

Reporte Proyecto Final

Estudiantes:

- **Juan Felipe Jojoa Crespo - A00382042**
- **Felipe Rojas Prado - A00393918**

Desarrollo

Metodología Ágil y Estrategia de Branching:

Para el desarrollo del proyecto se implementó la metodología ágil Scrum, debido a su enfoque iterativo e incremental, el cual facilita la entrega continua de valor, la adaptación a cambios y la gestión eficiente de tareas técnicas complejas. Scrum se aplicó en combinación con la herramienta Jira, utilizada para gestionar el backlog, planificar los sprints, documentar historias de usuario y dar seguimiento al progreso del equipo.

El proyecto se organizó en dos sprints de duración fija, cada uno con objetivos claros y entregables definidos. Durante la fase inicial, se construyó el Product Backlog, compuesto por historias de usuario priorizadas según los requerimientos del curso: infraestructura como código, CI/CD avanzado, observabilidad, seguridad, pruebas, documentación y despliegue. Cada historia de usuario se desglosó en subtareas técnicas para facilitar la planificación y el seguimiento.

En cada sprint se realizaron las actividades formales de Scrum:

- **Sprint Planning:** Definición de las historias de usuario a abordar, estimación del esfuerzo y asignación de tareas.
- **Daily Meetings:** Reuniones breves de seguimiento para revisar el progreso, identificar bloqueos y coordinar actividades.
- **Sprint Review:** Presentación del incremento desarrollado, demostración del avance y validación contra los criterios de aceptación establecidos.
- **Sprint Retrospective:** Análisis de lo que funcionó bien, los aspectos por mejorar y acuerdos para optimizar el siguiente sprint.

Jira se empleó como herramienta central de gestión, permitiendo mantener el backlog actualizado, visualizar el flujo de trabajo mediante tableros Kanban/Scrum, medir la capacidad del equipo mediante gráficos de burndown y mantener la trazabilidad entre historias, commits, pull requests y despliegues. Esto garantizó una administración clara, ordenada y transparente del proyecto como se puede observar en las capturas.

Buscar

+ Crear Mejorar versión

Espacio Implementación integral del ecosistema DevOps y arquitectura de microservicios ...

Resumen Cronograma Backlog Tablero Calendario Lista Formularios Metas Desarrollo Código More

Buscar en el backlog Filter

Tablero Sprint 1 20 oct - 20 nov (9 actividades)

	IMPLEMENTACIÓN C...	FINALIZADA	...		
IIDEDYADM-3	HU01 - Gestión Ágil del Proyecto y Estrategia de Branching	IMPLEMENTACIÓN C...	FINALIZADA	3	...
IIDEDYADM-7	HU04 - Implementación de Pipeline Completo CI/CD	IMPLEMENTACIÓN C...	FINALIZADA	8	...
IIDEDYADM-10	HU06 - Sistema de Change Management y Release Notes	IMPLEMENTACIÓN C...	FINALIZADA	5	...
IIDEDYADM-9	HU05 - Implementación de Pruebas Unitarias, Integración y E2E	IMPLEMENTACIÓN C...	FINALIZADA	5	...
IIDEDYADM-6	HU03 - Implementación y Documentación de Patrones de Diseño	IMPLEMENTACIÓN C...	FINALIZADA	8	...
IIDEDYADM-5	HU02 - Implementación de Infraestructura en la Nube usando Terraform	IMPLEMENTACIÓN C...	FINALIZADA	8	...
IIDEDYADM-12	HU09 - Documentación General y Presentación del Proyecto	IMPLEMENTACIÓN C...	TAREAS POR HACER	3	...
IIDEDYADM-11	HU08 - Endurecimiento de Seguridad en Infraestructura y Aplicación	IMPLEMENTACIÓN C...	TAREAS POR HACER	5	...

+ Crear

Ejemplo de una HU con su escenario y criterios de aceptación:

Cambiar tipo de actividad

IIDEDYADM-7

HU04 – Implementación de Pipeline Completo CI/CD

Finalizada ✓

✓ Listo

Descripción

Como DevOps,
quiero construir pipelines automáticas que
incluyan pruebas, análisis, despliegues y
aprobaciones,
para garantizar integraciones consistentes y
despliegues seguros.

Criterios Gherkin

- Dado que el código debe ser analizado
- Cuando se ejecute el pipeline de CI/CD
- Entonces SonarQube debe generar un informe de calidad
- Cuando se ejecuten los tests unitarios
- Dado que los contenedores deben ser construidos
- Cuando se construyan imágenes

Tablero de Jira

Buscar

+ Crear Mejorar versión

Espacio Implementación integral del ecosistema DevOps y arquitectura de microservicios ...

Resumen Cronograma Backlog Tablero Calendario Lista Formularios Metas Desarrollo Código More

Buscar tablero Filter

Completar sprint

POR HACER 2

EN CURSO

LISTO 7 ✓

HU09 – Documentación General y Presentación del Proyecto

IMPLEMENTACIÓN COMPLETA DE AR...

IIDEDYADM-12

HU08 – Endurecimiento de Seguridad en Infraestructura y Aplicación

IMPLEMENTACIÓN COMPLETA DE AR...

IIDEDYADM-11

HU01 – Gestión Ágil del Proyecto y Estrategia de Branching

IMPLEMENTACIÓN COMPLETA DE AR...

IIDEDYADM-3

HU04 – Implementación de Pipeline Completo CI/CD

IMPLEMENTACIÓN COMPLETA DE AR...

IIDEDYADM-2

Implementación de Observabilidad Completa

IMPLEMENTACIÓN COMPLETA DE AR...

IIDEDYADM-10

+ Crear

Para el desarrollo del proyecto se adoptó una estrategia de branching basada en **Git Flow**, ajustada a las necesidades del equipo y del alcance del sistema. Esta estructura garantiza control, estabilidad y trazabilidad en cada etapa del ciclo de vida del software. A continuación, se detalla por qué fue seleccionada:

1. Separación clara entre ambientes (Prod, QA/Preprod y Desarrollo)

La existencia de las ramas **Master**, **Stage** y **Dev** permite mapear directamente los tres entornos del proyecto:

- **Master** representa el estado del sistema en producción.
- **Stage** actúa como entorno de pruebas integrales y validación (QA/Preproducción).
- **Dev** sirve como espacio de integración continua donde convergen los desarrollos realizados por el equipo.

Esta separación reduce riesgos, facilita la revisión de cambios y permite validar correctamente nuevas funcionalidades antes de llegar a producción.

2. Control detallado del desarrollo mediante ramas temáticas

Se emplean ramas específicas para organizar y aislar los trabajos del equipo:

- **feature/** para nuevas funcionalidades.
- **bugfix/** para correcciones en desarrollo.
- **hotfix/** para arreglos urgentes que deben llegar rápidamente a producción.
- **test/** (opcional) para experimentos o pruebas independientes.

Esto permite que cada cambio esté encapsulado, mejorando la trazabilidad, evitando interferencias entre tareas y facilitando el code review.

3. Flujo ordenado y seguro hacia producción

La estrategia seleccionada establece un flujo claro:

feature/bugfix → Dev → Stage → Master

Este orden garantiza:

- Integración continua sin afectar ramas estables.
- Pruebas funcionales y de sistema en Stage antes de liberar versiones.

- Despliegues seguros a Master con menor riesgo de falla.

4. Reducción de complejidad (Git Flow simplificado)

Aunque Git Flow completo incluye ramas adicionales como *release*, se optó por una versión **simplificada**, más acorde al tamaño del equipo y la dinámica del proyecto académico. Esto evita sobrecarga operativa y mantiene un flujo más ágil y práctico.

