente para el desarrollo de microservicios, como Spring Boot y Micronaut.
- **Rendimiento y escalabilidad**: Java ofrece un rendimiento robusto y una escalabilidad eficiente, lo que permite a los microservicios manejar grandes cargas de trabajo y escalar vertical y horizontalmente según sea necesario.
- **Seguridad**: Java tiene un sólido sistema de seguridad integrado en su diseño, lo que ayuda a proteger los microservicios contra vulnerabilidades y amenazas de seguridad.
- **Madurez y confiabilidad**: Java es un lenguaje de programación maduro y ampliamente utilizado en la industria, lo que significa que los desarrolladores pueden confiar en su estabilidad, soporte a largo plazo y continua evolución.
- **Facilidad de mantenimiento y gestión**: La sintaxis limpia y la amplia disponibilidad de herramientas de desarrollo facilitan el mantenimiento y la gestión de los microservicios escritos en Java, lo que reduce los costos operativos a largo plazo.

Como tarea inicial se crea el repositorio en GitLab [5] para que los integrantes del equipo puedan acceder y ver el avance de la aplicación. Posteriormente, como la empresa ya posee un sistema implementado en el lenguaje de programación Natural

[6], hay que realizar la integración de este sistema con el nuevo sistema desarrollado en Java.

Inicialmente se implementaron dos subprogramas en lenguaje Natural [6] que permiten obtener información de los clientes con seguros de hogar y de los clientes con seguros para el automotor. La integración con la central telefónica, necesitó del uso de estos subprogramas para poder desarrollar las funcionalidades en la nueva aplicación Java.

Esta aplicación, posee dos dispositivos informáticos remotos que se comunican a través de una red a la que están conectados entre sí para envíar y recibir información (endpoints). Un endpoint es una URL (por sus siglas en inglés, Uniform Resource Locators) de una API (código que permite a diferentes aplicaciones comunicarse entre sí y compartir información) que se encarga de contestar a una solicitud del usuario. Es una ubicación digital concreta a la que se envían solicitudes de información con el objetivo de obtener cierta información de respuesta.

Los endpoints especifican los puntos en que una API (Interfaz de Programación de Aplicaciones) puede acceder para conseguir recursos y garantizar el funcionamiento correcto del software en el que se encuentra. Uno de los endpoints es para los clientes que contratan un seguro hogar y otro de los endpoints es para clientes que contratan un seguro para el automotor.

Se realizó la implementación de 2 endpoints para solicitar información al servidor. Por ejemplo, el CUIT/CUIL y la patente del vehículo para los clientes que contraten un seguro para el automotor.

Posteriormente, se define una capa denominada controlador, la cual es una parte de la arquitectura de software para el patrón de diseño Modelo Vista Controlador (MVC) [7]. La función principal de esta capa es actuar como un intermediario entre la interfaz de usuario (la vista) y los datos de la aplicación (el modelo). Luego, se inicia con la implementación de la capa de servicio, en esta capa se definen todos los servicios que brinda la aplicación.

A continuación, en la figura 1 se presenta una captura de pantalla para la estructura de archivos que sigue el patrón MVC [7].

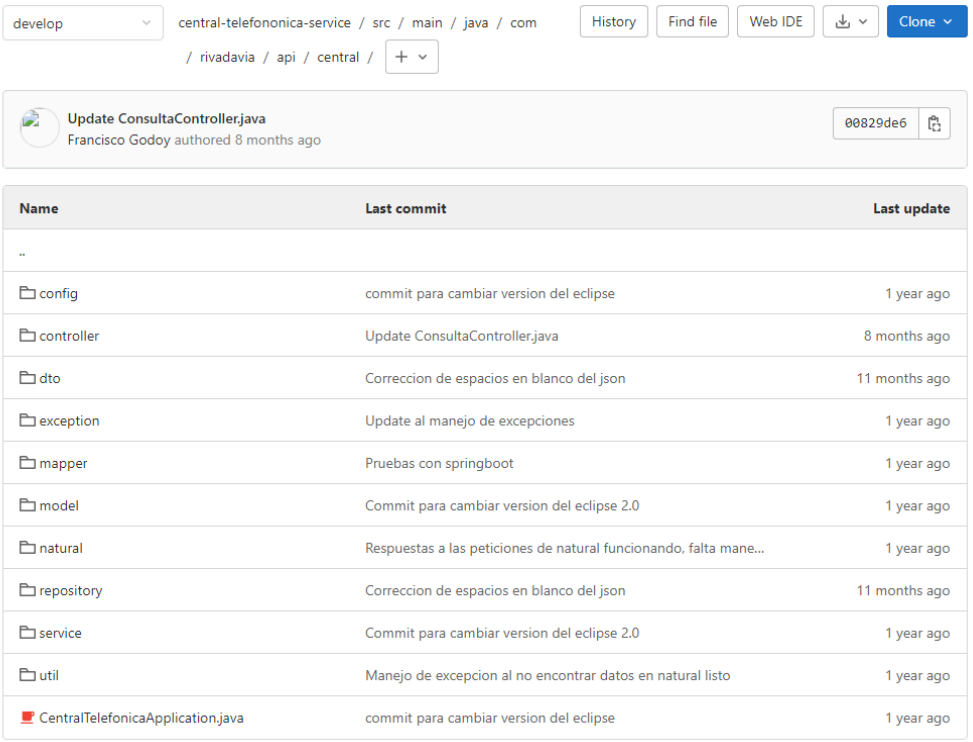

Figura 1. Captura de pantalla del repositorio de la central telefónica en GitLab.

En la carpeta repositorio (repository), es en donde se encuentra la mayor parte de la lógica del código.

Se configura una clase para la conexión con un componente que se utiliza para gestionar y facilitar la comunicación entre diferentes sistemas, en este caso, el objetivo es comunicar la nueva aplicación con el backend de la aplicación anterior. Si existe algún tipo de error se retorna una excepción que identifica dicho error, caso contrario, se genera un archivo con formato JSON de salida.

Una vez implementado el camino que realizan los datos de entrada y los datos de salida, se configura la clase principal (Main) para poder ejecutar la aplicación con la herramienta Spring Boot [8] [35] y así realizar las pruebas necesarias. Estas pruebas consisten en ingresar diferentes datos para monitorear el flujo de la aplicación. Cuando las pruebas finalizan, se actualiza en el repositorio con la versión correspondiente.

Una vez desarrollada la aplicación, se procede a la configuración de un conjunto de archivos especiales denominados manifiestos. A través de estos archivos se puede configurar ciertas características como el uso de la CPU, la memoria, cantidad de pods (contenedor en el que se presenta un conjunto de datos al usuario), volúmenes, namespaces (contenedor abstracto en el que un grupo de uno o más identificadores únicos pueden existir), entre otros. Además, se realizan una serie de pasos automatizados que se ejecutan para compilar, probar y desplegar la aplicación. Esto garantiza la entrega continua y la calidad del software. La integración con un repositorio en el proceso permite un seguimiento colaborativo del código y garantiza que todos los cambios se gestionan de manera efectiva.

Dentro de la plataforma de OpenShift [1] se crea un proyecto que contendrá todas las configuraciones necesarias para la aplicación. También se debe generar un archivo en donde se encuentra el código fuente de la aplicación. Luego, se realizan 4 tareas principales:

- La primera es “Git Clone”. Esta tarea consiste en clonar el código de la aplicación.
- La segunda tarea consiste en obtener la versión de la aplicación que se encuentra en un archivo determinado
- La tercera tarea crea la imagen a partir del código fuente. La imágen es un archivo ejecutable e independiente que se utiliza para crear un contenedor. Esta imagen de contenedor posee las bibliotecas, las dependencias y los archivos que el contenedor necesita para ejecutarse. Los contenedores son paquetes ligeros que incluyen el código de las aplicaciones junto con sus dependencias, como versiones concretas de entornos de ejecución de ciertos lenguajes de programación y bibliotecas indispensables para ejecutar los servicios de software.
- Por último, la cuarta tarea consiste en vincular la imagen para tener un control de versiones, es decir, asignar un valor para determinar la versión más reciente.

Estas cuatro tareas se llevan a cabo a través de un conjunto de procesos a ejecutar (pipeline). A continuación, en la figura 2 se presenta una captura de pantalla de este conjunto de procesos pertenecientes a la aplicación de la central telefónica.

