# 1) C4 – Contexto de Integración

**1.1 Propósito del diseño**

Proponer cómo **todos los sistemas** (canales, cores, pagos, riesgos, fraude y terceros) **se comunican de forma segura, desacoplada y confiable** mientras el banco migra gradualmente del **Core Tradicional** a un **Nuevo Core Digital** sin interrumpir el servicio.

**1.2 Alcance**

- **Canales**: Web Banking y App Móvil (puntos de entrada del cliente).
- **Capa de Integración**:
    - **API Gateway / API Manager**: entrada única, políticas de seguridad, cuotas, versionado.
    - **Orquestación / ESB ligero**: coordina procesos (ej., ejecución de pagos).
    - **Plataforma de Eventos (EDA)**: publica/suscribe eventos de negocio (desacople).
    - **Modelo Canónico**: lenguaje de datos común para reducir acoplamiento.
- **Backends**:
    - **Core Tradicional** (legado) y **Nuevo Core Digital** (coexistencia y migración).
    - **Plataforma de Pagos** (estándar ISO 20022).
    - **Riesgos** y **Fraude** (evaluaciones en línea).
    - **APIs de Terceros / Open Finance** (exposición controlada y cumplimiento).
- **Servicios Transversales**:
  **IAM** (OIDC/OAuth2), **WAF/Perímetro**, **Observabilidad** (logs/métricas/trazas), **Gestión de Secretos**.

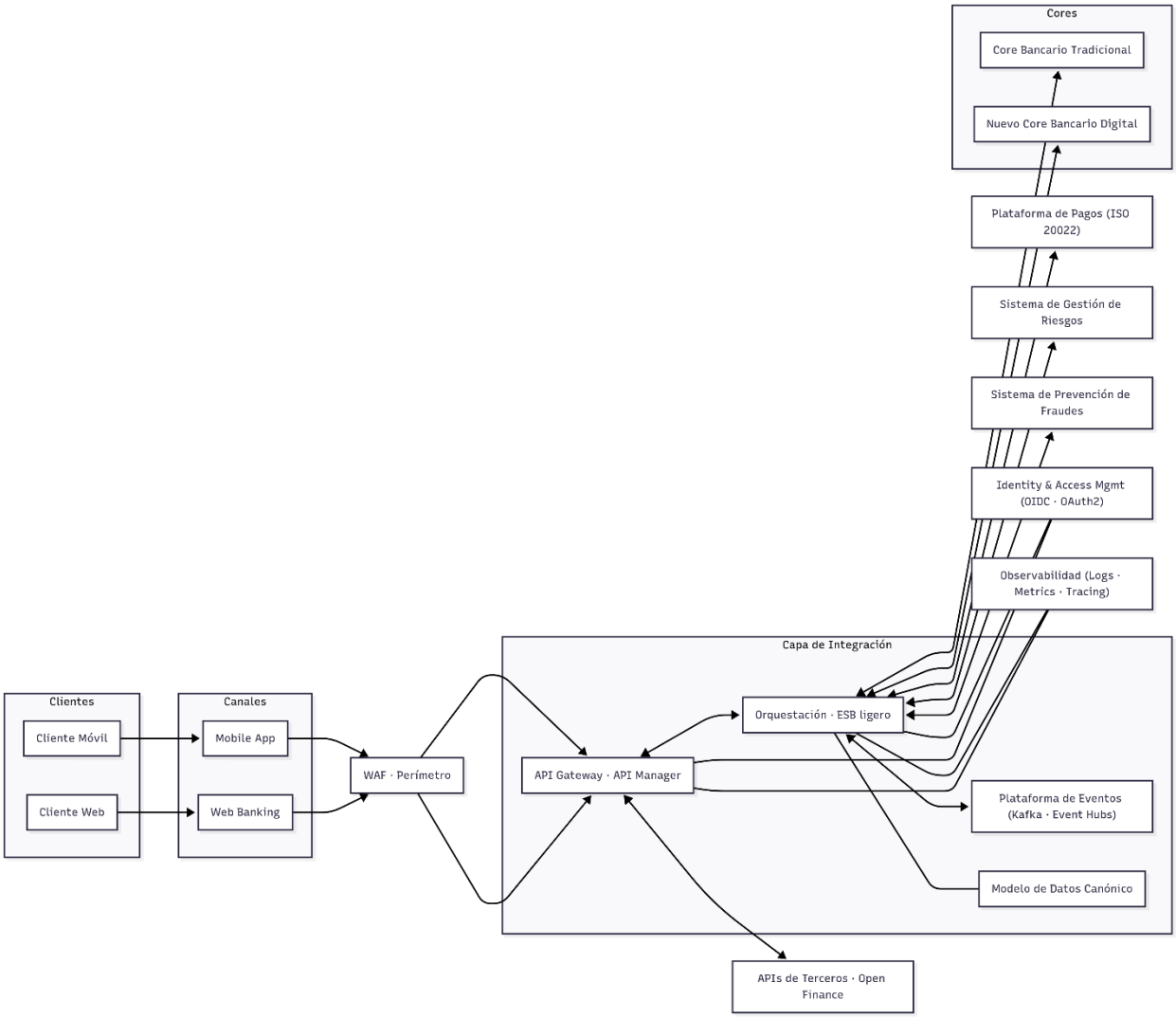
**Por qué así**: El **Gateway** protege y gobierna el acceso; la **Orquestación** implementa reglas y secuencias; la **Mensajería/EDA** desacopla (cada consumidor avanza a su ritmo); el **Modelo Canónico** evita dependencias frágiles entre canales y backends. Esto facilita la **migración por partes** (estrategia *Strangler Fig*).

