DOCUMENTO REFERENCIA DE ARQUITECTURA ORIENTADA A MICROSERVICOS - ADA

Fábrica de software

Fábrica de Desarrollo de Software – ADA S.A.S

**REVISION Y CONTROL DE CAMBIOS**

**Revisión y Versiones**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Autor | Versión | Descripción | Cargo | Fecha |
| Carlos Torres | 1.0 | Documento | Desarrollador | 19/02/2024 |
|  |  |  |  |  |

**Control de Cambios**

|  |  |  |  |
| --- | --- | --- | --- |
| Fecha | Autor | Versión | Descripción del cambio |
| 19/02/2024 | Carlos Torres | 1.0 | Liberación Documento |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

# TABLA DE CONTENIDO

[TABLA DE CONTENIDO 4](#_Toc161668184)

[RESUMEN 9](#_Toc161668185)

[INTRODUCCION 9](#_Toc161668186)

[PROPOSITO GENERAL 9](#_Toc161668187)

[1. Entender los Principios Fundamentales 10](#_Toc161668188)

[2. Destacar las Ventajas Estratégicas 10](#_Toc161668189)

[3. Guiar en la Implementación Efectiva 10](#_Toc161668190)

[4. Facilitar la Colaboración Interdisciplinaria 10](#_Toc161668191)

[5. Orientar hacia la Innovación Continua 10](#_Toc161668192)

[ALCANCE 11](#_Toc161668193)

[1. Fundamentación Teórica 11](#_Toc161668194)

[2. Estructura de la Arquitectura 11](#_Toc161668195)

[3. Tecnologías Utilizadas 11](#_Toc161668196)

[4. Proceso de Desarrollo 11](#_Toc161668197)

[5. Gestión de la Configuración y Versionamiento 11](#_Toc161668198)

[6. Monitoreo y Mantenimiento 12](#_Toc161668199)

[7. Pruebas y Validación 12](#_Toc161668200)

[8. Seguridad 12](#_Toc161668201)

[OBJETIVOS 12](#_Toc161668202)

[1. Clarificar los Principios Rectores 12](#_Toc161668203)

[2. Detallar la Estructura Microservicios 12](#_Toc161668204)

[3. Especificar Tecnologías Utilizadas 12](#_Toc161668205)

[4. Resaltar Estrategias de Desarrollo 13](#_Toc161668206)

[5. Enfatizar Gestión de Configuración y Versionamiento 13](#_Toc161668207)

[6. Abordar Monitoreo y Mantenimiento 13](#_Toc161668208)

[7. Destacar Estrategias de Pruebas 13](#_Toc161668209)

[8. Abordar Aspectos de Seguridad 13](#_Toc161668210)

[RESTRICIONES Y LIMITACIONES 13](#_Toc161668211)

[1. Complejidad Operativa: 14](#_Toc161668212)

[2. Cambio Cultural: 14](#_Toc161668213)

[3. Requisitos de Infraestructura: 14](#_Toc161668214)

[4. Coordinación entre Microservicios: 14](#_Toc161668215)

[5. Seguridad y Gobernanza: 14](#_Toc161668216)

[6. Costos Operativos: 14](#_Toc161668217)

[7. Complejidad en Pruebas: 14](#_Toc161668218)

[8. Mantenimiento de Versiones: 15](#_Toc161668219)

[9. Tolerancia a Fallos: 15](#_Toc161668220)

[10. Capacidades de Monitoreo: 15](#_Toc161668221)

[DEFINICIONES Y ABREVIATURAS 15](#_Toc161668222)

[Definiciones 15](#_Toc161668223)

[Microservicio: 15](#_Toc161668224)

[Orquestación: 15](#_Toc161668225)

[Contenerización: 15](#_Toc161668226)

[Despliegue Continuo: 16](#_Toc161668227)

[Escalabilidad Horizontal: 16](#_Toc161668228)

[Circuit Breaker: 16](#_Toc161668229)

[Integración Continua (CI): 16](#_Toc161668230)

[Entrega Continua (CD): 16](#_Toc161668231)

[Monitoreo de Microservicios: 16](#_Toc161668232)

[Abreviaturas 16](#_Toc161668233)

[REQUISITOS 17](#_Toc161668234)

[Requisitos Funcionales: 17](#_Toc161668235)

[1. Gestión de Microservicios: 17](#_Toc161668236)

[2. Orquestación y Coordinación: 17](#_Toc161668237)

[3. Despliegue Continuo: 18](#_Toc161668238)

[4. Interoperabilidad: 18](#_Toc161668239)

[5. Gestión de Configuración: 18](#_Toc161668240)

[Requisitos No Funcionales: 18](#_Toc161668241)

[1. Rendimiento: 18](#_Toc161668242)

[2. Escalabilidad: 18](#_Toc161668243)

[3. Seguridad: 18](#_Toc161668244)

[4. Fiabilidad y Disponibilidad: 18](#_Toc161668245)

[5. Mantenibilidad: 19](#_Toc161668246)

[Otros Requisitos: 19](#_Toc161668247)

[1. Documentación: 19](#_Toc161668248)

[2. Colaboración y Formación: 19](#_Toc161668249)

[3. Compatibilidad con Estándares: 19](#_Toc161668250)

[4. Monitorización y Análisis: 19](#_Toc161668251)

[DEFINICIÓN DE LA ARQUITECTURA 19](#_Toc161668252)

[Modelo de Referencia 20](#_Toc161668253)

[Angular Client 20](#_Toc161668254)

[Identity Provider 21](#_Toc161668255)

[API Gateway 21](#_Toc161668256)

[Static Content 21](#_Toc161668257)

[Microservices 21](#_Toc161668258)

[Sonarqube 21](#_Toc161668259)

[Kafka 21](#_Toc161668260)

[Portainer 21](#_Toc161668261)

[Docker 22](#_Toc161668262)

[Docker Compose 22](#_Toc161668263)

[Docker Network 22](#_Toc161668264)

[Database Config 22](#_Toc161668265)

[Database Client 22](#_Toc161668266)

[Nexus Repository 22](#_Toc161668267)

[OpenFeign 22](#_Toc161668268)

[Remote Services 22](#_Toc161668269)

[Ecosystem Core 23](#_Toc161668270)

[Ecosystem Domain 23](#_Toc161668271)

[Service Discovery 23](#_Toc161668272)

[Management 23](#_Toc161668273)

[Spring Config 23](#_Toc161668274)

[Gitlab Repository 23](#_Toc161668275)

[JUnit 23](#_Toc161668276)

[Maven 23](#_Toc161668277)

[Archetype Maven 24](#_Toc161668278)

[Diagrama de componente (Business Service) 24](#_Toc161668279)

[Capa de presentación: 25](#_Toc161668280)

[Clientes externos: 25](#_Toc161668281)

[Capa de servicio spring cloud 25](#_Toc161668282)

[Capa de negocio 25](#_Toc161668283)

[Capa de servicios externos 25](#_Toc161668284)

[Capa de persistencia 25](#_Toc161668285)

[Políticas y Lineamientos 25](#_Toc161668286)

[Arquitectura Orientada a Microservicios: 26](#_Toc161668287)

[Descomposición de Servicios: 26](#_Toc161668288)

[Interfaz RESTful: 26](#_Toc161668289)

[Contratos de API Bien Definidos: 26](#_Toc161668290)

[Gestión Centralizada de Configuración: 26](#_Toc161668291)

[Monitoreo y Registro: 26](#_Toc161668292)

[Seguridad y Autorización: 27](#_Toc161668293)

[Despliegue Continuo: 27](#_Toc161668294)

[Pruebas Automatizadas: 27](#_Toc161668295)

[Documentación Clara y Accesible: 27](#_Toc161668296)

[Vista Lógica 27](#_Toc161668297)

[Definición de la Estructura de un Microservicio 28](#_Toc161668298)

[Directorios 29](#_Toc161668299)

[Paquetes 29](#_Toc161668300)

[Archivos 30](#_Toc161668301)

[Reglas de Implementación 31](#_Toc161668302)

[Categorías y Directrices 31](#_Toc161668303)

[Tecnologías Utilizadas 34](#_Toc161668304)

[Repositorio de Código Fuente 35](#_Toc161668305)

[1. Colaboración y Desarrollo Distribuido: 35](#_Toc161668306)

[2. Ciclo de Vida Integrado: 35](#_Toc161668307)

[3. Integración Continua y Despliegue Continuo (CI/CD): 35](#_Toc161668308)

[4. Gestión Avanzada de Ramas y Fusiones: 35](#_Toc161668309)

[5. Gestión de Problemas Integrada: 35](#_Toc161668310)

[6. Control de Acceso y Seguridad: 35](#_Toc161668311)

[7. Registro y Seguimiento de Cambios: 36](#_Toc161668312)

[8. Gestión de Releases y Versiones: 36](#_Toc161668313)

[9. Soporte para Artefactos y Pipelines: 36](#_Toc161668314)

[Lenguaje Desarrollo Backend con Java 36](#_Toc161668315)

[Robustez de Java: 36](#_Toc161668316)

[Eficiencia de Spring Boot: 37](#_Toc161668317)

[Adopción en la Industria: 37](#_Toc161668318)

[Aspectos de Microservicios: 37](#_Toc161668319)

[Contenerización con Docker 38](#_Toc161668320)

[Orquestación con Kubernetes 38](#_Toc161668321)

[Gestión de Configuración con Spring Cloud Config 38](#_Toc161668322)

[Comunicación entre Microservicios con Spring Cloud y Feign 38](#_Toc161668323)

[Registro y Descubrimiento de Servicios con Eureka 38](#_Toc161668324)

[Gateway API con Spring Cloud Gateway 39](#_Toc161668325)

[Manejo de Eventos con Apache Kafka 39](#_Toc161668326)

[Persistencia de Datos con Spring Data JPA o MongoDB 39](#_Toc161668327)

[Pruebas con JUnit y Mockito 39](#_Toc161668328)

[Monitoreo y Trazabilidad con Spring Boot Actuator y Zipkin 39](#_Toc161668329)

[Gestión de Dependencias con Maven 40](#_Toc161668330)

[Repositorio de Código Fuente 40](#_Toc161668331)

[Servidores de Despliegues 40](#_Toc161668332)

[Ambiente Productivo 41](#_Toc161668333)

[Ambiente Desarrollo, QA, Presentación 41](#_Toc161668334)

[Equipo Desarrollador 41](#_Toc161668335)

[Equipo Tester 41](#_Toc161668336)

[Estrategias de Desarrollo 41](#_Toc161668337)

[Fase de Diseño 42](#_Toc161668338)

[Enfoque Microservicios 42](#_Toc161668339)

[Identificación y Definición de Límites del Servicio 42](#_Toc161668340)

[Elección de Tecnologías Adecuadas 42](#_Toc161668341)

[Creación de Interfaces Bien Definidas 42](#_Toc161668342)

[Descomposición de Monolitos 43](#_Toc161668343)

[Evaluación Inicial 43](#_Toc161668344)

[Definición de Dominios de Microservicios 43](#_Toc161668345)

[Estrategias de Descomposición 43](#_Toc161668346)

[Garantía de Continuidad del Servicio 44](#_Toc161668347)

[Prácticas de Pruebas 44](#_Toc161668348)

[Testing Unitario 44](#_Toc161668349)

[Implementación de Prácticas de Pruebas Unitarias 44](#_Toc161668350)

[Integración Continua y Pruebas Unitarias 45](#_Toc161668351)

[Cobertura de Código y Análisis Estático 45](#_Toc161668352)

[Escalabilidad y Mantenimiento 45](#_Toc161668353)

[Pruebas de Integración 46](#_Toc161668354)

[Pruebas de Estrés y Rendimiento 46](#_Toc161668355)

[Integración Continua 46](#_Toc161668356)

[Automatización de Construcción 46](#_Toc161668357)

[Retroalimentación Rápida 46](#_Toc161668358)

[Despliegue Continuo 47](#_Toc161668359)

[Automatización de Implementación 47](#_Toc161668360)

[Proceso de Despliegue Continuo 47](#_Toc161668361)

[Gestión de Versiones 48](#_Toc161668362)

[Estrategias y Prácticas 48](#_Toc161668363)

[Proceso de Implementación 48](#_Toc161668364)

[Garantizando Eficiencia y Calidad 49](#_Toc161668365)

[Monitoreo Continuo 49](#_Toc161668366)

[Estrategias y Prácticas 49](#_Toc161668367)

[Proceso de Monitoreo Continuo 49](#_Toc161668368)

[Prácticas de Seguridad 50](#_Toc161668369)

[Estrategias y Prácticas 50](#_Toc161668370)

[Proceso de Implementación Segura 50](#_Toc161668371)

# RESUMEN

El presente documento describe la arquitectura de software definida en la compañía para la construcción de soluciones basada en microservicios.