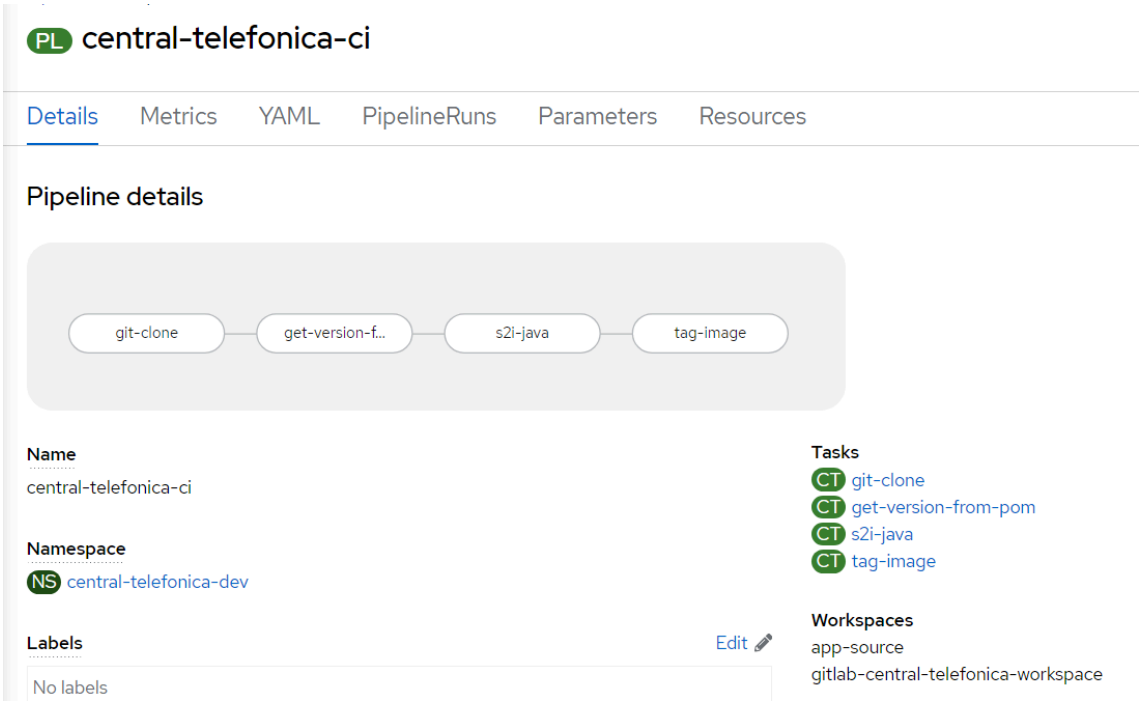

Figura 2. Captura de pantalla del pipeline de la central telefónica.

En OpenShift [1] se tiene un proyecto por cada etapa o ambiente de trabajo (Desarrollo, Testing y Producción). La ventaja de tener estos 3 ambientes distintos es que permite separar las etapas que involucra el proyecto.

Una vez que se tiene todo el desarrollo del software a ejecutar, se finaliza con la parte de Integración Continua (CI, por sus siglas en inglés Continuous Integration), y a continuación se inicia con la parte de Entrega Continua (CD, por sus siglas en inglés, Continuous Delivery).

En la entrega continua se utiliza otra herramienta perteneciente a OpenShift [1], ArgoCD [9]. Esta herramienta permite la creación de pods a partir de la imagen creada previamente. ArgoCD [9] automatiza el despliegue de aplicaciones basadas en Kubernetes [2], garantizando que la aplicación se ejecute correctamente en el entorno de producción. Esto es importante para mantener una infraestructura confiable y actualizada. Además, la configuración detallada de recursos como CPU y memoria en los manifiestos garantiza un uso eficiente de los recursos del clúster OpenShift [1], lo que es esencial para mantener un alto rendimiento y escalabilidad.

A continuación, en la figura 3 se presenta una captura de pantalla para la interfaz de la aplicación para la central telefónica en ArgoCD [9], en donde se visualizan todos los manifiestos y los pods que están en ejecución.

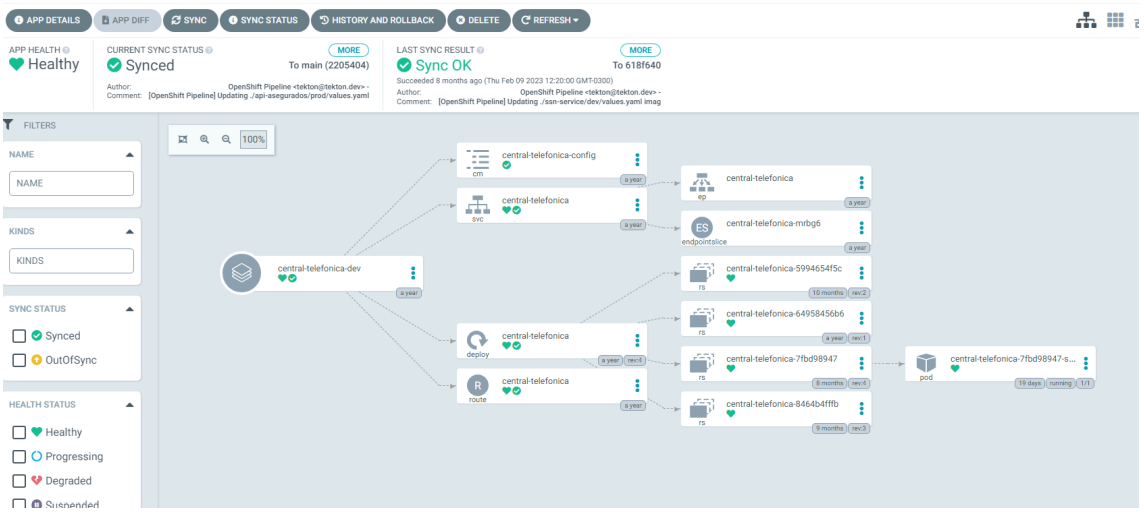

Figura 3. Captura de pantalla de ArgoCD para la central telefónica.

De no existir fallos, se puede acceder a la ruta creada dentro de la intranet de la empresa para probar el servicio correspondiente. Además, se realizan varias pruebas, testeos y monitoreos para detectar posibles errores en la aplicación. Luego de estas pruebas se dará por terminada la aplicación.

Aproximadamente, cada 3 semanas, el equipo de trabajo y los líderes del proyecto (PM, por sus siglas en inglés, Project Manager) se reúnen para realizar la asignación de tareas. Además, semanalmente se realiza una reunión entre el equipo de trabajo y los PM para comunicar los avances en las tareas. En esta reunión se notifica la cantidad de horas que fueron necesarias para la realización de cada tarea.

Estas reuniones son importantes para la comunicación efectiva y la gestión de proyectos. El seguimiento de las horas dedicadas a las tareas es esencial para la gestión del tiempo y la evaluación de la eficiencia en el desarrollo.

Todas las integraciones y tareas que son asignadas pertenecen al proyecto DEI (proyecto actual que se encuentra en desarrollo por parte de la empresa), el cual fue definido a principios del año 2022 y se espera finalizar con las integraciones a mediados del año 2024. Este proyecto refleja un compromiso a largo plazo que involucra un desarrollo de software de alta calidad y la mejora continua de los procesos.

El primer servicio desarrollado fue el de la central telefónica. Este servicio se utiliza en la actualidad para atención al cliente. Luego, se han desarrollado otros servicios. Por ejemplo, la integración con un servicio externo denominado Autoinspector. Este servicio consiste en recibir datos del cliente y realizar validaciones sobre dicha información. Esto es importante para facilitar a los productores y a los asegurados la inspección previa al alta de un seguro.

Para almacenar la información de las inspecciones que se realizan se utiliza una base de datos Oracle [11]. Las imágenes se encuentran almacenadas en AWS S3 (Amazon Web Service) [12]. En la nueva aplicación se deben manipular estas imágenes para almacenarlas en un formato binario en una de las bases de datos existentes en la empresa. Para ello, se implementó un método específico que se ejecuta cada cierto tiempo y obtiene los enlaces de las imágenes que aún no fueron persistidas en la base de datos final.

Como medida de seguridad para algunas aplicaciones de la empresa se utiliza la herramienta Keycloak [13] (software de código abierto que permite el inicio de sesión único con gestión de acceso e identidad dirigido a aplicaciones y servicios modernos). Esta herramienta utiliza Single Sign On (SSO), que es un esquema de autenticación que permite a los usuarios iniciar sesión una vez, utilizando un conjunto único de credenciales de inicio de sesión y obtener acceso seguro a varios servicios y aplicaciones relacionadas durante esa sesión sin necesidad de registrarse nuevamente.

