ANDES
RETAIL

capa de interoperabilidad

Propuesta de arquitectura. Prueba técnica para Arquitecto Senior de Software y Soluciones (Vortexbird).

Autor Andrés Ayala
Entrega 17/11/2026
Versión actualizada la noche del 14 al 15/11/2026 (primer borrador del 10/11/2026)

Contenido

1. Resumen ejecutivo

2. Preguntas, supuestos y riesgos
- 2.1 Preguntas y supuesto adoptado
- 2.2 Supuestos adicionales, sin pregunta asociada
- 2.3 Riesgos

3. Arquitectura de alto nivel
- 3.1 Antes y después
- 3.2 Contenedores
- 3.3 Decisión clave 1: qué es síncrono y qué es asíncrono
- 3.4 Decisión clave 2: rol del API Gateway
- 3.5 Decisión clave 3: dónde se transforma y se valida
- 3.6 Decisión clave 4: versionado de APIs
- 3.7 Disponibilidad 99.9%
- 3.8 Cómo se impide que la arquitectura se degrade
- 3.9 Despliegue y red del demo en AWS

4. Diseño de la capa de interoperabilidad
- 4.1 Bounded contexts
- 4.2 Servicios y responsabilidades
- 4.3 Patrón de integración: adaptadores como capa anticorrupción
- 4.4 Flujo A: crear orden, antes y después
- 4.5 Contrato 1: consultar stock
- 4.6 Contrato 2: crear orden
- 4.7 Modelo de errores
- 4.8 Idempotencia en crear orden

5. Seguridad
- 5.1 Flujos OAuth2 y OIDC
- 5.2 Qué se valida en el gateway y qué en los servicios
- 5.3 Autorización por roles, scopes, partner y canal
- 5.4 Rotación de llaves, expiración y secretos
- 5.5 Datos personales y de pago
- 5.6 Modelo de amenazas
- 5.7 Auditoría
- 5.8 Acceso de los operadores al clúster

6. Resiliencia, performance y escalabilidad
- 6.1 Rate limiting por partner y canal
- 6.2 Cache: dónde, qué, TTL e invalidación
- 6.3 Flujo B: consulta masiva en promo, antes y después
- 6.4 Protección de los legados (backpressure)
- 6.5 Reintentos y DLQ
- 6.6 Plan para promos
- 6.7 Escalabilidad
- 6.8 Evidencia

7. Observabilidad y operación
- 7.1 Campos mínimos de logging
- 7.2 Métricas
- 7.3 Trazas distribuidas y correlación
- 7.4 Alertas
- 7.5 Runbook: falla stock o precio
- 7.6 Herramientas

8. Plan de implementación (roadmap)
- 8.1 Estrategia y orden de migración
- 8.2 MVP: 4 a 6 semanas
- 8.3 A 3 meses
- 8.4 A 6 meses
- 8.5 Equipo y criterio de salida

9. Cultura y liderazgo técnico
- 9.1 Ambigüedad: cómo reducir riesgo sin frenar el avance
- 9.2 Cómo documentar decisiones para escalar el equipo
- 9.3 "Rápido" frente a seguridad y operación: cómo se negocia
- 9.4 Desacuerdos técnicos fuertes: cómo se gestionan
- 9.5 Calidad sin volver lento el delivery

Anexos
- Anexo A. Amenazas 7 a 9
- Anexo B. Métricas y alertas complementarias
- Anexo C. Catálogo de eventos
- Anexo D. Registro de decisiones (ADR)
- Anexo E. Contratos
- Anexo F. Escenarios de falla dirigida
- Anexo G. Siglas y términos

> Este documento sigue la estructura obligatoria de 9 secciones. Cada sección abre con su conclusión y cada decisión trae lo que sacrifica. Lo que todavía no se ha medido dice "por medir": ninguna cifra de resultado es inventada. Las cifras de volumen son supuestos de dimensionamiento (sección 2) y las de latencia o frescura son objetivos de diseño. Los diagramas están exportados como imagen; su fuente Mermaid está en `entregable/diagramas/` y el diseño completo, con contratos y registros de decisión, en el repositorio (`docs/` y `contracts/`). Las siglas se explican la primera vez que aparecen y se reúnen en el anexo G.