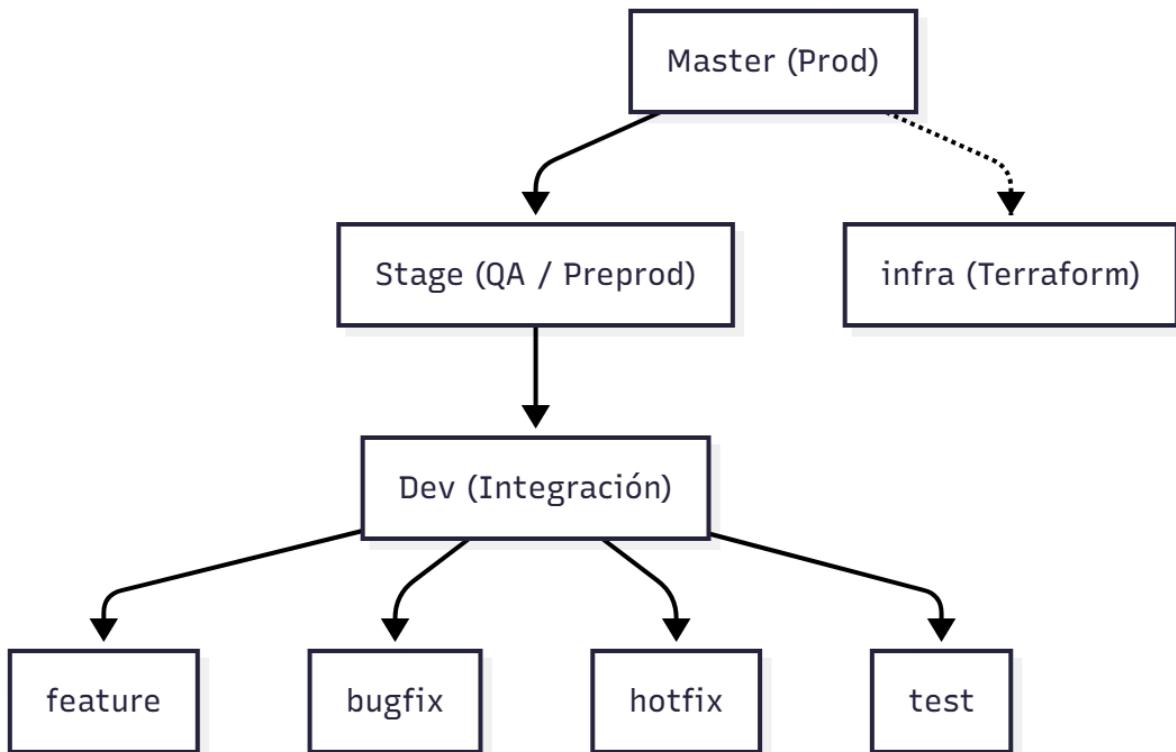
5. Integración con infraestructura como código

Se incluyó una única rama **infra** para gestionar los archivos de Terraform.

La decisión se basa en:

- Separar la infraestructura del código de aplicación.
- Permitir su versionado independiente.
- Mantener un repositorio coherente sin multiplicar ramas innecesariamente.

Al no requerir ambientes diferenciados de infraestructura, una sola rama **infra** es suficiente y más ordenada.



Infraestructura como código en Terraform

1. Objetivo

- **Declarar y versionar** toda la infraestructura base (RG, red, AKS, ACR) en Terraform, garantizando reproducibilidad entre entornos (dev, stage, prod).
- **Asegurar separación de responsabilidades** mediante módulos reutilizables y archivos por ambiente.

2. Alcance

- **Recursos Azure:** Resource Group, Virtual Network, Subnet para AKS, Network Security Group, Azure Kubernetes Service (AKS), Azure Container Registry (ACR).

3. Estructura del Repositorio IaC

```
terraform/
environments/
  dev/
    backend.tf
    main.tf
    variables.tf
    outputs.tf
    .terraform.lock.hcl
  stage/
    backend.tf
    main.tf
    variables.tf
    outputs.tf
  prod/
    backend.tf
    main.tf
    variables.tf
    outputs.tf
modules/
  networking/
    main.tf variables.tf outputs.tf
  aks/
    main.tf variables.tf outputs.tf
  acr/
    main.tf variables.tf outputs.tf
```

- **environments/**: instanciación por entorno.

- **modules/**: lógica reusable (principio DRY).

4. Módulos y Recursos

- **Networking Module**: VNet, Subnet AKS, NSG.
- **AKS Module**: azurerm_kubernetes_cluster con identidad SystemAssigned.
- **ACR Module**: azurerm_container_registry SKU Basic.
- **Environment**: azurerm_resource_group + tags.

5. Parámetros y Variables Clave (dev)

- **Localización**: "Chile Central".
- **Kubernetes**: versión "1.34.0".
- **Coste**: aks_node_count = 1, vm_size = "Standard_B2s".
- **Red**: 10.0.0.0/16 y 10.0.1.0/24.
- **ACR**: SKU Basic.

6. Tagging / Metadatos

- **Tags aplicados**: Environment, Project, ManagedBy=Terraform, CreatedDate.
- **Beneficio**: control de costes y auditoría.

7. Gestión de Estado

- **Backend remoto** en Azure Blob Storage (tfstate).
- **Beneficios**: locking, historial, consistencia colaborativa.

8. Flujo de Aprovisionamiento (Workflow)

1. **Push/Merge** en GitHub → GitHub Actions.
2. **Terraform**: init → validate → plan → apply.

3. **Post-aprovisionamiento:** obtener credenciales AKS, push imágenes a ACR, deploy con kubectl.

9. Dependencias / Orden

- **Secuencia:** Resource Group → Networking → AKS → ACR → Integración → Despliegue.
- **Uso de depends_on** para garantizar orden.

10. Outputs Estratégicos

- **ACR:** login server, credenciales.
- **AKS:** nombre, FQDN, kubeconfig.
- **next_steps:** guía inmediata tras apply.

11. Seguridad Inicial

- **NSG:** HTTP/HTTPS (HTTP recomendado solo en dev).
- **Identidad AKS:** SystemAssigned.
- **ACR admin:** habilitado solo para bootstrap.

12. Buenas Prácticas Aplicadas

- **Separación por ambiente.**
- **Uso de módulos.**
- **Validaciones en variables.**
- **Tags estándar.**
- **Costos optimizados en dev.**

13. Consideraciones de Coste

- **AKS:** 1 nodo B2s (bajo costo).
- **ACR:** SKU Basic.

- **VNet amplia:** facilita escalado futuro.

14. Operaciones Post-Despliegue (Checklist)

- Rotar credenciales ACR.
- Activar autoescalado.
- Plan antes de cada cambio.

15. Estrategia de Evolución / Mejoras

- Role Assignment automático AKS↔ACR.
- Módulo de monitoreo.
- Eliminar HTTP en producción.
- Key Vault para secretos.
- Workload Identity.

16. Trazabilidad

- Versionado Git/GitHub.
- `terraform.lock.hcl` para versiones reproducibles.

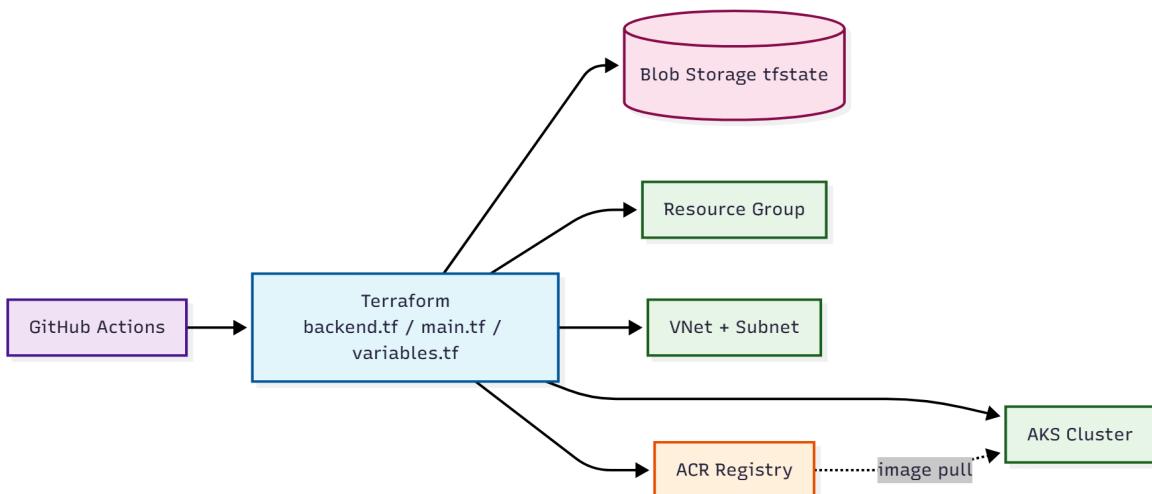
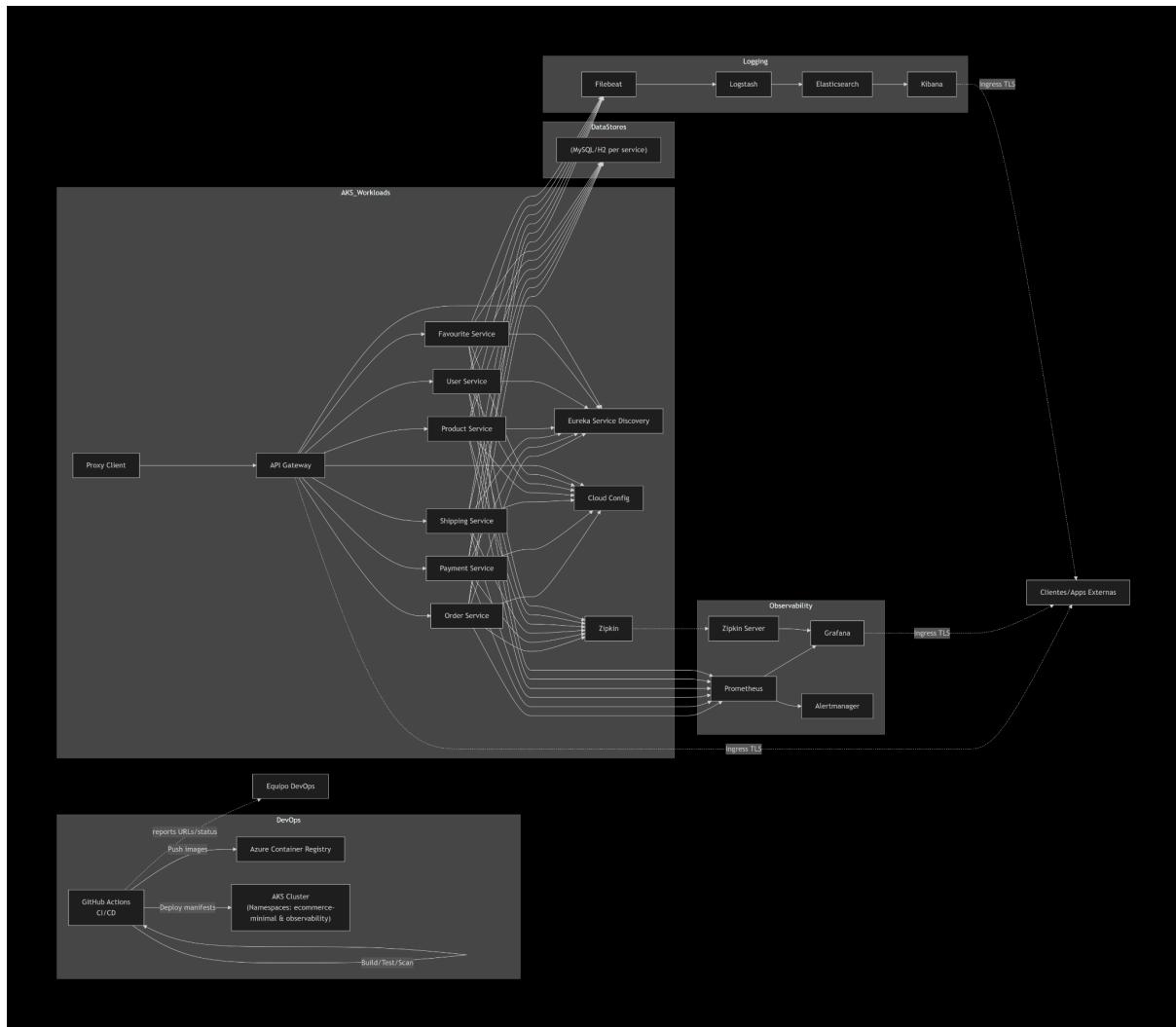