### 1.3 Supuestos clave

- Los canales hablan **HTTPs/REST** hacia el Gateway.
- Procesos de larga duración (p. ej., pagos con confirmación externa) usan **eventos**.
- Autenticación de clientes con **OIDC/OAuth2**; **mTLS** en el tráfico interno.
- La plataforma corre en nubes tipo **Azure/AWS** o *on-prem* con equivalentes.

### 1.4 No-objetivos (explícitos)

- No se define aún el esquema físico de bases de datos.
- No se selecciona un proveedor único (se dejan alternativas equivalentes).
- No se prescribe una sola herramienta ESB; se prioriza **ligereza + estándares**



- **APIGW**: entrada única, authz, *rate limits*, versionado, governance.
- **ORCH**: coordina pasos de negocio (p. ej., validar → autorizar → ejecutar pago).
- **EDA**: eventos "hechos" de negocio (PagoRealizado, ClienteActualizado).
- **CANON**: modelo compartido (reduce transformaciones punto-a-punto).
- **IAM**: autenticación/autorización (OIDC/OAuth2) y *mTLS* interno.
- **OBS**: trazabilidad extremo a extremo (correlation-id).
- **WAF**: protección perimetral (OWASP Top-10, DDoS, etc.).

### 1.6 Casos de uso representativos

**CU-1: Consultar saldo**

1. App Móvil → **API Gateway** (token OIDC)
2. Gateway → **Orquestación** → **Accounts** (a través de adaptador según core del cliente)
3. Respuesta normalizada con **Modelo Canónico**; se registra *trace* en Observabilidad.

   **Por qué**: latencia baja y lectura directa; el cliente no debe saber en qué core está su cuenta.

---

**CU-2: Ejecutar pago**

1. Canal (Web/App) llama POST /payments con **Idempotency-Key**.
2. **API Gateway** valida token y límites; pasa a **Orquestación**.
3. Orquestación valida con **Riesgos** y **Fraude**.
4. Llama **Plataforma de Pagos** usando mensajes **ISO 20022** y al **Core** correspondiente (Tradicional o Digital).
5. Publica **evento** PagoRealizado en **EDA** (Contabilidad, Notificaciones y Analytics se enteran sin acoplarse).

6. Respuesta al canal; si el canal reintenta, la **Idempotency-Key** evita duplicidades.

   **Por qué**: procesos sensibles usan *sagas* y eventos; resiliencia y auditabilidad.

---

**CU-3: Open Finance – Obtención de movimientos**

1. Tercero autorizado → **API Gateway (exposición externa)**
2. **Scopes**/consentimientos controlados por **IAM** y políticas del Gateway.
3. **Orquestación** agrega y normaliza datos desde ambos **cores**.