1. Resumen ejecutivo

Problema. Los canales de AndesRetail (e-commerce propio y marketplaces) hablan directo con cada sistema. Cada consulta de stock o precio golpea en línea al ERP (Enterprise Resource Planning, el sistema central de gestión, dueño de catálogo y precios) y al WMS (Warehouse Management System, el sistema de bodega, dueño del stock físico). Cada canal coordina a su manera la creación de órdenes contra el OMS (Order Management System, el sistema de órdenes) y nadie es dueño del stock que se puede vender. El resultado son caídas en promociones, sobreventa, órdenes pagadas sin confirmar y ninguna forma de seguir una orden de punta a punta.

Solución propuesta.

1. Una capa de interoperabilidad entre los canales y los sistemas, con APIs estándar y versionadas para stock, precios, órdenes y notificaciones.
2. Las consultas de stock y precio se responden desde un modelo de lectura propio, alimentado por eventos. La carga sobre ERP y WMS deja de depender del tráfico de los canales.
3. La capa se hace dueña del stock vendible: reserva antes de aceptar una orden, con vencimiento.
4. Crear orden es una sola saga orquestada (una secuencia de pasos con su forma de deshacerlos), con idempotencia y compensaciones: lo que se puede deshacer se deshace, lo que ya se cobró se termina.
5. Cada sistema queda detrás de un adaptador que traduce y lo protege con límites de concurrencia. Sumar un sistema es sumar un adaptador.
6. Seguridad uniforme: OAuth2 y OIDC (OpenID Connect), los estándares de autorización e identidad, con permisos por scope, políticas y cuotas por partner, y cifrado mutuo entre servicios. La capa nunca ve datos de tarjeta.
7. Trazabilidad de punta a punta con un identificador de correlación en peticiones, eventos y auditoría.
8. Es síncrono solo lo que la capa puede responder con sus propios datos; lo que depende de un tercero es asíncrono.
9. Migración gradual, canal por canal, con vuelta atrás en cada paso. MVP (minimum viable product: la primera entrega útil en producción) en 4 a 6 semanas con un marketplace piloto.
10. Cada requisito no funcional tiene una prueba automática definida para que la arquitectura no se degrade con el tiempo: unas ya son compuerta del pipeline y el resto se ejecuta aparte y entra a él según el roadmap.

Beneficio de negocio. Vender en promociones sin caídas; no prometer lo que no hay; saber siempre en qué va una orden; sumar un marketplace o cambiar un sistema en semanas y no en meses; y un solo lugar donde aplicar seguridad y cumplimiento.

Lo que cuesta. Más piezas que operar, consistencia eventual declarada en las lecturas, un crear orden que responde "aceptada" y no el resultado final, y un equipo de plataforma que hoy no existe. Las lecturas ya están medidas en el demo (sección 6.8): con el pico 10x, el AS-IS llega al timeout y el TO-BE responde en milisegundos sin cargar al legado. En órdenes, un reintento con la misma llave no duplica (0 de más en 20, contra 20 de 20 duplicadas en el AS-IS), y el pago rechazado, el webhook perdido y el OMS caído se resuelven sin intervención.

2. Preguntas, supuestos y riesgos

Conclusión. El AS-IS (la situación actual) no venía dado, así que se formuló como hipótesis y cada duda quedó como un par: la pregunta que se haría a los stakeholders (personas o áreas con autoridad sobre una decisión) y el supuesto con el que se avanzó. Cada pregunta dice qué decisión de arquitectura cambia si la respuesta es otra. Hay 15 preguntas, 27 supuestos y 9 riesgos. Las ocho que más mueven la arquitectura son P1, P2, P3, P5, P6, P7, P9 y P11.

2.1 Preguntas y supuesto adoptado

TI significa tecnología de la información, el área de sistemas de AndesRetail.

