CONFIDENCIAL

FINTRIXS PAY

Anexos técnicos

09.1 — ARQUITECTURA TÉCNICA DE FINTRIXS PAY

CÓDIGO	ANX-001
VERSIÓN	1.0 — 16 de abril de 2026
APROBADO POR	Representante Legal
VIGENCIA	—

FINTRIXS S.A.S. · NIT 901.994.194-3
Paquete de Due Diligence — Banco Sponsor Agregador
Emitido el 16 de abril de 2026 — Versión 1.0 del paquete

09.1 — ARQUITECTURA TÉCNICA DE FINTRIXS PAY · Código: ANX-001 · Versión: 1.0 — 16 de abril de 2026 · FINTRIXS S.A.S.

1. Visión general

Fintrixs Pay es una plataforma de pagos construida como un monorepo de microservicios. El backend está compuesto por ~15 servicios en NestJS (TypeScript 5.3 estricto), el frontend es una SPA en Vue 3 con Composition API, la capa de datos es PostgreSQL 16 con RLS, la mensajería asíncrona es Apache Kafka (Confluent/MSK) y el API Gateway es Kong.

2. Diagrama lógico (texto estructurado)

```
[Browser / Mobile]
       |
       ▼
[CloudFront + WAF (AWS)]
       |
       ▼
[Kong API Gateway (4000)]  ──── JWT RS256 + rate limiting
       |
       ├─► [fintrix-api-gateway :4000]  (Apollo Federation)
       ├─► [postgraphile-gateway :4001] (GraphQL auto-generado)
       |
       ├─► [auth-service :3001]
       ├─► [payments-api :3000]
       ├─► [tokenization-service :3002]
       ├─► [card-vault-service :3003]          ◄── CDE PCI aislado
       ├─► [webhooks-service :3004]
       ├─► [orchestration-service :3005]
       ├─► [email-service :3006]
       ├─► [onboarding-service :3007]
       ├─► [realtime-gateway :3008]            ── SSE al dashboard
       ├─► [admin-audit-service :3009]
       └─► [integration-runtime :3010]

Servicios generados por factory: ms-countries, ms-cities, ms-branches, ms-roles, ms-customers

[Apache Kafka (MSK)]  ◄── Outbox + Inbox pattern (idempotencia)
       |
       ├─► Consumidores: email-service, orchestration-service, integration-runtime, ...

[PostgreSQL 16 (RDS Multi-AZ)]  ◄── Schemas: payments_core, payments_api, vault (scope PCI)
       |                              RLS por app.current_merchant_id
       └─► Réplica de lectura + cross-region snapshot (DR)

[AWS KMS] — llaves gestionadas (AES-256, rotación automática)
[AWS S3 / MinIO dev] — documentos KYB, evidencias
[AWS Secrets Manager] — secretos runtime
[AWS CloudWatch + Prometheus + Grafana] — observabilidad
[SIEM] — correlación y detección de amenazas
```

3. Flujo transaccional (happy path)

1. Cliente ingresa datos en el iframe / dropin de Fintrixs.
2. PAN viaja únicamente al `card-vault-service` (VPC PCI) vía TLS + mTLS.
3. `card-vault-service` cifra con AES-256 (KMS) y devuelve un token opaco.
4. `tokenization-service` intercambia token interno ↔ token de red (Visa/Credibanco) cuando aplica.
5. `payments-api` crea la transacción, escribe evento en tabla `outbox`.

6. Publisher lee el outbox y publica evento en Kafka.
7. `payments-api` enruta la autorización a **Credibanco** (red Visa) con data 3DS 2.2.
8. Respuesta del emisor se propaga: `payments-api` → Kafka → consumidores → frontend vía SSE (`realtime-gateway`).
9. Subcomercio recibe webhook firmado por `webhooks-service`.
10. `admin-audit-service` registra cada paso con correlación UUID.

4. Microservicios — detalle por dominio

4.1. Servicios PCI (scope reducido)

- **card-vault-service (:3003)** — único custodio del PAN. VPC/subnet dedicada, network policies restrictivas, acceso sólo vía JWT con scope PCI + MFA.
- **tokenization-service (:3002)** — emite/valida tokens opacos; puente con tokenización de red Visa.