# INTRODUCCION

En el dinámico mundo de desarrollo de software, la evolución constante de las tecnologías y las crecientes demandas de los usuarios han impulsado un cambio significativo en las estrategias arquitectónicas. La arquitectura de microservicios ha surgido como una respuesta eficiente y escalable para enfrentar los desafíos contemporáneos en el desarrollo de software. Este enfoque modular y descentralizado ha ganado popularidad por su capacidad para proporcionar flexibilidad, escalabilidad y mantenibilidad en comparación con las arquitecturas monolíticas tradicionales.

En el contexto de una fábrica de software, donde la producción de soluciones tecnológicas de alta calidad es la principal meta, la adopción de la arquitectura de microservicios se presenta como una estrategia clave. Esta arquitectura permite la creación de servicios independientes y altamente especializados, cada uno enfocado en cumplir una función específica. Así, la fábrica de software puede ofrecer soluciones personalizadas, escalables y fácilmente actualizables, manteniendo una respuesta ágil a los cambios en los requisitos del cliente y del mercado.

Este documento explorará los principios fundamentales y las mejores prácticas asociadas con la implementación de una arquitectura de microservicios en la fábrica de software ADA. Desde el diseño modular hasta las estrategias de despliegue y la gestión continua, examinaremos cómo esta arquitectura puede potenciar la eficiencia operativa, acelerar los ciclos de desarrollo y mejorar la calidad de los productos de software, todo mientras se fomenta la colaboración y la innovación dentro del equipo de desarrollo. A lo largo de este viaje, descubriremos cómo los microservicios no solo transforman la estructura técnica, sino que también impactan positivamente en la cultura y la capacidad de entrega de la fábrica de software.

# PROPOSITO GENERAL

El propósito fundamental de esta descripción arquitectónica de microservicios en la fábrica de software de la compañía es proporcionar una visión integral y estratégica de la implementación de una arquitectura que revoluciona la manera en que concebimos, construimos y entregamos soluciones tecnológicas. Este documento tiene como objetivo principal:

## 1. Entender los Principios Fundamentales

Ofrecer una comprensión general de los principios esenciales que respaldan la arquitectura de microservicios. Esto incluye la modularidad, la independencia de servicios, la escalabilidad y la capacidad de despliegue continuo, entre otros aspectos clave.

## 2. Destacar las Ventajas Estratégicas

Identificar y resaltar las ventajas estratégicas que la arquitectura de microservicios aporta a una fábrica de software. Estas ventajas incluyen la agilidad en el desarrollo, la capacidad de adaptación a cambios rápidos en los requisitos y la mejora continua de la calidad del software.

## 3. Guiar en la Implementación Efectiva

Proporcionar orientación detallada sobre cómo implementar efectivamente la arquitectura de microservicios en el contexto específico de una fábrica de software. Esto abarcará desde el diseño inicial hasta la gestión operativa, destacando las mejores prácticas y abordando posibles desafíos.

## 4. Facilitar la Colaboración Interdisciplinaria

Fomentar la colaboración entre equipos interdisciplinarios dentro de la fábrica de software. La arquitectura de microservicios no solo transforma la estructura técnica, sino que también impacta en la cultura de desarrollo, promoviendo la colaboración, la innovación y la responsabilidad compartida.

## 5. Orientar hacia la Innovación Continua

Motivar a la fábrica de software a adoptar un enfoque proactivo hacia la innovación continua. La arquitectura de microservicios no solo es una evolución técnica, sino también una invitación a repensar y mejorar los procesos, la cultura organizativa y la entrega de productos de software.

En conjunto, esta descripción busca servir como un recurso integral que inspire y guíe a la fábrica de software hacia una arquitectura moderna y centrada en la eficiencia, la flexibilidad y la excelencia en el desarrollo de software.

# ALCANCE

La descripción abarcará diversos aspectos, desde los principios fundamentales hasta la implementación práctica, con el propósito de orientar a todos los actores involucrados en el desarrollo y gestión de software en ADA. El alcance del documento incluirá:

## 1. Fundamentación Teórica

Explorar los principios y conceptos fundamentales que respaldan la arquitectura de microservicios. Esto proporcionará una base sólida para comprender el porqué de la elección de esta arquitectura y cómo se alinea con los objetivos estratégicos de ADA.

## 2. Estructura de la Arquitectura

Detallar la estructura de la arquitectura de microservicios en ADA, incluyendo la división de responsabilidades entre los distintos microservicios, las interconexiones y la forma en que colaboran para cumplir los objetivos globales de la fábrica de software.

## 3. Tecnologías Utilizadas

Describir las tecnologías específicas empleadas en la implementación de microservicios en ADA. Esto abarcará desde el uso de Spring Boot y Java 21 hasta herramientas de contenerización con Docker y orquestación con Spring Cloud, proporcionando una visión integral de la pila tecnológica.

## 4. Proceso de Desarrollo

Examinar el proceso de desarrollo de software bajo la arquitectura de microservicios en ADA. Esto incluirá desde la fase de diseño y desarrollo hasta las pruebas unitarias, integración continua y despliegue continuo, destacando las metodologías y prácticas específicas adoptadas.

## 5. Gestión de la Configuración y Versionamiento

Abordar las estrategias implementadas para la gestión de la configuración y el versionamiento de los microservicios. Esto asegurará la coherencia y trazabilidad a lo largo del ciclo de vida del software.

## 6. Monitoreo y Mantenimiento

Detallar las prácticas y herramientas utilizadas para el monitoreo continuo de los microservicios en producción, así como las estrategias de mantenimiento y actualización para garantizar un rendimiento óptimo y una rápida respuesta a los cambios.

## 7. Pruebas y Validación

Explorar las estrategias de pruebas implementadas en ADA, tanto a nivel unitario como de integración, asegurando la calidad y confiabilidad de cada microservicio y del sistema en su conjunto.

## 8. Seguridad

Abordar las políticas y prácticas de seguridad implementadas para proteger los microservicios y los datos en ADA, garantizando la integridad y confidencialidad de la información.

Este documento tiene como objetivo ser un recurso integral que sirva de guía para todos los profesionales involucrados en el ciclo de vida del software en ADA, desde desarrolladores y arquitectos hasta gerentes de proyecto y responsables de calidad.

# OBJETIVOS

La descripción detallada de la arquitectura de microservicios en la fábrica de software ADA tiene como objetivos principales proporcionar una comprensión profunda de la implementación y funcionamiento de esta arquitectura. Los objetivos específicos son:

## 1. Clarificar los Principios Rectores

Explorar y comunicar de manera clara los principios fundamentales que guían la elección y aplicación de la arquitectura de microservicios en ADA. Esto incluirá la modularidad, independencia de servicios, escalabilidad y mantenibilidad.

## 2. Detallar la Estructura Microservicios

Proporcionar una descripción detallada de la estructura de los microservicios en ADA. Esto abarcará la identificación de servicios específicos, su responsabilidad y la interconexión entre ellos, dando una visión global de la arquitectura.

## 3. Especificar Tecnologías Utilizadas

Describir las tecnologías clave utilizadas en la implementación de microservicios en ADA, proporcionando una visión clara de la pila tecnológica. Esto incluirá detalles sobre frameworks, lenguajes de programación, herramientas de contenerización, orquestación y otras tecnologías relevantes.

## 4. Resaltar Estrategias de Desarrollo

Detallar las estrategias y procesos específicos adoptados para el desarrollo de microservicios en ADA. Esto incluirá desde la fase de diseño hasta las prácticas de pruebas, integración continua y despliegue continuo, garantizando la eficiencia y calidad en todo el ciclo de vida del software.

## 5. Enfatizar Gestión de Configuración y Versionamiento

Enfocarse en las prácticas y herramientas utilizadas para la gestión de configuración y el versionamiento de los microservicios en ADA. Esto asegurará la consistencia y trazabilidad a lo largo del tiempo.

## 6. Abordar Monitoreo y Mantenimiento

Explorar las estrategias de monitoreo implementadas para evaluar y mantener el rendimiento óptimo de los microservicios en producción. Además, detallar las estrategias de mantenimiento y actualización para garantizar la adaptabilidad a cambios y la resiliencia del sistema.

## 7. Destacar Estrategias de Pruebas

Enfatizar las estrategias de pruebas utilizadas en ADA, tanto a nivel unitario como de integración. Esto garantizará la calidad y fiabilidad de cada microservicio y del sistema en su conjunto.

## 8. Abordar Aspectos de Seguridad

Detallar las políticas y prácticas de seguridad implementadas para salvaguardar los microservicios y los datos en ADA, asegurando la integridad y confidencialidad de la información.

Estos objetivos buscan proporcionar una descripción completa y coherente de la arquitectura de microservicios en la fábrica de software ADA, sirviendo como guía integral para todos los involucrados en el proceso de desarrollo y gestión de software.

# RESTRICIONES Y LIMITACIONES

A pesar de los beneficios inherentes a la arquitectura de microservicios, es crucial reconocer y abordar las restricciones y limitaciones específicas que la fábrica de software ADA puede enfrentar durante la implementación. Estas restricciones son esenciales para una comprensión completa y realista de la arquitectura, y ayudarán a los equipos a tomar decisiones informadas. Entre las restricciones y limitaciones relevantes se encuentran:

## 1. Complejidad Operativa:

La adopción de microservicios introduce una complejidad operativa adicional, especialmente en términos de monitoreo, orquestación y mantenimiento. La fábrica de software ADA debe abordar este desafío con herramientas y prácticas sólidas.

## 2. Cambio Cultural:

La transición hacia microservicios a menudo requiere un cambio cultural significativo en términos de desarrollo, despliegue y colaboración entre equipos. Puede haber resistencia al cambio que necesita ser gestionada cuidadosamente.

## 3. Requisitos de Infraestructura:

La implementación de microservicios puede requerir una infraestructura específica y robusta. Asegurarse de que los recursos de hardware y red sean adecuados es esencial para evitar cuellos de botella y garantizar un rendimiento óptimo.