La seguridad es necesaria para aquellas aplicaciones que exponen su URL pública a organizaciones externas, ya que al ser pública puede estar expuesta a una gran cantidad de accesos con mala intención. Por lo tanto, la implementación de una capa de seguridad es una de las tareas más importantes al implementar microservicios.

En el trabajo que se realiza diariamente existe una retroalimentación continua del conocimiento para abordar las problemáticas presentadas. Todas las integraciones realizadas se pueden ver como el desarrollo de microservicios que pueden trabajar independientemente del resto de la aplicación facilitando la división de tareas y el manejo de los errores existentes.

2. Informe Técnico

1. Resumen

El proyecto DEI (sigla designada por el equipo de gestión, la cual no posee ningún significado especial) se destaca por su enfoque a largo plazo, compromiso con la calidad del software y adopción de nuevas tecnologías. La comunicación efectiva y de gestión proactiva del tiempo contribuyen al éxito del proyecto. Este proyecto posee una numerosa lista de integraciones que el equipo de desarrolladores debe realizar utilizando todas las herramientas que se precisen.

Durante la Práctica Profesional Supervisada (PPS), se han estudiado y analizado aspectos para el desarrollo de soluciones tecnológicas en microservicios. Se llevó a cabo un programa de capacitación en el uso de herramientas como OpenShift [32] y Kubernetes [2]. Se han explorado metodologías para soluciones eficientes y escalables. Se han desarrollado aplicaciones en Java [3], utilizando Eclipse [4] y siguiendo el patrón MVC [7].

Se ha desarrollado un microservicio para una central telefónica de la empresa Seguros Rivadavia siguiendo el diseño MVC [7] y usando GitLab [5] para el control de versiones. Se aplicaron CI/CD, y se garantizó la seguridad mediante Keycloak [13] para el inicio de sesión único (SSO) [25].

Se ha implementado Autoinspector, un microservicio que integra información de los vehículos asegurados utilizando Oracle [11] y AWS S3 [12]. Se utilizó SQL Developer [19] para la gestión de la base de datos. Se implementaron webhooks para recibir datos y automatización de tareas en Java [3], empleando MapStruct [21] y JPA [24].

Para garantizar la seguridad, se implementó Keycloak [13] [34] y se exploró 3scale [10] como alternativa para controlar, distribuir y monetizar APIs. Se realizaron pruebas exhaustivas antes de la exposición a producción.

Se utilizó Scrum [15] con Sprints semanales y reuniones regulares. Jira [14] se utilizó para la gestión de tareas y GitLab [5] para el seguimiento del código. La adaptación al trabajo remoto se gestionó mediante reuniones virtuales y herramientas colaborativas.

2. Palabras claves

Gestión de proyectos, Microservicios, Metodologías ágiles, aplicaciones web, Ingeniería de Software, Bases de Datos, Integración de proyectos y servicios en la nube.

3. Contexto

Para una constante comunicación y/o coordinación, se realizan 1 o 2 reuniones semanales con el objetivo de exponer los avances y/o inconvenientes. Generalmente, se realiza una reunión al inicio de la semana y otra al finalizarla.

Para centralizar la información del proyecto, se utiliza GitLab [5] para que haya un seguimiento de versiones y en el caso de que sea necesario se puede descargar los últimos cambios y realizar las modificaciones que correspondan.

Además, se utiliza la herramienta Jira [14], en la cual se especifican los avances en formato de texto. Es decir, en esta herramienta, aparecen los proyectos en el cual un desarrollador está asignado, dentro de esos proyectos las integraciones, y a su vez, cada integración posee su lista de tareas. El líder de proyecto puede asignar semanalmente tareas a los desarrolladores para que realicen durante el transcurso de los días y luego comentar en formato de texto los avances de la misma. También se suele especificar un estado para la tarea, ya sea pendiente, en curso, pausada o finalizada.

Se hace uso de Scrum [15]. Scrum es una metodología de diseño ágil y se basa en realizar sprints semanales (reuniones). El sprint es parte del método Scrum y facilita la división de un proyecto en etapas. Entre los principales beneficios de utilizar Scrum, se encuentra la reducción en tiempos y la no conformidad en las entregas. Scrum se centra en la mejora continua, principio básico de la metodología ágil [36].

Cada 3 semanas aproximadamente se realiza la planificación del sprint. En esta planificación se determinan las tareas a desarrollar o nuevas tareas que no estén en la lista.

El trabajo en equipos pequeños beneficia mantener una comunicación fluida y agiliza la coordinación de tareas. Cada integrante tiene la posibilidad de participar en una tarea de interés o en donde posea experiencia con la tecnología utilizada.

La comunicación del grupo de trabajo se realiza a través de una cuenta de correo electrónico (Gmail). La empresa posee un dominio propio (@segurosrivadavia.com) en donde cada empleado posee su propia cuenta. Además, cuenta con una intranet en donde se encuentran todos los proyectos internos, herramientas y programas que solo son utilizados por algunos empleados de la empresa. Para las reuniones virtuales se utiliza la herramienta “Meet” que brinda la plataforma de Google.

Además, se tiene una plataforma para realizar pedidos por parte del equipo de trabajo, por ejemplo, instalar un nuevo software en una máquina propietaria o solicitar algún tipo de soporte para alguna aplicación específica. Las solicitudes realizadas, poseen una prioridad y una solicitud de alta prioridad deberá ser atendida lo antes posible.

Las prioridades dependen exclusivamente del solicitante. El desarrollador determina con cuánta urgencia necesita determinada solicitud. Por ejemplo, si existe un problema que corresponde a una aplicación que se encuentra en producción se

debe informar con determinada urgencia. Esto se realiza generando un “ticket” o solicitud en donde se especifica prioridad alta, ya que si es un servicio activo podría generar pérdidas considerables.

Las aplicaciones siguen el patrón Modelo Vista Controlador (MVC) [7]. Además, se implementaron capas especiales para garantizar un flujo efectivo de datos. El uso de endpoints para la comunicación remota entre dispositivos informáticos y la creación de nuevas capas de repositorios permitieron un desarrollo estructurado y eficiente.

MVC [7] es un patrón de diseño que considera dividir una aplicación en tres módulos claramente identificables y con funcionalidad bien definida:

- **El modelo (model)**: es un conjunto de clases que representan la información del mundo real que el sistema debe procesar sin tener en cuenta la forma en la que esa información será presentada.
- **La vista (view)**: es el conjunto de clases que se encargan de presentar al usuario final la información contenida en el modelo. Una vista está asociada a un modelo, pudiendo existir varias vistas asociadas al mismo modelo.
- **El controlador (controller)**: se encarga de dirigir el flujo de control de la aplicación ante eventos externos. A partir de estos eventos, el controlador se encarga de modificar el modelo o de abrir y cerrar vistas.

Para ciertas aplicaciones que realizamos, algunas deben utilizar una base de datos temporal. Por lo cual, dentro de la aplicación contamos con el uso de la librería JPA [24] para el manejo de la base de datos.

JPA (Java Persistence API) es una especificación de Java [3] que describe el manejo de datos en aplicaciones Java [3] mediante la persistencia en bases de datos relacionales. La persistencia se refiere a la capacidad de almacenar y recuperar objetos en una base de datos. Proporciona una interfaz de programación de aplicaciones (API) estándar para la gestión de entidades y relaciones en Java [3]. Permite escribir código Java [3] para interactuar con bases de datos de manera más fácil y eficiente, sin tener que preocuparse por detalles específicos de la implementación de la base de datos subyacente. Las principales características y funciones de JPA incluyen: Mapeo Objeto-Relacional (ORM), Operaciones CRUD (Crear, Leer, Actualizar, Eliminar), Consultas JPQL (Java Persistence Query Language), Transacciones, Gestión de relaciones y Portabilidad. [24]

Para algunas otras aplicaciones actualmente contamos con el uso de AWS [12] para el guardado de imágenes en la nube. A través de AWS [12] se pueden almacenar grandes cantidades de imágenes y escalar su capacidad según sea necesario sin preocuparse por la gestión de la infraestructura subyacente. También, garantizar una alta disponibilidad de los datos.

Al usar estos servicios, las imágenes estarán distribuidas automáticamente entre distintos servidores y ubicaciones geográficas, esto garantiza una alta disponibilidad incluso ante situaciones problemáticas en alguno de los servidores.

Además, se han configurado mecanismos de seguridad robustos mediante políticas de acceso y autenticación. AWS [12] ofrece soporte para una variedad de lenguajes de programación, incluido Java [3]. Esto facilita la integración de servicios de almacenamiento sin tener que preocuparse por detalles de implementación complejos.