Diagrama de arquitectura



Patrones de diseño

Para los patrones de diseño se identificaron los patrones implementados en la infraestructura del proyecto existente, y los patrones que se lograron identificar fueron los siguientes al detalle:

1. Microservices Architecture

Problema: Monolitos complican escalado y velocidad de cambio.

Solución/Intento: Dividir el dominio en servicios autónomos (órdenes, pagos, productos, usuarios, envíos, favoritos).

Implementación: Carpetas por servicio (order-service/, payment-service/, etc.) cada una con su pom.xml, Dockerfile y despliegue independiente.

Beneficios: Escalado granular, libertad tecnológica, ciclos de release separados.

Riesgos: Complejidad operacional (observabilidad, trazabilidad distribuida).

Mitigación: Uso de Eureka, Config Server y API Gateway.

2. API Gateway

Problema: Clientes deben conocer múltiples endpoints y protocolos.

Solución: Un punto de entrada único para enrutar, filtrar, agregar y aplicar cross-cutting concerns.

Implementación: Módulo api-gateway con dependencia spring-cloud-starter-gateway.

Beneficios: Simplificación para clientes, centralización de autenticación, logging y límites de tasa.

Riesgos: Punto único de fallo, latencia adicional.

Mitigación: Escalado horizontal y health checks.

3. Service Discovery (Eureka)

Problema: Direcciones IP/puertos cambian dinámicamente en entornos orquestados.

Solución: Registro y descubrimiento dinámico de servicios.

Implementación: service-discovery como Eureka Server y clientes con spring-cloud-starter-netflix-eureka-client.

Beneficios: Desacoplamiento de endpoints, auto-escalado.

Riesgos: Dependencia de disponibilidad del server.

Mitigación: Replicar Eureka, habilitar timeouts.

4. Centralized / Externalized Configuration

Problema: Config duplicada y difícil de modificar sin redeploy.

Solución: Config Server suministra propiedades versionadas.

Implementación: Módulo cloud-config y spring-cloud-starter-config en servicios.

Beneficios: Cambios controlados, rollback, consistencia.

Riesgos: Latencia inicial de bootstrap, falla del servidor de configuración.

Mitigación: Caché local, perfiles fallback.

5. Database per Service

Problema: Acoplamiento y congestión de un único esquema.

Solución: Cada servicio maneja su propio modelo y almacenamiento.

Implementación: Dependencias JPA, MySQL/H2 por servicio (order-service/pom.xml).

Beneficios: Autonomía de evolución, aislamiento de fallos.

Riesgos: Complejidad de integridad referencial entre servicios.

Mitigación: Uso de IDs y consistencia eventual (ver recomendación Saga).

6. Layered Architecture (Controller – Service – Repository)

Problema: Mezcla de lógica de negocio, transporte y persistencia reduce mantenibilidad.

Solución: Separar responsabilidades en capas definidas.

Implementación: OrderResource (API), OrderService (negocio), OrderRepository (persistencia).

Beneficios: Testeabilidad, claridad, reutilización.

Riesgos: Capas excesivas pueden añadir complejidad.

Mitigación: Mantener lógica de negocio en servicios, mapping en helpers.

7. Repository Pattern

Problema: Dependencia directa de la lógica sobre detalles de acceso a datos.

Solución: Abstracción mediante interfaces que delegan a JPA.

Implementación: Interfaces Spring Data *Repository (ej. OrderRepository).

Beneficios: Menos código boilerplate, fácil test con mocks.

Riesgos: Consultas complejas pueden quedar ocultas.

Mitigación: Queries explícitas con @Query cuando sea necesario.

8. Service Layer

Problema: Lógica dispersa en controladores.

Solución: Encapsular operaciones de negocio en servicios.

Implementación: OrderServiceImpl maneja validaciones y mapping.

Beneficios: Reutilización, cohesión.

Riesgos: Sobre-crecimiento de métodos.

Mitigación: Refactor a dominios específicos cuando se expanda.

9. Data Transfer Object (DTO)

Problema: Exponer entidades JPA directamente crea acoplamiento y riesgos de seguridad.

Solución: Objetos planos para intercambio en API.

Implementación: OrderDto, CartDto, ProductDto etc.

Beneficios: Control de campos expuestos, versión de contrato.

Riesgos: Duplicidad y esfuerzo de mapeo.

Mitigación: Helpers y generación automática en futuros (MapStruct).

10. Data Mapper

Problema: Lógica de conversión Domain ↔ DTO dispersa.

Solución: Centralizar mapping en clase helper.

Implementación: OrderMappingHelper.map() (métodos estáticos).

Beneficios: Consistencia y reutilización.

Riesgos: Crecimiento de métodos si se agregan muchos casos.

Mitigación: Segregar por agregado o adoptar MapStruct.

11. Builder Pattern

Problema: Constructores con muchos parámetros dificultan legibilidad.

Solución: API fluida para construcción de objetos inmutables/convenientes.

Implementación: Lombok @Builder en Order, OrderDto, Cart, etc.

Beneficios: Claridad, evita telescoping constructors.

Riesgos: Coste de generar objetos intermedios.

Mitigación: Usar builder solo en objetos con múltiples campos opcionales.

12. Dependency Injection / IoC

Problema: Creación manual de dependencias aumenta acoplamiento.

Solución: Inversión de control via Spring contenedor.

Implementación: @Service, @RequiredArgsConstructor para inyección por constructor.

Beneficios: Testeabilidad (mocks), desacoplamiento.

Riesgos: Configuración oculta si hay demasiados beans implícitos.

Mitigación: Mantener claridad en configuración y perfiles.

13. REST Controller Pattern

Problema: Manejo ad-hoc de endpoints.

Solución: Definir controladores anotados que exponen recursos HTTP.

Implementación: OrderResource y otras clases con @RestController.

Beneficios: Organización clara de endpoints.

Riesgos: Inflación de controladores con demasiada lógica.

Mitigación: Delegar al Service Layer.

14. Validation Pattern

Problema: Datos inválidos ingresan a lógica de negocio.

Solución: Anotaciones Bean Validation (JSR-380) en parámetros y DTOs.

Implementación: @NotBlank, @NotNull, @Valid en OrderResource.

Beneficios: Consistencia y reducción de errores.

Riesgos: Mensajes genéricos de error si no se personalizan.