   **Por qué**: cumplimiento regulatorio y control fino de acceso vía scopes y consent.

**1.7 Mapeo a BIAN (nivel contexto)**

| **Dominio BIAN (ejemplos)** | **Elemento en el diseño** | **Justificación** |
|---|---|---|
| **Current Account** | Adaptadores de Cuentas en Orquestación | Gestión de cuentas y saldos. |
| **Payment Execution / Clearing** | Orquestación + Plataforma de Pagos | Estándares ISO 20022 y conciliación. |
| **Fraud Detection** | Integración con **Fraud** | Validaciones en línea por transacción. |
| **Risk Analytics** | Integración con **Risk** | Scoring/limites antes de autorizar. |
| **Party/Customer** | Modelo Canónico + Clientes | Identidad de cliente y KYC/AML. |

| Dominio BIAN (ejemplos) | Elemento en el diseño | Justificación |
|---|---|---|
| **Channel Activity** | Canales + Gateway | Exposición segura y gobernada. |
| **Identity & Access** | **IAM** (OIDC/OAuth2) | Autenticación/autorización estandarizadas. |

**Por qué mencionar BIAN**: da un **lenguaje común** entre negocio y tecnología, mejora alineación y facilita la **migración por capacidades**.

### 1.8 Decisiones arquitectónicas (resumen y por qué)

- **API Gateway/Manager**: seguridad, versionado y gobierno central.
- **EDA + eventos de negocio**: desacople, escalabilidad y extensibilidad.
- **Orquestación ligera**: reglas claras de proceso sin bloquear canales.
- **Modelo Canónico**: evita N×M transformaciones; facilita multicore.
- **Coexistencia de cores** (*Strangler Fig*): migración por dominios, riesgo bajo.
- **Transversales obligatorios**: IAM, WAF, Observabilidad, Gestión de Secretos.

---

### 1.9 Riesgos clave y mitigaciones

- **Duplicidad de pagos** → *Idempotency-Key*, *outbox*, *exactly-once* lógico.
- **Latencia** → caché/QoS en consultas, asincronía en procesos largos.
- **Acoplamiento accidental** → modelo canónico, *contract testing*, versionado.
- **Fallas de un core** → *circuit breaker*, *retries*, *fallbacks*, DR.
- **Cumplimiento/PII** → tokenización, cifrado, *least privilege*, auditoría.

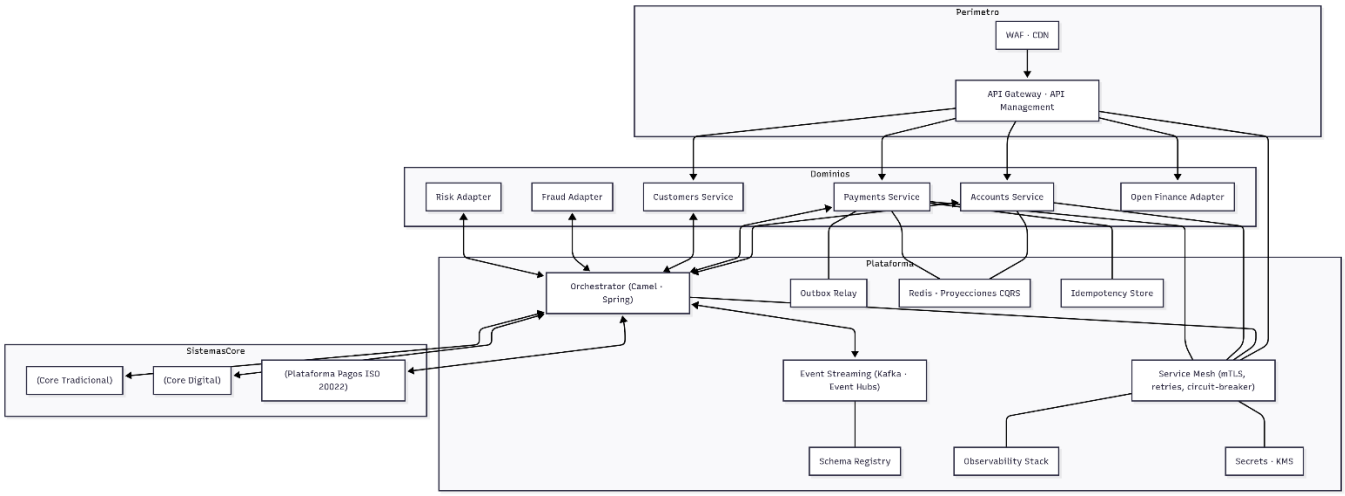
## 2) C4 – Contenedores (Plataforma y piezas desplegables)

### 2.1 Objetivo

Mostrar **qué se despliega** (plataforma, servicios, adaptadores) y **cómo se conectan** para soportar los casos de uso del banco con **seguridad, HA/DR y observabilidad**.