#	Pregunta	A quién	Supuesto adoptado	Decisión que cambia
P1	¿El OMS orquesta hoy pago y despacho, o solo registra y enruta órdenes?	TI	S1: solo registra y enruta	Dueño de la saga: la capa o el OMS
P2	¿Quién es dueño del stock vendible (físico menos reservado)?	Negocio, bodega	S2: nadie hoy; la capa crea el servicio de disponibilidad y reservas	Modelo de reserva y fuente de verdad
P3	¿ERP y WMS emiten eventos o permiten leer sus cambios?	TI	S3: no emiten eventos; lectura de cambios viable en la base del ERP, sondeo incremental en el WMS	Cómo se alimenta el modelo de lectura y qué frescura se promete
P4	¿Qué volúmenes hay en día normal y en promo; cuántos SKU, tiendas y partners?	Negocio, TI	S4: 200 consultas por segundo en base, 2.000 en pico, 50.000 SKU (stock keeping unit, el código de cada producto vendible), 40 ubicaciones, 10 partners, 20.000 órdenes al día	Dimensionamiento de cache, mensajería y cuotas
P5	¿Cómo es el enlace entre on-prem y AWS (Amazon Web Services)?	Infraestructura	S5: VPN sitio a sitio, sin enlace dedicado	La ruta de las APIs no puede depender del enlace
P6	¿Qué desactualización de stock y precio es aceptable para el negocio?	Negocio	S6: stock, segundos; precio, hasta un minuto. La verdad se verifica al reservar	Frescura objetivo, umbral de `stale`, ventana de tolerancia de precio
P7	¿La pasarela ofrece checkout alojado, tokenización y webhooks firmados?	Seguridad	S7: sí; la capa nunca ve datos de tarjeta	Alcance de PCI DSS (Payment Card Industry Data Security Standard, el estándar de seguridad de la industria de tarjetas; sección 5.5) y diseño de payment-service
P8	¿Dónde viven OMS y CRM y qué APIs exponen?	TI	S8: SaaS o nube, con API REST	Diseño de los adaptadores
P9	¿Los marketplaces aceptan confirmación asíncrona de la orden?	Negocio, partners	S9: sí, respuesta 202 con estado consultable y webhooks	Frontera entre sync y async en crear orden
P10	¿Hay restricciones sobre dónde pueden residir los datos personales?	Legal, seguridad	S10: sin restricción de región; aplica la Ley 1581 de 2012 de Colombia	Región de AWS, cifrado, retención de datos personales
P11	En las órdenes de marketplace, ¿quién cobra: el marketplace o AndesRetail?	Finanzas, negocio	S11: depende del canal; el diseño soporta ambos con `payment.mode` (`CAPTURE` o `PREPAID`)	Si la saga llama o no a la pasarela
P12	¿Qué ventanas de mantenimiento y qué disponibilidad tienen ERP y WMS?	TI	S12: ventanas nocturnas semanales	Tiempo máximo que la capa sirve dato desactualizado y tamaño de los colchones
P13	¿Las tiendas físicas venden del mismo inventario que los canales digitales, y cada cuánto lo reportan al WMS?	Operación de tiendas	S13: sí comparten inventario; el WMS se entera en menos de un minuto	Colchón de seguridad por SKU y canal
P14	¿Cuál es el costo de una sobreventa frente al de una venta perdida?	Negocio	S14: la sobreventa cuesta más (reembolso, reclamo, sanción del marketplace)	Qué tan conservadores son colchones y vencimientos

#	Pregunta	A quién	Supuesto adoptado	Decisión que cambia
P15	Si el precio cambia mientras el cliente paga, ¿cuánto tiempo se respeta el precio que vio, y el ERP distingue una corrección de un cambio comercial?	Negocio, TI	S15: se respeta lo mismo que dura la reserva del canal (15 minutos por defecto); el ERP registra el motivo de cada cambio de lista de precios, y una corrección invalida de inmediato las versiones anteriores	Ventana de tolerancia de precio y campo `correction` de `PriceChanged` (ADR-015)