Algunas de las ventajas de utilizar AWS [12] son:

- **Acceso global**: Los datos almacenados en AWS pueden ser accedidos desde cualquier parte del mundo donde haya conectividad a internet. No importa dónde estén ubicados físicamente los servidores de AWS, los usuarios pueden acceder a sus datos desde cualquier lugar.
- **Acceso remoto**: Los usuarios pueden acceder a sus datos almacenados en AWS de forma remota, lo que significa que no necesitan estar en la misma ubicación física que los servidores de AWS para acceder a sus datos. Esto es especialmente útil en entornos distribuidos o cuando se trabaja de forma remota.
- **Escalabilidad y disponibilidad**: AWS está diseñado para ser altamente escalable y disponible. Esto significa que incluso si hay un aumento repentino en la demanda de acceso a los datos, AWS puede manejarlo y garantizar que los datos estén disponibles de manera rápida y confiable.

Por cuestiones de seguridad, se trabaja a través de una máquina virtual instalada en las computadoras personales de cada empleado y en donde el ingreso es a través de credenciales. Una vez que se inicia sesión se tiene acceso a las herramientas para realizar los desarrollos.

Para trabajos futuros o posibles nuevas estrategias, el equipo suele realizar reuniones presenciales. Estas reuniones son esenciales para el buen funcionamiento y la evolución constante del equipo. En este contexto, las reuniones proporcionan un espacio vital para compartir conocimientos, experiencias y perspectivas entre los miembros del equipo. Además, existe la oportunidad de revisar y mejorar constantemente los procesos existentes.

4. Trabajo Realizado

En este trabajo se abordan diversos aspectos para el desarrollo de soluciones tecnológicas en el ámbito de los microservicios. Se llevó a cabo un programa de capacitación en el uso de herramientas específicas, como OpenShift [1] basada en Kubernetes [31] para la implementación y despliegue de microservicios. En este contexto, se estudiaron diversas metodologías y herramientas esenciales para el desarrollo de soluciones eficientes y escalables [28, 29, 30].

OpenShift [1] es una plataforma unificada para diseñar e implementar aplicaciones según sus necesidades. Además, facilita la implementación y gestión de aplicaciones en contenedores. Permite trabajar de manera rápida e inteligente con un conjunto completo de servicios que brinda la posibilidad de comercializar aplicaciones con la infraestructura que se elija. Es una plataforma de aplicaciones en la nube híbrida impulsada por Kubernetes [2] que combina servicios para reducir los problemas a la hora de desarrollar y gestionar aplicaciones. Es un sistema de código abierto para implementar y administrar aplicaciones alojadas en contenedores. Permite automatizar tareas operativas para la administración de aplicaciones.

A continuación, en la figura 4, se presenta una captura de pantalla de OpenShift [1]. A la izquierda de la imagen se tiene la barra de navegación con las funcionalidades principales.

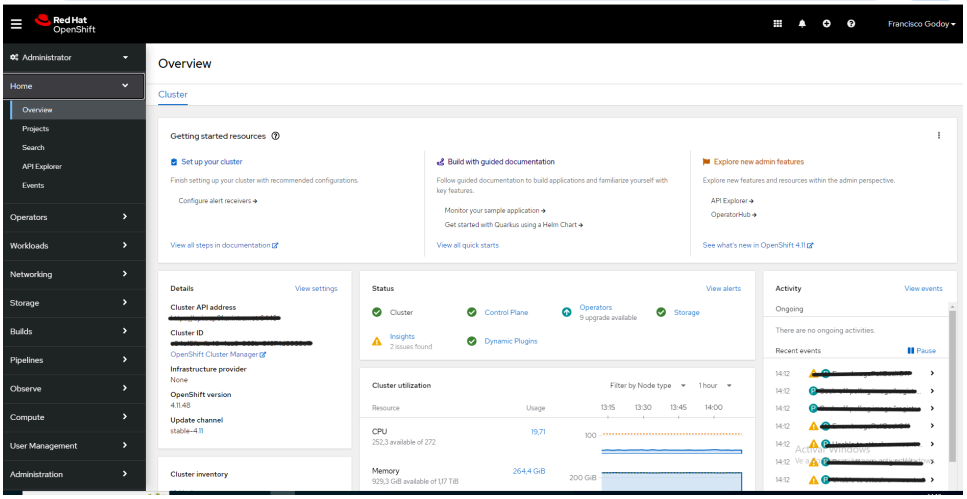

Figura 4. Captura de pantalla de OpenShift. Las partes ocultas pertenecen a datos que son sensibles para la empresa.

La plataforma OpenShift [1] se utilizó para gestionar proyectos en diferentes entornos de trabajo, como desarrollo, prueba y producción. La automatización del proceso de despliegue se llevó a cabo con tareas específicas, como clonar el código, obtener la versión de la aplicación, crear la imagen y vincularla para el control de versiones.

La CI (Integración Continua) y CD (Entrega Continua) es un término general que abarca varias fases de DevOps [16].

La CI es la práctica de integrar cambios de código en un repositorio múltiples veces al día.

La CD posee dos significados:

- La entrega continua, que automatiza las integraciones de código y,
- La implementación continua que entrega automáticamente las versiones finales a los usuarios finales.

Las pruebas de CI/CD reducen los errores y defectos del código, son importantes para el flujo de trabajo. La práctica de CI implica integrar pequeños subconjuntos de cambios en un período corto, en lugar de integrar actualizaciones importantes con menos frecuencia.

Automatizar los flujos de trabajo para probar, fusionar y verificar los cambios en un repositorio compartido significa que los equipos pueden entregar un código más claro. La práctica de la entrega continua significa que los desarrolladores pueden invertir menos tiempo realizando pruebas internas, ya que la práctica garantiza un código estable en la fase de entrega. Se simplifica el proceso de detección de errores, lo que acelera el tiempo de resolución.

Las tareas de CI se realizan dentro de la plataforma OpenShift [1], esto permite manipular los pipelines y otros recursos para la creación de una instancia de la aplicación.

Las tareas de CD, se realizan en la plataforma ArgoCD [9]. Esta es una herramienta de entrega continua para Kubernetes [2]. Utiliza repositorios Git [22] como fuente de origen para definir el estado deseado de la aplicación, es decir la configuración deseada que se le quiera dar al servicio.

Se aplicó la metodología Scrum [15] y se utilizó Jira [14] para especificar avances, asignar tareas y realizar un seguimiento y control del proyecto.

A continuación, en la figura 5 se presenta una captura de pantalla de la plataforma Jira [14] en donde se muestran algunas de las tareas asignadas.

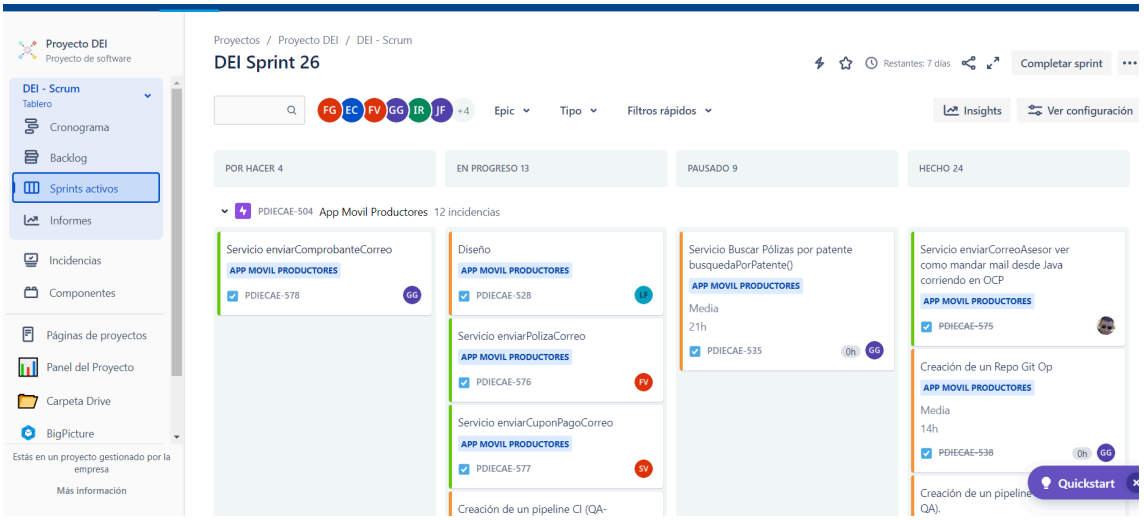

Figura 5. Captura de pantalla de la integración App Móvil Productores en Jira.

