ENTREGABLE 2-B

**MATRIZ DE RIESGO,**

**REGLAS, INDICADORES Y PERFILES DE RIESGOS DE LA DGA**



**Ventanilla Única de Comercio Exterior 2.0**

**Gestor de Filenet**

**Documento Manual Tecnico Desarrollo**

**Versión 1.0**

**9 de enero de 2025**

#### Historial de Versiones

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Versión** | **Fecha** | **Descripción** | **Elaborado por** | **Revisado por** |
| **1.0** | 26/12/24 | Elaboración del documento | Miguel Montoya |  |

Tabla de contenido

[Introducción 7](#_Toc187337476)

[1. Objetivo 7](#_Toc187337477)

[2. Alcance 7](#_Toc187337478)

[3. Glosario de Términos 7](#_Toc187337479)

[4. Infraestructura Tecnológica 8](#_Toc187337480)

[4.1 Sistemas operativos de servidores 8](#_Toc187337481)

[4.2 Sistemas de gestión de base de datos 9](#_Toc187337482)

[4.3 Servidor Web 10](#_Toc187337483)

[4.4 Herramientas de codificación 10](#_Toc187337484)

[5 Arquitectura del Sistema 11](#_Toc187337485)

[5.1 Arquitectura lógica 11](#_Toc187337486)

[5.2 Arquitectura física 12](#_Toc187337487)

[5.2.1 Arquitectura física del ambiente de desarrollo 13](#_Toc187337488)

[5.2.2 Arquitectura física del ambiente de certificación y capacitación 13](#_Toc187337489)

[5.2.3 Arquitectura física del ambiente de producción 14](#_Toc187337490)

[6 Marco de Trabajos, Librerías y Estándares 15](#_Toc187337491)

[6.1 Marco de trabajo 15](#_Toc187337492)

[6.1.1 Marco de trabajo para los proyectos de microservicios 16](#_Toc187337493)

[6.2 Librerías 17](#_Toc187337494)

[6.2.1 Librerías para los proyectos de microservicios 17](#_Toc187337495)

[6.3 Estándares 18](#_Toc187337496)

[7 Estructuración de Código Fuente 18](#_Toc187337497)

[7.1 Proyectos de código fuente de microservicios 18](#_Toc187337498)

[7.2 Proyectos en GIT 18](#_Toc187337499)

[7.3 Proyectos en Sonarqube 19](#_Toc187337500)

[7.4 Proyectos en Jenkins 19](#_Toc187337501)

[7.5 Proyectos en OpenShift 19](#_Toc187337502)

[8 Herramienta para la Construcción y Gestión del Proyecto 20](#_Toc187337503)

[8.1 Archivos de configuración generados por Gradle 20](#_Toc187337504)

[8.2 Archivo build.gradle 21](#_Toc187337505)

[8.3 Estructura de proyecto de código fuente microservicios 22](#_Toc187337506)

[9 Base de Datos 24](#_Toc187337507)

[9.1 BD PostgreSQL 24](#_Toc187337508)

[9.1.1 Esquemas de base de datos 24](#_Toc187337509)

[9.1.2 Acceso a base de datos 24](#_Toc187337510)

[9.1.3 Modelo de base de datos 25](#_Toc187337511)

[10 Funcionalidad del Sistema 25](#_Toc187337512)

[10.1 Requerimientos funcionales 25](#_Toc187337513)

[10.2 Requerimientos no funcionales 25](#_Toc187337514)

[11 Componentes y Despliegue 27](#_Toc187337515)

[11.1 Diagramas de componentes 27](#_Toc187337516)

[11.2 Diagrama de despliegue 28](#_Toc187337517)

[11.3 Deployment 30](#_Toc187337518)

[11.4 Service e Ingress 31](#_Toc187337519)

[11.5 JenkinsFile 32](#_Toc187337520)

[11.6 DockerFile 33](#_Toc187337521)

[11.7 Canalización de trabajos para el despliegue en Jenkins 34](#_Toc187337522)

[11.8 Configuración en openShift 34](#_Toc187337523)

[11.9 Configuración de microservicio Config 34](#_Toc187337524)

[Anexo A: Archivos de Configuración de Spring Boot 35](#_Toc187337525)

[A-1: Microservicio vuce-filenet-api 35](#_Toc187337526)

[A-2: Microservicio vuce-filechunk-api 35](#_Toc187337527)

[Anexo B: Archivos build.gradle 35](#_Toc187337528)

[A-1: Microservicio vuce-filenet-api 35](#_Toc187337529)

[A-2: Microservicio vuce-filechunk-api 35](#_Toc187337530)

#### Índice de tablas

[Tabla 1. Glosario de términos 7](#_Toc187337531)

[Tabla 2. Sistemas operativos de servidores 9](#_Toc187337532)

[Tabla 3. Resumen de herramientas de desarrollo 10](#_Toc187337533)

[Tabla 4. Librerías principales de los proyectos de microservicios 17](#_Toc187337534)

[Tabla 5. Estándares de desarrollo 18](#_Toc187337535)

[Tabla 6. Proyectos de microservicios 18](#_Toc187337536)

[Tabla 7. Proyectos en GIT 18](#_Toc187337537)

[Tabla 8. Proyectos en Sonarqube 19](#_Toc187337538)

[Tabla 9. Proyectos en Jenkins 19](#_Toc187337539)

[Tabla 10. Proyectos en OpenShift 19](#_Toc187337540)

[Tabla 11. Esquemas de BD 24](#_Toc187337541)

[Tabla 12. Login para acceso en BD 24](#_Toc187337542)

[Tabla 13. RNF01 25](#_Toc187337543)

[Tabla 14. RNF02 25](#_Toc187337544)

[Tabla 15. RNF03 26](#_Toc187337545)

[Tabla 16. RNF04 26](#_Toc187337546)

[Tabla 17. RNF05 26](#_Toc187337547)

[Tabla 18. Ejecutables JAR 27](#_Toc187337548)

[Tabla 19. Archivos Deployment 30](#_Toc187337549)

[Tabla 20. Archivos Service 31](#_Toc187337550)

[Tabla 21. Archivos Ingress 31](#_Toc187337551)

#### Índice de figuras

[Figura 1. Diagrama de arquitectura lógica 12](#_Toc187338650)

[Figura 2. Diagrama de arquitectura física del ambiente de desarrollo 13](#_Toc187338651)

[Figura 3. Diagrama de arquitectura física del ambiente de certificación 14](#_Toc187338652)

[Figura 4. Diagrama de arquitectura física del ambiente de producción 15](#_Toc187338653)

[Figura 5. Dependencias que incluye Spring Boot Starter Web 16](#_Toc187338654)

[Figura 6. Archivos de configuración generados por Gradle 20](#_Toc187338655)

[Figura 7. Estructura principal de un proyecto de microservicio 22](#_Toc187338656)

[Figura 8. Estructura de los proyectos vuce-filenet-api y vuce-filechunk-api 23](#_Toc187338657)

[Figura 9. Diagrama de base de datos 25](#_Toc187338658)

[Figura 10. Diagrama de sistemas 27](#_Toc187338659)

[Figura 11. Diagrama de componentes de vuce-filenet-api 28](#_Toc187338660)

[Figura 12. Diagrama de componentes de vuce-filechunk-api 28](#_Toc187338661)

[Figura 13. Diagrama de despliegue 29](#_Toc187338662)

[Figura 14. Archivo deployment 30](#_Toc187338663)

[Figura 15. Archivo Service 31](#_Toc187338664)

[Figura 16. Archivo Ingress 32](#_Toc187338665)

[Figura 17. Archivo JenkinsFile 33](#_Toc187338666)

[Figura 18. Archivo DockerFile 33](#_Toc187338667)

[Figura 19. Url Jenkins 34](#_Toc187338668)

[Figura 20. Componentes en jenkins 34](#_Toc187338669)

# Introducción

Este documento técnico provee una visión general de la arquitectura del proyecto Gestor de Filenet; así como, la especificación técnica de la solución, usando diferentes vistas para apreciar los diferentes aspectos del Sistema.

# Objetivo

El objetivo principal de este manual es mostrar mediante diagramas, modelos y artefactos, las consideraciones técnicas y tecnológicas (plataforma) en la que está implementada la solución. Esbozar los aspectos funcionales de la aplicación, definir los mecanismos de despliegue y distribución del sistema, así como también esbozar el modelo de la arquitectura de datos a desarrollar.

Objetivos específicos:

* Facilitar una guía técnica de las configuraciones existentes al personal técnico responsable del mantenimiento del sistema.
* Brindar información técnica acerca del sistema y su modelo de datos representado a nivel conceptual, físico y lógicos; además se agrega el diccionario de datos con las estructuras de tablas de Base de datos.

# Alcance

Este documento proporciona los aspectos técnicos e informáticos del desarrollo para que los usuarios que sean asignados para dar mantenimiento al sistema lo hagan de forma idónea conociendo los requerimientos y la arquitectura utilizada para la construcción del sistema, la infraestructura tecnológica usada, el modelo que soporta los datos, la forma en que se estructura el código fuente, como se establece la conexión a base de datos, entre otros.

Es importante tener en cuenta que en el presente manual se mencionan las especificaciones de hardware, software y herramienta que se utilizaron.

Cómo se mencionó anteriormente, este manual está orientado al personal técnico delegado para el mantenimiento del sistema por lo cual se espera que las persona a cargo tenga las habilidades y conocimientos necesarios para poder entender y hacer suya la información brindada; teniendo además los permisos y autorizaciones necesarias.

# Glosario de Términos

Tabla 1. Glosario de términos

| **ACRÓNIMO/CONCEPTO** | **DESCRIPCIÓN** |
| --- | --- |
| Repositorio GIT | Un repositorio Git es un almacenamiento virtual del proyecto. Lo que permite guardar versiones del código a las que puedes acceder cuando se necesitan. |
| Branch de Repositorio en GIT | En Git, las ramas son parte del proceso de desarrollo diario. Las ramas de Git son un puntero eficaz para aplicar los cambios en el código. Estos cambios pueden ser por añadir una nueva función, solucionar un error o verificar por una optimización; esto permite poder trabajar y tener siempre a disposición el código original (master). |
| GitLab | GitLab es una suite completa que permite gestionar, crear y conectar los repositorios con diferentes aplicaciones y hacer todo tipo de integraciones con ellas, ofreciendo un ambiente y una plataforma en cual se puede realizar las varias etapas de su SDLC/ADLC y DevOps. |
| Jenkins | Jenkins es un servidor de automatización open source escrito en Java. Ayuda en la automatización de parte del proceso de desarrollo de software mediante integración continua y facilita ciertos aspectos de la entrega continua. |
| API | Las API (del inglés Application Programming Interface) son mecanismos que permiten a dos componentes de software comunicarse entre sí mediante un conjunto de definiciones y protocolos. |
| REST | Significa transferencia de estado representacional. REST define un conjunto de funciones como GET, PUT, DELETE, etc. que los clientes pueden utilizar para acceder a los datos del servidor. Los clientes y los servidores intercambian datos mediante HTTP. La principal característica de la [API de REST](https://docs.aws.amazon.com/apigateway/latest/developerguide/http-api-vs-rest?pg=wianapi&cta=restapi" \t "_blank) es la ausencia de estado. La ausencia de estado significa que los servidores no guardan los datos del cliente entre las solicitudes. |
| Back end | El término es utilizado para referirse al área lógica del sistema. Se refiere a la arquitectura interna del sistema que asegura que todos elementos desarrollen la función correcta. No está visible a ojos del usuario y no incluye ningún tipo de elemento gráfico. |
| IDE | Por sus siglas en inglés Integrated Development Environment: es una aplicación informática que proporciona servicios integrales para el desarrollo de sistemas en un lenguaje de programación. |

# Infraestructura Tecnológica

La Infraestructura Tecnológica usada es On-Premises donde la instalación del sistema es dentro de una ubicación física brindada por VUCE - MINCETUR donde es un entorno de TI in-house, dado a que el centro de datos se encuentra en las instalaciones de la entidad donde el equipo de TI tiene control y administración sobre los equipos, servidores y servicios usados.