Mitigación: Custom handlers y mensajes internacionalizados.

15. Exception Wrapping

Problema: Errores técnicos se muestran sin contexto de negocio.

Solución: Excepciones específicas (ej. OrderNotFoundException).

Implementación: Lanzada en OrderServiceImpl.findById.

Beneficios: Claridad semántica, mejor logging.

Riesgos: Exceso de clases de excepción.

Mitigación: Agrupar por dominios y usar jerarquías.

16. Containerization & Orchestrated Deployment

Problema: Deploy inconsistente entre entornos.

Solución: Docker + Kubernetes/Helm para empaquetar y orquestar.

Implementación: Dockerfile, compose.yml, manifiestos en k8s/, charts en helm/.

Beneficios: Portabilidad, escalado automático.

Riesgos: Complejidad de configuración y observabilidad.

Mitigación: Automatización CI/CD y monitoreo centralizado.

17. Centralized Logging Pattern

Problema: Dificultad para rastrear flujo entre servicios.

Solución: Uso consistente de logging (@Slf4j).

Implementación: Mensajes de log en OrderServiceImpl, OrderResource.

Beneficios: Depuración y auditoría.

Riesgos: Volumen alto de logs.

Mitigación: Ajustar niveles y agregar correlación (futuro: sleuth/zipkin).

Patrones Implementados

• Resiliencia: Circuit Breaker + Retry + Bulkhead

Implementación: shipping-service en OrderItemServiceImpl mediante anotaciones @CircuitBreaker, @Retry, @Bulkhead sobre las llamadas externas a Order y Product (métodos fetchOrder, fetchProduct).

Configuración: shipping-service/src/main/resources/application.yml sección

resilience4j.circuitbreaker|retry|bulkhead.instances.shippingService

Beneficios: aislamiento ante fallos, reintentos controlados, limitación de concurrencia para evitar saturación.

Anexo del fragmento de código:

```
shipping-service > src > main > java > com > selimhorri > app > service > impl > OrderItemServiceImpl.java > ...
You, 3 seconds ago | 3 authors (SelimHorri and others)
1 package com.selimhorri.app.service.impl;
2
3 import java.util.List;
4 import java.util.stream.Collectors;      SelimHorri, 4 years ago • impl business layer for s
5
6 import javax.transaction.Transactional;
7
8 import org.springframework.beans.factory.annotation.Value;
9 import org.springframework.stereotype.Service;
10 import org.springframework.web.client.RestTemplate;
11
12 import com.selimhorri.app.constant.AppConstant;
13 import com.selimhorri.app.domain.id.OrderItemId;
14 import com.selimhorri.app.dto.OrderDto;
15 import com.selimhorri.app.dto.OrderItemDto;
16 import com.selimhorri.app.dto.ProductDto;
17 import com.selimhorri.app.exception.wrapper.OrderItemNotFoundException;
18 import com.selimhorri.app.helper.OrderItemMappingHelper;
19 import com.selimhorri.app.repository.OrderItemRepository;
20 import com.selimhorri.app.service.OrderItemService;
21
22 import io.github.resilience4j.bulkhead.annotation.Bulkhead;
23 import io.github.resilience4j.circuitbreaker.annotation.CircuitBreaker;
24 import io.github.resilience4j.retry.annotation.Retry;
25 import lombok.RequiredArgsConstructor;
26 import lombok.extern.slf4j.Slf4j;
27
28 You, 3 seconds ago | 3 authors (SelimHorri and others) | Qodo: Test this class
29 @Service
30 @Transactional
31 @Slf4j
32 @RequiredArgsConstructor
33 public class OrderItemServiceImpl implements OrderItemService {
34
35     private final OrderItemRepository orderItemRepository;
36     private final RestTemplate restTemplate;
```

```
37 @Value("${features.enrich-order-item-details:true}")
38 private boolean enrichOrderItemDetails;
39
40 Qodo: Test this method
41 @Override
42 @CircuitBreaker(name = "shippingService")
43 @Retry(name = "shippingService")
44 @Bulkhead(name = "shippingService")
45 public List<OrderItemDto> findAll() {
46     log.info("**** OrderItemDto List, service; fetch all orderItems **");
47     return this.orderItemRepository.findAll()
48         .stream()
49         .map(OrderItemMappingHelper::map)
50         .map(o -> {
51             if (enrichOrderItemDetails) {
52                 o.setProductDto(fetchProduct(o.getProductDto().getProductId()));
53                 o.setOrderDto(fetchOrder(o.getOrderDto().getOrderId()));
54             }
55             return o;
56         })
57         .distinct()
58         .collect(Collectors.toUnmodifiableList());
59
60 Qodo: Test this method
61 @Override
62 @CircuitBreaker(name = "shippingService")
63 @Retry(name = "shippingService")
64 @Bulkhead(name = "shippingService")
65 public OrderItemDto findById(final OrderItemId orderItemId) {
66     log.info("**** OrderItemDto, service; fetch orderItem by id **");
67     return this.orderItemRepository.findById(orderItemId)
68         .map(OrderItemMappingHelper::map)
69         .map(o -> {
70             if (enrichOrderItemDetails) {
```

```
shipping-service > src > main > java > com > selimhorri > app > service > impl > OrderItemServiceImpl.java > ...
32  public class OrderItemServiceImpl implements OrderItemService {
34      public OrderItemDto findById(final OrderItemId orderItemId) {
35          OrderItem o = this.orderItemRepository.findById(orderItemId);
36          if (o != null) {
37              o.setOrderDto(fetchOrder(o.getOrderDto().getOrderId()));
38          }
39          return o;
40      }
41      .orElseThrow(() -> new OrderItemNotFoundException(String.format(format: "Order
42      })
43  }
44
45  Qodo: Test this method
46  @Override
47  public OrderItemDto save(final OrderItemDto orderItemDto) {
48      log.info("**** OrderItemDto, service; save orderItem **");
49      return OrderItemMappingHelper.map(this.orderItemRepository
50          .save(OrderItemMappingHelper.map(orderItemDto)));
51  }
52
53  Qodo: Test this method
54  @Override
55  public OrderItemDto update(final OrderItemDto orderItemDto) {
56      log.info("**** OrderItemDto, service; update orderItem **");
57      return OrderItemMappingHelper.map(this.orderItemRepository
58          .save(OrderItemMappingHelper.map(orderItemDto)));
59  }
60
61  Qodo: Test this method
62  @Override
63  public void deleteById(final OrderItemId orderItemId) {
64      log.info("**** Void, service; delete orderItem by id **");
65      this.orderItemRepository.deleteById(orderItemId);
66  }
67
68  Qodo: Test this method
69  @CircuitBreaker(name = "shippingService", fallbackMethod = "fallbackProduct")
70  @Retry(name = "shippingService")
71  @Bulkhead(name = "shippingService")
72  private ProductDto fetchProduct(Integer productId) {
73      return this.postTemplate.getForObject(
74
```

```
shipping-service > src > main > java > com > selimhorri > app > service > impl > OrderItemServiceImpl.java > ...
32  public class OrderItemServiceImpl implements OrderItemService {
101     private ProductDto fetchProduct(Integer productId) {
102         return this restTemplate.getForObject(
103             AppConstant.DiscoveredDomainsApi.PRODUCT_SERVICE_API_URL + "/" + productId,
104             responseType: ProductDto.class
105         );
106     }
107 }
108 Qodo: Test this method
109 private ProductDto fallbackProduct(Integer productId, Throwable t) {
110     log.warn("Product service fallback for id={}, reason={}", productId, t.toString());
111     return ProductDto.builder().productId(productId).build();
112 }
113 Qodo: Test this method
114 @CircuitBreaker(name = "shippingService", fallbackMethod = "fallbackOrder")
115 @Retry(name = "shippingService")
116 @Bulkhead(name = "shippingService")
117 private OrderDto fetchOrder(Integer orderId) {
118     return this restTemplate.getForObject(
119         AppConstant.DiscoveredDomainsApi.ORDER_SERVICE_API_URL + "/" + orderId,
120         responseType: OrderDto.class
121     );
122 }
123 Qodo: Test this method
124 private OrderDto fallbackOrder(Integer orderId, Throwable t) {
125     log.warn("Order service fallback for id={}, reason={}", orderId, t.toString());
126     return OrderDto.builder().orderId(orderId).build();
127 }
128 }
129
130
131
132
133 }
```

```
shipping-service > src > main > resources > application.yml
You, 1 hour ago | 2 authors (SelimHorri and one other)
1     SelimHorri, 4 years ago • add properties for domain micros ...
2 server:
3   servlet:
4     context-path: /shipping-service
5
6 spring:
7   zipkin:
8     base-url: ${SPRING_ZIPKIN_BASE_URL:http://localhost:9411}
9 config:
10    import: ${SPRING_CONFIG_IMPORT:optional:configserver:http://localhost:9296}
11 application:
12   name: SHIPPING-SERVICE
13 profiles:
14   active:
15     - dev
16
17 # Feature toggles (can be overridden via Config Server or env vars)
18 features:
19   enrich-order-item-details: ${FEATURE_ENRICH_ORDER_ITEM_DETAILS:true}
20
21 # HTTP client timeouts (ms)
22 http:
23   client:
24     connect-timeout-ms: ${HTTP_CLIENT_CONNECT_TIMEOUT_MS:2000}
25     read-timeout-ms: ${HTTP_CLIENT_READ_TIMEOUT_MS:3000}
26
27 resilience4j:
28   circuitbreaker:
29     instances:
30       shippingService:
31         register-health-indicator: true
32         event-consumer-buffer-size: 10
33         automatic-transition-from-open-to-half-open-enabled: true
34         failure-rate-threshold: 50
35         minimum-number-of-calls: 5
36         permitted-number-of-calls-in-half-open-state: 3
```

```

36     |   permitted-number-of-calls-in-half-open-state: 3
37     |   sliding-window-size: 10
38     |   wait-duration-in-open-state: 5s
39     |   sliding-window-type: COUNT_BASED
40   retry:
41     instances:
42       shippingService:
43         max-attempts: 3
44         wait-duration: 200ms
45         retry-exceptions:
46           - java.io.IOException
47           - org.springframework.web.client.ResourceAccessException
48           - org.springframework.web.client.HttpServerErrorException
49   bulkhead:
50     instances:
51       shippingService:
52         max-concurrent-calls: 20
53         max-wait-duration: 500ms
54
55 management:
56   health:
57     circuitbreakers:
58       enabled: true
59   endpoint:
60     health:
61       show-details: always