El primer microservicio realizado es el de la central telefónica. Se han implementado endpoints específicos para obtener información de clientes con seguros de hogar y automotor. La primera etapa consiste en investigar de forma exhaustiva los datos requeridos y la planificación de reuniones colaborativas con todas las partes interesadas.

El diseño es una tarea importante del desarrollo, ayuda a establecer objetivos y los servicios necesarios. Generalmente el diseño está a cargo del líder del equipo, el cual posteriormente se debate entre los miembros que van a desarrollar el servicio para mejorar el diseño o sugerir cambios para bien. La idea es que se llegue a un acuerdo para llegar a determinar el diseño y que luego en la aplicación no haya problemas. Esta tarea es de las más importantes y no se la debe tomar como algo secundario, si es necesario, la tarea de diseño puede durar hasta 2 o 3 semanas, ya que un buen diseño implica mayor facilidad a la hora de implementar el código.

A continuación, en la figura 6, se presenta el diseño realizado para el servicio que genera un archivo con formato pdf que luego será enviado o presentado al cliente.

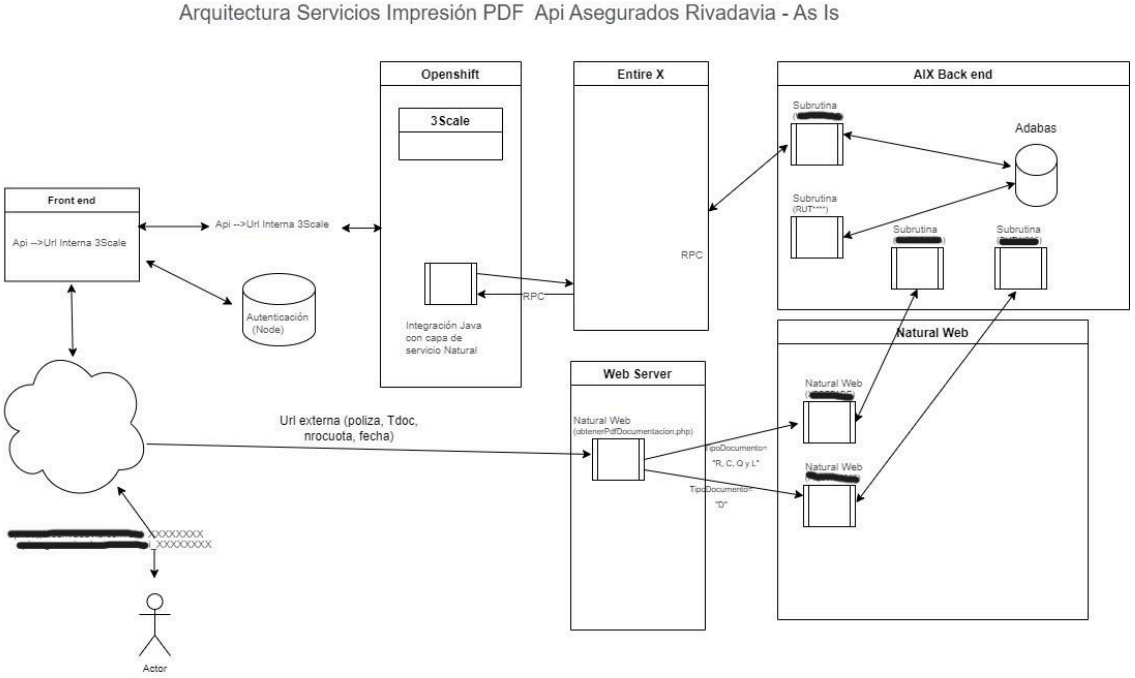

Figura 6. Diseño para la generación de archivos con formato pdf. Las partes ocultas pertenecen a datos que son sensibles para la empresa.

Luego de realizar el diseño, se procede con el desarrollo del código. La aplicación se estructura mediante el diseño de endpoints específicos para clientes con seguros de hogar y automotor. Se implementaron programas específicos para las funcionalidades requeridas. Además, se siguió el patrón de diseño Modelo Vista Controlador dividiendo la aplicación en capas para mejorar la organización y mantenimiento del código.

Se configura la aplicación mediante la creación de clases para la conexión con componentes externos y la gestión de errores. Se emplea Spring Boot [8] para ejecutar la aplicación y realizar las pruebas exhaustivas. La interfaz OpenApi [18] proporciona una forma amigable de probar los servicios y verificar su correcto funcionamiento.

A continuación, en la figura 7, se presentan dos servicios pertenecientes a la central telefónica. Para /clienteHogar se solicita enviar el DNI, mientras que para /clienteAutomotor se solicita enviar el DNI y la patente del automotor.

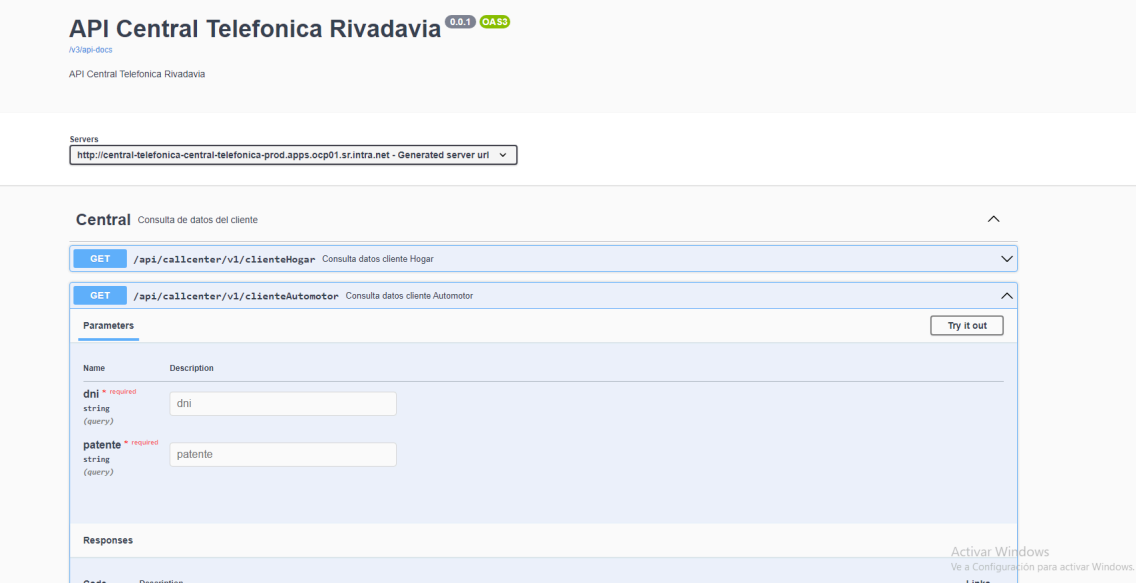

Figura 7. Captura de pantalla de Swagger del servicio Central Telefónica API.

Una vez desarrollada la aplicación, se realizó la configuración de los archivos de manifiestos. En estos archivos se tienen que definir características como el uso de CPU, memoria, cantidad de pods, entre otros. Estos archivos sirven para automatizar tareas relacionadas con la compilación, las pruebas y el despliegue de la aplicación.

En OpenShift [1] se crea el proyecto que contiene todas las configuraciones necesarias para la aplicación. Estas configuraciones incluyen la creación de archivos específicos para las tareas del despliegue. La ejecución de estas tareas se realiza a través de un conjunto completo de procesos.

Una vez que finaliza el despliegue y las pruebas se realiza un seguimiento para determinar si existe algún problema.

Otro de los microservicios desarrollados es Autoinspector. Este microservicio permite la recepción de datos, incluyendo formularios y fotos de vehículos, facilitando la evaluación de un seguro. Este microservicio se basa en inteligencia artificial y verifica la información proporcionada por los clientes para brindar validaciones esenciales en el proceso de aseguramiento.

Como motor base de datos se utilizó Oracle [11] y las imágenes se almacenaron en AWS S3 [12], diseñando un método específico para manipular tales imágenes.

La interacción con el motor de base de datos Oracle, se realizó mediante SQL developer [19].

El microservicio Autoinspector posee webhooks [20]. Un webhook es un mensaje automatizado que se envía a una aplicación externa cuando ocurre un evento. Es el retorno de una llamada HTTP o una petición HTTP POST generada por la notificación de un evento. Esta petición HTTP es registrada y se utiliza para almacenar datos en formato JSON.