## 4. Coordinación entre Microservicios:

La coordinación entre microservicios, especialmente en situaciones de dependencias complejas, puede ser un desafío. La fábrica de software ADA debe abordar la comunicación y la sincronización efectiva entre servicios.

## 5. Seguridad y Gobernanza:

La fragmentación de servicios puede plantear desafíos en términos de seguridad y gobernanza. La implementación de medidas de seguridad robustas y políticas de gobernanza es crucial.

## 6. Costos Operativos:

Aunque la arquitectura de microservicios ofrece escalabilidad y flexibilidad, también puede aumentar los costos operativos, especialmente si no se gestiona adecuadamente. La fábrica de software ADA debe estar consciente de los costos asociados con la infraestructura y las herramientas.

## 7. Complejidad en Pruebas:

La implementación de microservicios puede introducir complejidades adicionales en las estrategias de prueba. Garantizar una cobertura de prueba adecuada y la integración continua es vital.

## 8. Mantenimiento de Versiones:

El mantenimiento de versiones y la gestión de dependencias entre microservicios pueden resultar desafiantes. Es esencial establecer políticas y prácticas claras para evitar conflictos.

## 9. Tolerancia a Fallos:

La arquitectura de microservicios debe ser capaz de manejar la tolerancia a fallos de manera efectiva. La fábrica de software ADA debe abordar la resiliencia y el manejo de errores de manera proactiva.

## 10. Capacidades de Monitoreo:

La implementación de sistemas de monitoreo efectivos para evaluar el rendimiento y la salud de los microservicios puede ser un desafío. La fábrica de software ADA debe invertir en herramientas y procesos adecuados.

Reconocer estas restricciones y limitaciones permitirá a la fábrica de software ADA abordar proactivamente los desafíos, mitigar riesgos y optimizar la implementación de la arquitectura de microservicios.

# DEFINICIONES Y ABREVIATURAS

## Definiciones

### Microservicio:

Un componente de software independiente y auto contenible que realiza una función específica dentro del contexto de la arquitectura de microservicios.

### Orquestación:

La coordinación y gestión centralizada de los microservicios para lograr un flujo de trabajo empresarial coherente.

### Contenerización:

La técnica de encapsular una aplicación y sus dependencias en contenedores ligeros, como Docker, para garantizar la portabilidad y consistencia en distintos entornos.

### Despliegue Continuo:

La práctica de implementar automáticamente los cambios en el entorno de producción después de pasar satisfactoriamente por las etapas de integración y pruebas.

### Escalabilidad Horizontal:

La capacidad de aumentar el rendimiento y la capacidad de los microservicios mediante la adición de instancias adicionales en lugar de mejorar los recursos de una única instancia.

### Circuit Breaker:

Un patrón de diseño que detecta y gestiona errores en microservicios, evitando la propagación de fallas y mejorando la resiliencia.

### Integración Continua (CI):

La práctica de combinar continuamente cambios en el código fuente en un repositorio compartido, seguido de pruebas automáticas y análisis estático.

### Entrega Continua (CD):

La práctica de llevar automáticamente el código desde la integración continua hasta un entorno de producción de manera confiable y eficiente.

### Monitoreo de Microservicios:

Definición: La observación y registro continuo de la actividad y rendimiento de los microservicios en producción para garantizar la disponibilidad y detectar posibles problemas.

## Abreviaturas

|  |  |
| --- | --- |
| Termino | Definición |
| Springboot | Spring Boot es un framework de código abierto para Java que facilita y acelera el desarrollo de aplicaciones web y microservicios. |
| Docker | Docker es una plataforma de código abierto que facilita el desarrollo, la implementación y la ejecución de aplicaciones mediante contenedores. |
| CI/CD | Integración Continua/Entrega Continua. |
| JVM | Máquina Virtual de Java. |
| API | Interfaz de Programación de Aplicaciones. |
| DBMS | Sistema de Gestión de Bases de Datos. |
| REST | Transferencia de Estado Representacional. |
| DNS | Sistema de Nombres de Dominio. |
| JWT | Token de Seguridad JSON. |
| SLA | Acuerdo de Nivel de Servicio. |
| Feign | Biblioteca Java de código abierto desarrollada por Netflix que simplifica la creación de clientes HTTP. |

Estas definiciones y abreviaturas se han establecido para garantizar una comunicación clara y coherente dentro de la fábrica de software ADA, facilitando así la comprensión y colaboración efectiva entre todos los miembros del equipo involucrados en la arquitectura de microservicios.

# REQUISITOS

La descripción de la arquitectura de microservicios en la fábrica de software ADA debe abordar requisitos funcionales, no funcionales y otros para garantizar una implementación efectiva y exitosa.

A continuación, se presentan los requisitos clave:

## Requisitos Funcionales:

### 1. Gestión de Microservicios:

* RF1: Los desarrolladores deben poder registrar nuevos microservicios en el sistema.
* RF2: El sistema debe permitir la actualización y retirada de microservicios de manera controlada.
* RF3: Se debe proporcionar una interfaz para gestionar las dependencias entre microservicios.

### 2. Orquestación y Coordinación:

* RF4: La arquitectura debe admitir mecanismos de orquestación para coordinar la ejecución de microservicios.
* RF5: Debe ser posible definir flujos de trabajo empresariales utilizando servicios orquestados.

### 3. Despliegue Continuo:

* RF6: La arquitectura debe facilitar el despliegue continuo de nuevos microservicios y actualizaciones.
* RF7: Se debe proporcionar una interfaz para gestionar y monitorizar los despliegues en tiempo real.

### 4. Interoperabilidad:

* RF8: Los microservicios deben ser interoperables y capaces de comunicarse eficientemente entre sí mediante interfaces bien definidas.
* RF9: Se deben establecer estándares para la comunicación entre microservicios, como RESTful APIs.

### 5. Gestión de Configuración:

* RF10: La arquitectura debe admitir la gestión de configuración para garantizar la consistencia entre entornos.
* RF11: Los cambios en la configuración deben ser rastreables y reversibles.

## Requisitos No Funcionales:

### 1. Rendimiento:

* RNF1: La arquitectura debe garantizar un rendimiento aceptable incluso bajo cargas de trabajo elevadas.
* RNF2: El tiempo de respuesta promedio para las solicitudes de microservicios debe cumplir con los SLA definidos.

### 2. Escalabilidad:

* RNF3: La arquitectura debe ser fácilmente escalable horizontalmente para gestionar aumentos en la carga de trabajo.
* RNF4: La escalabilidad debe ser automática y basada en políticas.

### 3. Seguridad:

* RNF5: Se debe implementar un modelo de seguridad robusto para proteger la integridad y confidencialidad de los datos.
* RNF6: Los microservicios deben ser capaces de autenticar y autorizar solicitudes de manera segura.

### 4. Fiabilidad y Disponibilidad:

* RNF7: La arquitectura debe ser altamente confiable y capaz de recuperarse automáticamente de fallos.
* RNF8: La disponibilidad del sistema debe cumplir con los objetivos de SLA.

### 5. Mantenibilidad:

* RNF9: La arquitectura debe facilitar la detección y corrección eficientes de errores.
* RNF10: Debe ser posible realizar actualizaciones de microservicios sin afectar la disponibilidad del sistema.

## Otros Requisitos:

### 1. Documentación:

* RO1: Se debe proporcionar documentación exhaustiva que explique la arquitectura, los microservicios y las prácticas recomendadas.
* RO2: La documentación debe mantenerse actualizada a medida que evoluciona la arquitectura.

### 2. Colaboración y Formación:

* RO3: Fomentar la colaboración entre equipos mediante la creación de foros y sesiones de formación.
* RO4: Facilitar la formación continua para que los equipos estén al tanto de las últimas prácticas y tecnologías.

### 3. Compatibilidad con Estándares:

* RO5: Asegurar que la arquitectura sea compatible con estándares de la industria para garantizar la interoperabilidad.

### 4. Monitorización y Análisis:

* RO6: Implementar herramientas de monitorización para recopilar datos sobre el rendimiento y la utilización de los microservicios.
* RO7: Facilitar el análisis de datos recopilados para la mejora continua de la arquitectura.

Estos requisitos garantizarán que la descripción de la arquitectura de microservicios en la fábrica de software ADA sea completa, comprensible y capaz de guiar la implementación exitosa de esta arquitectura moderna.

# DEFINICIÓN DE LA ARQUITECTURA

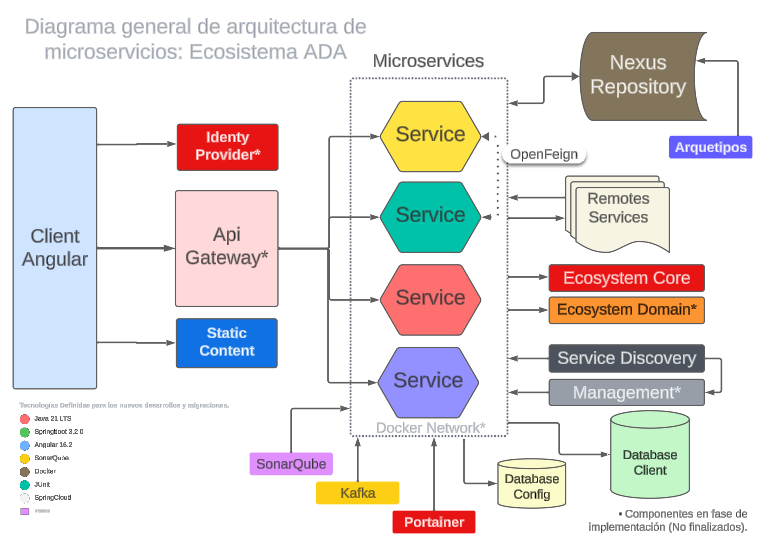
La arquitectura orientada a microservicios es un enfoque de diseño de software que estructura una aplicación como una colección de servicios independientes, pequeños y altamente especializados, conocidos como microservicios. En contraste con las arquitecturas monolíticas, donde una aplicación se desarrolla como una única unidad, los microservicios permiten dividir una aplicación en partes más manejables y autónomas.

Se define el uso de arquitectura orientada a microservicios para la implementación de las nuevas soluciones en la fábrica de software ADA.

A continuación, se presenta un modelo de referencia que describe la implementación de la misma.

## Modelo de Referencia

El modelo de referencia para la arquitectura de microservicios en la fábrica de software ADA define una estructura modular y escalable que incorpora diversos artefactos esenciales para el desarrollo, despliegue y gestiones eficientes de aplicaciones. A continuación, se describen los artefactos clave teniendo como base el siguiente gráfico de referencia:



### Angular Client

El artefacto Angular Client representa la interfaz de usuario basada en el framework Angular. Se encarga de proporcionar una experiencia interactiva y amigable para los usuarios finales, interactuando con los microservicios a través de solicitudes HTTP.

### Identity Provider

El Identity Provider gestiona la autenticación y autorización de usuarios en el sistema. Proporciona un servicio centralizado para la gestión de identidades y la emisión de tokens de seguridad.

### API Gateway

El API Gateway actúa como un punto de entrada único para todas las solicitudes de clientes. Proporciona funciones como enrutamiento, equilibrio de carga y autenticación, simplificando la interacción entre los clientes y los microservicios.