```

- **Configuración: Feature Toggle (enriquecimiento de OrderItem)**

Implementación: propiedad features.enrich-order-item-details (por defecto true). Uso en OrderItemServiceImpl para activar/desactivar llamadas a servicios externos.

Configuración: application.yml y variables de entorno (FEATURE_ENRICH_ORDER_ITEM_DETAILS).

Beneficios: activar/desactivar comportamiento sin redeploy; mitigación rápida ante incidentes.

Anexos del fragmento del código:

shipping-service > src > main > java > com > selimhorri > app > service > impl > OrderItemServiceImpl.java > OrderItemServiceImpl

You, 1 hour ago | 3 authors (SelimHorri and others)

```
1 package com.selimhorri.app.service.impl;
2
3 import java.util.List;
4 import java.util.stream.Collectors;
5
6 import javax.transaction.Transactional;
7
8 import org.springframework.beans.factory.annotation.Value;
9 import org.springframework.stereotype.Service;
10 import org.springframework.web.client.RestTemplate;
11
12 import com.selimhorri.app.constant.AppConstant;
13 import com.selimhorri.app.domain.id.OrderItemId;
14 import com.selimhorri.app.dto.OrderDto;
15 import com.selimhorri.app.dto.OrderItemDto;
16 import com.selimhorri.app.dto.ProductDto;
17 import com.selimhorri.app.exception.wrapper.OrderItemNotFoundException;
18 import com.selimhorri.app.helper.OrderItemMappingHelper;
19 import com.selimhorri.app.repository.OrderItemRepository;
20 import com.selimhorri.app.service.OrderItemService;
21
22 import io.github.resilience4j.bulkhead.annotation.Bulkhead;
23 import io.github.resilience4j.circuitbreaker.annotation.CircuitBreaker;
24 import io.github.resilience4j.retry.annotation.Retry;
25 import lombok.RequiredArgsConstructor;
26 import lombok.extern.slf4j.Slf4j;
```

You, 1 hour ago | 3 authors (SelimHorri and others) | Qodo: Test this class

```
28 @Service
29 @Transactional
30 @Slf4j
31 @RequiredArgsConstructor
32 public class OrderItemServiceImpl implements OrderItemService {
33
34     private final OrderItemRepository orderItemRepository;
35     private final RestTemplate restTemplate;
```

```
34     private final OrderItemRepository orderItemRepository;
35     private final RestTemplate restTemplate;
36
37     @Value("${features.enrich-order-item-details:true}")
38     private boolean enrichOrderItemDetails;
39
40     Qodo: Test this method
41     @Override
42     @CircuitBreaker(name = "shippingService")
43     @Retry(name = "shippingService")
44     @Bulkhead(name = "shippingService")
45     public List<OrderItemDto> findAll() {
46         log.info("**** OrderItemDto List, service; fetch all orderItems **");
47         return this.orderItemRepository.findAll()
48             .stream()
49             .map(OrderItemMappingHelper::map)
50             .map(o -> {
51                 if (enrichOrderItemDetails) {
52                     o.setProductDto(fetchProduct(o.getProductDto().getProductId()));
53                     o.setOrderDto(fetchOrder(o.getOrderDto().getOrderId()));
54                 }
55                 return o;
56             })
57             .distinct()
58             .collect(Collectors.toUnmodifiableList());
59
60     Qodo: Test this method
61     @Override
62     @CircuitBreaker(name = "shippingService")
63     @Retry(name = "shippingService")
64     @Bulkhead(name = "shippingService")
65     public OrderItemDto findById(final OrderItemId orderItemId) {
66         log.info("**** OrderItemDto, service; fetch orderItem by id **");
67         return this.orderItemRepository.findById(orderItemId)
68             .map(OrderItemMappingHelper::map)
69             .map(o -> {
```

```
shipping-service > src > main > java > com > selimhorri > app > service > impl > OrderItemServiceImpl.java > OrderItemService
32     public class OrderItemServiceImpl implements OrderItemService {
33         public OrderItemDto findById(final OrderItemId orderItemId) {
34             return orderItemId.map(o -> {
35                 if (enrichOrderItemDetails) {
36                     o.setProductDto(fetchProduct(o.getProductDto().getProductId()));
37                     o.setOrderDto(fetchOrder(o.getOrderDto().getOrderId()));
38                 }
39                 return o;
40             })
41             .orElseThrow(() -> new OrderItemNotFoundException(String.format(format: "Order
42             })
43         }
44
45         Qodo: Test this method
46     @Override
47     public OrderItemDto save(final OrderItemDto orderItemDto) {
48         log.info("**** OrderItemDto, service; save orderItem **");
49         return OrderItemMappingHelper.map(this.orderItemRepository
50             .save(OrderItemMappingHelper.map(orderItemDto)));
51     }
52
53         Qodo: Test this method
54     @Override
55     public OrderItemDto update(final OrderItemDto orderItemDto) {
56         log.info("**** OrderItemDto, service; update orderItem **");
57         return OrderItemMappingHelper.map(this.orderItemRepository
58             .save(OrderItemMappingHelper.map(orderItemDto)));
59     }
60
61         Qodo: Test this method
62     @Override
63     public void deleteById(final OrderItemId orderItemId) {
64         log.info("**** Void, service; delete orderItem by id **");
65         this.orderItemRepository.deleteById(orderItemId);
66     }
67
68         Qodo: Test this method
69     @CircuitBreaker(name = "shippingService", fallbackMethod = "fallbackProduct")
```

```
shipping-service > src > main > java > com > selimhorri > app > service > impl > OrderItemServiceImpl.java > OrderItemService
32  public class OrderItemServiceImpl implements OrderItemService {
98      @CircuitBreaker(name = "shippingService", fallbackMethod = "fallbackProduct")
99      @Retry(name = "shippingService")
100     @Bulkhead(name = "shippingService")
101     private ProductDto fetchProduct(Integer productId) {
102         return this restTemplate.getForObject(
103             AppConstant.DiscoveredDomainsApi.PRODUCT_SERVICE_API_URL + "/" + productId,
104             responseType: ProductDto.class
105         );
106     }
107
108     Qodo: Test this method
109     private ProductDto fallbackProduct(Integer productId, Throwable t) {
110         log.warn("Product service fallback for id={}, reason={}", productId, t.toString());
111         return ProductDto.builder().productId(productId).build();
112     }
113
114     Qodo: Test this method
115     @CircuitBreaker(name = "shippingService", fallbackMethod = "fallbackOrder")
116     @Retry(name = "shippingService")
117     @Bulkhead(name = "shippingService")
118     private OrderDto fetchOrder(Integer orderId) {
119         return this restTemplate.getForObject(
120             AppConstant.DiscoveredDomainsApi.ORDER_SERVICE_API_URL + "/" + orderId,
121             responseType: OrderDto.class
122         );
123     }
124
125     Qodo: Test this method
126     private OrderDto fallbackOrder(Integer orderId, Throwable t) { You, 1 hour ago • fea
127         log.warn("Order service fallback for id={}, reason={}", orderId, t.toString());
128         return OrderDto.builder().orderId(orderId).build();
129     }
130 }
```

```
shipping-service > src > main > resources > application.yml
You, 1 hour ago | 2 authors (SelimHorri and one other)
1 |     SelimHorri, 4 years ago • add properties for domain micros
2 server:
3   servlet:
4     context-path: /shipping-service
5
6 spring:
7   zipkin:
8     base-url: ${SPRING_ZIPKIN_BASE_URL:http://localhost:9411}
9   config:
10    import: ${SPRING_CONFIG_IMPORT:optional:configserver:http://localhost:9296}
11   application:
12     name: SHIPPING-SERVICE
13   profiles:
14     active:
15       - dev
16
17 # Feature toggles (can be overridden via Config Server or env vars)
18 features:
19   enrich-order-item-details: ${FEATURE_ENRICH_ORDER_ITEM_DETAILS:true}
20
21 # HTTP client timeouts (ms)
22 http:
23   client:
24     connect-timeout-ms: ${HTTP_CLIENT_CONNECT_TIMEOUT_MS:2000}
25     read-timeout-ms: ${HTTP_CLIENT_READ_TIMEOUT_MS:3000}
26
27 resilience4j:
28   circuitbreaker:
29     instances:
30       shippingService:
31         register-health-indicator: true
32         event-consumer-buffer-size: 10
33         automatic-transition-from-open-to-half-open-enabled: true
34         failure-rate-threshold: 50
35         minimum-number-of-calls: 5
36         permitted-number-of-calls-in-half-open-state: 3
```

```

36      permitted-number-of-calls-in-half-open-state: 3
37      sliding-window-size: 10
38      wait-duration-in-open-state: 5s
39      sliding-window-type: COUNT_BASED
40  retry:
41    instances:
42      shippingService:
43        max-attempts: 3
44        wait-duration: 200ms
45        retry-exceptions:
46          - java.io.IOException
47          - org.springframework.web.client.ResourceAccessException
48          - org.springframework.web.client.HttpServerErrorException
49  bulkhead:
50    instances:
51      shippingService:
52        max-concurrent-calls: 20
53        max-wait-duration: 500ms
54
55  management:
56    health:
57      circuitbreakers:
58        enabled: true
59      endpoint:
60        health:
61        | show-details: always
62