4.2. Servicios core

- **payments-api (:3000)** — transacciones, outbox, RLS, orquestación síncrona.
- **auth-service (:3001)** — JWT RS256 (llaves en KMS), login, refresh, revocación.
- **webhooks-service (:3004)** — entrega firmada (HMAC SHA-256) con reintentos exponenciales y DLQ.
- **orchestration-service (:3005)** — ruteo de eventos internos y notificaciones.
- **admin-audit-service (:3009)** — auditoría inmutable (append-only) de acciones administrativas.

4.3. Servicios de soporte

- **onboarding-service (:3007)** — KYC/KYB, carga de documentos a S3 con preSigned URLs.
- **email-service (:3006)** — consumidor Kafka; entrega via SES/SMTP; plantillas.
- **realtime-gateway (:3008)** — SSE `fintrix.ui.*` al frontend.
- **integration-runtime (:3010)** — ejecutor YAML-driven para integraciones.

4.4. Gateways GraphQL

- **fintrix-api-gateway (:4000)** — Apollo Federation.
- **postgraphile-gateway (:4001)** — GraphQL auto-generado (solo lectura sobre `payments_api`).

5. Patrones arquitectónicos

5.1. Outbox + Inbox transaccional

- Evento insertado en tabla `outbox` dentro de la misma transacción del cambio de estado.

- Publisher separado publica en Kafka.
- Consumidores usan `@fintrix/event-inbox-pg` para idempotencia.
- Prohibido publicar a Kafka directamente desde un handler HTTP.

5.2. Multi-tenancy con RLS

- Paquete `@fintrix/tenant-context` inyecta `app.current_merchant_id` desde el JWT.
- Cada política RLS en PostgreSQL filtra por merchant.
- Pruebas en `@fintrix/rls-testkit` contra DB real (nunca mock).

5.3. Scope PCI estrictamente acotado

- Solo `card-vault-service` y `tokenization-service` manejan PANs.
- Logger `@fintrix/logging` con masking automático; bypass prohibido por CI.

5.4. Service Factory

- Servicios CRUD simples se generan desde YAML (`backend/packages/fintrix-cli`).
- No se editan a mano los servicios en `apps/generated/` .

6. Infraestructura

6.1. Cloud

- AWS `us-east-1` (producción primaria).
- AWS `us-west-2` (DR warm standby).
- EKS (Kubernetes) como orquestador.
- RDS PostgreSQL 16 Multi-AZ + réplica cross-region asíncrona.
- MSK (Kafka 7.5.0) — 3 brokers, RF ≥ 3.
- CloudFront + WAF.
- Route53 con health checks.

6.2. Ambientes

Ambiente	Cuenta AWS	Propósito
Dev	fintrixs-dev	Desarrollo interno
Staging	fintrixs-staging	Pre-producción, pruebas E2E
Prod	fintrixs-prod	Producción
DR	fintrixs-prod (us-west-2)	Recuperación ante desastres

6.3. CI/CD

- GitHub Actions: `backend-ci.yml`, `microservices-eks-deploy.yml`, `kong-deck-sync.yml`, `terraform.yml`.
- IaC en Terraform.
- Docker images firmadas, escaneadas por Trivy.

7. Seguridad en la arquitectura

- **Zero Trust** entre servicios (mTLS para comunicación interna).
- **Segmentación de red** por VPC/subnet y network policies.
- **RBAC** a nivel de aplicación (JWT + scopes).
- **RLS** a nivel de base de datos.
- **DLP** para egress de datos.
- **SIEM** con correlación multi-fuente.
- **Auditoría inmutable** en `admin-audit-service`.
- **Secrets Manager** para credenciales.
- **KMS** para llaves de cifrado con rotación automatizada.

8. Observabilidad

- Logs estructurados via `@fintrix/logging` (correlación por UUID).
- Métricas Prometheus via `prom-client` + `@fintrix/metrics`.
- Dashboards Grafana por dominio.
- Trazas distribuidas (OpenTelemetry → Jaeger).
- Alertas en PagerDuty / Opsgenie.

9. Contratos

- **OpenAPI** para APIs REST.
- **AsyncAPI** para eventos Kafka.
- **GraphQL SDL** para gateways.
- Validación en CI: `npm run docs:openapi:lint`, `docs:asyncapi:lint`, `docs:graphql:check-breaking`.