Una de las tareas realizadas, es la configuración del webhook. La aplicación cliente proporciona una URL a la API del servidor y especifica el evento sobre el cual desea obtener información. Luego, el cliente ya no tiene necesidad de realizar nuevamente más consultas al servidor, dado que este le enviará de forma automática cuando suceda el evento especificado.

A continuación, en la figura 8, se presenta una captura de pantalla de la configuración de los webhooks realizada. El microservicio Autoinspector posee 5 eventos (creación de inspección, inició de Inspección, inspección completa, imagen procesada e inspección bloqueada). Por lo tanto, se pueden añadir URLs para que automáticamente notifique a la aplicación ante cualquiera de estos eventos.

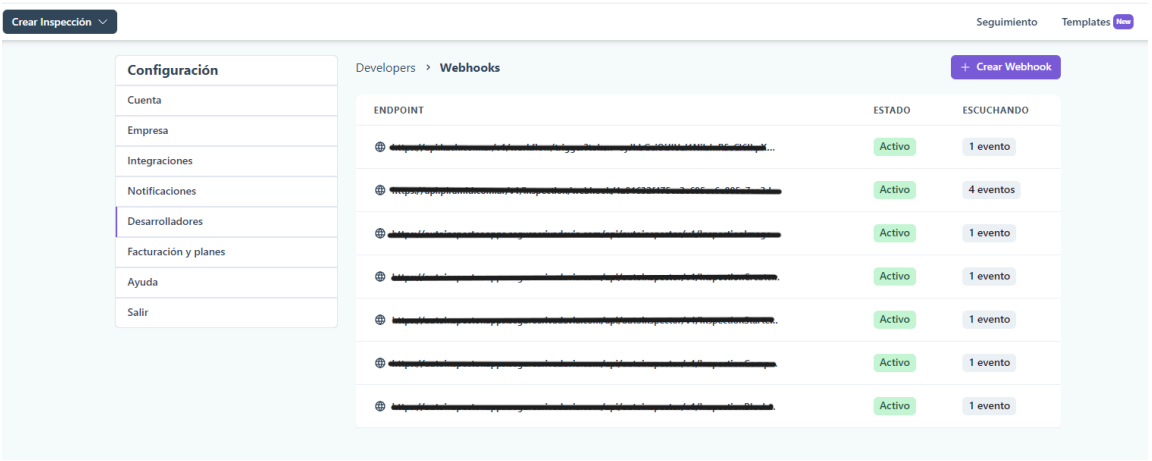

Figura 8. Captura de pantalla de la configuración de webhooks de Autoinspector. Las partes ocultas pertenecen a datos que son sensibles para la empresa.

También, se cuenta con la posibilidad de tener una URL con todos los eventos configurados y no tener una URL por cada evento. Antes de trabajar con los webhooks, se realizó una investigación exhaustiva sobre el funcionamiento de los mismos. Además, se investigó sobre la automatización de tareas. En este caso, cada minuto, se realiza una consulta a la base de datos para obtener cierta información. Este proceso automático se realizó en Java [3]. Este proceso se ejecuta cada minuto para enviar a través de los webhooks los enlaces a las imágenes, estos enlaces expiran cada 5 minutos. Por lo tanto, se debe evitar que esto suceda, en caso contrario se vuelve a realizar una petición de la información correspondiente.

Otra tarea realizada es la de mapear los datos que vienen en determinado formato desde Autoinspector para transformarlos y almacenarlos en la base de datos temporal Oracle y en la base de datos del sistema Natural [6] donde se guardan de manera permanente. Para ello, se utilizó la librería MapStrut [21]. Está librería posee anotaciones que permiten mapear los campos de una clase a otra, especificando si cambia el tipo de dato y si cambia el nombre. De no existir cambios, automáticamente realiza el mapeo y transfiere el dato.

Una vez que ha finalizado el desarrollo del código, hay que garantizar la seguridad de la aplicación. Para ello, se configuró la herramienta Keycloak [13], un software de código abierto que facilita el inicio de sesión único (SSO). El SSO es un esquema de autenticación que permite a los usuarios iniciar sesión una vez y obtener

acceso seguro a varios servicios sin necesidad de volver a registrarse. Implementar esta medida de seguridad, es esencial para las URLs públicas a organizaciones externas.

A continuación, en la figura 9, se presenta la interfaz del SSO de RedHat, la cual es utilizada para la seguridad de algunas de las aplicaciones de la empresa.

Figura 9. Captura de pantalla del SSO de RedHat. Las partes ocultas pertenecen a datos que son sensibles para la empresa.

SSO funciona iniciando sesión en un sistema denominado Proveedor de Identidad (IdP). Durante este proceso, el IdP verifica las credenciales del usuario (nombre de usuario y contraseña). Si la autenticación es exitosa, el IdP genera un token de sesión que contiene información sobre la identidad del usuario y otros atributos relevantes.

Cuando un usuario intenta acceder a otro sistema o aplicación que forma parte del mismo entorno de SSO, el sistema solicita que valide el token de sesión del usuario. Si el token es válido y aún no ha expirado, se confirma la identidad del usuario y permite el acceso sin requerir que el usuario vuelva a ingresar sus credenciales [25].

Otra alternativa, es el desarrollo de aplicaciones de seguridad a través de 3scale [10] de RedHat. Esta herramienta ayuda a las organizaciones a controlar y distribuir sus APIs. Además, permite la administración y el acceso a las APIs para supervisar su uso, aplicar políticas de seguridad y realizar un análisis detallado. La integración de OpenShift [1] y 3scale [10] facilitó la gestión de APIs en el entorno basado en contenedores.

A continuación, en la figura 10, se presenta la interfaz de 3Scale [10] y parte de la configuración para asignar una clave al servicio de la central telefónica. Es decir, en el caso de exponer una URL pública, para lograr acceder al servicio se requiere de la clave administrada por 3Scale [10].

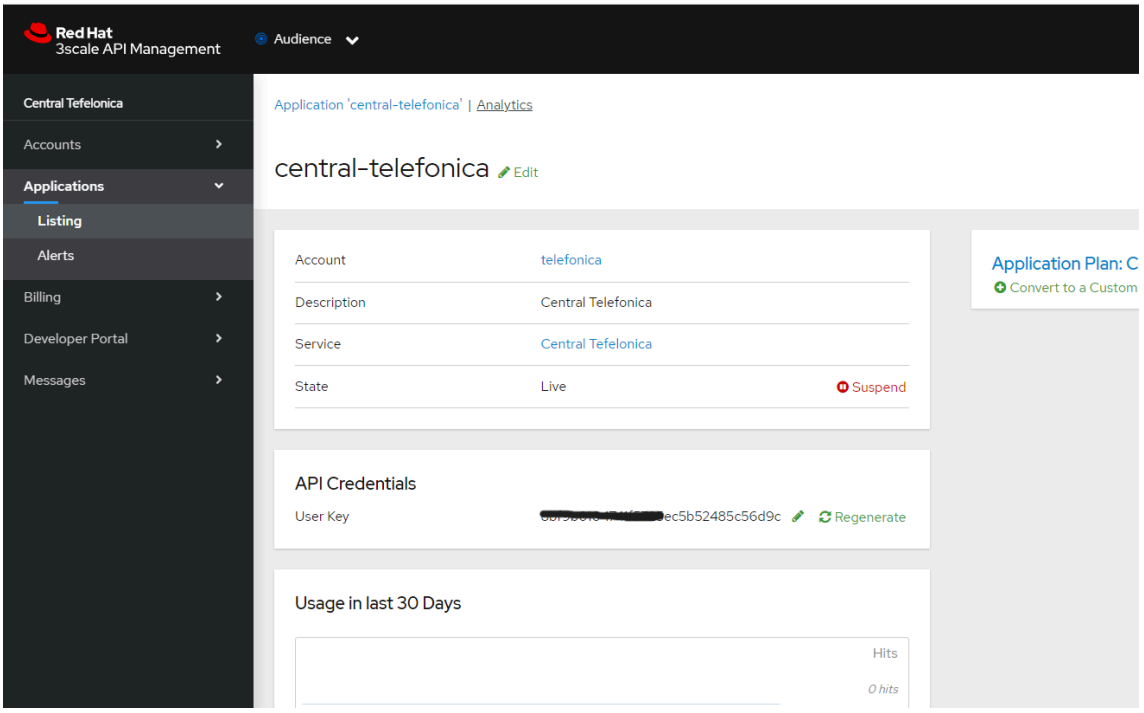

Figura 10. Captura de pantalla del 3Scale de RedHat. Las partes ocultas pertenecen a datos que son sensibles para la empresa.