### Static Content

El artefacto Static Content se encarga de la gestión y entrega de contenido estático, como imágenes, hojas de estilo y archivos JavaScript, optimizando el rendimiento de la interfaz de usuario.

### Microservices

Los Microservicios son componentes independientes y autocontenibles que ejecutan funciones específicas de negocio. Se comunican entre sí de manera eficiente para lograr operaciones más complejas.

### Sonarqube

Sonarqube proporciona análisis estático de código para evaluar la calidad del código fuente. Identifica posibles problemas y áreas de mejora, contribuyendo a la mantenibilidad y fiabilidad del sistema.

### Kafka

Kafka actúa como un sistema de mensajería y transmisión de eventos, permitiendo la comunicación asincrónica entre microservicios y la transmisión en tiempo real de eventos clave.

### Portainer

Portainer es una interfaz de usuario para gestionar y visualizar contenedores Docker. Facilita la administración eficiente de los contenedores en entornos Docker.

### Docker

Docker proporciona la capacidad de contenerizar aplicaciones, encapsulando dependencias y configuraciones en contenedores ligeros y portátiles.

### Docker Compose

Docker Compose se utiliza para definir y gestionar aplicaciones Docker multi-contenedor. Facilita la orquestación de contenedores y la configuración de servicios relacionados.

### Docker Network

La red Docker permite la comunicación eficiente entre contenedores Docker, asegurando una interacción fluida entre los microservicios.

### Database Config

La configuración de la base de datos aborda los aspectos relacionados con la configuración y conexión de bases de datos, garantizando una gestión eficiente de los datos persistentes.

### Database Client

El cliente de base de datos facilita la interacción con las bases de datos, ejecutando consultas y transacciones necesarias para el funcionamiento de los microservicios.

### Nexus Repository

Nexus Repository actúa como un repositorio centralizado para gestionar y almacenar artefactos de software, como bibliotecas, dependencias y archivos JAR.

### OpenFeign

OpenFeign simplifica la creación de clientes HTTP para la comunicación entre microservicios. Facilita la integración y el consumo de servicios remotos.

### Remote Services

Los Remote Services son microservicios externos que se consumen desde la arquitectura principal, permitiendo la colaboración con servicios externos para ampliar la funcionalidad.

### Ecosystem Core

El Ecosystem Core representa el núcleo central de la arquitectura, donde se definen y gestionan los servicios fundamentales compartidos por todos los microservicios.

### Ecosystem Domain

El Ecosystem Domain aborda los aspectos específicos del dominio de la aplicación, definiendo microservicios especializados para cumplir con los requisitos del negocio.

### Service Discovery

El Service Discovery se encarga de la identificación y registro dinámico de los microservicios en la arquitectura, facilitando la comunicación y la escalabilidad.

### Management

El artefacto de Management aborda aspectos relacionados con la gestión y supervisión de los microservicios, garantizando un rendimiento óptimo y una resiliencia adecuada.

### Spring Config

Spring Config proporciona un sistema centralizado para la gestión de configuraciones, asegurando una coherencia y actualización eficientes en los microservicios.

### Gitlab Repository

El repositorio Gitlab es un sistema centralizado para gestionar y versionar el código fuente, facilitando el desarrollo colaborativo y la integración continua.

### JUnit

Junit es un marco de pruebas unitarias para el desarrollo de software en Java. Se utiliza para garantizar la calidad y fiabilidad del código a nivel de unidad.

### Maven

Maven es una herramienta de gestión de proyectos que simplifica la construcción, prueba y distribución de aplicaciones Java.

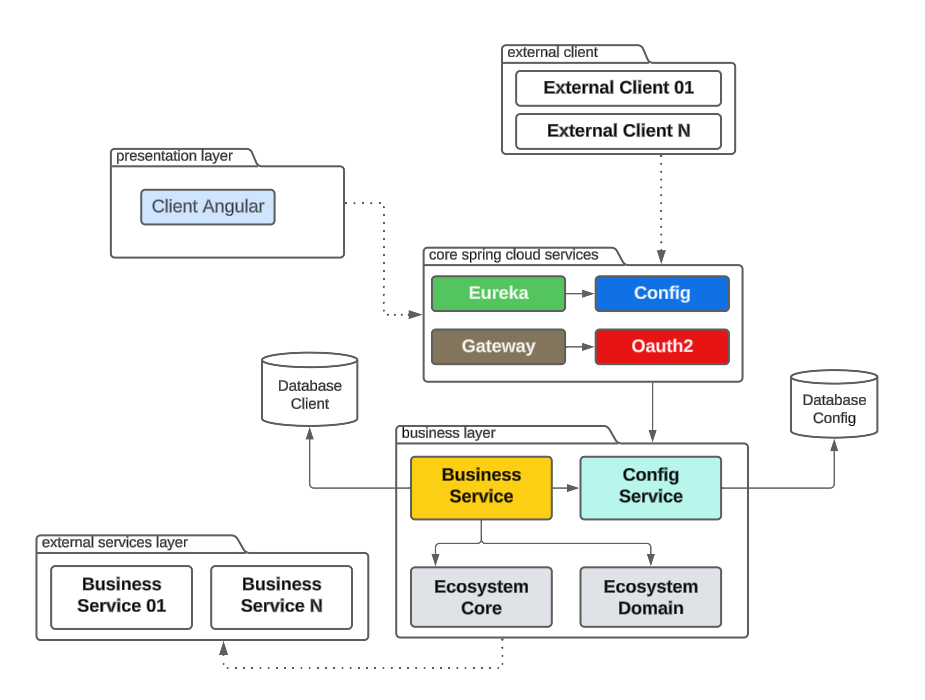
### Archetype Maven

Los Archetypes Maven son plantillas predefinidas que simplifican la creación de nuevos proyectos Maven, asegurando una estructura coherente y mejores prácticas.

Este modelo de referencia establece una base sólida para la arquitectura de microservicios en la fábrica de software ADA, abordando de manera integral los artefactos clave necesarios para el desarrollo y despliegue eficientes de aplicaciones modulares y escalables.

## Diagrama de componente (Business Service)

El diagrama general de componentes ayuda a visualizar la estructura y las interacciones clave entre los componentes que componen un servicio específico. Como se puede visualizar en la siguiente imagen:



A continuación, se describen los componentes comunes que interactúan en dicho diagrama:

### Capa de presentación:

* Cliente Angular: Frontend que puede requerir el microservicio.

### Clientes externos:

Es posible que la solución sea consumida por un servicio externo por lo tanto la solución es desplegada bajo el marco de autorización spring security y Oauth2 para la correcta validación y verificación de acceso a los recursos.

### Capa de servicio spring cloud

* Eureka: Servicio de registro.
* Config: Servicio de configuraciones.
* Gateway: Servicio de enrutamiento.
* Oauth2: Servicio de autorización y autenticación.

### Capa de negocio

* Business Service: Servicio del negocio. Contine la lógica de la solución.
* Config Service: Servicio que provee configuraciones del ecosistema de servicios.
* EcosystemCore: Librería que contiene clases base para la correcta operación del servicio.
* EcosystemDomain: Librería que contiene el modelo de datos (dto) del negocio.

### Capa de servicios externos

Algunos soluciones requieren de otros servicios para operar. Esta capa provee acceso a esos servicios por medio de interfaces Feign.

### Capa de persistencia

* Database Config: Base de datos de configuración del ecosistema de servicios.
* Database Client: Base de datos del negocio del cliente.

## Políticas y Lineamientos

La aplicación de políticas y lineamientos para la construcción de microservicios en la fábrica de software ADA puede variar según las necesidades y requisitos específicos de la organización y los clientes. Sin embargo, se definen de manera general de las políticas y lineamientos que deben considerarse a la hora de implementar soluciones:

### Arquitectura Orientada a Microservicios:

**Descripción**: La fábrica de software ADA sigue un enfoque de arquitectura orientada a microservicios para el desarrollo de sistemas.

**Política**: Todos los nuevos desarrollos seguirán la arquitectura de microservicios para garantizar la escalabilidad, flexibilidad y mantenibilidad.

### Descomposición de Servicios:

**Descripción**: Cada microservicio deberá tener una responsabilidad única y estar completamente desacoplado de otros servicios.

**Política**: La descomposición de servicios se realizará cuidadosamente para evitar dependencias innecesarias y promover la independencia de cada microservicio.

### Interfaz RESTful:

**Descripción**: Las interfaces de los microservicios seguirán los principios RESTful para la comunicación entre servicios.

**Política**: Todas las comunicaciones entre microservicios se realizarán a través de interfaces RESTful para garantizar la simplicidad y la interoperabilidad.

### Contratos de API Bien Definidos:

**Descripción**: Cada microservicio tendrá un contrato de API bien definido que documenta las operaciones y los formatos de datos admitidos.

**Política**: Los equipos de desarrollo deben seguir los contratos de API establecidos y mantener la documentación actualizada.

### Gestión Centralizada de Configuración:

**Descripción**: La configuración de los microservicios se gestionará centralmente para facilitar la administración y actualizaciones.

**Política**: La configuración se almacenará en un lugar central y se accederá a través de servicios de configuración para garantizar consistencia y control.

### Monitoreo y Registro:

**Descripción**: Todos los microservicios incluirán capacidades de monitoreo y registro para facilitar la identificación y resolución de problemas.

**Política**: La implementación de monitoreo y registro será obligatoria, y los equipos de operaciones tendrán acceso a herramientas centralizadas.

### Seguridad y Autorización:

**Descripción**: La seguridad será una consideración clave en el diseño de microservicios, incluida la autenticación y autorización.

**Política**: Se seguirán las mejores prácticas de seguridad para garantizar la protección de datos y sistemas, y se implementarán mecanismos de autorización adecuados.

### Despliegue Continuo:

**Descripción**: La fábrica de software ADA adoptará prácticas de despliegue continuo para acelerar la entrega de nuevas funcionalidades.

**Política**: Se establecerán pipelines de CI/CD para automatizar pruebas, construcción y despliegue, y se fomentará la entrega frecuente y pequeñas iteraciones.

### Pruebas Automatizadas:

**Descripción**: La automatización de pruebas será fundamental para garantizar la calidad y estabilidad de los microservicios.

**Política**: Todos los microservicios deberán contar con suites de pruebas automatizadas, incluidas pruebas unitarias, de integración y de extremo a extremo.

### Documentación Clara y Accesible:

**Descripción**: La documentación de microservicios será clara, completa y accesible para facilitar la comprensión y el uso por parte de otros equipos.

**Política**: Cada microservicio deberá contar con documentación actualizada, incluidos detalles sobre la API, dependencias y configuración.

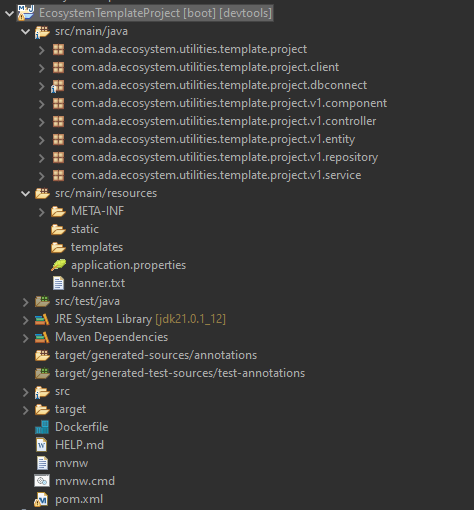
## Vista Lógica

La Vista Lógica del documento de arquitectura de la fábrica de software ADA se enfoca en la descomposición de una aplicación general[[1]](#footnote-1) en elementos clave, centrándose específicamente en los microservicios, bases de datos y sistemas de colas de mensajería[[2]](#footnote-2). Estos componentes son fundamentales para comprender la implementación detallada de la arquitectura de microservicios.