La principal ventaja de la infraestructura On-Premises es lo fácil que es de controlar ya que, al residir dentro de las propias instalaciones de la entidad, el personal a cargo puede acceder a ella de forma directa, controlando cada uno de sus componentes decidiendo qué hacer con ellos y cómo disponerlos.

De lo anterior también se adiciona que todos los datos almacenados de la lógica de negocio de la VUCE se encuentran sobre la infraestructura, sin que haya terceros que puedan acceder a ella, lo cual se vuelve sumamente valioso dado a que la información es de carácter sensible.

## Sistemas operativos de servidores

Un Sistema Operativo de Servidor es aquel que es usado en un servidor el cual contiene un conjunto de procesos, programas y utilidades para gestionar los recursos de hardware y software; y proporcionar servicios para ser usados por uno o varios sistemas(s).

A continuación, se presenta los sistemas operativos conforme los servicios que son usados:

Tabla 2. Sistemas operativos de servidores

| **SERVICIO** | **TIPO** | **VERSION** | **PAGINA WEB** | **S. OPERATIVO** |
| --- | --- | --- | --- | --- |
| **Sistema de Control de versiones** | **GitLab** | 12.4.1 | <https://about.gitlab.com/> | Red Hat Enterprise Linux 7.2 (Maipo) |
| **Servicio de automatización para el proceso de desarrollo con integración continua** | **Jenkins** | 2.319.1 | <https://www.jenkins.io/> | Red Hat Enterprise Linux 7.8 (Maipo) |
| **Contenedor de microservicios – Openshift** | OpenShift - Wikipedia | 1.16.3 | <https://www.redhat.com/en/technologies/cloud-computing/openshift> | Red Hat Enterprise Linux 7.2 (Maipo) |
| **API Manager** |  | 4.0.0 | https://wso2.com/es/api-manager/ | Red Hat Enterprise Linux 7.2 (Maipo) |
| **Repositorio de artefactos** |  | 7.11.5 | <https://jfrog.com/> | Red Hat Enterprise Linux 7.2 (Maipo) |
| **Servicio de gestión de colas** |  | 3.2 | <https://kafka.apache.org/> | Red Hat Enterprise Linux 7.2 (Maipo) |
| **Sistema de gestión de contenido** |  | 5.5.4 | <https://www.ibm.com/es-es/products/filenet-content-manager> | Red Hat Enterprise Linux 7.2 (Maipo) |

## Sistemas de gestión de base de datos

Los microservicios vuce-filenet-api y vuce-filechunk-api utilizan la base de datos PostgreSQL para almacenar la información de la configuración de las carpetas en el Filenet por cada componente que utilice estos microservicios.

## Servidor Web

NGINX, es un servidor web de código abierto de alto rendimiento licenciado bajo la licencia BSD simplificada. Es multiplataforma, por lo que corre en sistemas tipo Unix (GNU/Linux, BSD, Solaris, Mac OS X, etc.) y Windows. NGINX fue inicialmente desarrollado con el fin explícito de superar el rendimiento ofrecido por el servidor web Apache. Sirviendo archivos estáticos, NGINX usa menos memoria que Apache, y puede manejar aproximadamente cuatro veces más solicitudes por segundo. Este servidor web es usado en los diferentes ambientes definidos para el Gestor de Filenet, utilizándose en su versión 1.22.

Las características básicas que tiene este servidor son:

* Características básicas del servidor web.
* Servidor de archivos estáticos, índices y autoindexado.
* Proxy inverso con opciones de caché.
* Balanceo de carga.
* Tolerancia a fallos.
* Soporte de HTTP y HTTP2 sobre SSL.
* Soporte para FastCGI con opciones de caché.
* Servidores virtuales basados en nombre y/o en dirección IP.
* Streaming de archivos FLV y MP4.24​
* Soporte para autenticación.
* Compatible con IPv6.
* Soporte para protocolo SPDY.
* Compresión gzip.
* Habilitado para soportar más de 10 000 conexiones simultáneas.

Apache Tomcat, es usado a nivel de implementación para los microservicios; este se usa de forma embebida el cual ofrece una forma consistente de empaquetamiento para los proyectos siguiendo el enfoque basado en una arquitectura de microservicios. También facilita la distribución a través de contenedores Docker y su gestión a través de un servicio de orquestación de contenedores, como Kubernetes.

## Herramientas de codificación

Para la codificación del Gestor de Filenet se utilizó:

Tabla 3. Resumen de herramientas de desarrollo

| TIPO | NOMBRE | DESCRIPCION | LOGO |
| --- | --- | --- | --- |
| Entorno de desarrollo | IntelliJ IDEA v2021.3.2 | Es un IDE inteligente y sensible al contexto para trabajar con Java y otros lenguajes JVM como Kotlin, Scala y Groovy en todo tipo de aplicaciones. |  |
| Lenguajes | Java OpenJDK v11 | Es un lenguaje de programación ampliamente utilizado para codificar aplicaciones web creado por Sun Microsystems en 1995. Se emplea Java para la implementación de los proyectos de los microservicios del backend del sistema. |  |
| Java Spring Boot  (Spring Boot) | Es una herramienta que hace que el desarrollo de aplicaciones web y microservicios con Spring Framework sea más rápido y fácil a través de: Configuración automática, Un enfoque obstinado de la configuración y La capacidad de crear aplicaciones independientes. |  |
| Heramientas | Postman | Postman es una popular herramienta utilizada para probar APIs, permitiendo a los desarrolladores enviar peticiones a servicios web y ver respuestas. |  |
| Swagger v2 | Swagger, herramienta usado para documentar y utilizar servicios web rest. |  |
| Navegadores | Imagen que contiene Logotipo  Descripción generada automáticamente | | |

# Arquitectura del Sistema

El componente transversal, fue diseñado a nivel de back end usando una arquitectura de microservicios que es un enfoque arquitectónico y organizativo donde el sistema está compuesto por pequeños servicios independientes y autónomos que se comunican a través de API bien definidas proporcionando las funcionalidades del negocio completa. Los microservicios que componen esta solución están siendo administrados a través de la plataforma de Kubernetes, donde cada uno implementa un conjunto de funciones coherentes y estrechamente relacionados comunicándose mediante protocolo http (REST) pero también de forma asíncrona con Apache Kafka en base a eventos para la actualización de la base de datos.

Para mayor información revisar el documento VUCE2-DocumentoArquitectura-GestorFilenet.v1.0.

## Arquitectura lógica

La arquitectura lógica del sistema representa de manera abstracta por los microservicios implementados que son los componentes que llevan a cabo un conjunto de operaciones relacionados a una funcionalidad donde puede existir comunicación entre ellos conforme los escenarios que hayan sido definidos.

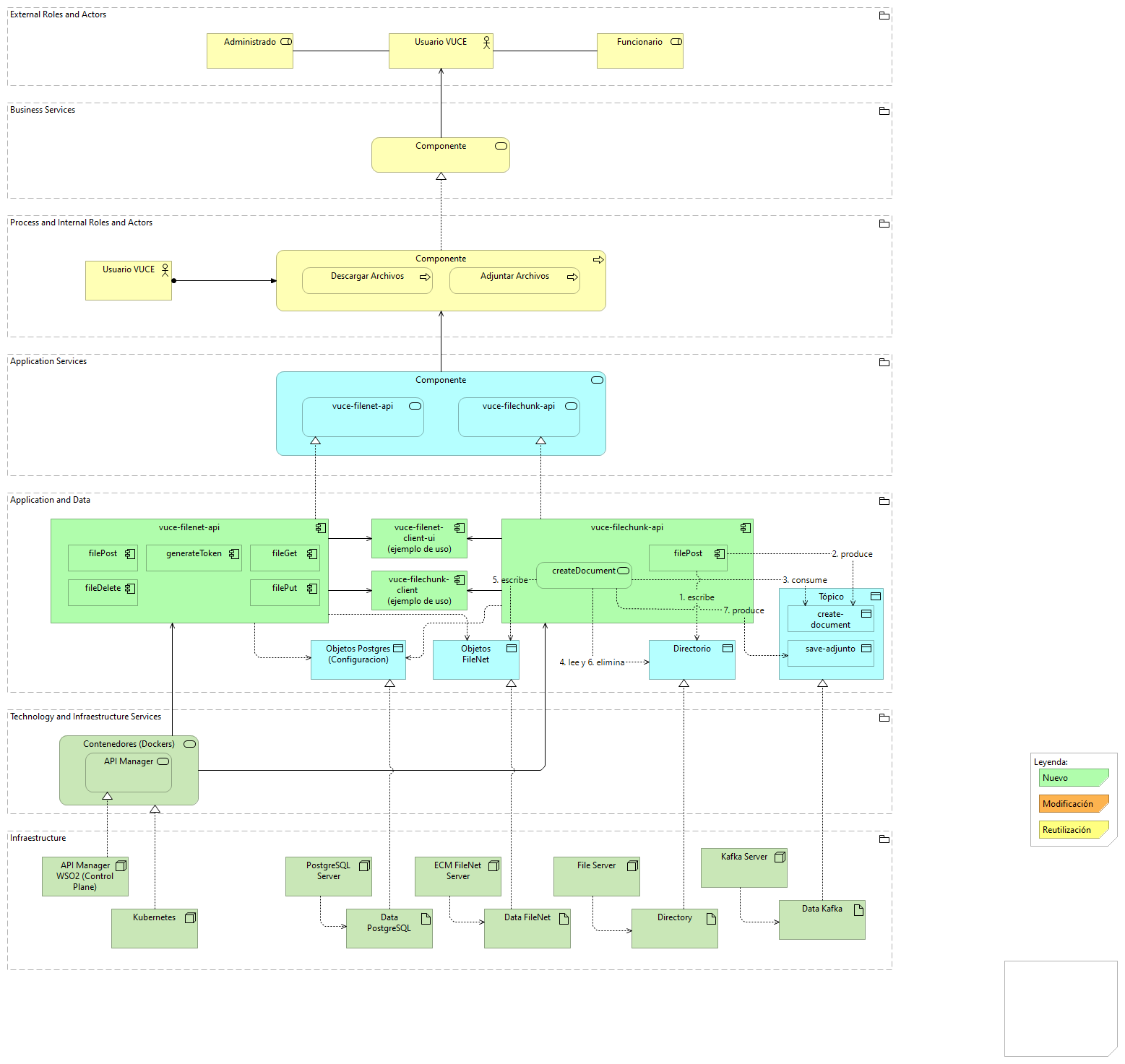


Figura 1. Diagrama de arquitectura lógica

## Arquitectura física

La arquitectura física del sistema representa la relación de todos los componentes físicos o hardware que intervienen para su correcto funcionamiento. Tomando en cuenta que la arquitectura lógica se basa en microservicios y microfrontend se debe mencionar que no hay coincidencia con la arquitectura física ya que estos elementos deben ser autónomo y permitir que el código con su estado se versione, implementen y escalen de forma independiente del hardware.

A continuación, se presentan los diagramas de arquitectura física que integran los cuatro ambientes definidos para el sistema que son: desarrollo, certificación, capacitación y producción.