Entre las funcionalidades configuradas en 3scale [10] se tiene el control de acceso a la API. Además, 3scale [10] ofrece herramientas analíticas que permiten obtener información detallada sobre el uso de la API y se pueden generar informes personalizados para comprender cómo se utilizan los servicios y tomar decisiones al respecto.

Al integrar 3scale [10] con OpenShift [1] facilitó la escalabilidad de los servicios según sea necesario. Se aprovechan las capacidades de administración de contenedores para gestionar de manera eficiente el despliegue y la escalabilidad. La diferencia entre 3scale [10] y SSO es que no hace falta la obtención de un token, con la clave del usuario es posible acceder al servicio. En este caso, SSO es más seguro que 3scale [10], pero este requiere la generación del token previamente.

Además, existen aplicaciones desarrolladas las cuales no poseen ningún tipo de seguridad. Estas son aplicaciones que se utilizan dentro de la intranet de la empresa y no es posible acceder desde un navegador estando por fuera de la misma. Estos servicios o aplicaciones se utilizan para trasladar información interna. Por ejemplo, una de las aplicaciones desarrolladas realiza el alta de casos en los que existe una destrucción total o parcial de los bienes asegurados.

El flujo en la aplicación inicia cuando el cliente solicita realizar la denuncia por algún siniestro. A través de una serie de 6 pasos, se solicitan los datos necesarios. No obstante, existe un paso 0 que no depende del tipo de siniestro ocasionado.

En la aplicación se desarrollan 7 endpoints, uno por cada paso. A medida que el cliente completa los datos solicitados se genera la información en el formato correspondiente para que sea almacenada en la base de datos.

Los endpoints del tipo POST van a recibir datos que serán almacenados. Recopilan la información de todos los pasos y la envían a través de la herramienta de OpenFeign [40] a otra aplicación propia de la empresa. Esta aplicación retorna un número para identificar el siniestro para el cliente.

En este caso, no es necesario aplicar seguridad a la aplicación que posee los endpoints para los 7 pasos, ya que será accedida y utilizada solamente por otras aplicaciones pertenecientes a la empresa. Además, fue necesario realizar la división de cada uno de los pasos por tipo de siniestro. En cada paso existen diferentes tipos de siniestros, lo que implica que por ejemplo en el primer paso puede llegar un conjunto de datos relacionados a un siniestro de tipo cristales, como también un conjunto de datos relacionado a un siniestro de tipo incendio. Estos datos pueden diferir en algunos campos como también pueden ser similares.

Es importante el diseño de la aplicación para poder obtener de forma discriminada los datos necesarios para cada tipo de siniestro. En esta tarea, se utilizó JsonSubTypes [41] para mapear correctamente los datos correspondiente a cada paso.

Se utiliza la notación @JsonSubTypes, esta es una anotación proporcionada por la biblioteca Jackson en Java para la serialización y deserialización de objetos JSON. Permite definir subtipos de una clase abstracta o de una interfaz y asociar estos subtipos con nombres o identificadores específicos en el JSON. Esto es útil cuando se trabaja con jerarquías de clases o interfaces y se necesita conservar la información sobre el tipo de objeto al serializar y deserializar.

```
    @JsonTypeInfo(use = JsonTypeInfo.Id.NAME, include = JsonTypeInfo.As.PROPERTY, property = "tipoSiniestro")
@JsonSubTypes({
            @JsonSubTypes.Type(name = "Cerraduras.",value = CerraduraDTO.class ),
            @JsonSubTypes.Type(name = "Cristales.",value = CristalDTO.class ),
            @JsonSubTypes.Type(name = "Daños por estacionamiento.",value = UbicacionDTO.class ),
            @JsonSubTypes.Type(name = "Granizo.",value = BasePaso1DTO.class ),
            @JsonSubTypes.Type(name = "Inmersión por inundación.",value = InmersionDTO.class ),
            @JsonSubTypes.Type(name = "Incendio.",value = IncendioDTO.class ),
            @JsonSubTypes.Type(name = "Robo total sin aparición de la unidad.",value = RoboTotalSinAparicionDTO.class ),
            @JsonSubTypes.Type(name = "Robo total con aparición posterior de la unidad.",value = RoboTotalConAparicionDTO.class ),
            @JsonSubTypes.Type(name = "Robo parcial.",value = RoboParcialDTO.class ),
            @JsonSubTypes.Type(name = "Accidente y/o Daños por choques, vuelcos, etc.",value = AccidenteParcialDTO.class )})
protected T data;
```

Figura 11. Implementación de una clase Java para definir los tipos de siniestros utilizando @JsonSubTypes.

Esta implementación permite definir una clase genérica que depende del tipo de siniestro. Con esta anotación se pueden definir todos los tipos de siniestros por un nombre y la especificación de la clase que va a implementar dicha clase. El controlador recibe una instancia de esta clase y dependiendo el valor del tipo de siniestro, hará un mapeo interno a la clase específica que corresponda.

Otra de las tareas que se está realizando es la utilización de aplicaciones de monitoreo. Prometheus [38] es un sistema de código abierto para el monitoreo de

aplicaciones. Permite recopilar métricas en tiempo real, almacenarlas de manera eficiente y proporciona consultas flexibles y potentes.

Grafana [39] es una herramienta de código abierto y permite visualizar métricas que se integran con varios sistemas de monitoreo, incluido Prometheus [38]. Proporciona paneles de control dinámicos y flexibles que permiten a los usuarios crear visualizaciones personalizadas de los datos.

Ventajas de monitorizar aplicaciones Spring Boot con Prometheus [38] y Grafana [39]:

- **Visibilidad en tiempo real**: Prometheus permite recopilar métricas en tiempo real. Permite tener una visión actualizada del estado de la aplicación y los sistemas subyacentes.
- **Alertas proactivas**: Prometheus puede configurarse para generar alertas basadas en umbrales predefinidos o patrones de comportamiento anómalo. Esto permite identificar y solucionar problemas antes de que afecten a los usuarios finales.
- **Análisis detallado**: Grafana proporciona una amplia gama de opciones de visualización que permiten a los usuarios analizar datos de manera detallada y comprender el rendimiento y comportamiento de la aplicación.
- **Escalabilidad**: Prometheus y Grafana están diseñados para escalar horizontalmente. Ambas herramientas pueden administrar grandes volúmenes de datos y proporcionar un buen rendimiento en entornos de producción de gran escala.
- **Compatibilidad con contenedores y orquestadores**: Spring Boot, Prometheus y Grafana son compatibles con contenedores como Docker y Kubernetes, lo que facilita su implementación y gestión para este tipo de entornos.

A continuación podemos ver una interfaz de Grafana en donde podemos encontrar varias de las funciones que ofrece la herramienta para el monitoreo de las aplicaciones. [Figura 12]

Figura 12. Interfaz principal de Grafana. La imagen no pertenece a Seguros Rivadavia, fue extraída de internet (https://grafana.com)

Conclusiones generales del trabajo realizado

Este trabajo proporciona una experiencia integral y un marco sólido para el desarrollo de habilidades técnicas. En el transcurso de este proceso, se abordaron diversas áreas, desde la investigación inicial de requisitos y la selección de herramientas hasta la implementación, las pruebas y el despliegue continuo de microservicios.

El enfoque principal fue el desarrollo de servicios eficientes y escalables para satisfacer las necesidades de la empresa y los clientes. La capacitación inicial sobre el uso de herramientas como OpenShift[1] y Kubernetes [2], sentó las bases para una comprensión de la infraestructura necesaria para desplegar microservicios.

El énfasis en la seguridad, reflejado en la implementación de SSO a través de Keycloak [13], subraya la importancia de salvaguardar los servicios implementados, especialmente en un entorno donde la exposición a redes externas es una consideración crítica.

La implementación de microservicios, desde el diseño hasta las pruebas y la integración con sistemas existentes, aporta capacidades para el trabajo en ambientes heterogéneos y complejos. La elección del lenguaje Java [3] y la integración con el motor de bases de datos Oracle [11] resaltan la adaptabilidad y la consideración cuidadosa de las tecnologías existentes en el entorno de la empresa.

La metodología ágil Scrum [15] proporcionó un marco de trabajo sólido, permitiendo entregas regulares y una colaboración efectiva en equipos pequeños. La planificación de sprints y la comunicación constante en reuniones semanales aseguraron un progreso continuo y una alineación con los objetivos del proyecto.

La incorporación de nuevas tecnologías, como ArgoCD [9] para la entrega continua, ilustra el compromiso con la innovación y la mejora continua. Las reuniones presenciales periódicas para discutir nuevas herramientas y estrategias de arquitectura subrayan el esfuerzo por mantenerse al día con las tendencias y optimizar los procesos de desarrollo.