### Definición de la Estructura de un Microservicio

La estructura de un microservicio define la organización y disposición de sus componentes, asegurando la modularidad, mantenibilidad y escalabilidad del servicio.

A continuación, se presenta una descripción detallada de los elementos que conforman la estructura de un microservicio en la fábrica de software ADA:



La imagen se basa en el arquetipo Maven de un proyecto estándar con acceso a datos con el cual se describirá la estructura estándar de un microservicio en la arquitectura propuesta.

Esta estructura proporciona una organización clara y eficiente para el desarrollo, prueba, empaquetado y despliegue de un microservicio en la fábrica de software ADA. Siguiendo estas convenciones, se facilita la colaboración entre equipos y se asegura la coherencia en la implementación de microservicios dentro del ecosistema de la fábrica de software.

#### Directorios

* **src/main/java**: Directorio que contiene el código fuente de la solución.
* **src/main/resources**: Directorio de recursos que implementa la solución como imágenes, archivos de configuración, plantillas.
* **src/test/java**: Directorio que contiene las pruebas unitarias.
* **target**: Artefacto generado de la solución.

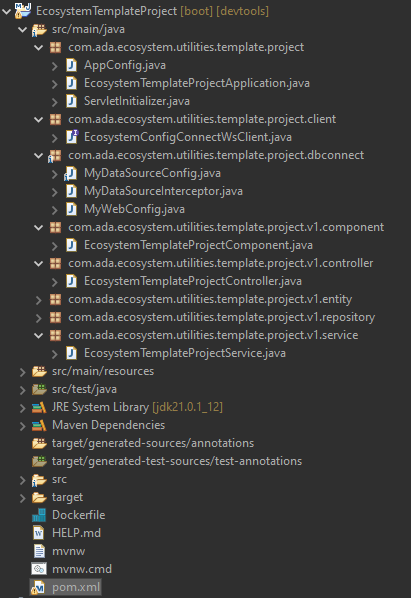
#### Paquetes

La definición de los paquetes se hace con el paquete modelo del arquetipo [com.ada.ecosystem.utilities.template.project] sin embargo esa estructura será cambiada por los parámetros ingresados en la generación del proyecto.

* [**com.ada.ecosystem.utilities.template.project**]: Paquete principal del microservicio.
* [com.ada.ecosystem.utilities.template.project].**client**: Paquete que contiene los clientes feign que consumen servicios externos
* [com.ada.ecosystem.utilities.template.project].**dbconnect**: Paquete que implementa el modelo multicliente.
* [com.ada.ecosystem.utilities.template.project].**component**: Contiene la lógica del negocio de la solución.
* [com.ada.ecosystem.utilities.template.project].**controller**: Contiene los endpoint expuesto en la solución.
* [com.ada.ecosystem.utilities.template.project].**entity**: Contiene las entidad que requiere la solución.
* [com.ada.ecosystem.utilities.template.project].**repository**: Contiene los repositorios que utilizan las entidades para el acceso a datos por medio de la interfaz JPA.
* [com.ada.ecosystem.utilities.template.project].**service**: Orquesta la invocación de los componentes.

### Archivos

A continuación, se describen los archivos genéricos del proyecto, se excluyen los paquetes entity y repository porque dependen del modelo de datos de la solución:



* **AppConfig.java**: Clase de configuración del microservicio. Contiene beans necesarios para el enrutamiento estático y la documentación OpenApi.
* **\*EcosystemTemplateProjectApplication.java**: Clase principal del microservicio. Será cambiada por el nombre ingresado en la generación del arquetipo concatenado por la palabra Application.
* **\*ServletInitializer.java**: Clase utilizada para la ejecución del microservicio.
* **\*EcosystemConfigConnectWsClient.java**: Cliente necesario para leer las configuraciones de orígenes de datos que utilizará el microservicios para la conexión multi-cliente.
* **MyDataSourceConfig.java**: Clase de configuración del modelo de conexión multi-cliente, se deben agregar los paquetes de los repositorios y las entidades para la correcta inicialización del microservicio.
* **\*MyDataSourceInterceptor.java**: Clase que se utiliza para interceptar el token de identificación de la conexión del cliente de la petición.
* **\*MyWebConfig.java**: Clase que implementa la configuración del interceptor y carga las configuraciones de conexión.
* **EcosystemTemplateProjectComponent.java**: Clase base para la implementación de la lógica del negocio. Será cambiada por el nombre ingresado en la generación del arquetipo concatenado por la palabra Component.
* **EcosystemTemplateProjectController.java**: Clase base para publicar los end point de la solución. Será cambiada por el nombre ingresado en la generación del arquetipo concatenado por la palabra Controller.
* **EcosystemTemplateProjectService.java**: Clase base que orquesta el consumo de los componentes. Será cambiada por el nombre ingresado en la generación del arquetipo concatenado por la palabra Service.
* **pom.xml**: Archivo de configuración del proyecto, contiene las dependencias, repositorios y configuración de generación del artefacto de la solución.
* **Dockerfile**: Archivo de configuración de contenerización Docker.
* **application.properties**: Archivo de configuración de la aplicación.
* **banner.txt**: Archivo que contiene mensajes de inicialización del servicio, útil para identificar la versión y el nombre del microservicio al iniciar.

\*Clases de solo lectura en la solución.

### Reglas de Implementación

Se proporcionan reglas de implementación teniendo presente las Políticas y Lineamientos definidas anteriormente para la implementación del código fuente en la fábrica de software ADA, asegurando coherencia, mantenibilidad y alineación con los estándares actuales.

#### Categorías y Directrices

* Clases de Entidades:
  + Definición Clara: Cada clase de entidad debe tener una definición clara de su propósito y responsabilidades.
  + Anotaciones JPA: Se deben aplicar anotaciones JPA como @Entity, @Id de manera consistente para facilitar el mapeo objeto-relacional.
  + Documentación: Se debe documentar la entidad como sus columnas con el marco de documentación OpenApi.
  + Secuencias: No se deben utilizar secuencias, existe un servicio que devuelve el identificador según la estructura que representa.
  + Consultas: Si la entidad representa una consulta esta debe definirse @Immutable y el @Subselect debe contener la sentencia sin parámetros.
  + Serialización: Las entidades deben ser serializadas implementando la interface Serializable.
  + Reglas y Validaciones: Toda entidad debe aplicar sus validaciones a nivel de campo.
  + Restricción de exposición de modelos de datos: No se deben exponer entidades en métodos públicos en la clases con anotaciones @Controller y @Service solo debe utilizarse en los componentes que son los que deben incluir la lógica del negocio.
* DTOs (Objetos de Transferencia de Datos):
  + Separación de Responsabilidades: Los DTOs deben estar separados de las clases de entidad para evitar acoplamiento innecesario.
  + Anotaciones de Validación: Se deben usar anotaciones de validación, como @NotNull, @Size, para asegurar la integridad de los datos transferidos.
  + Serialización: Los dtos deben ser serializados.
  + Documentación: Se debe documentar el dto como sus columnas con el marco de documentación OpenApi.
  + Son el único medio para exponer información compleja en las capas @Controller y @Service ya que se deben restringir la exposición de modelos de datos.
* Repositorios:
  + Métodos Consistentes: Los métodos en los repositorios deben seguir una nomenclatura y estructura consistentes para facilitar su comprensión.
  + Evitar Consultas Nativas: Se dará preferencia a consultas JPQL o métodos derivados sobre consultas nativas. Solo en casos estrictamente necesarios cuando una implementación JPA sea compleja se podrá utilizar. Ejemplos:
    - Consultas de paginación complejas.
    - Sentencias insert/select cuando tienen inicialización de valores por defecto de tablas con más de 20 campos.
    - Sentencias de actualización, inserción o eliminación que surgen de condiciones complejas (varias tablas).
  + Capa Core: Todo repositorio que utilice funcionalidades de la capa Core debe extender la clase EcosystemRepository.
* Servicios/Componentes:
  + Separación de Responsabilidades: La lógica de negocio debe estar separada de las operaciones CRUD y otros aspectos técnicos en la capa @Component. La capa @Service debe ser la única que consume los @Component de esta manera aseguramos la restricción de exposición de modelos de datos.
  + Manejo de Excepciones: Se deben lanzar excepciones específicas y personalizadas para facilitar la identificación y gestión de errores.
  + Capa Core: Todo componente que utilice funcionalidades de la capa Core debe extender la clase EcosystemService.
* Controladores:
  + Manejo de Endpoints: Los controladores deben manejar endpoints específicos de manera clara y coherente.
  + Validación de Entradas: Se realizará una validación adecuada de las entradas recibidas, con respuestas de error detalladas.
  + Documentación: Se debe documentar cada endpoint con sus argumentos (sin los incluye) con el marco de documentación OpenApi.
  + Capa Core: Todo controlador que utilice funcionalidades de la capa Core debe extender la clase CustomResponseAdvise.
* Configuraciones:
  + Externalización de Configuraciones: Configuraciones como URL de bases de datos, credenciales, etc., deben externalizarse y no estar codificadas en el propio código. Para eso existen 2 servicios encargados de almacenar esas configuraciones Spring Config Cloud para la centralización de configuraciones de perfiles y EcosystemConfigWS para las configuraciones del negocio de cada cliente.
* Utilidades:
  + Funciones Reutilizables: Las clases de utilidades deben contener funciones reutilizables y genéricas para promover la eficiencia del desarrollo.
* Arquetipo por Defecto:
  + Estructura Proyectual: Se proporcionará un arquetipo por defecto que seguirá una estructura proyectual clara y bien definida.
* Reglas del Proyecto:
  + Revisión y Aprobación: Cada nuevo proyecto deberá ser revisado y aprobado para garantizar que cumple con las reglas de implementación y estándares establecidos.
* Lanzamiento de Excepciones:
  + Excepciones Personalizadas: Las excepciones lanzadas deben ser personalizadas del tipo EcosystemException la cual extiende de RuntimeException y debe proporcionar mensajes descriptivos de errores y soluciones posibles.
* Control de Errores:
  + Centralización de Manejo de Errores: El manejo de errores debe centralizarse en aspectos específicos para evitar duplicación de lógica. Es implementación esta soportada en la clase CustomResponseAdvise la cual debe ser extendida en los controladores del microservicios.
* Comunicación entre Servicios (Feign):
  + Interfaces Claras: Las interfaces Feign deben ser claras y bien definidas, siguiendo el contrato de comunicación entre servicios.
* Estructura de Mensajes:
  + Formato Estandarizado: Los mensajes intercambiados entre servicios deben seguir un formato estandarizado, preferiblemente JSON, para facilitar el análisis y la depuración.
* Generación de Mensajes de Logs:
  + Detalles Relevantes: Los mensajes de logs deben contener detalles relevantes, como identificadores únicos, para facilitar la correlación en entornos distribuidos.
* Respuestas de Errores y Exitosas:
  + Formato Consistente: Las respuestas de errores y exitosas deben seguir un formato consistente para facilitar su procesamiento por parte de los consumidores.
* Configuración de Perfiles Properties:
  + Uso Eficiente: Se deben utilizar perfiles de propiedades de manera eficiente, por ejemplo, para gestionar configuraciones específicas para entornos de desarrollo, prueba y producción.
* Políticas y Buenas Prácticas:
  + Consistencia de Estilo: Se adoptará un estilo de codificación consistente en toda la fábrica, basado en estándares como Google Java Style o similares.
  + Revisión de Código: Todas las implementaciones pasarán por un proceso de revisión de código para garantizar la calidad y cumplimiento de las directrices.
  + Documentación: Cada clase y método debe estar debidamente documentado, explicando su propósito y cualquier detalle importante.
  + Pruebas Unitarias: Se deben escribir pruebas unitarias para validar la funcionalidad de cada componente, asegurando robustez y facilitando futuras modificaciones.
  + Actualizaciones Constantes: Las reglas de implementación se revisarán y actualizarán regularmente para reflejar los cambios en las mejores prácticas y estándares de la industria.
* Proceso de Implementación:
  + Planificación y Diseño: Antes de la implementación, se realizará una planificación detallada y un diseño claro de la solución.
  + Revisión de Diseño: Un equipo revisará el diseño propuesto para asegurar que cumple con las reglas y políticas establecidas.
  + Implementación y Revisión de Código: La implementación seguirá las reglas establecidas, y cada componente pasará por una revisión de código antes de la fusión.
  + Pruebas Unitarias y Funcionales: Cada componente se someterá a pruebas unitarias y funcionales para garantizar su correcto funcionamiento.
* Beneficios:
  + Mantenibilidad Mejorada: La implementación coherente facilita la comprensión y mantenimiento del código a lo largo del tiempo.
  + Reducción de Errores: La aplicación de reglas claras contribuye a la reducción de errores y mejora la calidad del software.
  + Eficiencia en el Desarrollo: Un conjunto claro de reglas agiliza el proceso de desarrollo al proporcionar guías claras.
* Mejora Continua:
  + Retroalimentación del Equipo: Se fomenta la retroalimentación del equipo para identificar áreas de mejora y ajustar las reglas de implementación según sea necesario.
  + Evaluación Periódica: Las reglas y políticas se evaluarán periódicamente para asegurar que sigan siendo relevantes y efectivas en el entorno de desarrollo actual.