### Arquitectura física del ambiente de desarrollo

Este diagrama presenta los componentes de infraestructura y software del ambiente de desarrollo durante el proyecto:

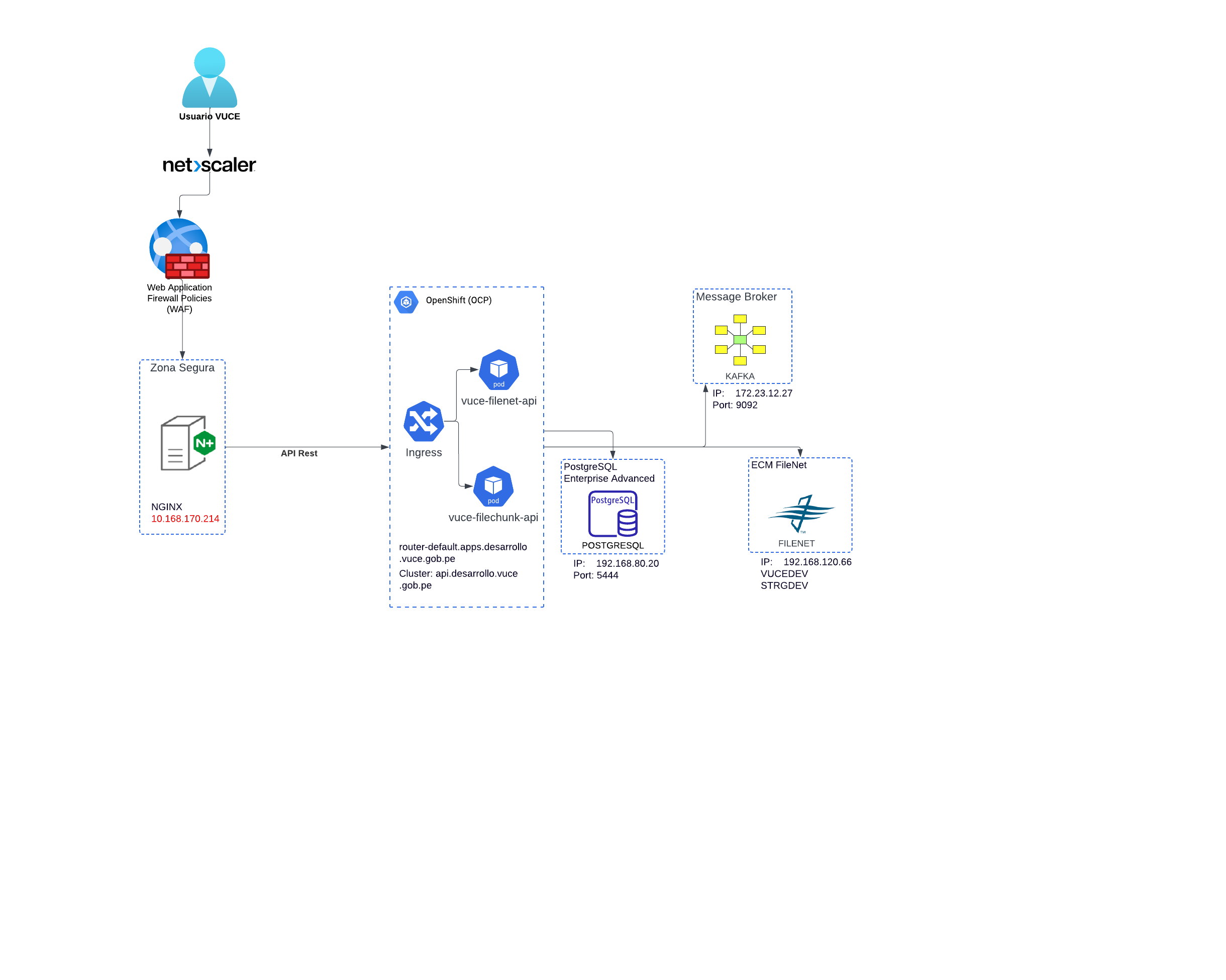


Figura 2. Diagrama de arquitectura física del ambiente de desarrollo

### Arquitectura física del ambiente de certificación y capacitación

Este diagrama presenta los componentes de infraestructura y software del ambiente de certificación y capacitación para evaluar la calidad del sistema y del buen funcionamiento de los procesos desarrollados durante el proyecto:



Figura 3. Diagrama de arquitectura física del ambiente de certificación

### Arquitectura física del ambiente de producción

Este diagrama presenta los componentes de infraestructura y software del ambiente de producción en donde se desplegará el sistema:

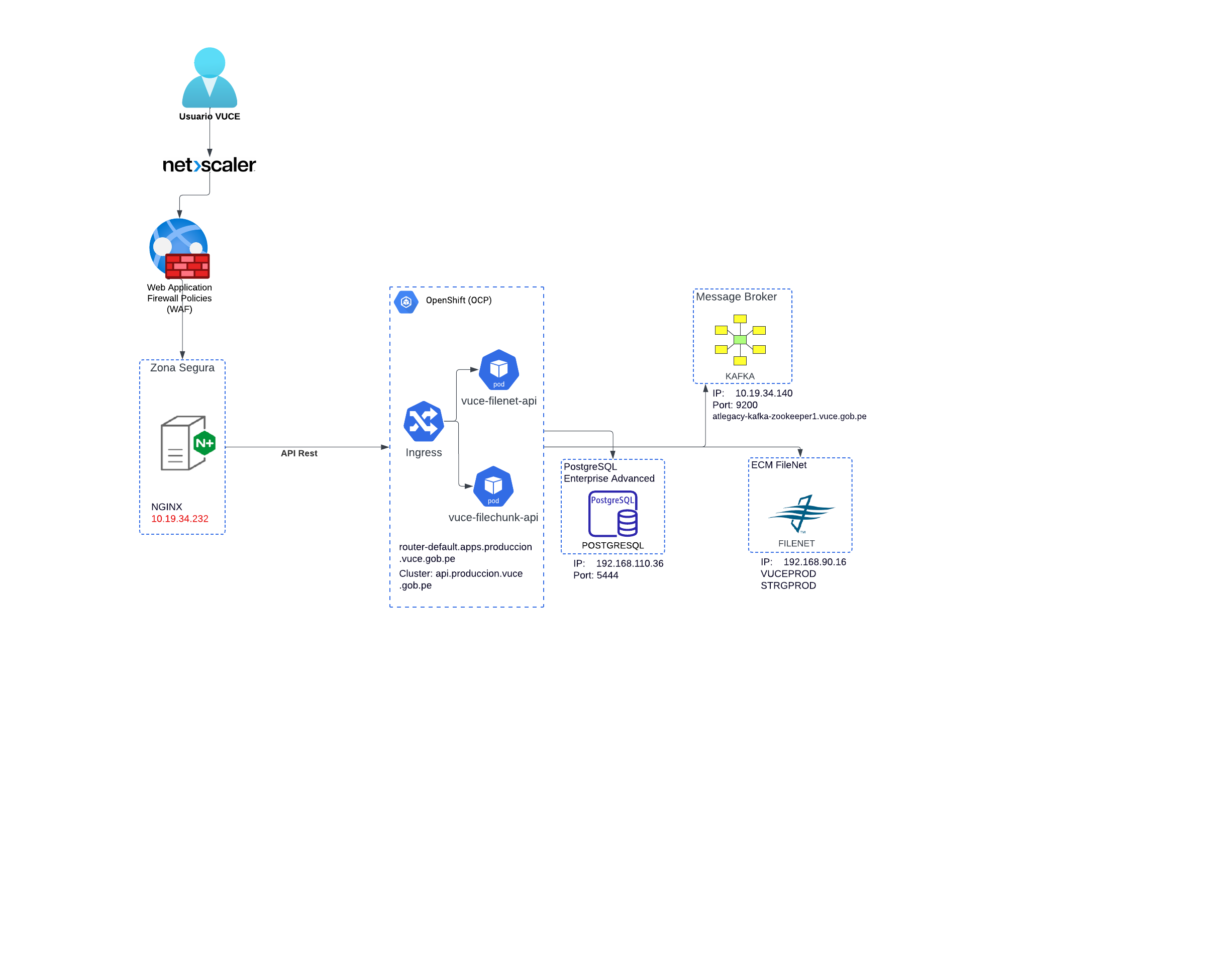


Figura 4. Diagrama de arquitectura física del ambiente de producción

# Marco de Trabajos, Librerías y Estándares

El lenguaje de programación usado para los microservicios del Gestor de Filenet es Java en su versión OpenJDK 11. Java es un lenguaje orientado a objetos, multiplataforma y centrado en la red; es un lenguaje de programación robusto, seguro y fiable. Este lenguaje cuenta con una máquina virtual conocida como JVM (Java Virtual Machine) que actúa como una capa de abstracción adicional entre la plataforma Java y el hardware de la maquina subyacente; por lo mencionado el código fuente de Java solo puede ejecutarse en aquellas maquinas que tienen la JVM instalada.

## Marco de trabajo

Un Marco de Trabajo (FrameWork) es un conjunto estandarizado de conceptos, prácticas y criterios para enfocar un tipo de problemática particular que sirve como referencia, para enfrentar y resolver nuevos problemas de índole similar. En el contexto de desarrollo de sistema un marco de trabajo está relacionado con patrones de diseño implementados en un lenguaje de programación, proveyendo diferentes librerías, funciones, interfaces y api; que un desarrollador podrá usarlas y de esta forma brindar una base para codificar el contenido y las funcionalidades para un sistema.

### Marco de trabajo para los proyectos de microservicios

El marco principal que se usa para el Gestor de Filenet en los proyectos de microservicios (BackEnd) es Spring que es un marco ligero de código abierto basado en Java, ampliamente utilizado para desarrollar aplicaciones empresariales el cual brinda un enfoque simplificado y modular. El uso del marco de trabajo sus módulos y librerías se pueden localizar en el archivo build.xml de Gradle. Por lo anterior se mencionan:

**Spring Boot**: Se usó para la configuración de los diferentes proyectos de los microservicios basados en Spring, delegándole las dependencias y el despliegue. Este fue utilizado en el proyecto haciendo uso del plugin para Gradle de la versión 2.7.14.

id 'org.springframework.boot' version '2.7.14.RELEASE'

id 'io.spring.dependency-management' version '1.0.15.RELEASE'

**Spring Boot Starter Web:** Este módulo se usó ya que permite construir aplicaciones RESTful usando Spring MVC. Spring Boot Starter Tomcat es el contenedor incrustado predeterminado para Spring Boot Starter Web.