```

- **Configuración: Timeouts de HTTP externos**

Implementación: shipping-service ClientConfig configura RestTemplate con connect-timeout y read-timeout externos.

Configuración: http.client.connect-timeout-ms,

http.client.read-timeout-ms (overridable por env

HTTP_CLIENT_CONNECT_TIMEOUT_MS / HTTP_CLIENT_READ_TIMEOUT_MS).

Beneficios: evita hilos bloqueados y reduce impacto de latencias anómalas.

Anexos del fragmento del código:

ClientConfig.java X ▶ v ESP-IDF: Search Error Hint

shipping-service > src > main > java > com > selimhorri > app > config > client > ClientConfig.java > {} com.selimhorri.app

You, 1 hour ago | 2 authors (SelimHorri and one other)

```
1 package com.selimhorri.app.config.client;           SelimHorri, 4 years ago • add data layer ...
2 
3 import org.springframework.beans.factory.annotation.Value;
4 import org.springframework.cloud.client.loadbalancer.LoadBalanced;
5 import org.springframework.context.annotation.Bean;
6 import org.springframework.context.annotation.Configuration;
7 import org.springframework.web.client.RestTemplate;
8 import org.springframework.http.client.SimpleClientHttpRequestFactory;
9 
10 You, 1 hour ago | 2 authors (You and one other) | Qodo: Test this class
11 @Configuration
12 public class ClientConfig {
13 
14     @Value("${http.client.connect-timeout-ms:2000}")
15     private int connectTimeoutMs;
16 
17     @Value("${http.client.read-timeout-ms:3000}")
18     private int readTimeoutMs;
19 
20     Qodo: Test this method
21     @LoadBalanced
22     @Bean
23     public RestTemplate restTemplateBean() {
24         SimpleClientHttpRequestFactory factory = new SimpleClientHttpRequestFactory();
25         factory.setConnectTimeout(connectTimeoutMs);
26         factory.setReadTimeout(readTimeoutMs);
27         RestTemplate restTemplate = new RestTemplate();
28         restTemplate.setRequestFactory(factory);
29         return restTemplate;
30     }
31 }
```

application.yml

shipping-service > src > main > resources > application.yml

You, 1 hour ago | 2 authors (SelimHorri and one other)

```
1 server:
2   servlet:
3     context-path: /shipping-service
4
5 spring:
6   zipkin:
7     base-url: ${SPRING_ZIPKIN_BASE_URL:http://localhost:9411/}
8   config:
9     import: ${SPRING_CONFIG_IMPORT:optional:configserver:http://localhost:9296}
10  application:
11    name: SHIPPING-SERVICE
12  profiles:
13    active:
14      - dev
15
16 SelimHorri, 4 years ago • add properties for domain micros
17 # Feature toggles (can be overridden via Config Server or env vars)
18 features:
19   enrich-order-item-details: ${FEATURE_ENRICH_ORDER_ITEM_DETAILS:true}
20
21 # HTTP client timeouts (ms)
22 http:
23   client:
24     connect-timeout-ms: ${HTTP_CLIENT_CONNECT_TIMEOUT_MS:2000}
25     read-timeout-ms: ${HTTP_CLIENT_READ_TIMEOUT_MS:3000}
26
27 resilience4j:
28   circuitbreaker:
29     instances:
30       shippingService:
31         register-health-indicator: true
32         event-consumer-buffer-size: 10
33         automatic-transition-from-open-to-half-open-enabled: true
34         failure-rate-threshold: 50
35         minimum-number-of-calls: 5
36         permitted-number-of-calls-in-half-open-state: 3
```

```

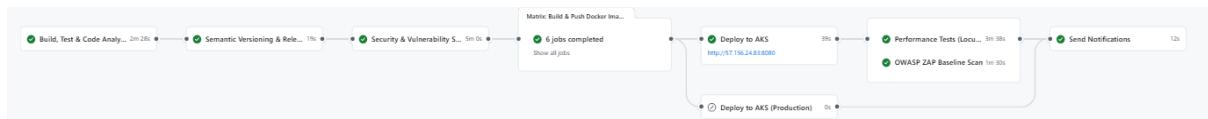
5   permitted-number-of-calls-in-half-open-state: 3
6   sliding-window-size: 10
7   wait-duration-in-open-state: 5s
8   sliding-window-type: COUNT_BASED
9
10  retry:
11    instances:
12      shippingService:
13        max-attempts: 3
14        wait-duration: 200ms
15        retry-exceptions:
16          - java.io.IOException
17          - org.springframework.web.client.ResourceAccessException
18          - org.springframework.web.client.HttpServerErrorException
19
20  bulkhead:
21    instances:
22      shippingService:
23        max-concurrent-calls: 20
24        max-wait-duration: 500ms
25
26
27  management:
28    health:
29      circuitbreakers:
30        enabled: true
31    endpoint:
32      health:
33        show-details: always
34
35

```

CI/CD Avanzado

- **Activadores y parámetros:**

El workflow se lanza con push o pull_request a master/develop, y permite workflow_dispatch con inputs (environment, deploy). Esto habilita ejecuciones manuales para dev/stage/prod y un flag para saltar despliegues cuando solo se necesita build/scan.



- **Variables globales:**

Define versión de Java, flags de Maven, nombres de ACR/AKS y namespace base. Esto evita cambios en línea en cada job y asegura consistencia entre etapas.

Repository secrets		New repository secret
Name	Last updated	
ACR_PASSWORD	last week	
ACR_USERNAME	last week	
ARM_CLIENT_ID	last week	
ARM_CLIENT_SECRET	last week	
ARM_SUBSCRIPTION_ID	last week	
ARM_TENANT_ID	last week	
AZURE_CREDENTIALS	last week	
ECOMMERCEROJAS435ACR	last week	
SLACK_WEBHOOK_URL	2 days ago	
SONAR_TOKEN	last week	
TERRAFORM_BACKEND_KEY	last week	

- Stage Build & Test:

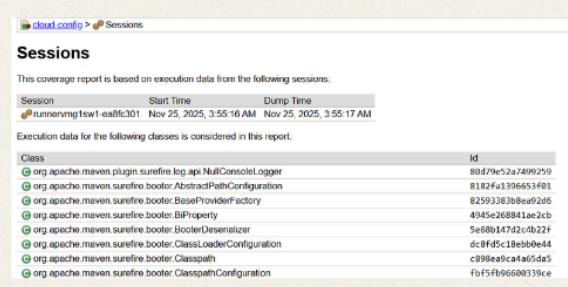
Clona repo con historia completa (necesario para Sonar/GitVersion), cachea Maven, compila (mvn clean compile), corre pruebas con JaCoCo, genera reportes agregados, publica resultados en GitHub Actions y ejecuta análisis SonarCloud con cobertura. Esta etapa produce artefactos .jar empaquetados y reportes de cobertura almacenados como artifacts.



```

268 [INFO] -----
269 [INFO] BUILD SUCCESS
270 [INFO] -----
271 [INFO] Total time: 12.271 s
272 [INFO] Finished at: 2025-11-25T03:55:11Z
273 [INFO] -----