2.2 Supuestos adicionales, sin pregunta asociada

#	Supuesto
S16	Los partners pueden adoptar OAuth2 Client Credentials en el plazo del MVP
S17	Existe o se puede formar un equipo de plataforma de al menos una persona desde el MVP
S18	Hay un entorno de pruebas de la pasarela de pagos
S19	El `sku` es el mismo identificador en ERP, WMS y OMS; si no, el adaptador mantiene la tabla de equivalencias
S20	La conciliación completa del WMS corre a las 03:00 hora de Colombia (08:00 UTC), fuera de la ventana de mantenimiento del WMS; el código de bodega del WMS es el `locationId` canónico, y si no, el adaptador mantiene la equivalencia
S21	El ERP permite leer su catálogo y su lista de precios completos por páginas (una foto), además de sus cambios: la lectura de cambios solo trae lo posterior a su activación, y el modelo de lectura necesita el estado inicial. El erp-adapter la lee una vez, al arrancar sin cursor guardado
S22	El ERP guarda cada precio como una fila con vigencia (`validFrom` , `validTo`), varias por SKU, canal y lista, como los registros de condición de un ERP: un precio programado es una fila nueva, no la modificación del vigente. Un alcance no tiene más de 10 filas no vencidas a la vez
S23	El ERP no borra filas de precio ni les cambia el `validFrom` : una fila que deja de regir se cierra con `validTo` . Si el ERP borrara una fila, su ventana seguiría publicada hasta vencer; la salida sería identificar cada fila por su número de registro de condición
S24	Las ubicaciones (tiendas y bodegas) son un conjunto acotado, del orden de cientos: por eso `location` puede ser etiqueta de una métrica y el SKU no (ADR-021)
S25	Una suscripción de webhooks se pausa cuando una entrega agota sus 8 intentos y no hubo ninguna entrega exitosa en 24 horas; la reactiva el partner o `ops-admin`
S26	Cada sistema comprado del AS-IS se dibuja en el C4 de contenedores con aplicación y base de datos, sin nombrar tecnología. El lote nocturno WMS a ERP es la integración 11 y el enlace entre la nube y el on-prem es el de la pregunta 5 (VPN sitio a sitio). Cómo llegan los marketplaces al WMS y al ERP no está definido: se dibujan directo
S27	Los identificadores que un partner envía en la orden (`externalOrderId` , SKU, tienda, cliente) caben en 64 caracteres de letras ASCII, dígitos y `.` , `_` , `:` y `-` . Si uno trae espacios, barras u otros caracteres, el partner los traduce antes de llamar, como traduce sus estados (ADR-026)

Los volúmenes de S4 son cifras para dimensionar, no datos del cliente. Todas las decisiones posteriores que dependen de ellos lo dicen.

2.3 Riesgos

#	Riesgo	Probabilidad	Impacto	Mitigación	Señal temprana
R1	La capa acumula lógica de negocio y se vuelve un ESB (Enterprise Service Bus: bus de integración centralizado que termina siendo cuello de botella)	Media	Alto	Gateway delgado por diseño, formato corto de rutas, reglas de dependencias en el pipeline, revisión de arquitectura de todo servicio nuevo	Peticiones de "una transformación pequeña" en el gateway
R2	La capa se vuelve punto único de falla: si cae, caen todos los canales	Baja	Muy alto	Tres zonas, servicios sin estado, degradaciones preparadas, presupuesto de error con alertas	Consumo del presupuesto de error
R3	Dato desactualizado provoca sobreventa	Media	Alto	Reserva atómica como verdad, colchones, frescura medida y con alerta, conciliación nocturna	Deriva de inventario, rechazos después de aceptar
R4	El supuesto S1 es falso y hay doble orquestación con el OMS	Media	Alto	Validarlo en la primera semana; la saga está aislada en un servicio y se puede reducir a un pase directo al OMS	Estados que cambian en el OMS sin evento de la capa
R5	El enlace on-prem o la lectura de cambios del ERP no rinden lo esperado	Media	Medio	Sondeo incremental como respaldo, tópico crudo para reproceso, umbral de `stale`	Frescura creciente en horas pico