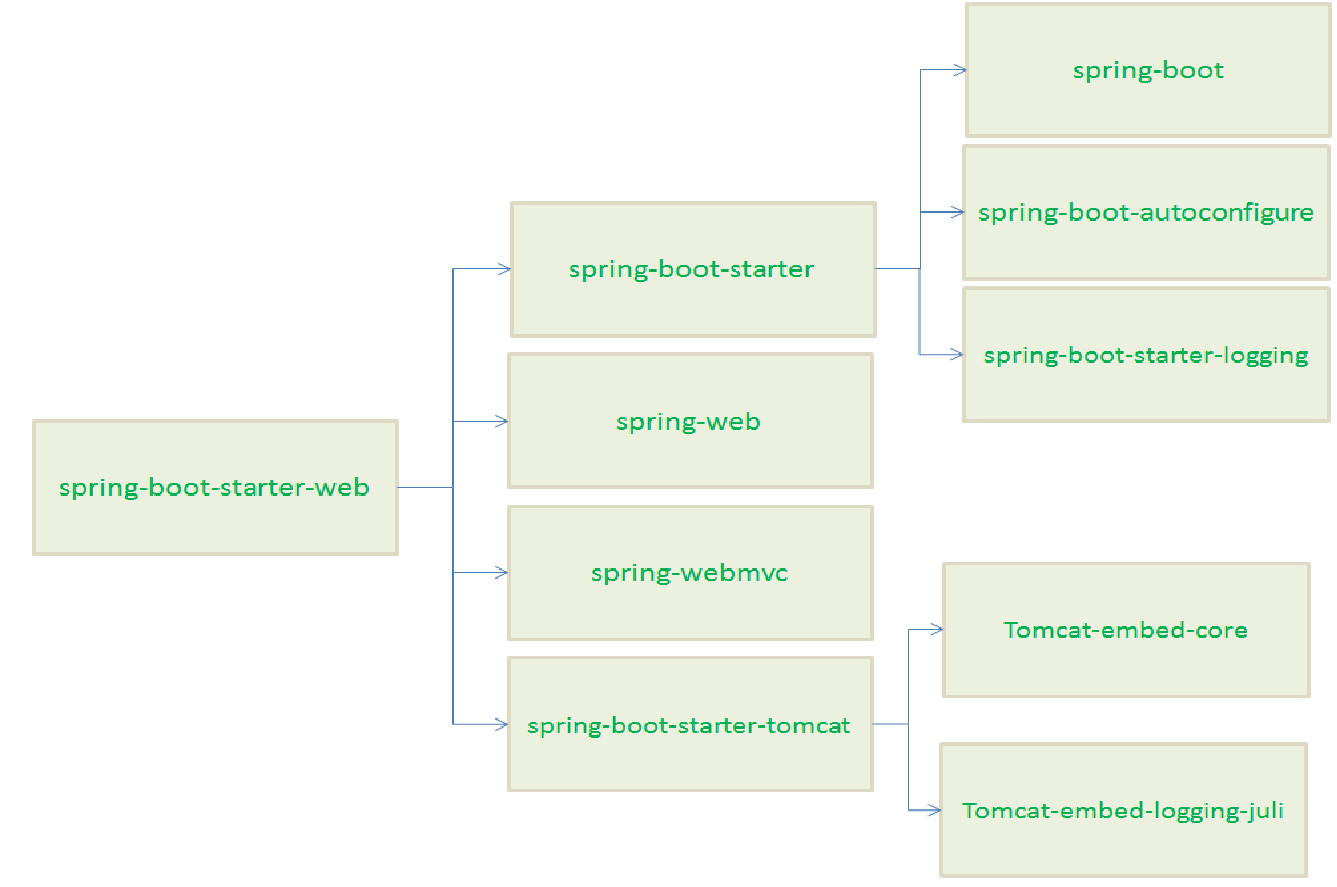
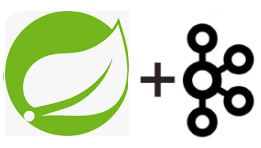


Figura 5. Dependencias que incluye Spring Boot Starter Web

**Spring JPA**: Este módulo pertenece a la familia de Spring Data y se utilizó para implementar repositorios basados en JPA (Java Persistence API). En el proyecto, se integró mediante la siguiente dependencia:

org.springframework.boot:spring-boot-starter-data-jpa



**Spring for Apache Kafka**: Este proyecto permitió producir o consumir mensajería basados en tópicos definidos con Apache Kafka con el objetivo de hacer integraciones con otros componentes. En el proyecto, se integró mediante la siguiente dependencia:

implementation 'org.springframework.kafka:spring-kafka'

La configuracion de Spring Boot se realiza en el archivo de configuración application.properties, ubicado en la carpeta de recursos del proyecto (src/main/resources). Este archivo permite configurar diversas propiedades de Spring y definir configuraciones personalizadas para la aplicación.

## Librerías

Una librería en Java comprende un conjunto de clases que facilitan operaciones y tareas ofreciendo al desarrollador funcionalidad ya implementada y lista para ser usada través de una API (por el anglicismo Application Programming Interface).

### Librerías para los proyectos de microservicios

A continuación se enumeran las principales librerías que usan los proyectos de microservicios los cuales se pueden localizar en el archivo build.xml de Gradle.

Tabla 4. Librerías principales de los proyectos de microservicios

| **LIBRERÍA** | **DESCRIPCION** | **DEPENDENCIA** |
| --- | --- | --- |
| **Project Lombok** | Gestión automática de recursos, generación automática de getters, setters, equals, hashCode y toString, entre otros. | org.projectlombok:lombok |
| **openapi** | Es una especificación para describir, producir, consumir y visualizar servicios web RESTful. Es un estándar ampliamente adoptado que permite a los desarrolladores definir la interfaz de sus APIs de manera independiente del lenguaje de programación | org.springdoc:springdoc-openapi-ui:1.2.32 |
| **junit** | JUnit es un marco de pruebas unitarias para el lenguaje de programación Java. Fue creado por Erich Gamma y Kent Beck y es una de las bibliotecas más utilizadas para realizar pruebas automatizadas en aplicaciones Java, JUnit permite a los desarrolladores escribir y ejecutar pruebas para verificar que su código funciona como se espera | org.mockito:mockito-junit-jupiter:5.8.0 |
| **jace** | Jace es una librería que permite a las aplicaciones Java integrarse con el IBM FileNet. | group: 'Jace', name: 'Jace', version: 'Jace' |
| **TUS** | TUS es un proyecto que tiene como objetivo hacer que las cargas de archivos reanudables sean fáciles de usar y estén ampliamente disponibles | me.desair.tus:tus-java-server:1.0.0-2.0 |

## Estándares

A continuación, los estándares usados para este proyecto:

Tabla 5. Estándares de desarrollo

| **Id** | **Nombre** | **Referencia** |
| --- | --- | --- |
| [1](https://github.com/angular/angular-cli/) | Estándares Desarrollo | VUCE2-LineamientosDesarrollo |
| [2](https://github.com/angular/angular-cli/) | Estándares Base de Datos | Documento estándares BD postgres |
| 3 | Métricas de Calidad de Código | Metricas e Indicadores de Calidad |
| 4 | Arquitectura Patrón | Lineamientos del Ciclo DevOps |

# Estructuración de Código Fuente

## Proyectos de código fuente de microservicios

A continuación, se listan los proyectos de los microservicios que conforman el back end del Gestor de FileNet:

Tabla 6. Proyectos de microservicios

| **Nro.** | **PROYECTO** | **NOMBRE** | **DEPLOYMENT** | **TIPO** | **LENGUAGE** |
| --- | --- | --- | --- | --- | --- |
| 1 | API transversal para la gestión de archivos en el FileNet | vuce-filenet-api | vuce-filenet-api | Proyecto Java | OpenJDK 11 |
| 2 | API transversal para la carga reanudable de archivos en el FileNet | vuce-filechunk-api | vuce-filechunk-api | Proyecto Java | OpenJDK 11 |

## Proyectos en GIT

Para el control de versiones del código fuente se hace uso de GIT, el cual es un software de control de versiones diseñado por Linus Torvalds, pensando en la eficiencia, la confiabilidad y compatibilidad del mantenimiento de versiones de aplicaciones cuando estas tienen un gran número de archivos de código fuente. Con el propósito de llevar registro de los cambios de los archivos, incluyendo la coordinación del trabajo de las personas realizan sobre estos al compartir el repositorio de código fuente.

Se debe de mencionar que se utiliza la plataforma de GitLab que además de brindar el sistema de control de versiones basado en GIT también brinda ciertas operaciones de DevOps basadas en integración continua y entrega continua (CI/CD, por sus siglas en ingles Continuous Integration y Continuous Delivery).

Tabla 7. Proyectos en GIT

|  |  |  |  |
| --- | --- | --- | --- |
| **Nro.** | **Proyecto** | **Nombre** | **Ruta GIT** |
| 1 | API transversal para la gestión de archivos en el FileNet | vuce-filenet-api | <http://gitlab.vuce.gob.pe/arquitectura/vuce-filenet-api> |
| 2 | API transversal para la carga reanudable de archivos en el FileNet | vuce-filechunk-api | <http://gitlab.vuce.gob.pe/arquitectura/vuce-filechunk-api> |
| 3 | Aplicación de ejemplo para consumir el API Filenet | poc-vuce-filenet-client | <http://gitlab.vuce.gob.pe/arquitectura/poc-vuce-filenet-client> |
| 4 | Aplicación de ejemplo para consumir el API Filechunk | vuce-filechunk-client | <http://gitlab.vuce.gob.pe/arquitectura/vuce-filechunk-client> |

## Proyectos en Sonarqube

Para el análisis de código fuente de los componentes del proyecto tenemos la siguiente ruta donde se puede validar las métricas de calidad.

Tabla 8. Proyectos en Sonarqube

|  |  |  |  |
| --- | --- | --- | --- |
| **Nro.** | **Proyecto** | **Nombre** | **URL Proyecto Sonarqube** |
| 1 | vuce-filenet-api | vuce-filenet-api |  |
| 2 | vuce-filechunk-api | vuce-filechunk-api |  |

## Proyectos en Jenkins

Para la automatización de los despliegues en los diferentes entornos se ha configurado los siguientes proyectos en la herramienta Jenkins:

Tabla 9. Proyectos en Jenkins

|  |  |  |  |
| --- | --- | --- | --- |
| **Nro.** | **Proyecto** | **Nombre** | **URL Proyecto Jenkins** |
| 1 | vuce-filenet-api | vuce-filenet-api | <http://172.23.12.29:8080/job/arquitectura/job/certificacion/job/vuce-filenet-api/> |
| 2 | vuce-filechunk-api | vuce-filechunk-api | <http://172.23.12.29:8080/job/arquitectura/job/certificacion/job/vuce-filechunk-api/> |

## Proyectos en OpenShift

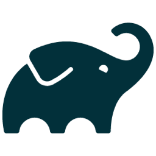
Se ha desplegado en la plataforma de contenedores OpenShift los siguientes proyectos.

Tabla 10. Proyectos en OpenShift

|  |  |  |  |
| --- | --- | --- | --- |
| **Nro.** | **Proyecto** | **Tipo** | **URL Proyecto Openshift** |
| 1 | vuce-filenet-api | vuce-filenet-api |  |
| 2 | vuce-filechunk-api | vuce-filechunk-api |  |

# Herramienta para la Construcción y Gestión del Proyecto

Las herramientas utilizadas para la gestión y construcción de los proyectos son:



Gradle, es una herramienta de código abierto que permite la automatización de la compilación de código fuente, la cual se encuentra centrada en la flexibilidad y el rendimiento; además dispone de un sistema de gestión de dependencias sólido. Gradle basa la construcción de proyectos en algunos conceptos de Apache Ant y Apache Maven, cambiando la configuración de forma XML a script utilizando un lenguaje especifico de dominio Groovy o Kotlin DSL (Domain Specific Language); uno de los archivos definidos a la hora de usar Gradle tiene el nombre de build con extensión .gradle, este se encuentra en la raíz del directorio de cada proyecto y se usó Groovy DSL para su contenido. Esta herramienta fue utilizada para los proyectos de microservicios basados en Java y la versión usada fue 7.4.-bin

## Archivos de configuración generados por Gradle

Al usarse Gradle como herramienta de construcción y gestión de proyecto esta crea una estructura de directorios con archivos, por lo cual se explican a continuación:

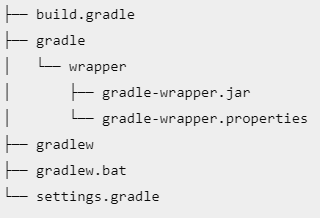


Figura 6. Archivos de configuración generados por Gradle

Gradle creó dos archivos en la raíz del proyecto con los nombres build.gradle y settings.gradle. Estos archivos son utilizados por Gradle para la configuración de la construcción del proyecto.