```



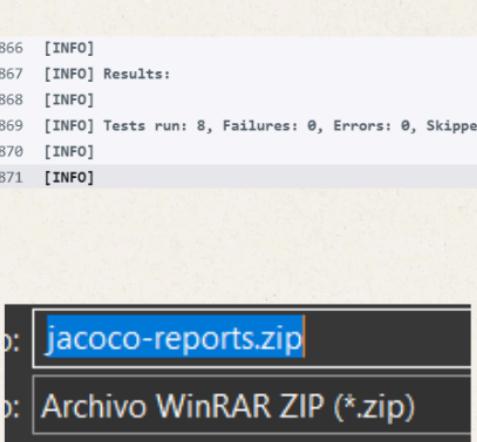
Sessions

This coverage report is based on execution data from the following sessions:

Session	Start Time	Dump Time
#runnervmgtsw1-eabfffc301	Nov 25, 2025, 3:55:16 AM	Nov 25, 2025, 3:55:17 AM

Execution data for the following classes is considered in this report:

Class	Id
org.apache.maven.plugin.surefire.log.api.NullConsoleLogger	80d79e52a7499259
org.apache.maven.surefire.booter.AbstractPathConfiguration	8182fa1396651f01
org.apache.maven.surefire.booter.BaseServiceProviderFactory	82593383b8e9d206
org.apache.maven.surefire.booter.BipProperty	4945c268841ee2c2
org.apache.maven.surefire.booter.BooterDeserializer	5e68b147d2c4b22f
org.apache.maven.surefire.booter.ClassLoaderConfiguration	dc8fd5c18ebbe644
org.apache.maven.surefire.booter.Classpath	c998ea9ca4a50da5
org.apache.maven.surefire.booter.ClasspathConfiguration	fbd5fb6660339ce



- **Stage Semantic Version:**

Usa GitVersion para calcular un semver; si falla, genera fallback con fecha+run. Solo en master y push crea notas, tag v<semver> y release GitHub (con fallback auto-notes). La salida version alimenta etapas posteriores para etiquetar imágenes.

- **Stage Security Scan:**

Descarga artefactos, ejecuta Trivy (filesystem) y OWASP Dependency Check. Publica SARIF a GitHub Security y el HTML de OWASP como artifact. Esto garantiza que dependencias y paquetes estén auditados antes de construir imágenes.

The screenshot shows a GitHub Actions step titled "Upload Trivy results to GitHub Security". The log output is as follows:

```

1 ► Run github/codeql-action/upload-sarif@v
15 Post-processing sarif files: ["trivy-fs-"
16 Validating trivy-fs-results.sarif
17 Adding fingerprints to SARIF file. See https://support-for-code-scanning#providing-data
18 (node:2231) [DEP0169] DeprecationWarning:
    API instead. CVEs are not issued for `ur
19 (Use `node --trace-deprecation ...` to s
20 ► Uploading code scanning results
23 ► Waiting for processing to finish

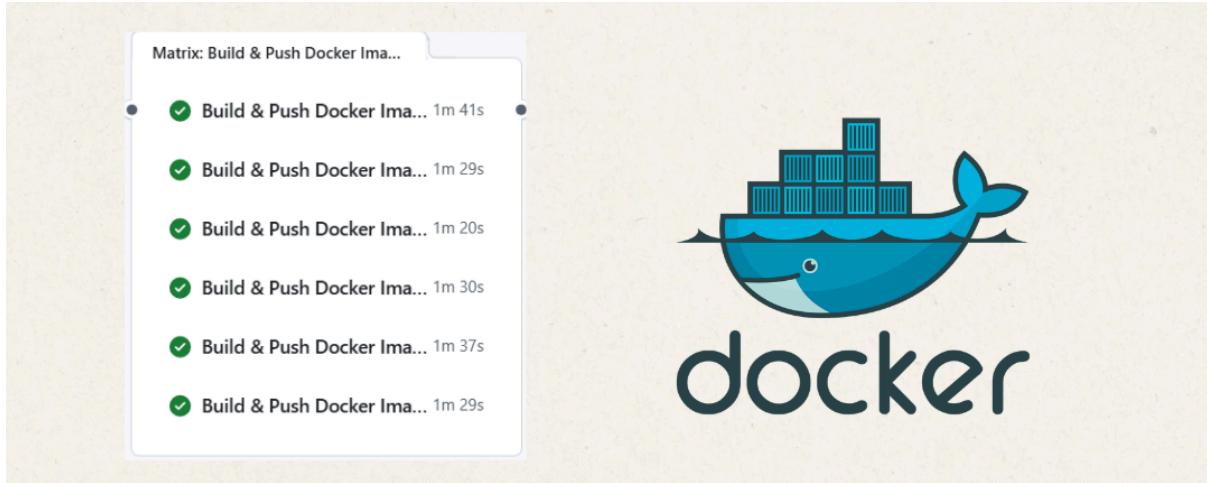
```

On the right, there is a "Code scanning" interface showing a list of vulnerabilities found in the master branch. The list includes:

- spring-framework: RCE via Data Binding on JDK 9+ (Critical) Library #1278 opened last week • Detected by Trivy in app/.libs/spring-webflux-5.3.13.jar...
- springframework: Security Bypass With Un-Prefixed Double Wildcard Pattern (Critical) Library #1188 opened last week • Detected by Trivy in app/.libs/spring-webflux-5.3.13.jar...
- spring-framework: RCE via Data Binding on JDK 9+ (Critical) Library #1187 opened last week • Detected by Trivy in app/.libs/spring-webflux-5.3.13.jar...
- spring: HttpClientServiceExporter readRemoteInvocation method untrusted java serialization (Critical) Library #1187 opened last week • Detected by Trivy in app/.libs/spring-web-5.3.13.jar...
- spring-framework: RCE via Data Binding on JDK 9+ (Critical) Library #1179 opened last week • Detected by Trivy in app/.libs/spring-beans-5.3.13.jar...
- spring-boot: Security Bypass With Wildcard Pattern Matching on Cloud Foundry (Critical) Library #1179 opened last week • Detected by Trivy in app/.libs/spring-beans-5.3.13.jar...

• Stage Build Docker Images:

Por cada servicio en matriz, descarga jars, monta Dockerfile dinámico con Temurin JRE, compila imagen multi-tag (pom version, short SHA, latest, y semver si existe). Autentica con ACR, sube imágenes y corre Trivy sobre la imagen resultante, enviando hallazgos a GitHub Security.



• Stage Deploy to AKS (dev/stage):

Solo en push a master o cuando deploy=true. Inicia sesión en Azure, configura contexto AKS, crea namespace si falta, aplica k8s-6-services.yaml, espera rollout para cada deployment, lista pods/servicios, y ejecuta smoke test resolviendo IP del LoadBalancer y chequeando /actuator/health.



- **Stage Performance Test:**

Depende de deploy-to-aks; si el despliegue tuvo éxito, usa Locust headless (configurable vía requirements-performance.txt) para stress de 3 minutos, genera logs y CSV, y sube reports/performance.

766	GET [Read]	Browse Products	2002	0(0.00%)		150	131	362	140 11.13
767	GET [Read]	View Product	946	0(0.00%)		150	131	335	140 5.26
768	----- -----	----- -----	----- -----	----- -----	----- -----	----- -----	----- -----	----- -----	----- -----
769	Aggregated		2949	0(0.00%)		150	131	362	140 16.40
0.00			0.00			0.00			0.00

- **Stage ZAP Scan:**

También depende del despliegue exitoso; obtiene IP del gateway, corre OWASP ZAP Baseline con opciones agresivas (-a -d -I), mueve ocho formatos de reporte a reports/security y los publica como artifacts.