### 2.2 Decisiones clave (y por qué)

- **API Gateway/Manager** (Azure APIM, AWS API Gateway, Apigee o Kong): *entry point*, políticas de seguridad, cuotas, versionado y developer portal.
- **Service Mesh (Istio/Linkerd)**: mTLS, *circuit breaker/retry/timeout*, *traffic shaping* (blue/green, canary).
- **Orquestador/ESB ligero (Apache Camel/Spring)**: coordina flujos de negocio y transforma datos hacia/desde el **Modelo Canónico**.
- **Plataforma de Eventos (Kafka/Event Hubs)**: EDA para desacople, *outbox pattern*, DLQ y reintentos.
- **Schema Registry** (Avro/JSON Schema): contratos de eventos y compatibilidad.
- **Adapters** a: Core Tradicional, Core Digital, Pagos (ISO 20022), Riesgos y Fraude.
- **Observabilidad** (OpenTelemetry + Prometheus + Grafana + ELK/Opensearch): métricas, logs y trazas end-to-end.
- **IAM** (OIDC/OAuth2 con AD B2C/Cognito/Keycloak): autenticación y autorización consistente.
- **Gestión de secretos** (Vault/KeyVault/Secrets Manager) y **KMS** para claveo.
- **Redis** para cachés/proyecciones de lectura (latencia baja) y **Idempotency Store**.




### 2.4 Interfaces y protocolos

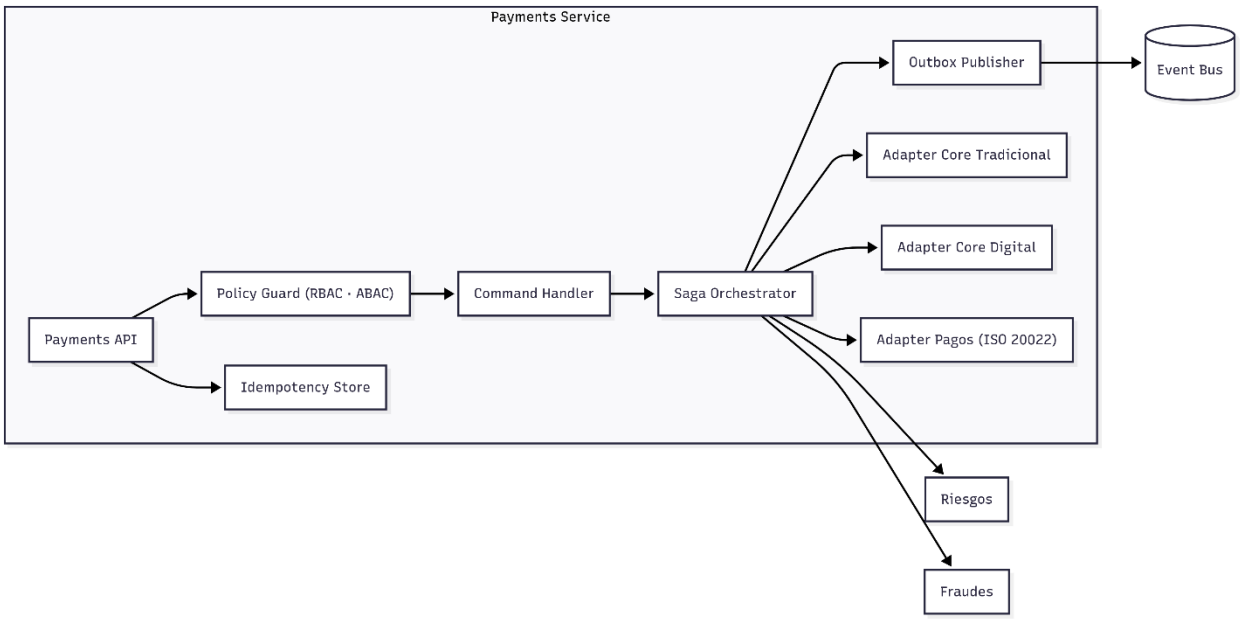
- **REST/HTTPs** para canales y terceros (OpenAPI).
- **Mensajería** (Kafka) para eventos de negocio (AsyncAPI).
- **ISO 20022** hacia *clearing* de pagos.
- **mTLS** interno + **W3C Trace Context** para trazas.