* **build.gradle**, es un archivo de texto plano cuyo contenido se usó Groovy DSL el cual contiene los plugins usados, los repositorios configurados, las dependencias usadas, las propiedades de descripción del proyecto definidos y configuraciones de tareas realizadas.
* **settings.gradle**, es un archivo de texto plano que indica a Gradle los módulos que deberá incluir al momento de compilar el proyecto, en el caso del Gestor de Filenet al seguir una arquitectura de microservicios este archivo únicamente va a contener el nombre del mismo proyecto.

También nos creó un directorio Gradle el cual contiene la carpeta wrapper. La carpeta wraper contiene unos archivos cuyo objetivo es permitir descargar una versión declarada de Gradle conforme la versión que fue creada el proyecto si es necesario. Los archivos que componen wrapper son:

* **Gradle-wrapper.jar,** es un archivo con extensión .jar que contiene el código para descargar la distribución Gradle.
* **Gradle-wrapper.properties**, es un archivo de texto plano de propiedades responsable de configurar el comportamiento del tiempo de ejecución del wrapper para la versión que se está usando.
* **Gradlew**, es un archivo de texto plano a nivel de script para sistemas basado en UNIX, almacenado en el directorio raíz del proyecto para ejecutar gradle-wrappe.jar.
* **Gradlew.bat**, es un archivo con extensión .bat, que es un script para sistemas basado en Windows, almacenado en el directorio raíz del proyecto para ejecutar gradle-wrapper.jar.

## Archivo build.gradle

Los archivos build.gradle de los proyectos tienen estructuras parecidas teniendo algunas diferencias en las dependencias usadas. A continuación, se presenta la estructura del archivo build.gradle del proyecto vuce-filenet-api, si se desea revisar todos los archivos de los proyectos ver el Anexo B.

**La sección de los plugins usados**:

plugins {

id 'org.springframework.boot' version '2.7.14'

id 'io.spring.dependency-management' version '1.0.15.RELEASE'

id 'java'

}

**La sección de los repositorios usados**:

repositories {

mavenCentral()

maven {

url 'https://repo.spring.io/libs-milestone'

}

maven {

url "http://10.19.34.189:8082/artifactory/vuce-libs-micro-local/"

allowInsecureProtocol = true

credentials {

username "micro"

password "simplEd17y\*"

}

}

}

**La sección de dependencias del marco de trabajo, librerías y librerías comunes**:

dependencies {

implementation 'org.springframework.boot:spring-boot-starter-web'

implementation 'org.springframework.boot:spring-boot-starter-actuator'

implementation 'org.springframework.boot:spring-boot-starter-data-jpa'

implementation 'org.springdoc:springdoc-openapi-ui:1.6.15'

implementation group: 'Jace', name: 'Jace', version: 'Jace'

implementation group: 'log4j', name: 'log4j', version: '1.2.13'

implementation group: 'p8cel10n', name: 'p8cel10n', version: 'p8cel10n'

implementation group: 'stax-api', name: 'stax-api', version: 'stax-api'

implementation group: 'xlxpScanner', name: 'xlxpScanner', version: 'xlxpScanner'

implementation group: 'xlxpScannerUtils', name: 'xlxpScannerUtils', version: 'xlxpScannerUtils'

runtimeOnly 'org.postgresql:postgresql'

compileOnly 'org.projectlombok:lombok'

annotationProcessor 'org.projectlombok:lombok'

testImplementation('org.springframework.boot:spring-boot-starter-test') {

exclude group: 'org.junit.vintage', module: 'junit-vintage-engine'

}

}

## Estructura de proyecto de código fuente microservicios

La herramienta de gestión de proyecto utilizada para los microservicios es gradle, como se menciono en el apartado principal de esta sección, por lo cual la estructura principal es la siguiente:

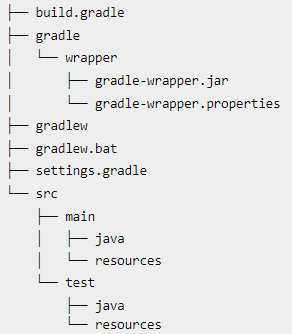


Figura 7. Estructura principal de un proyecto de microservicio

Como se presentó en la sección 7.1, el Gestor de Filenet esta compuesto por dos microservicios, los cuales tienen una estructura similar de paquetes. A continuación, se describe esta estructura:

* **src/main/java:** Directorio donde se encuentra el código fuente principal del proyecto. Contiene una serie de carpetas que representan la estructura de paquetes del proyecto. En el caso de vuce-filenet-api, el paquete principal es pe.gob.vuce.filenet.api.
* **src/main/resources:** Directorio destinado a los recursos de configuración y auxiliares, como archivos XML, propiedades (properties), imágenes, entre otros. Aquí se encuentra el archivo de configuración de Spring Boot.
* **src/test/java:** Directorio donde se ubica el código fuente de las pruebas, con una estructura de paquetes similar a la del directorio src/main/java.
* **src/test/resources:** Directorio reservado para los recursos de configuración y auxiliares relacionados con las pruebas, como archivos XML, propiedades (properties), imágenes, entre otros.

|  |  |
| --- | --- |
|  |  |

Figura 8. Estructura de los proyectos vuce-filenet-api y vuce-filechunk-api

A continuación, se presenta una descripción de los principales paquetes de estos proyectos:

* **pe.gob.vuce.filenet.api**: Contiene las clases controladoras encargadas de servir los diferentes métodos del API, implementados siguiendo el estándar REST. Estas clases utilizan anotaciones como @PostMapping, @GetMapping, @PutMapping y @DeleteMapping para definir los endpoints.
* **pe.gob.vuce.filenet.configuration**: Contiene las clases responsables de la configuración de la aplicación, como las relacionadas con aspectos de seguridad.
* **pe.gob.vuce.filenet.ecm**: Contiene las clases encargadas de realizar operaciones sobre Filenet, como la subida y descarga de archivos.
* **pe.gob.vuce.filenet.entity**: Contiene las clases de tipo Entity, que representan los datos de la base de datos. Cada propiedad de estas clases está mapeada a las columnas correspondientes de las tablas en la base de datos.
* **pe.gob.vuce.filenet.login**: Contiene las clases destinadas al manejo de logging. Estas proporcionan funcionalidad para registrar mensajes en el log, siguiendo los lineamientos establecidos en el documento de arquitectura.
* **pe.gob.vuce.filenet.model**: Contiene las clases que definen la estructura de los datos devueltos por los métodos del microservicio.
* **pe.gob.vuce.filenet.repository**: Contiene las clases encargadas de realizar operaciones con la base de datos, como consultas y actualizaciones.
* **pe.gob.vuce.filenet.service**: Contiene las clases que implementan la lógica de negocio de la aplicación.
* **pe.gob.vuce.filenet.tool**: Contiene las clases utilitarias que proporcionan funcionalidades auxiliares para el proyecto.
* **pe.gob.vuce.filenet.kafka:** Contiene las clases relacionadas con la operaciones en Kafka, como la configuracion y la definicion de los Producers y Consumers.

# Base de Datos

La base de datos del Gestor Filenet está implementada en el gestor de Base de Datos PostgreSQL.

## BD PostgreSQL

La configuración se realiza en el esquema VUCE\_FILENET**.**

### Esquemas de base de datos

Tabla 11. Esquemas de BD

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Nro.** | **Esquema** | **Descripción** | **Condición** | **Consideraciones para instalación** |
| 1 | filenet | Esquema para las tablas, secuencias, vistas e índice | Nuevo | Ninguna |

### Acceso a base de datos

Tabla 12. Login para acceso en BD

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Nro.** | **Login** | **Descripción** | **Uso** | **Condición** | **Consideraciones para instalación** |
| 1 | filenetapp | Usuario que tiene acceso a los objetos de BD VUCE\_FILENET | Aplicación | Nuevo | Ninguna |

### Modelo de base de datos

A continuación, se muestra el modelo de base de datos:



Figura 9. Diagrama de base de datos

# Funcionalidad del Sistema

## Requerimientos funcionales

Los requisitos funcionales del Gestor de Filenet son los siguientes:

* Cargar un documento en Filenet.
* Cargar un documento de manera reanudable en Filenet, para archivos de gran tamaño.
* Actualizar las propiedades de un documento.
* Consultar un documento en Filenet.
* Eliminar un documento de Filenet.

## Requerimientos no funcionales

Los requerimientos no funcionales del Gestor de Filenet son los siguientes:

Tabla 13. RNF01

|  |  |
| --- | --- |
| **Número** | RNF01 |
| **Nombre del requerimiento** | Documentación mediante OpenAPI versión 3.0 |
| La documentación de todas las APIs del microservicio deben realizarse utilizando el estándar OpenAPI versión 3.0 | |

Tabla 14. RNF02

|  |  |
| --- | --- |
| **Número** | RNF02 |
| **Nombre del requerimiento** | Securización de APIs |
| Las APIs deben estar protegidas contra accesos no autorizados mediante el uso de WSO2 API Manager. Esto incluye la implementación de autenticación y control de acceso basado en tokens OAuth2. | |

Tabla 15. RNF03

|  |  |
| --- | --- |
| **Número** | RNF03 |
| **Nombre del requerimiento** | Gestión de APIs |
| Se debe utilizar WSO2 API Manager para la gestión centralizada de las APIs, Esto permitirá una visibilidad completa sobre el uso de las APIs, patrones de tráfico, y posibles cuellos de botella o ataques de seguridad. | |

Tabla 16. RNF04

|  |  |
| --- | --- |
| **Número** | RNF04 |
| **Nombre del requerimiento** | Realizar Pruebas unitarias |
| Se realizarán pruebas unitarias. El objetivo es probar el componente de manera aislada de la aplicación, los errores o defectos secundarios de otros componentes deben ser eliminados. | |

Tabla 17. RNF05

|  |  |
| --- | --- |
| **Número** | RNF05 |
| **Nombre del requerimiento** | Cumplir con métricas de clean code y buenas prácticas de programación bajo principios SOLID |
| Se mejorará el código fuente considerando los siguientes puntos:   * Código innecesario < 1%. * Código repetido < 1%. * Documentación del código (comments) > 20% * Cobertura > 70% * Cumplimiento de reglas > 90% * Calificación de mantenibilidad = A * Deuda técnica < 4h * Seguridad = A | |

# Componentes y Despliegue

Todos los componentes que entregan un ejecutable como .jar, deben cumplir un estándar de nombre por cada ambiente. Si el ejecutable se instala en algún servidor entonces debe ser configurado en los archivos de canalización para que el despliegue se realice bajo el estándar requerido.