La experiencia de trabajar en un entorno remoto, con reuniones virtuales regulares y una infraestructura de colaboración digital, resalta la capacidad para adaptarse a los desafíos contemporáneos de la vida laboral. La combinación de GitLab [5], Jira [14] y otras herramientas facilitó una gestión eficiente del código, la comunicación y la asignación de tareas.

5. Líneas Futuras

Desarrollo de una App móvil para asegurados: actualmente ya existe una App, pero se proyecta la implementación de una aplicación móvil totalmente nueva. Se implementará en Java y estará disponible para ser descargada tanto en iOS como en Android.

Desarrollo de una App móvil productores: se proyecta implementar una aplicación móvil para los productores de la empresa desde donde podrán operar mediante diferentes funcionalidades.

Desarrollo de una API Rivadavia: consiste en el desarrollo de una API general la cual contenga servicios que sean reutilizables. El objetivo es minimizar la duplicación de código para diferentes servicios.

Aplicaciones de monitoreo: el objetivo es desarrollar monitoreos de una manera legible y sencilla para los microservicios.

6. Referencias Bibliográficas:

1. OpenShift: https://docs.openshift.com/ (Septiembre 2023)
2. Kubernetes: https://kubernetes.io/docs/home/ (Septiembre 2023)
3. Java: https://www.oracle.com/ar/java/technologies/javase/jdk11-archive-downloads.html (Septiembre 2023)
4. Eclipse: https://www.eclipse.org/?gclid=CjwKCAjwgsqoBhBNEiwAwe5w08WLK6JkuE8y_IsKtvJERQBqgCqjVWtUUdDKs7xTCtulS70bZJmRrxoC_zAQAvD_BwE (Octubre 2023)
5. GitLab: https://about.gitlab.com/ (Octubre 2023)
6. Natural: https://es.wikipedia.org/wiki/Natural_(lenguaje_de_programaci%C3%B3n) (Septiembre 2023)
7. MVC: https://www.campusmvp.es/recursos/post/que-es-el-patron-mvc-en-programacion-y-por-que-es-util.aspx (Octubre 2023)
8. Spring Boot: https://docs.spring.io/spring-boot/docs/current/reference/htmlsingle/ (Septiembre 2023)
9. ArgoCD: https://argoproj.github.io/cd/ (Octubre 2023)
10. 3Scale: https://www.redhat.com/es/technologies/jboss-middleware/3scale (Octubre 2023)
11. Oracle: https://www.oracle.com/ar/database/ (Octubre 2023)
12. AWS S3: https://aws.amazon.com/es/free/?all-free-tier.sort-by=item.additionalFields.SortRank&all-free-tier.sort-order=asc&awsf.Free%20Tier%20Categories=categories%23storage&trk=21bd62b1-8884-4ab6-99ad-4add1b4d4da1&sc_channel=ps&ef_id=CjwKCAjwgsqoBhBNEiwAwe5w03oGg6y5xZX8310LdfWEyhX7840YKp75eBNS5fkFu_MXybeNvAgI_xoCovcQAvD_BwE:G:s&s_kwcid=AL!4422!3!648041610136!e!!g!!aws%20s3!19658922915!146731952875&awsf.Free%20Tier%20Types=*all (Octubre 2023)
13. Keycloak: https://www.keycloak.org/ (Octubre 2023)
14. Jira: https://www.atlassian.com/es/software/jira?&aceid=&adposition=&adgroup=143040524765&campaign=19324540226&creative=642190148804&device=c&keyword=jira&matchtype=e&network=g&placement=&ds_kids=p74609443754&ds_e=GOOGLE&ds_eid=700000001558501&ds_e1=GOOGLE&gad_source=1&gclid=CjwKCAjwv-2pBhB-EiwAtsQZFK3p7SVAcCLS_wwzzIdsoJQ4UIVo2klZo7X6hfhaCbuvyNGkvvsf-hoCyKsQAvD_BwE&gclsrc=aw.ds (Octubre 2023)
15. Scrum: https://asana.com/es/resources/what-is-scrum (Octubre 2023)

16. DevOps: https://www.redhat.com/es/topics/devops/devops-engineer#:~:text=El%20ingeniero%20de%20DevOps%20incorpora,el%20mantenimiento%20y%20las%20actualizaciones (Octubre 2023)
17. RedHat: https://www.redhat.com/es (Octubre 2023)
18. OpenApi: https://www.openapis.org/ (Octubre 2023)
19. SQL Developer: https://www.oracle.com/ar/database/sqldeveloper/ (Noviembre 2023)
20. Webhook: https://www.redhat.com/es/topics/automation/what-is-a-webhook (Noviembre 2023)
21. MapStruct: https://www.arquitecturajava.com/java-mapping-con-mapstruts-y-anotaciones/ (Noviembre 2023)
22. Git: https://git-scm.com/ (Noviembre 2023)
23. Spring Scheduler: https://windoctor7.github.io/Tareas-con-Spring-Scheduler.html (Noviembre 2023)
24. JPA: https://www.ibm.com/docs/es/was-liberty/nd?topic=overview-java-persistence-api-jpa (Noviembre 2023)
25. SSO: https://blog.hubspot.es/website/que-es-sso (Diciembre 2023)
26. **Red Hat OpenShift I: contenedores & Kubernetes**: Zach Gutterman, Dan Kolepp, Eduardo Ramirez Ronco, Jordi Sola Alaball, Richard Allred, Michael Jarrett, Harpal Singh, Federico Fapitalle, Maria Fernanda Ordonez Casado. do180.pdf
27. **Cloud-native API Administration with Red Hat 3scale API Management:** Alejandro Serna-Borja, Enol Alvarez de Prado, Marek Czernek, Guy Bianco IV, Iván Chavero. do240.pdf
28. **Red Hat OpenShift Administration II: Operating a Production Kubernetes Cluster:** Zach Gutterman, Dan Kolepp, Eduardo Ramirez Ronco, Jordi Sola Alaball, Richard Allred, Michael Jarrett, Harpal Singh, Federico Fapitalle, Maria Fernanda Ordonez Casado, Andres Hernandez, Ivan Chavero. do280.pdf
29. **Red Hat OpenShift Development II: Containerizing Applications:** Zach Gutterman, Richard Allred, Ricardo Jun, Ravishankar Srinivasan, Fernando Lozano, Ivan Chavero, Dan Kolepp, Jordi Sola Alaball, Manuel Aude Morales, Eduardo Ramírez Martínez, Guy Bianco, Randy Thomas, Marek Czernek. do288.pdf
30. **Red Hat OpenShift Administration III : Scaling Kubernetes Deployments in the Enterprise:** Alejandro Coma, Alex Corcoles, Ivan Chavero, Federico Fapitalle, Andres Hernandez, James Mighion, Joel Birchler, Michael Phillips, Christopher Caillouet, Harpal Singh, Razique Mahroua, Dan Kolepp. do380.pdf
31. **Introducción a Kubernetes:** **https://medium.com/@diego.coder/introducci%C3%B3n-a-kubernetes-b23fee249254 (diego.coder)** (Febrero 2024)
32. **¿Por qué elegir Red Hat OpenShift? Características y beneficios:** **https://www.chakray.com/es/por-que-elegir-red-hat-openshift-caracteristicas-y-beneficios/ (Amanda Vallès Garrido)** (Febrero 2024)
33. **Cómo crear microservicios con Red Hat:** **https://www.chakray.com/es/como-crear-microservicios-con-red-hat-quarkus/ (Daniel Santiago Blanco Cuadrado)** (Febrero 2024)
34. **Keycloak - Identity and Access Management for Modern Applications:** Stian Thorgersen, Pedro Igor Silva.

35. **Beginning Spring Boot 3: Build Dynamic Cloud-Native Java Applications and Microservices:** K. Siva Prasad Reddy, Sai Upadhyayula.
36. **Métodos Ágiles. Scrum, Kanban, Lean:** Carmen Lasa Gómez, Alonso Álvarez García, Rafael de las Heras del Dedo
37. Amazon: https://www.amazon.com (Febrero 2024)
38. Prometheus: https://prometheus.io (Marzo 2024)
39. Grafana: https://grafana.com (Marzo 2024)
40. OpenFeign: https://spring.io/projects/spring-cloud-openfeign (Marzo 2024)
41. JsonSubTypes:https://www.baeldung.com/java-jackson-polymorphic-deserialization (Marzo 2024)