#	Riesgo	Probabilidad	Impacto	Mitigación	Señal temprana
R6	Adopción lenta de los partners	Alta	Medio	Entorno de pruebas, documentación generada de los contratos, convivencia con la integración vieja y fecha de retiro acordada	Partners sin tráfico en el entorno de pruebas
R7	La operación de Kafka, la malla y el clúster supera la capacidad del equipo	Media	Medio	Servicios administrados, plantilla de servicio, runbooks, empezar con lo mínimo	Alertas ignoradas, despliegues que se frenan
R8	Convivencia: canales no migrados venden sin reservar	Alta durante la migración	Medio	Colchón mayor temporal, migrar primero los canales de más órdenes	Deriva de inventario concentrada en ciertos SKU
R9	Un precio mal cargado a la baja se sigue vendiendo durante la ventana de pago	Media	Medio	Ventana de tolerancia igual al vencimiento de la reserva, no mayor; el ERP marca las correcciones y una corrección invalida las versiones anteriores (ADR-015)	Órdenes aceptadas con una versión de precio ya reemplazada

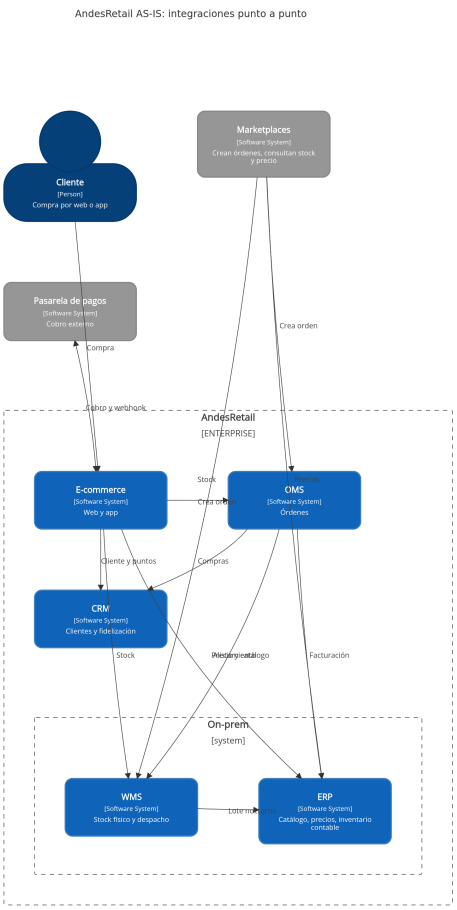
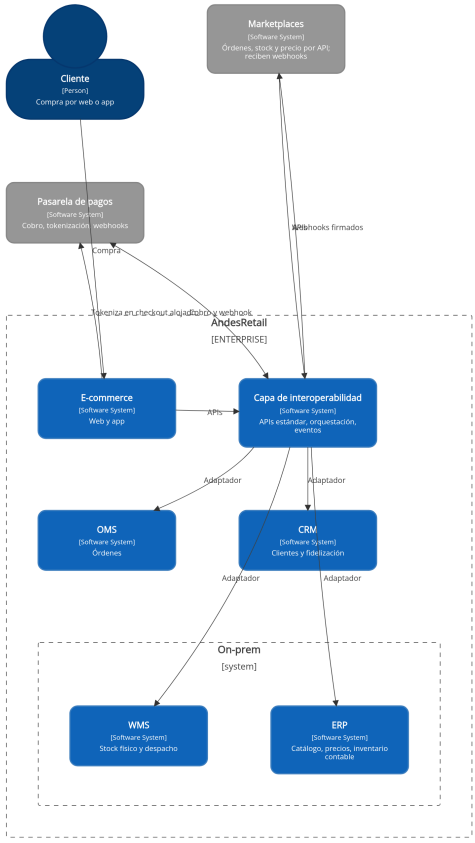
El riesgo R4 merece una línea más: si el OMS ya orquesta pago y despacho, la saga de la capa sobra en parte. El daño está acotado porque la saga vive en un solo servicio con contratos propios; reducirla a un pase directo al OMS no toca a los canales ni a los demás servicios.