Tabla 18. Ejecutables JAR

|  |  |  |  |
| --- | --- | --- | --- |
| **Nro.** | **Ambiente** | **Deployment** | **Ejemplo** |
| 1 | Desarrollo | nombreProyecto-VERSIÓN- SNAPSHOT.extensión | filechunk-client-SNAPSHOT.jar |
| 2 | Certificación | nombreProyecto-VERSIÓN-RELEASE.extensión | filechunk-client-1.0-RELEASE.jar |
| 3 | Producción | nombreProyecto-VERSIÓN.extensión | filechunk-client-1.0.jar |

## Diagramas de componentes

El siguiente diagrama muestra, de manera general, la arquitectura del Gestor de Filenet y su relación con las demás aplicaciones y almacenes de datos:

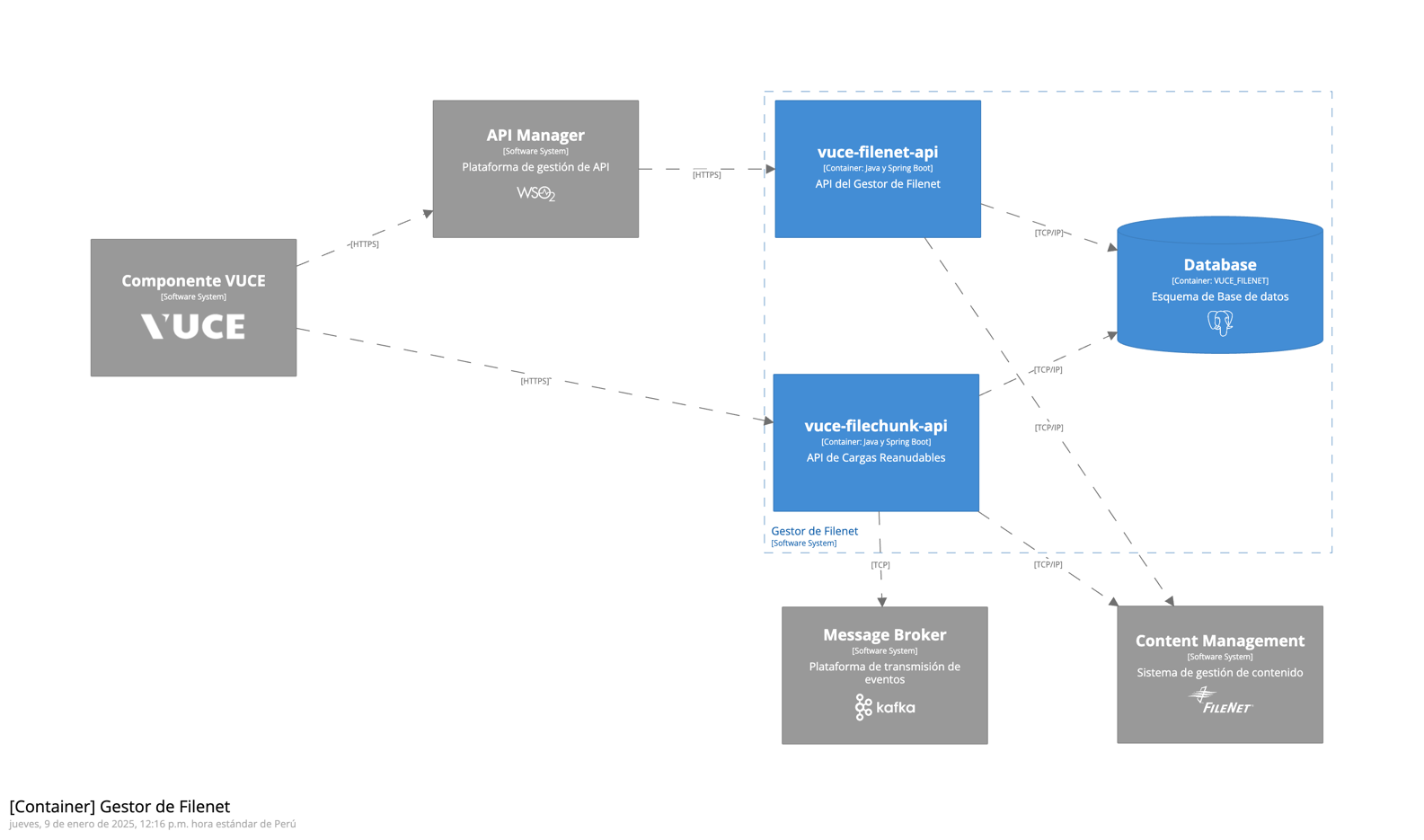


Figura 10. Diagrama de sistemas

En el diagrama anterior, se muestran las aplicaciones que conforman el Gestor de Filenet dentro del rectángulo azul. Este incluye los dos microservicios que componen el sistema, así como la base de datos. A continuación, en el siguiente diagrama, se detalla la estructura de los componentes del microservicio vuce-filenet-api:

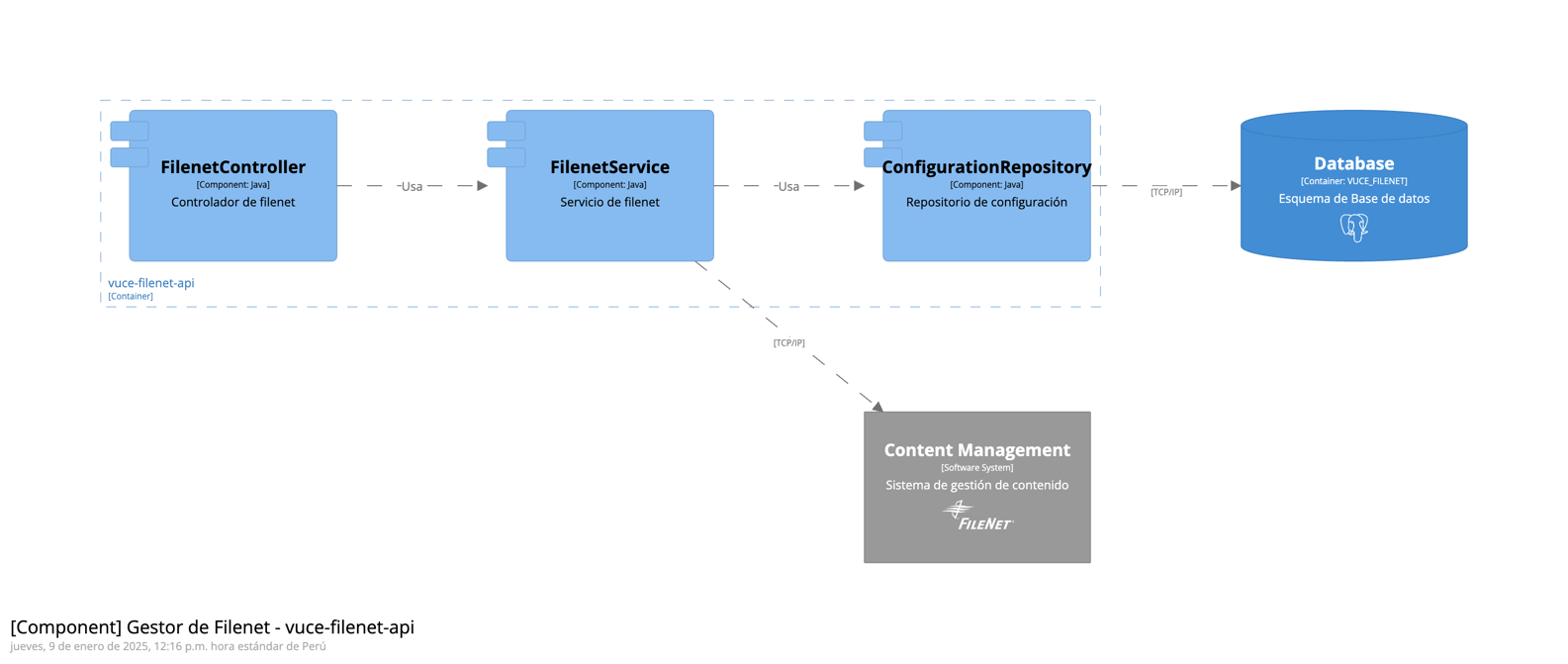


Figura 11. Diagrama de componentes de vuce-filenet-api

Por último, en el siguiente diagrama se presenta el detalle de los componentes que conforman el microservicio vuce-filechunk-api:

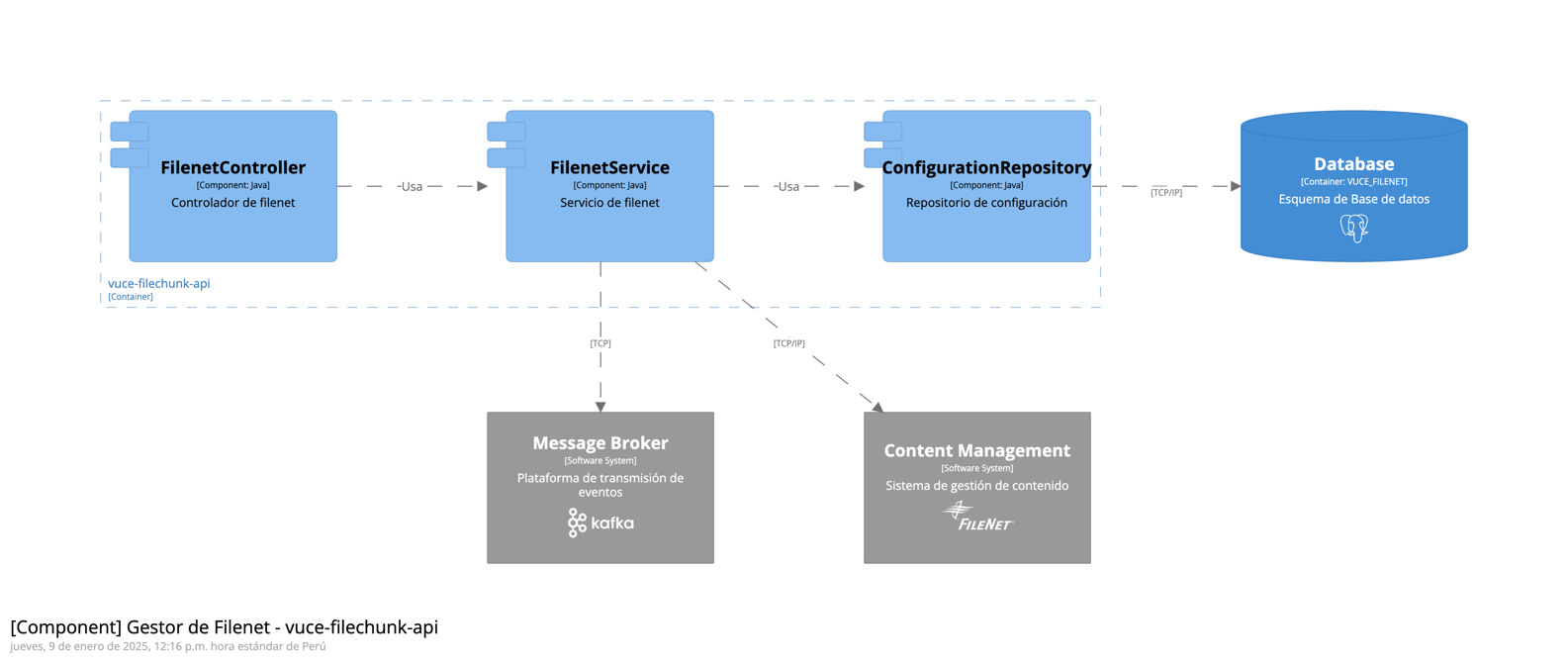


Figura 12. Diagrama de componentes de vuce-filechunk-api

## Diagrama de despliegue

El siguiente diagrama de despliegue muestra los componentes del Gestor de Filenet dentro de la infraestructura. Los microservicios vuce-filenet-api y vuce-filechunk-api están desplegados en pods gestionados por OpenShift. Estos interactúan con el servidor de base de datos PostgreSQL, el servidor de mensajería Apache Kafka y el servidor de gestión de archivos IBM Filenet.