## Tecnologías Utilizadas

Una arquitectura de microservicios utiliza una combinación de tecnologías y herramientas para lograr la modularidad, la independencia y la escalabilidad deseada. A continuación, se describe un conjunto común de tecnologías utilizadas en la fábrica de software ADA:

### Repositorio de Código Fuente

Se define GitLab como el repositorio central para la arquitectura de microservicios en la Fábrica de Software ADA, esta elección se basa en una serie de consideraciones estratégicas y características clave que satisfacen las necesidades específicas de desarrollo, colaboración y gestión de código en un entorno de microservicios. A continuación, se proporciona una descripción de las razones detrás de esta elección:

#### 1. Colaboración y Desarrollo Distribuido:

GitLab facilita la colaboración en el desarrollo distribuido al proporcionar herramientas robustas para la gestión de repositorios Git. Sus funcionalidades, como ramas, solicitudes de extracción y comentarios integrados, permiten una colaboración efectiva entre equipos de desarrollo distribuidos.

#### 2. Ciclo de Vida Integrado:

GitLab ofrece un ciclo de vida completo para el desarrollo de software, desde la planificación hasta la implementación y la supervisión. Su integración con herramientas como CI/CD, gestión de problemas y seguimiento de versiones proporciona un flujo de trabajo coherente y eficiente.

#### 3. Integración Continua y Despliegue Continuo (CI/CD):

GitLab CI/CD está integrado directamente en la plataforma, permitiendo la automatización de la construcción, prueba y despliegue de microservicios de manera sencilla. Esta integración contribuye a la entrega continua y la implementación eficiente de cambios en la arquitectura de microservicios.

#### 4. Gestión Avanzada de Ramas y Fusiones:

La gestión de ramas en GitLab es robusta, permitiendo la creación, fusión y gestión eficiente de ramas, lo cual es esencial en un entorno de desarrollo de microservicios, donde la modularidad y la independencia son críticas.

#### 5. Gestión de Problemas Integrada:

GitLab incluye una herramienta de gestión de problemas que permite un seguimiento efectivo de problemas y la integración con el código fuente. Esto es crucial para abordar problemas, mejoras y nuevas características en el contexto de la arquitectura de microservicios.

#### 6. Control de Acceso y Seguridad:

GitLab proporciona funcionalidades avanzadas de control de acceso y seguridad, asegurando que solo usuarios autorizados tengan acceso a los repositorios y a las operaciones críticas. La seguridad en el código fuente es esencial en un entorno de microservicios.

#### 7. Registro y Seguimiento de Cambios:

El registro detallado de cambios en GitLab facilita la auditoría y el seguimiento de las modificaciones realizadas en los microservicios. Esto es esencial para la trazabilidad y la resolución de problemas en una arquitectura distribuida.

#### 8. Gestión de Releases y Versiones:

GitLab ofrece herramientas para gestionar versiones y releases, lo que facilita la coordinación y la implementación de cambios en los microservicios, asegurando la coherencia en el entorno de producción.

#### 9. Soporte para Artefactos y Pipelines:

Descripción: GitLab permite la gestión de artefactos y la configuración de pipelines de CI/CD, lo que simplifica la construcción, prueba y despliegue de microservicios y sus dependencias.

En resumen, la elección de GitLab para la arquitectura de microservicios en la Fábrica de Software ADA se basa en su capacidad integral para soportar un ciclo de vida completo de desarrollo, colaboración efectiva, CI/CD integrado y gestión avanzada de código fuente, todo lo cual es esencial en un entorno de microservicios dinámico y ágil.

### Lenguaje Desarrollo Backend con Java

La elección de Java y Spring Boot para la arquitectura de microservicios en la fábrica de software ADA se basa en varias razones que incluyen la robustez de la plataforma Java, la eficacia de Spring Boot en el desarrollo de microservicios y la amplia adopción de estas tecnologías en la industria. Aquí hay una descripción detallada de la razones analizadas:

#### Robustez de Java:

* **Portabilidad**: Java es conocido por su portabilidad, lo que significa que las aplicaciones escritas en Java pueden ejecutarse en cualquier lugar donde exista una máquina virtual Java (JVM). Esto facilita la gestión de la infraestructura y permite a los desarrolladores centrarse en la lógica de negocio en lugar de preocuparse por los detalles de la plataforma subyacente.
* **Lenguaje Orientado a Objetos**: Java es un lenguaje de programación orientado a objetos, lo que facilita la creación de código modular y mantenible. Este enfoque se alinea bien con la arquitectura de microservicios, donde la modularidad es fundamental.
* **Ecosistema Maduro**: Java tiene un ecosistema maduro y ampliamente adoptado con una gran cantidad de bibliotecas y frameworks que pueden ser utilizados para diversos propósitos, desde la persistencia de datos hasta la seguridad.

#### Eficiencia de Spring Boot:

* **Desarrollo Rápido**: Spring Boot, basado en el framework Spring, facilita el desarrollo rápido de aplicaciones gracias a sus convenciones y configuraciones por defecto. Los desarrolladores pueden centrarse en la implementación de la lógica de negocio sin preocuparse demasiado por la configuración.
* **Contenedor Integrado**: Spring Boot incluye un contenedor integrado (por ejemplo, Tomcat o Jetty), lo que significa que los microservicios se pueden empaquetar como JARs autocontenidos. Esto simplifica la implementación y mejora la portabilidad.
* **Spring Ecosystem**: El ecosistema Spring proporciona módulos y extensiones para tareas comunes en el desarrollo de microservicios, como seguridad, gestión de configuración, descubrimiento de servicios, y más.

#### Adopción en la Industria:

* **Comunidad Activa**: Tanto Java como Spring tienen comunidades activas que ofrecen soporte, documentación y soluciones a problemas comunes. Esto es esencial para la sostenibilidad y resolución rápida de problemas.
* **Estándares de la Industria**: Java y Spring Boot son estándares de la industria para el desarrollo de aplicaciones empresariales y microservicios. La elección de tecnologías ampliamente aceptadas puede facilitar la colaboración y la integración con otras soluciones.

#### Aspectos de Microservicios:

* **Escalabilidad y Despliegue Independiente**: La arquitectura de microservicios se beneficia de la capacidad de escalar y desplegar servicios de forma independiente. Java y Spring Boot proporcionan las herramientas necesarias para lograr esto eficientemente.
* **Facilita la Descomposición de Aplicaciones**: Java y Spring Boot permiten la descomposición de aplicaciones monolíticas en servicios más pequeños y manejables, lo que facilita la evolución hacia una arquitectura de microservicios.

Versión: Amazon Correto 21

Link: <https://github.com/corretto/corretto-21/releases>

Framework:

* Springboot 3.2.0 Para servicios con Paginación de consultas (hasta esa versión se soporta Oracle 11gR2).
* Springboot 3.2.3: Si no requiere paginación de consultas.

### Contenerización con Docker

Docker facilita la contenerización de aplicaciones, lo que permite empaquetar un microservicio junto con sus dependencias y configuraciones en un contenedor ligero y portátil.

Version: Docker version 25.0.3, build 4debf41

### Orquestación con Kubernetes

Kubernetes es una plataforma de orquestación de contenedores que facilita la implementación, escalabilidad y gestión de aplicaciones contenerizadas en clústeres. Se utiliza para administrar y escalar microservicios de manera eficiente.

Version: v1.29

### Gestión de Configuración con Spring Cloud Config

Spring Cloud Config proporciona una solución para la gestión externa de configuraciones, permitiendo a los microservicios recuperar configuraciones específicas de manera dinámica.

Versión: La misma utilizada en el framework principal (3.2x)

### Comunicación entre Microservicios con Spring Cloud y Feign

Spring Cloud facilita la creación de patrones de comunicación entre microservicios. Feign es una biblioteca de cliente declarativa que simplifica la creación de clientes HTTP para la comunicación entre servicios.

Versión: La misma utilizada en el framework principal (3.2x)

### Registro y Descubrimiento de Servicios con Eureka

Eureka, parte de Spring Cloud, proporciona un servicio de registro y descubrimiento que permite a los microservicios encontrar y comunicarse entre sí de manera dinámica.

Versión: La misma utilizada en el framework principal (3.2x)

### Gateway API con Spring Cloud Gateway

Spring Cloud Gateway es una herramienta que actúa como gateway API, proporcionando una entrada única para las solicitudes de clientes y dirigiéndolas a los microservicios correspondientes.

Versión: La misma utilizada en el framework principal (3.2x)

### Manejo de Eventos con Apache Kafka

Apache Kafka es una plataforma de transmisión de eventos que facilita la comunicación asíncrona entre microservicios, permitiendo la transmisión en tiempo real de eventos clave.

Versión: La misma utilizada en el servidor.

### Persistencia de Datos con Spring Data JPA o MongoDB

Descripción: Spring Data JPA se utiliza comúnmente para acceder a bases de datos relacionales (Oracle, MySQL, SQLServer, etc), mientras que MongoDB se utiliza para almacenar datos de forma no relacional. La elección depende de los requisitos específicos de persistencia de datos.