3. Arquitectura de alto nivel

Conclusión. La capa se pone entre los canales y los sistemas existentes y cambia tres cosas. Uno: las lecturas de stock y precio salen de un modelo de lectura propio, así que la carga sobre ERP y WMS depende de cuánto cambian los datos y no de cuántos los consultan. Dos: crear orden es una sola saga explícita, con reserva de stock, idempotencia y compensaciones, en lugar de una cadena repetida en cada canal. Tres: cada sistema existente queda detrás de un adaptador; agregar o reemplazar un sistema es agregar o cambiar un adaptador. El precio es más piezas que operar y consistencia eventual declarada en las lecturas.

3.1 Antes y después

Los diagramas usan el modelo C4 (diagramas de arquitectura en cuatro niveles de zoom: contexto, contenedores, componentes y código). Aquí se muestran los tres primeros niveles, sin bajar a clases ni métodos: el contexto de los dos lados (nivel 1, con los sistemas en las mismas posiciones para que se comparen), los contenedores del AS-IS (esta sección) y del TO-BE (sección 3.2), y los componentes de dos piezas del TO-BE (sección 3.2). El AS-IS es una hipótesis de trabajo construida desde los cinco problemas del enunciado.

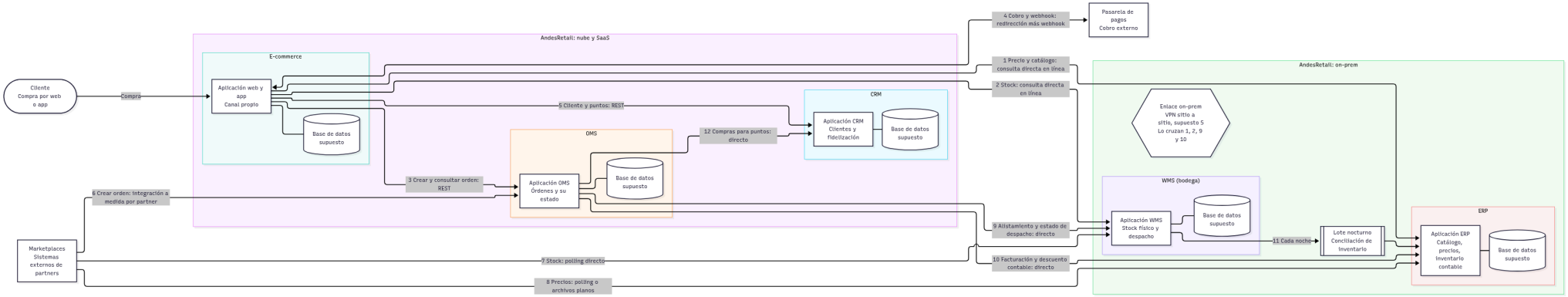
AS-IS: integraciones punto a punto	TO-BE: capa de interoperabilidad
	

	AS-IS	TO-BE
Conexiones de canales a sistemas	8 directas, de 12 integraciones en total	0: todo pasa por la capa
Quién conoce el formato de cada sistema	Todos sus consumidores	Solo su adaptador

	AS-IS	TO-BE
Identidad	Una por integración	Un proveedor de identidad, un token, scopes
Dueño del stock vendible	Nadie	inventory-service
Dueño del flujo de crear orden	Cada canal	order-service

Las integraciones internas entre sistemas (OMS con WMS, OMS con ERP, lote nocturno de WMS a ERP) no se tocan en el MVP; se evalúan en la hoja de ruta (sección 8).

Por dentro, el AS-IS en el nivel 2 del C4 (contenedores):