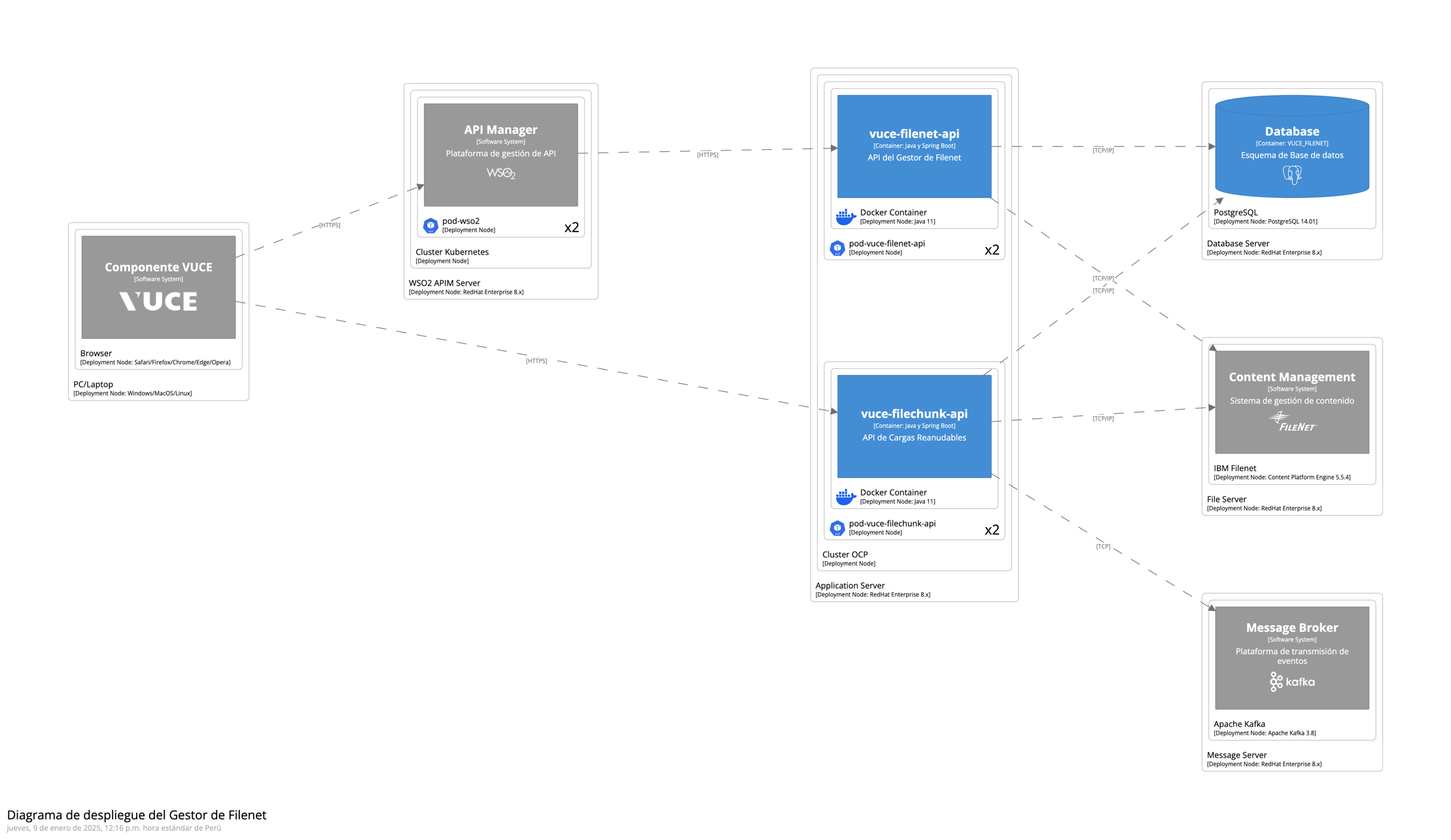


Figura 13. Diagrama de despliegue

## Deployment

Los archivos de deployment se realizan por cada uno de los componentes desplegados que tiene el proyecto vuce-sunat-api. En él se define el namespace, labels, puerto y variables de entorno que se usan en el proyecto. Los nombres asignados a estos archivos tendrán la estructura definida por el equipo de arquitectura: vuce-filenet-api-app.yml. Ubicación de los archivos: ocp/{ambiente}/filenet.

Tabla 19. Archivos Deployment

| **Orden** | **Nombre del proyecto** | **Nombre del deployment** | **Ambiente** |
| --- | --- | --- | --- |
| 1 | vuce-filenet-api | vuce-filenet-api-app.yml | desa, certi, capa, prod |
| 2 | vuce-filechunk-api | vuce-filechunk-api-app.yml | desa, certi, capa, prod |

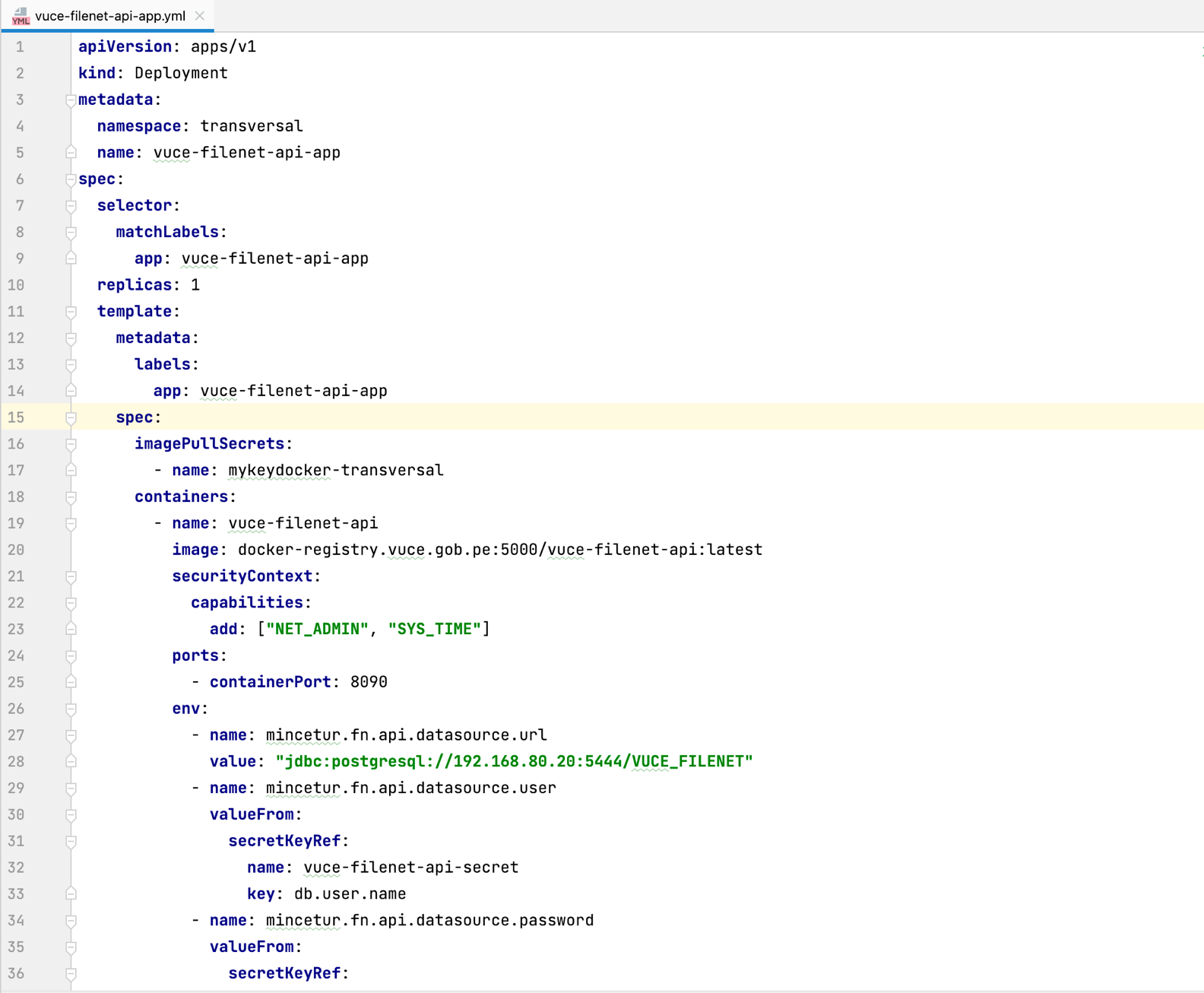


Figura 14. Archivo deployment

## Service e Ingress

Con la definición de los archivos deploymet se garantiza que los servicios se estén ejecutando. Pero para que puedan ser usados o accedidos desde el exterior es necesario definir los archivos Service e Ingress. Se tiene que manejar que internamente los pods pueden acceder dentro de la red del clúster.

Los archivos de Services ayudarán a definir un objeto en Openshift que recibe solicitudes HTTP o HTTPS y poder balancear entre los Pods definidos. Los nombres asignados a estos archivos tendrán la estructura: componente- service.yml.

En el caso del archivo Ingress permite definir un controlador para vincular el DNS Host y elegir qué solicitudes van a qué servicio según la condición que especifique (en nuestro caso, será la ruta de la URL). Este archivo es definido por cada namespace.

Tabla 20. Archivos Service

| **Orden** | **Nombre del proyecto** | **Nombre del service** | **Ambiente** |
| --- | --- | --- | --- |
| 1 | vuce-filenet-api | vuce-filenet-api-service.yml | desa, certi, capa, prod |
| 2 | vuce-filechunk-api | vuce-filechunk-api-service.yml | desa, certi, capa, prod |

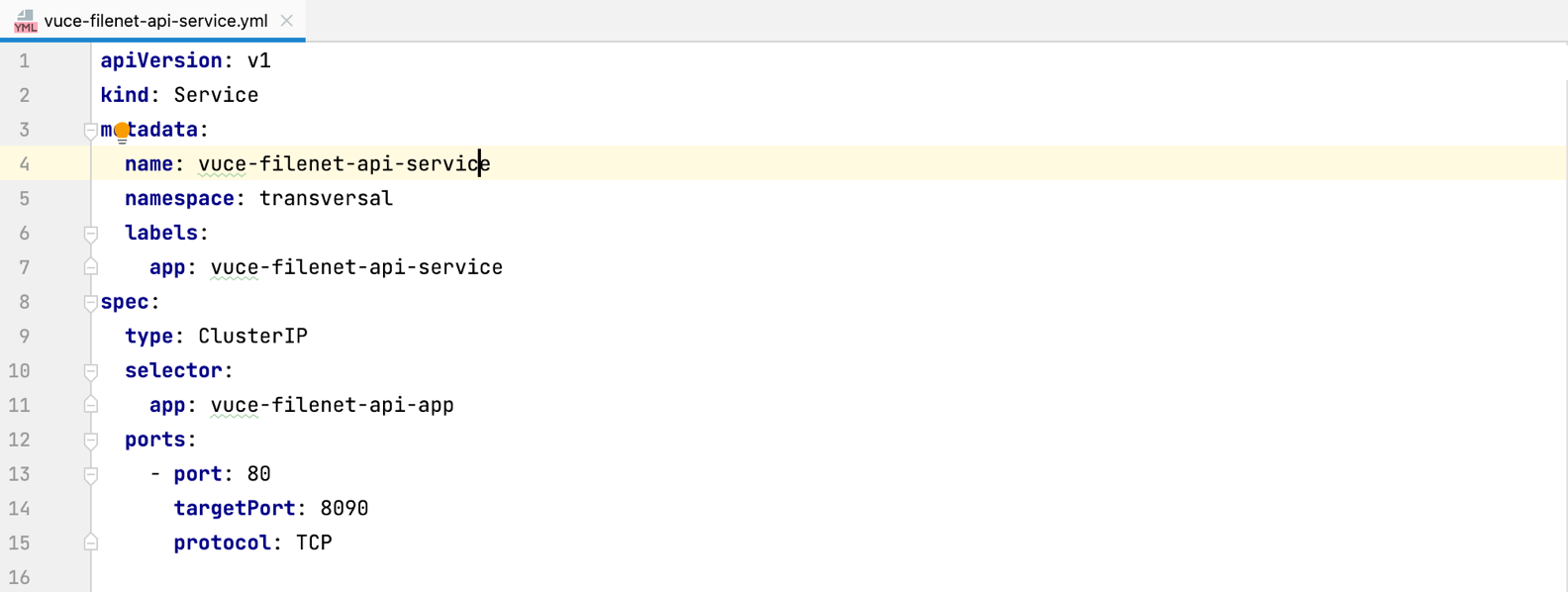


Figura 15. Archivo Service

Tabla 21. Archivos Ingress

| **Orden** | **Nombre del ingress** | **Ambiente** |
| --- | --- | --- |
| 1 | 31-rules-transversal.yaml | desa |
| 2 | 31-rules-transversal.yaml | certi |
| 3 | 31-rules-transversal.yaml | capa |
| 4 | 31-rules-transversal.yaml | prod |