Versión: La misma utilizada en el framework principal (3.2x)

### Pruebas con JUnit y Mockito

JUnit es un marco de pruebas unitarias estándar para Java, y Mockito se utiliza para simplificar la creación de objetos simulados para pruebas.

Versión: La misma utilizada en el framework principal (3.2x)

### Monitoreo y Trazabilidad con Spring Boot Actuator y Zipkin

Descripción: Spring Boot Actuator proporciona puntos finales para monitorear y gestionar la aplicación. Zipkin es una herramienta de trazabilidad que ayuda a visualizar y analizar la latencia en microservicios. (Sección por evaluar Elastict)

Versión: La misma utilizada en el framework principal (3.2x)

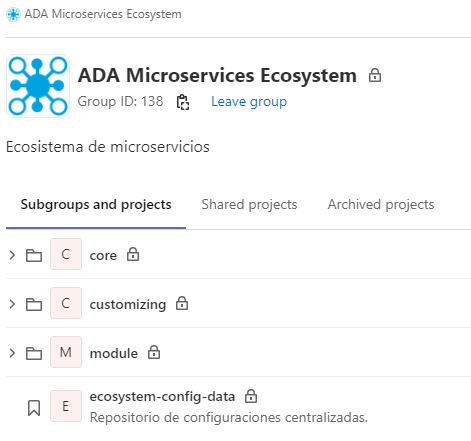
### Gestión de Dependencias con Maven

Maven es una herramienta de gestión de dependencias que facilitan la construcción, prueba y gestión de proyectos Java.

Versión: La misma utilizada en el framework principal (3.2x)

### Repositorio de Código Fuente

Se crea nuevo repositorio para almacenar el código fuente de los microservicios. El cual está en la siguiente ruta: <http://10.1.140.120/ada-microservices-ecosystem>



El repositorio ha sido organizado con siguiente estructura:

* core: Servicios base del ecosistema de microservicios.
* customizing: Servicios relacionados con las personalizaciones de usuarios como logos, gui, etc.
* module: En este grupo se almacenan los servicios del negocio.
* Ecosystem-config-data: Repositorio para las configuraciones centralaizadas estáticas de los microservicios.

### Servidores de Despliegues

#### Ambiente Productivo

|  |  |
| --- | --- |
| Sistema Operativo | Ubuntu Server 20.04 LTS |
| Memoria RAM | 64GB |
| HDD | 256GB |
| Docker | version 24.0.7, build afdd53b |
| Docker Compose | version v2.23.1 |
| kafka | Versión 3.5.1 |
| Java JDK | Amazon Corretto 21.0.1.12 |

#### Ambiente Desarrollo, QA, Presentación

|  |  |
| --- | --- |
| Sistema Operativo | Ubuntu Server 20.04 LTS |
| Memoria RAM | 32GB |
| HDD | 256GB |
| Docker | version 24.0.7, build afdd53b |
| Docker Compose | version v2.23.1 |
| kafka | Versión 3.5.1 |
| Java JDK | Amazon Corretto 21.0.1.12 |

### Equipo Desarrollador

|  |  |
| --- | --- |
| Sistema Operativo | Ubuntu Server 20.04 LTS / Window 11 Profesional |
| Memoria RAM | 16GB |
| HDD | 512GB |
| Docker | version 24.0.7, build afdd53b |
| Docker Compose | version v2.23.1 |
| kafka | Versión 3.5.1 |
| Java JDK | Amazon Corretto 21.0.1.12 |

### Equipo Tester

|  |  |
| --- | --- |
| Sistema Operativo | Ubuntu Server 20.04 LTS / Window 11 Profesional |
| Memoria RAM | 12GB |
| HDD | 512GB |

## Estrategias de Desarrollo

### Fase de Diseño

#### Enfoque Microservicios

En esta fase, la fábrica de software ADA adopta un enfoque específico para el diseño de microservicios

##### Identificación y Definición de Límites del Servicio

Durante la fase de diseño, la fábrica de software ADA se dedica a identificar y establecer claramente los límites de cada microservicio. Esto implica un análisis exhaustivo de las funciones y responsabilidades de cada componente del sistema, dividiéndolo en servicios más pequeños y manejables.

* **Descomposición Funcional**: Se realiza una descomposición cuidadosa de la funcionalidad global en servicios individuales. Cada microservicio es responsable de una tarea específica y tiene límites bien definidos para evitar la interdependencia excesiva con otros servicios.
* **Delimitación de Contexto**: Se define el contexto de cada microservicio, estableciendo claramente qué datos y lógica pertenecen a ese servicio en particular. Esta delimitación ayuda a evitar acoplamientos innecesarios y facilita la independencia de cada microservicio.

##### Elección de Tecnologías Adecuadas

La fábrica de software ADA realiza una evaluación cuidadosa de las tecnologías disponibles y selecciona aquellas que mejor se adapten a las necesidades específicas de cada microservicio.

* **Tecnologías Específicas**:Dependiendo de los requisitos y características particulares de cada microservicio, se eligen tecnologías específicas, incluyendo lenguajes de programación, bases de datos, frameworks y herramientas de desarrollo.
* **Consistencia Tecnológica**: Aunque cada microservicio puede utilizar tecnologías diferentes, se establecen pautas y mejores prácticas para garantizar una cierta consistencia y facilitar la gestión del ecosistema de microservicios.

##### Creación de Interfaces Bien Definidas

La definición clara de interfaces es esencial para la comunicación efectiva entre microservicios. La fábrica de software ADA se centra en establecer interfaces bien definidas y fomentar la comunicación a través de contratos claros.

* **Contratos API**: Cada microservicio expone un conjunto de contratos API que define cómo interactuar con él. Estos contratos son documentados de manera exhaustiva, especificando los puntos finales, los tipos de datos y los métodos admitidos.
* **Documentación Rigurosa**: Se pone un énfasis significativo en la documentación de las interfaces, lo que facilita la comprensión y el consumo por parte de otros equipos de desarrollo y sistemas.
* **Compatibilidad y Versionado**: La fábrica implementa prácticas de gestión de versiones para garantizar la compatibilidad entre microservicios y permitir actualizaciones sin problemas. Esto se traduce en un entorno dinámico y adaptable a medida que evolucionan los requisitos del sistema.

#### Descomposición de Monolitos

Si hay migración o evolución desde una arquitectura monolítica, se destacarán las estrategias utilizadas para descomponer y refactorizar aplicaciones existentes en microservicios.

##### Evaluación Inicial

Antes de iniciar el proceso de descomposición, la fábrica de software ADA realiza una evaluación exhaustiva de la arquitectura monolítica existente.

* **Análisis de Dependencias**: Se identifican las dependencias entre los distintos módulos o componentes del monolito. Esto ayuda a comprender la complejidad de la aplicación y a planificar una descomposición efectiva.
* **Identificación de Dominios**: Se analiza la aplicación para identificar dominios específicos de negocio que puedan convertirse en microservicios independientes. Esto se hace considerando las funcionalidades y la lógica del negocio de cada componente.

##### Definición de Dominios de Microservicios

La fábrica de software ADA define los límites y las responsabilidades de los futuros microservicios. Esta etapa implica una cuidadosa planificación y diseño de la arquitectura de microservicios.

* **Modelado de Dominios**: Se utiliza modelado de dominios para visualizar y entender mejor las áreas de negocio que serán convertidas en microservicios. Esto ayuda a identificar las relaciones y las interacciones entre los distintos servicios.
* **Establecimiento de Contratos**: Para cada dominio identificado, se definen los contratos API y las interfaces que guiarán la interacción entre los microservicios. Esto garantiza una comunicación clara y consistente.

##### Estrategias de Descomposición

La fábrica de software ADA implementa estrategias específicas para descomponer el monolito en microservicios.

* **Desglose por Funcionalidad**: La aplicación monolítica se descompone en microservicios basados en la funcionalidad, dividiendo las características de negocio en servicios independientes y especializados.
* **Refactorización Gradual**: Se sigue un enfoque de refactorización gradual, donde se identifican módulos específicos dentro del monolito que pueden convertirse en microservicios sin interrupciones significativas.
* **Implementación de Pasarelas**: Se pueden implementar pasarelas o puntos de entrada para enrutar el tráfico desde la aplicación monolítica hacia los nuevos microservicios, permitiendo una migración progresiva.

##### Garantía de Continuidad del Servicio

Durante el proceso de descomposición, la fábrica de software ADA implementa medidas para garantizar la continuidad del servicio.

* **Pruebas Rigurosas**: Se realizan pruebas extensivas para asegurar que los nuevos microservicios funcionen correctamente y cumplan con los requisitos de negocio.
* **Monitoreo Continuo**: Se establecen herramientas y prácticas de monitoreo continuo para identificar posibles problemas de rendimiento o errores después de la descomposición.
* **Gestión de Transiciones**: Se planifican cuidadosamente las transiciones desde el monolito hacia la arquitectura de microservicios, minimizando cualquier impacto negativo en la experiencia del usuario y en las operaciones del negocio.

### Prácticas de Pruebas

#### Testing Unitario

La fábrica de software ADA sigue prácticas de pruebas unitarias para cada microservicio individual. Esto garantiza la fiabilidad y calidad del código en niveles de unidad, identificando y corrigiendo problemas a una escala más pequeña.

##### Implementación de Prácticas de Pruebas Unitarias

* **Definición de Casos de Prueba:**
  + La fábrica de software ADA inicia el desarrollo de cada microservicio con la definición clara de casos de prueba unitarios.
  + Se identifican escenarios críticos y límites para garantizar una cobertura exhaustiva.
* **Herramientas de Pruebas:**
  + Se utilizan herramientas de pruebas unitarias, como JUnit o TestNG, para implementar y ejecutar los casos de prueba de manera automatizada.
  + La automatización de pruebas agiliza el proceso y facilita la identificación temprana de posibles problemas.
* **Mocking y Stubs:**
  + Se emplean técnicas de mocking y stubbing para simular el comportamiento de dependencias externas o servicios, permitiendo pruebas aisladas y controladas.
  + Esto asegura que las pruebas se centren específicamente en el microservicio en cuestión.

##### Integración Continua y Pruebas Unitarias

* **Integración en Fases Tempranas:**
  + Las pruebas unitarias se integran en un proceso de integración continua desde las primeras etapas del desarrollo.
  + Cada confirmación de código desencadena automáticamente la ejecución de pruebas unitarias.
* **Retroalimentación Rápida:**
  + La integración continua proporciona retroalimentación inmediata a los desarrolladores sobre la validez de sus cambios.
  + Los resultados de las pruebas unitarias se comunican de manera clara, permitiendo correcciones rápidas en caso de fallos.

##### Cobertura de Código y Análisis Estático

* **Cobertura de Código:**
  + La fábrica de software ADA monitorea activamente la cobertura de código de las pruebas unitarias.
  + Se establece un umbral mínimo de cobertura para garantizar que se prueben todas las partes críticas del código.
* **Análisis Estático:**
  + Herramientas de análisis estático se utilizan para identificar posibles problemas en el código fuente que podrían pasar desapercibidos durante las pruebas unitarias.
  + Se abordan y corrigen las cuestiones identificadas para mejorar la calidad del código.

##### Escalabilidad y Mantenimiento

* **Escalabilidad de Pruebas:**
  + - La fábrica de software ADA establece procesos y estructuras que permiten la escalabilidad de las pruebas unitarias a medida que el número de microservicios crece.
    - Se definen patrones y prácticas para facilitar la incorporación eficiente de nuevas pruebas.
  + **Mantenimiento de Pruebas:**
    - Las pruebas unitarias son parte integral de la definición de "Done" en el ciclo de desarrollo.
    - Se asigna tiempo para el mantenimiento continuo de las pruebas a medida que evoluciona el código y los requisitos.