C4 de contenedores AS-IS

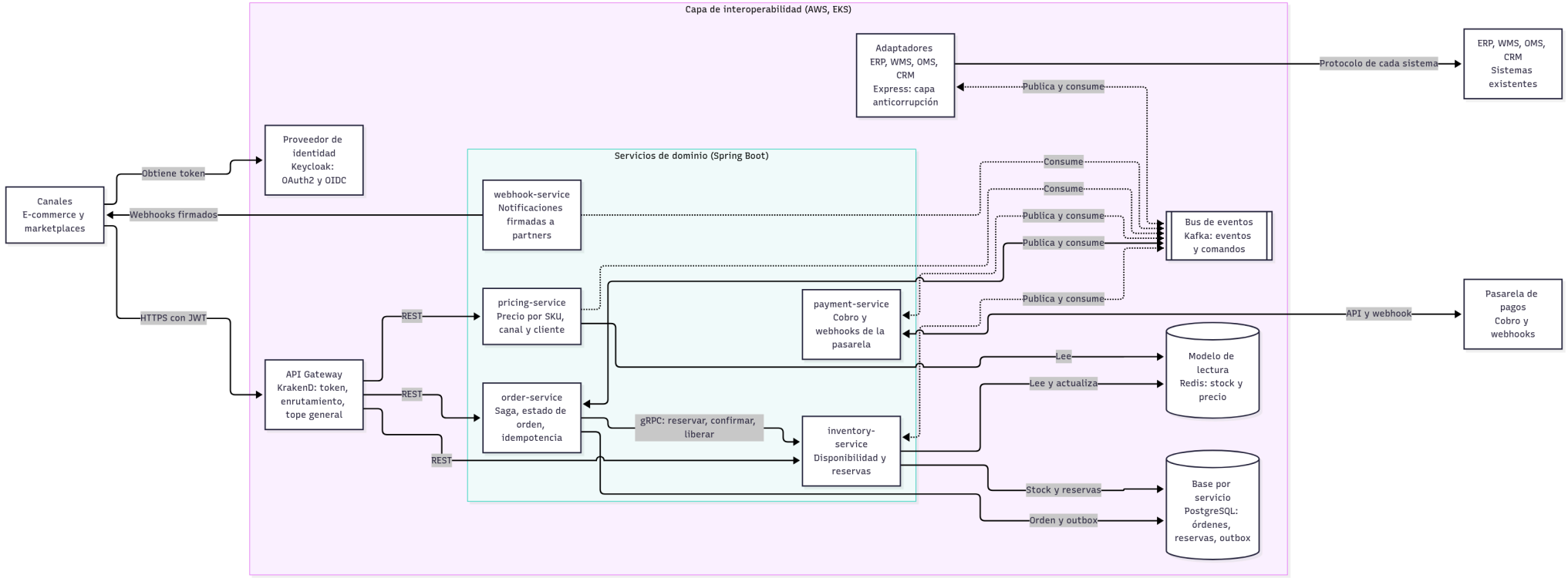
Muestra cada sistema como una aplicación con su base de datos, el lote nocturno que concilia WMS con ERP y el enlace VPN entre la nube y el on-prem, con las 12 integraciones numeradas como en la sección 8.1. Lo que se ve es el problema: seis de las doce integraciones (1, 2, 7, 8, 9 y 10) llegan a las aplicaciones de ERP y WMS, las dos que están on-prem, y cada una usa su propio formato y su propia autenticación. Lo asumido y no dicho por el enunciado, la base de datos de cada sistema y el enlace, está registrado como supuesto S26.

Cada problema del enunciado, su causa y el mecanismo que lo resuelve:

Problema	Causa en el AS-IS	Mecanismo en el TO-BE	Cómo se mide
Integraciones punto a punto frágiles	12 conexiones directas; la coordinación vive repetida en cada canal	Adaptador por sistema, modelo canónico, contratos versionados, saga única	Canales con acceso directo al legado: 0. Tiempo para sumar un sistema nuevo
Inconsistencias en stock, precios y estado	Sin dueño del stock vendible; lote nocturno; webhooks perdidos	Reserva atómica antes de aceptar; modelo de lectura por eventos con frescura medida; outbox y deduplicación	Órdenes rechazadas por sobreventa después de aceptadas: 0. Frescura p95. Órdenes atascadas
Incidentes en picos	Las lecturas de los canales llegan en línea al ERP y al WMS	Lecturas desde Redis; cuotas por partner; adaptadores con límite de concurrencia	p95 bajo 300 ms a 10x. Llamadas al legado planas durante el pico
Poca trazabilidad y auditoría	Ningún identificador cruza los sistemas	`correlationId` y traza distribuida en HTTP, gRPC (protocolo de llamadas remotas, sección 3.3) y eventos; registro de auditoría inmutable	Porcentaje de órdenes con traza completa de punta a punta
Seguridad inconsistente	Una autenticación por integración; sin permisos por canal	OAuth2 y OIDC central, scopes, política por partner, cifrado mutuo interno, secretos gestionados	Rutas sin scope: 0. Credenciales directas al legado en manos de canales: 0

El p95 (percentil 95: el valor bajo el cual queda el 95% de las mediciones) es la forma en que el enunciado fija la latencia, y así se mide aquí.