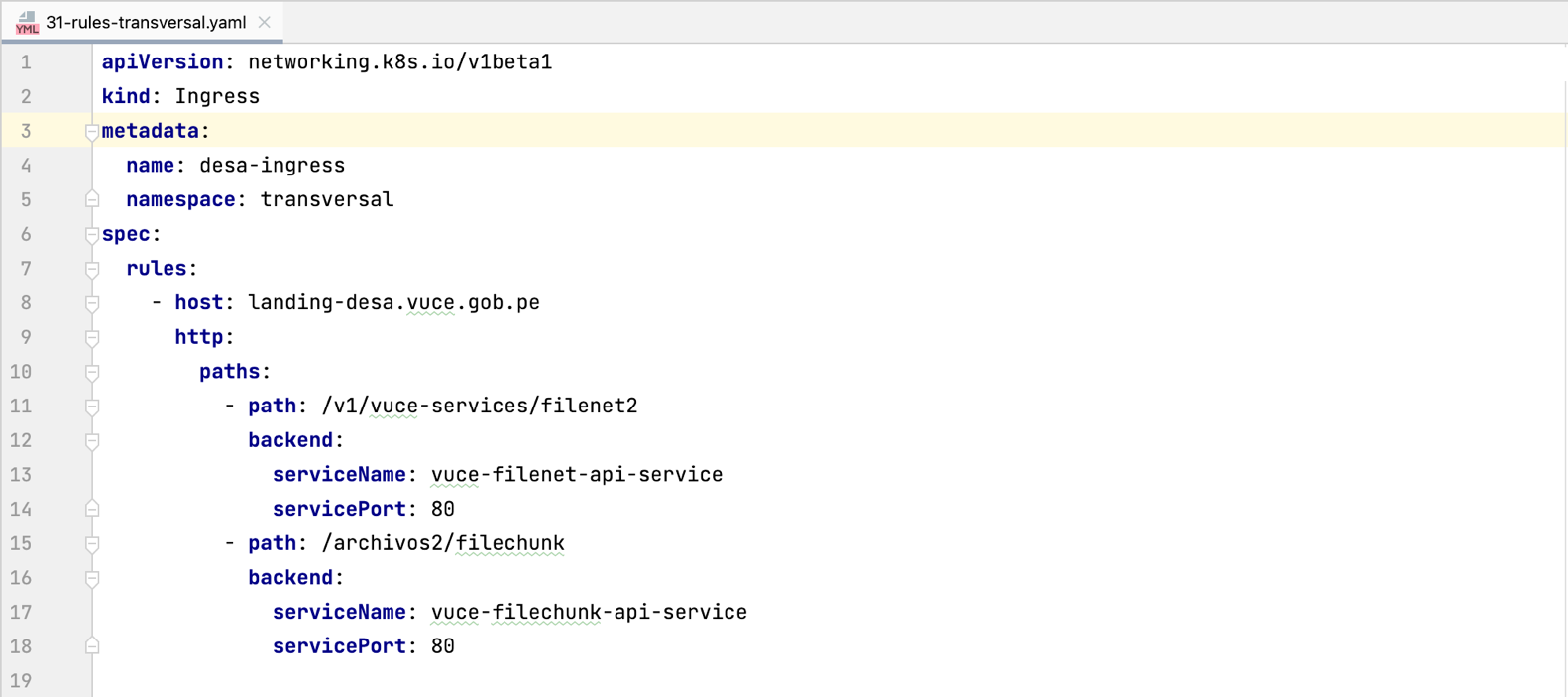


Figura 16. Archivo Ingress

## JenkinsFile

En estos archivos especifica las etapas para el proceso de construcción y despliegue del microservicio, conectándose al repositorio de GitLab para extraer los códigos fuente conforme la rama configurada. El nombre de este archivo será Jenkinsfile para todos los componentes desplegados.

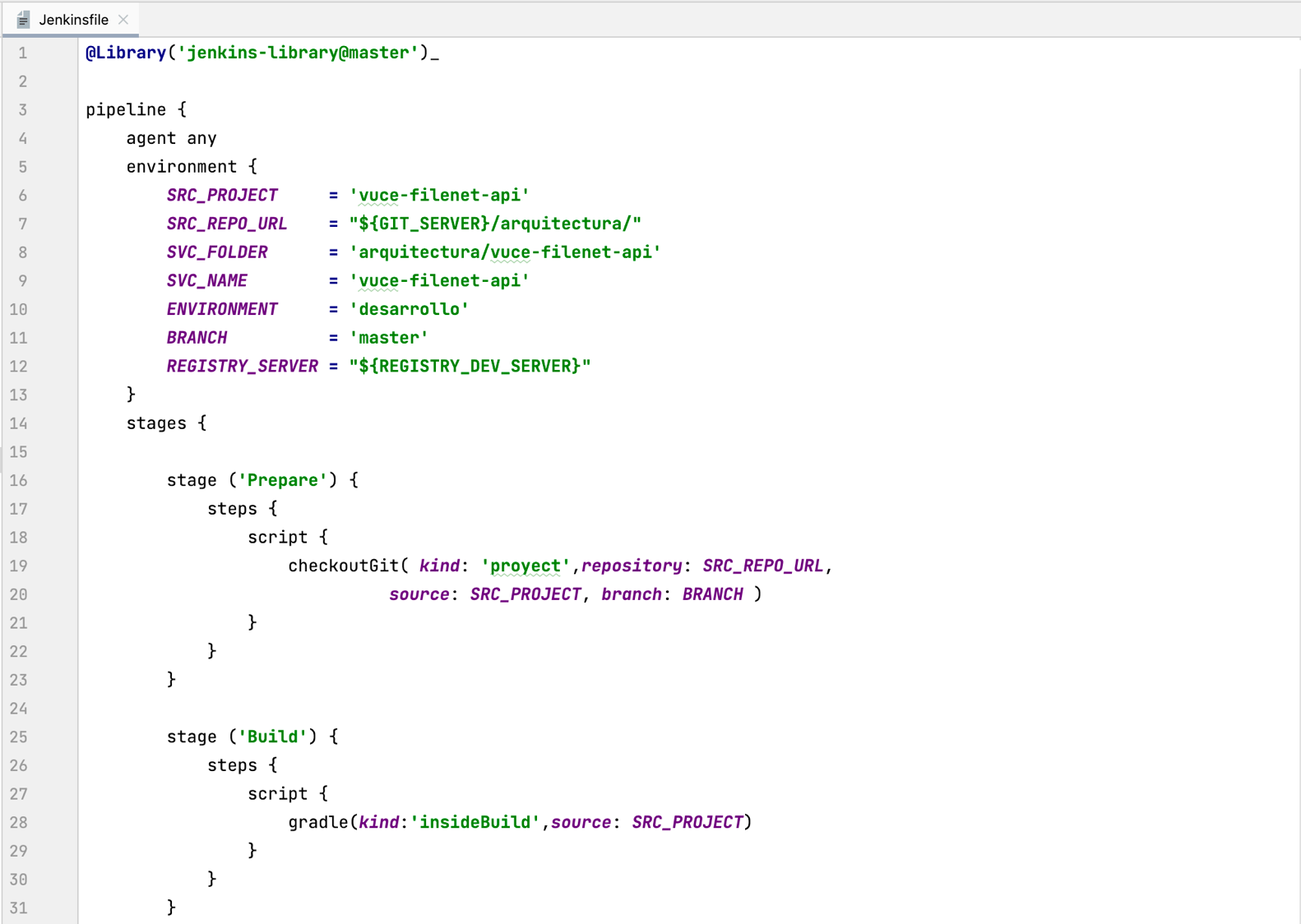


Figura 17. Archivo JenkinsFile

## DockerFile

Estos archivos se encargan de generar la imagen con la que se va a generar un contenedor posterior a subir al registro o repositorio de imágenes de docker, en este se especifica la versión de java, el puerto, el nombre del jar, ubicación de los archivos html, javascript, etc. El nombre de este archivo será Dockerfile para todos los componentes desplegados.

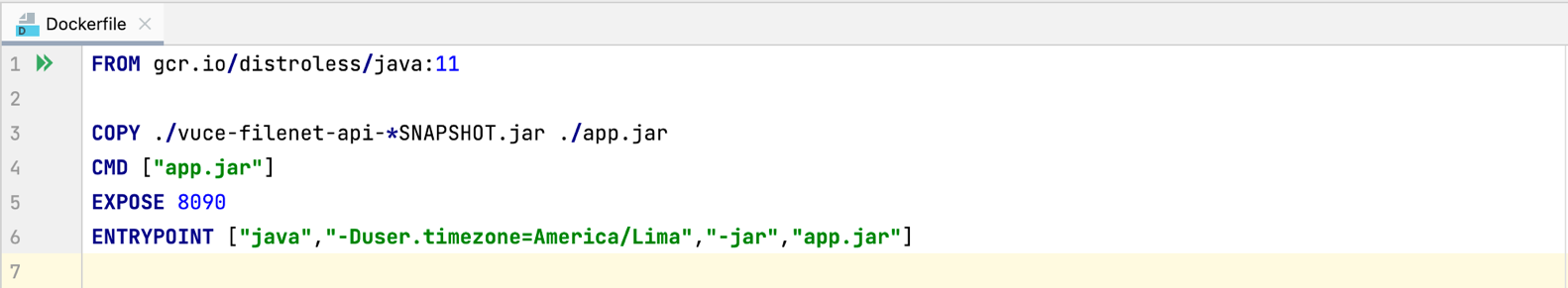


Figura 18. Archivo DockerFile

## Canalización de trabajos para el despliegue en Jenkins

La siguiente tabla guarda la relación de cada uno de los componentes con sus respectivos Jobs de Jenkins, la ruta de acceso a Jenkins fue coordinado con el equipo de arquitectura. Los pipelines en el Jenkins fueron realizados para cada ambiente de VUCE (desa, certi).

Figura 19. Url Jenkins

|  |  |  |  |
| --- | --- | --- | --- |
| **Nro.** | **Proyecto** | **Nombre** | **URL Proyecto Jenkins** |
| 1 | vuce-filenet-api | vuce-filenet-api | <http://172.23.12.29:8080/job/arquitectura/job/certificacion/job/vuce-filenet-api/> |
| 2 | vuce-filechunk-api | vuce-filechunk-api | <http://172.23.12.29:8080/job/arquitectura/job/certificacion/job/vuce-filechunk-api/> |

**

Figura 20. Componentes en jenkins

## Configuración en openShift

## Configuración de microservicio Config

# Anexo A: Archivos de Configuración de Spring Boot

## A-1: Microservicio vuce-filenet-api

## A-2: Microservicio vuce-filechunk-api

# Anexo B: Archivos build.gradle

## A-1: Microservicio vuce-filenet-api

## A-2: Microservicio vuce-filechunk-api