## 3) C4 – Componentes (detalle de 1–2 servicios críticos)

### 3.1 Payments Service (ejemplo completo)

**Componentes internos:**

- **Payments API** (REST) con **Idempotency-Key**.
- **Command Handler** (valida y encola comandos).
- **Saga Orchestrator** (coordina pasos: riesgo → fraude → core → pagos).
- **Adapters** (Core Tradicional, Core Digital, Pagos ISO 20022).
- **Idempotency Store** (evita duplicados).
- **Outbox Publisher** (publica evento PagoRealizado).
- **Policy Guard** (ABAC/RBAC por monto/segmento).




**Por qué así**

- **Sagas**: manejan compensaciones (si falla el clearing, se revierte).
- **Idempotencia** + **Outbox**: consistencia y "exactly-once lógico".
- **Adapters**: esconden diferencias entre cores/pagos; el canal no se entera.

**3.2 Accounts Service**

- **API de Cuentas** → **Query Layer** → **Proyecciones (Redis/CQRS)** para lecturas rápidas.
- **Adapters** a ambos cores para *cache warmup* y *misses*.
- Justifica **CQRS**: latencias bajas y escalado independiente lectura/escritura.

**4) Patrones de integración y tecnologías (justificación breve)**

- **API-First (OpenAPI/AsyncAPI)**: contrato claro y *contract testing*.
- **Modelo Canónico**: reducir transformaciones N×M; estabilidad de interfaces.
- **EDA (eventos)** + **Outbox**: desacople y resiliencia.
- **Saga**: consistencia en procesos distribuidos.
- **Strangler Fig**: migrar por capacidades sin "big bang".

- **Circuit Breaker/Retry/Timeout** (mesh): robustez ante fallos intermitentes.
- **CQRS** (lecturas): experiencia rápida y coste controlado.
- **Idempotency-Key**: evita pagos duplicados.

---

**5) Seguridad, cumplimiento y protección de datos**

**5.1 Autenticación y autorización**

- **OIDC + OAuth2** (Auth Code + PKCE en móviles).
- **Scopes** granulares por dominio (p.ej., payments.execute, accounts.read).
- **MFA** y **step-up** para montos altos.
- **mTLS** servicio-servicio (mesh).

**5.2 Protección de datos (PII/PCI)**

- **Cifrado** en tránsito (TLS 1.2+) y en reposo (KMS).
- **Tokenización/enmascarado** de PAN/PII.
- **Minimización de datos** y **retención limitada**.
- **Auditoría** no repudiable (logs inmutables).
- **DLP** y segregación de ambientes.
- **Cumplimiento**: **PCI DSS** (pagos), **GDPR/ley local de datos**, lineamientos de **Open Finance**.

**5.3 Controles en el API Gateway**

- **WAF**, **rate limits**, **quotas**, **schema validation**, **CORS**, **bot protections**.
- **Políticas** por versión de API y por *consumer* (contratos/SLA).

## 6) Alta disponibilidad (HA) y Recuperación ante Desastres (DR)

### 6.1 Objetivos

- **RPO ≤ 5 minutos**, **RTO ≤ 30 minutos** (declárales metas).
- **Multi-AZ** para servicios *stateless* (réplicas ≥3 + HPA).
- **DB** con réplica síncrona (HA) y asíncrona (DR inter-región).
- **Kafka** con múltiples *brokers* y *replication factor* ≥3.
- **Backups** automáticos + **pruebas de restore**.
- **Infra as Code** y **runbooks** (procedimientos) para DR.
- **Chaos drills** periódicos.

### 6.2 Estrategia

- **Active-Active** para gateway, mesh, orquestación y mensajería.
- **Active-Passive** para sistemas que no soportan AA (p.ej., Core Tradicional).