Run ZAP Baseline Scan

```

1 ► Run zapproxy/action-baseline@v0.14.0
22 starting the program
23 github run id :19657675868
24 /usr/bin/chmod +xw /home/runner/work/ecommerce-microservice-backend-app/ecommerce-microservice-backend-app
25 /usr/bin/docker pull ghcr.io/zaproxy/zaproxy:stable -q
26 ghcr.io/zaproxy/zaproxy:stable
27 /usr/bin/docker run -v /home/runner/work/ecommerce-microservice-backend-app/ecommerce-microservice-backend-app:/zap/wrk/:rw --network=host -e ZAP_AUTH_HEADER
-e ZAP_AUTH_HEADER_VALUE -e ZAP_AUTH_HEADER_SITE -t ghcr.io/zaproxy/zaproxy:stable zap-baseline.py -t http://57.156.48.54:8080 -J report_json.json -w
report_md.md -r report_html.html -a -d -I

```

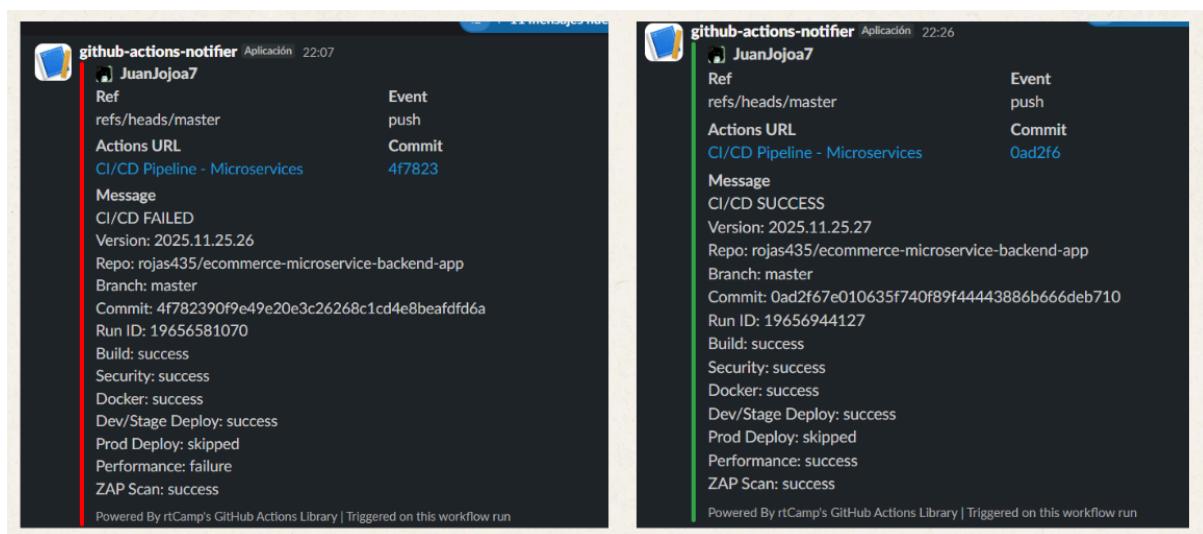
zap-scan-report.zip
Archivo WinRAR ZIP (*.zip)

• Stage Deploy to Prod:

Solo vía workflow_dispatch con environment=prod. Replica login/namespace/manifest apply, pero permite tolerar fallas de rollout (usa || true), y ejecuta smoke test final. Esta etapa requiere aprobación del entorno en GitHub Environments.

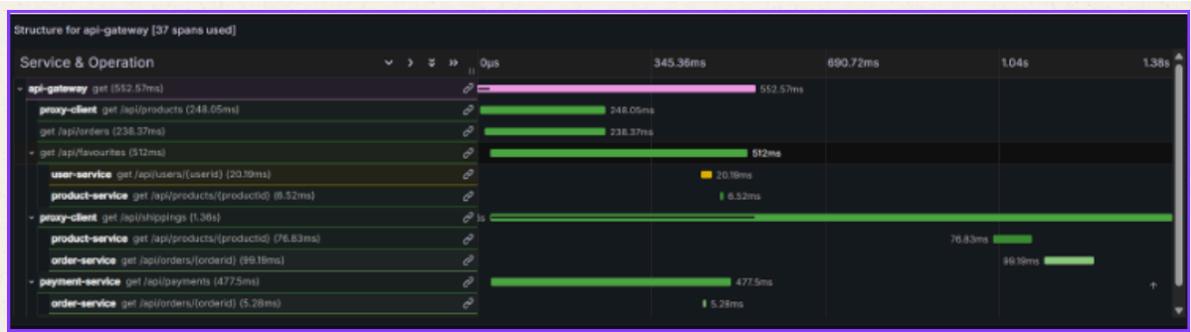
• Stage Notify:

Siempre se ejecuta; revisa resultados agregados de todos los jobs y determina un estado general. Imprime resumen en logs, verifica si el secret de Slack está presente; si sí, envía mensaje con versión, branch y estado de cada etapa. En caso de fallo global, crea un issue automático con enlace al run.



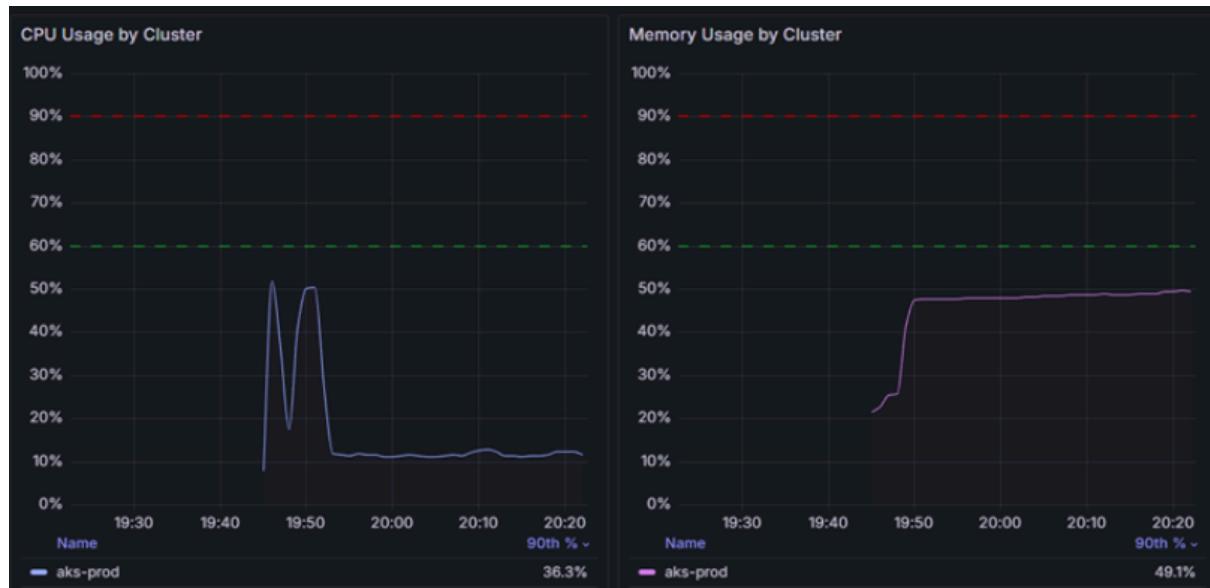
Trazas Zipkin en Grafana:

Aquí podemos visualizar todos los microservicios funcionando en el panel y viendo el tiempo en ms de cada uno lo que demora en responder cada uno de ellos, estas métricas son importantes para conocer el estado de vida de los mismos y sus tiempos de respuesta.



Métricas de CPU y RAM:

En esta sección podemos visualizar el panel del cluster donde tenemos métricas como el uso de CPU y ram, asimismo vemos los niveles de alerta de uso y de alerta crítica, en la imagen podemos ver que el cluster se comporta de manera normal y contiene los microservicios.



Resultados Finales

Pruebas Locust

Métrica Clave	Valor Promedio (Estable)	Estado
Usuarios Concurrentes	40 usuarios	Meta alcanzada
Tasa de Peticiones (RPS)	~16.5 req/s	Flujo constante
Tasa de Fallos (Errores)	0% (0 fallos)	Excelente
Tiempo de Respuesta Promedio	~151 ms	Muy rápido
Percentil 95% (P95)	~300 ms	Aceptable

Podemos ver que obtuvimos un gran resultado al hacer pruebas de rendimiento con 40 usuarios simultáneos consiguiendo métricas buenas para lo que es el proyecto, dado que

hay que considerar que no tenemos el mejor cómputo, y estamos economizando costos, es un gran paso para la optimización de programas a gran escala, además los patrones ayudaron mucho con esta carga y la distribución de la misma.

Pruebas ZAP

ZAP Scanning Report

Encontramos que en las pruebas no tenemos ninguna alerta de alto nivel solo informacional, esto sugiere que el código esta bien escrito y sin alguna vulnerabilidad grave o brecha de seguridad.

Costos de Azure:

Managed Disk 32 GB SSD	\$2-3 USD/ mes	<u>aBuATfSNBtwxtQrwK--BrbuLvKffsDnS1mCoAAaAIWTEALw_wcB</u>
ACR Basic	\$5 USD/ mes	
Storage Account	\$0.50 USD/ mes	
Load Balancer	\$20 USD/ mes	
Public IP	\$3 USD/ mes	
Bandwidth	\$1 USD/ mes	

Conclusiones

- El proyecto quedó desplegable end-to-end: código, contenedores y pipeline CI/CD listos para ambientes reales.
- Observabilidad y seguridad dejaron de ser opcionales; ahora son parte del flujo estándar.

- Lecciones clave: automatizar desde el inicio evita reprocesos; documentar hosts/secretos ahorra soporte; y validar clusters antes de aplicar manifiestos es vital para no perder tiempo.

Video explicativo:

<https://youtu.be/ioxino6q5X8>