#### Pruebas de Integración

Se emplean pruebas de integración para validar la comunicación y colaboración adecuada entre microservicios. Estas pruebas aseguran que los servicios puedan trabajar eficientemente juntos como un sistema cohesivo.

#### Pruebas de Estrés y Rendimiento

La fábrica realiza pruebas de estrés y rendimiento para evaluar la capacidad y escalabilidad de los microservicios bajo diversas condiciones de carga. Esto es crucial para garantizar un rendimiento eficiente en entornos de producción.

### Integración Continua

#### Automatización de Construcción

La fábrica utiliza prácticas de integración continua para automatizar la construcción y las pruebas de cada microservicio en cada cambio de código. Esto ayuda a identificar rápidamente problemas de integración y mantener un flujo de desarrollo constante.

* **Proceso Continuo**: La fábrica adopta prácticas de integración continua, donde cada cambio de código desencadena automáticamente la construcción y las pruebas del microservicio afectado.
* **Entorno de Construcción Estandarizado**: Se establece un entorno de construcción estandarizado, asegurando consistencia y reproducibilidad en el proceso de construcción, independientemente de quién realice los cambios.
* **Automatización de Tareas de Construcción**: Se automatizan todas las tareas relacionadas con la construcción del microservicio, desde la compilación del código fuente hasta la generación de artefactos ejecutables.
* **Pruebas Automáticas**: Se incorporan pruebas automáticas como parte integral del proceso de construcción. Esto incluye pruebas unitarias, pruebas de integración y cualquier otra prueba relevante para validar la funcionalidad y la integridad del microservicio.
* **Herramientas**: Se utilizará Jenkins para el proceso de integración continua.

Retroalimentación Rápida

La implementación de integración continua proporciona retroalimentación inmediata a los desarrolladores sobre la calidad y compatibilidad de sus cambios, permitiendo correcciones rápidas y reduciendo los posibles conflictos de integración. A continuación, se describe el enfoque en el cual se utilizará en la fábrica de software:

* **Detección Temprana de Problemas**: Identificación de errores o problemas de integración tan pronto como se realizan los cambios de código.
* **Correcciones Eficientes**: Recibir retroalimentación inmediata, los desarrolladores para abordar rápidamente cualquier problema identificado, lo que lleva a correcciones eficientes y reducción del tiempo dedicado a la resolución de conflictos.
* **Reducción de Conflictos de Integración**: Al integrar continuamente los cambios y resolver problemas de manera temprana, se minimiza la posibilidad de conflictos de integración prolongados que podrían surgir al final del ciclo de desarrollo.
* **Fomento de la Mejora Continua**: La retroalimentación constante crea un ciclo de mejora continua, ya que los desarrolladores reciben información inmediata sobre la calidad de su trabajo y pueden ajustar sus prácticas en consecuencia.
* **Informes y Dashboards**: Se utilizarán informes y dashboards automatizados para presentar de manera clara los resultados de las pruebas y la retroalimentación a los equipos de desarrollo.
* **Herramientas**: GitLab CI, Kibana.
* **Registro de Retroalimentación**: Se mantiene un registro detallado de la retroalimentación proporcionada a los desarrolladores, lo que facilita la revisión y el análisis retrospectivo.
* **Revisión Periódica del Proceso**: El proceso de retroalimentación rápida se revisa periódicamente para identificar oportunidades de mejora y optimización.
* **Feedback a Equipos de Desarrollo**: Los resultados y la eficacia del proceso se comparten con los equipos de desarrollo, fomentando una cultura de mejora continua y colaboración.

### Despliegue Continuo

#### Automatización de Implementación

La fábrica implementa despliegue continuo, lo que significa que los microservicios pasan automáticamente por un proceso de implementación después de las pruebas exitosas. Esto garantiza una entrega rápida y eficiente de nuevas funcionalidades o correcciones.

##### Proceso de Despliegue Continuo

* **Pruebas Exitosas**: Después de que los microservicios han pasado con éxito las pruebas unitarias, de integración y otras pruebas definidas, se consideran listos para la implementación.
* **Automatización de Implementación**: Se emplea un proceso de automatización de implementación que toma los artefactos de la construcción del microservicio y los implementa automáticamente en los entornos correspondientes.
* **Despliegue Automático**: Los microservicios se despliegan automáticamente en los entornos de desarrollo, prueba o producción, según la etapa del ciclo de vida del software.
* **Herramientas**: Por defecto, se emplean herramientas como Jenkins, GitLab CI/CD, o herramientas específicas de implementación continua para automatizar el proceso de despliegue según requerimientos específicos del cliente.

#### Gestión de Versiones

Estrategias claras para la gestión de versiones, lo que permite la implementación de nuevas características sin afectar negativamente a los servicios existentes.

##### Estrategias y Prácticas

* **Versionamiento Semántico**: Se adopta el versionamiento semántico para asignar números de versión de manera significativa. Esto asegura una comprensión clara de los cambios realizados en cada versión.
* **Compatibilidad Hacia Atrás**: Las nuevas versiones están diseñadas para ser compatibles hacia atrás, asegurando que los servicios existentes no se vean afectados negativamente al implementar nuevas funcionalidades.
* **Git Flow**: Se sigue un modelo de flujo de trabajo Git, como Git Flow, para gestionar ramas y versiones de manera estructurada. Esto facilita la colaboración entre equipos y la gestión de lanzamientos.
* **Ramificación por Funcionalidades**: La implementación de nuevas características se realiza a menudo en ramas separadas, lo que permite aislar y probar cambios antes de fusionarlos con la rama principal.
* **Etiquetas para Versiones**: Cada versión se etiqueta claramente en el sistema de control de versiones (por ejemplo, Git) para facilitar la identificación y recuperación de versiones específicas.
* **Documentación de Cambios**: Se documentan claramente los cambios realizados en cada versión, proporcionando información detallada sobre nuevas características, correcciones de errores y posibles impactos.

##### Proceso de Implementación

* **Pruebas Exhaustivas**: Antes de lanzar una nueva versión, se realizan pruebas exhaustivas, incluyendo pruebas unitarias, de integración y pruebas específicas para las nuevas características.
* **Despliegue Gradual**: Se adopta un enfoque de despliegue gradual, implementando la nueva versión en entornos de desarrollo y prueba antes de llegar a producción. Esto permite identificar posibles problemas antes de llegar al usuario final.
* **Monitoreo Continuo**: Se implementa un sistema de monitoreo continuo para evaluar el rendimiento y la estabilidad de la nueva versión una vez que se encuentra en producción.

### Garantizando Eficiencia y Calidad

#### Monitoreo Continuo

La fábrica implementa soluciones de monitoreo continuo para evaluar el rendimiento en tiempo real y abordar cualquier problema potencial antes de que afecte a los usuarios finales.

##### Estrategias y Prácticas

* **Herramientas de Monitoreo**: Se utilizan herramientas avanzadas de monitoreo, como Prometheus y Grafana, para recopilar métricas esenciales sobre el rendimiento de los microservicios en tiempo real.
* **Alertas Proactivas**: Se configuran alertas proactivas que notifican automáticamente al equipo de operaciones sobre cualquier anomalía o tendencia preocupante en el rendimiento, permitiendo una respuesta rápida.
* **Métricas Clave**: Se definen métricas clave relacionadas con la utilización de recursos, tiempos de respuesta, errores y otros indicadores críticos para evaluar la salud general del sistema.
* **Registro Detallado**: Los registros detallados de eventos y errores se centralizan para facilitar la identificación y resolución eficiente de problemas potenciales.
* **Monitoreo de Infraestructura**: Además de los microservicios, se monitorea la infraestructura subyacente (servidores, redes, contenedores) para asegurar que todos los componentes contribuyan al rendimiento general.

##### Proceso de Monitoreo Continuo

* **Recopilación de Datos**: Se realiza una recopilación constante de datos, tanto de métricas internas como de servicios externos y dependencias, para tener una visión completa del entorno operativo.
* **Análisis de Tendencias**: El equipo de operaciones analiza tendencias a largo plazo para anticipar cambios en la carga y tomar medidas preventivas antes de que impacten significativamente en el rendimiento.
* **Optimización Proactiva**: Basándose en las métricas y análisis, se lleva a cabo una optimización proactiva, identificando áreas de mejora en el código, la configuración o la infraestructura.
* **Escalabilidad Automática**: Se implementa escalabilidad automática para adaptarse dinámicamente a cambios en la demanda, garantizando un rendimiento constante incluso en situaciones de carga variable.

#### Prácticas de Seguridad

Estrategias para garantizar la seguridad en todas las etapas del ciclo de vida del software, desde el diseño hasta la implementación, para proteger los microservicios contra posibles amenazas.

##### Estrategias y Prácticas

* **Análisis de Amenazas**: Se realiza un análisis exhaustivo de amenazas durante la fase de diseño y desarrollo para identificar posibles vulnerabilidades y escenarios de riesgo.
* **Diseño Seguro**: Se adopta un enfoque de "seguridad por diseño" donde se integran controles de seguridad desde la fase inicial del diseño de microservicios, asegurando una base sólida.
* **Autenticación y Autorización**: Se implementan mecanismos robustos de autenticación y autorización para garantizar que solo usuarios autorizados tengan acceso a los microservicios y a los recursos protegidos.
* **Gestión de Identidades**: Se establece un sistema de gestión de identidades efectivo para la administración centralizada de usuarios, roles y permisos, reduciendo el riesgo de acceso no autorizado.
* **Cifrado de Datos**: La información sensible se cifra durante la transmisión y almacenamiento, protegiendo la integridad y confidencialidad de los datos intercambiados entre microservicios.
* **Monitoreo de Seguridad**: Se implementan soluciones avanzadas de monitoreo de seguridad que alertan sobre posibles brechas o actividades sospechosas en tiempo real, facilitando respuestas rápidas.
* **Actualizaciones y Parches**: Se establece un proceso regular de aplicar actualizaciones y parches de seguridad para los sistemas operativos, marcos y bibliotecas utilizados por los microservicios.
* **Pruebas de Penetración**: Se realizan pruebas de penetración de manera periódica para evaluar la resistencia de los microservicios contra ataques externos y asegurar la eficacia de las medidas de seguridad.

##### Proceso de Implementación Segura

* **Revisiones de Código**: Todas las modificaciones de código pasan por revisiones de seguridad para detectar posibles problemas de seguridad antes de la implementación.
* **Auditorías de Seguridad**: Auditorías regulares de seguridad se llevan a cabo para evaluar la robustez del sistema y garantizar el cumplimiento continuo de los estándares de seguridad.
* **Capacitación Continua**: El equipo de desarrollo recibe capacitación continua sobre las últimas amenazas de seguridad y prácticas recomendadas, promoviendo una cultura de seguridad dentro de la fábrica de software ADA.

1. Una aplicación general se define para efectos del presente documento como una solución que utiliza repositorios de almacenamiento del código, acceso a bases de datos y consumo de otros microservicios. [↑](#footnote-ref-1)
2. Aplica sólo para procesamiento masivo los cuales se definen como procesos asíncronos. [↑](#footnote-ref-2)