---

## 7) Estrategia de integración multicore (convivencia legado + digital)

### 7.1 Principios

- **Capa de abstracción** en la **Capa de Integración** (adapters + modelo canónico).
- **Ruteo** por reglas (producto, rango de cuenta, cohortes de clientes).
- **Shadow/dual-write** controlado (eventos + outbox) donde aplique.
- **Criterios de corte** por **dominio BIAN** (ej.: nuevas cuentas → digital).

### 7.2 Datos y reconciliación

- **Eventos de negocio** para *sync* eventual.
- **Herramientas de reconciliación** diarias (saldos, movimientos, ledger).
- **Estrategia de "freeze windows"** al mover cohortes sensibles.

---

## 8) Gestión de identidad y acceso (IAM)

### 8.1 Flujos soportados

- **Clientes**: OIDC (Auth Code + PKCE), refresco de tokens, *consent* para Open Finance.
- **Sistemas internos**: mTLS + *client credentials* (OAuth2).
- **Administración**: RBAC/ABAC (según rol y atributos del cliente).

**8.2 Políticas**

- **Least privilege**, **separación de funciones**, rotación de credenciales, *just-in-time access*.
- **Auditoría** de accesos y decisiones de autorización.

---

**9) APIs internas/externas y mensajería (estándares)**

**9.1 Contratos**

- **OpenAPI** para REST, **AsyncAPI** para eventos.
- **Versionado** (v1/v2) y política de **deprecación** con fechas.
- **Contract testing** (PACT) y validación de esquemas en gateway/consumidores.

**9.2 Requisitos técnicos**

- **Idempotency-Key** en POST /payments.
- **Correlation-ID** (W3C Trace Context).
- **Errores estandarizados** (RFC 7807 – problem+json).
- **Límites**: rate limiting por *consumer* y *burst control*.

Ejemplo de encabezados

**Idempotency-Key: 3b2f7d84-4a2c-4a27-8f0e-1e9c9c9f8ad1**

**X-Correlation-ID: 6c1a1e2f-65e1-4d51-bcb0-4f5e2c1c8fa0**

**Authorization: Bearer <access_token>**

**Content-Type: application/json**

### 9.3 Mensajería

- **Tópicos por dominio** (payments.events, accounts.events).
- **Schema Registry** obligatorio, **DLQ**, reintentos con *backoff*.
- **Idempotencia** por eventId + *store* en consumidores críticos.

## 10) Gobierno de APIs y microservicios

### 10.1 Proceso

- **API Design Review** (arquitectura + seguridad).
- Registro en **Developer Portal** con SLA, cuotas, T&C y *changelog*.
- **Service Catalog** (Backstage): dueño, SLO, *oncall*, dependencias.

### 10.2 Políticas

- **Políticas globales** en Gateway: authn, authz, CORS, límites, validación.
- **Scorecards** de calidad (linters OpenAPI, cobertura de observabilidad).
- **SLOs** y **error budgets** (latencia P95, tasa de errores, disponibilidad).

---

## 11) Plan de migración gradual (minimizar riesgo operativo)

### 11.1 Fases

1. **Fundación de plataforma**: cluster, gateway, mesh, observabilidad, IAM, Kafka, Vault.
2. **Modelo Canónico + Adapters** (read-only primero). **Shadow traffic** desde canales.
3. **Dominios piloto**: *Accounts (read)* y *Payments (read)* con **CQRS**.
4. **Escrituras controladas** en *Payments*: **Idempotency + Outbox + Saga**; *feature flags*.
5. **Cohortes**: nuevas cuentas → Core Digital; migraciones por lotes del legado.
6. **Decom** progresivo del Core Tradicional por capacidades.

### 11.2 Métricas de éxito (reporta en tablero)

- **Latencia P95** por endpoint, **Disponibilidad** (99.9+), **Errores** (<0.1%).

- % de tráfico en Core Digital, **tasa de fraude detenido**, **incidentes** y **MTTR**.
- Cumplimiento de **RPO/RTO**, **pruebas de restore** exitosas.

**11.3 Riesgos y mitigación**

- **Regulatorio** → gobierno de datos/retención, DPIA/PIA, auditorías.
- **Datos inconsistentes** → reconciliación + *event sourcing* para dominios críticos.
- **Deuda técnica** → *gates* de calidad y ventanas de *hardening* por fase.