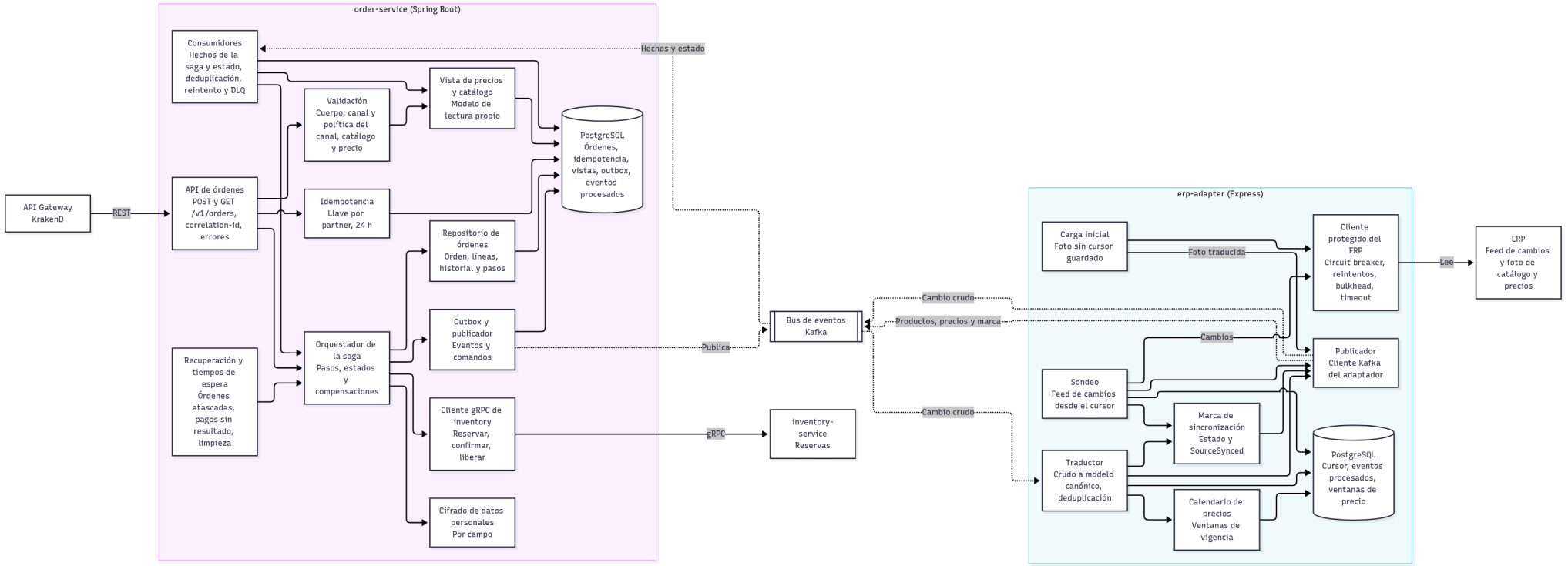
3.2 Contenedores



C4 de contenedores TO-BE

Es el nivel 2 del C4 (contenedores), con sus mismos elementos y relaciones, dibujado como diagrama de flujo para que las flechas se lean. Las líneas punteadas son eventos por Kafka. La capa corre en AWS sobre EKS (Elastic Kubernetes Service, el Kubernetes administrado de AWS), como sugiere el enunciado.

Un nivel más adentro, el nivel 3 del C4 (componentes) de dos de esos contenedores: `order-service`, el que orquesta la saga de crear orden, y `erp-adapter`, el adaptador del ERP. Cada caja es un paquete o un archivo del código; no se llega a clases ni métodos.



C4 de componentes TO-BE

En `order-service`, la API valida idempotencia, cuerpo, canal, catálogo y precio contra su propia vista de precios (alimentada por eventos), reserva stock con una sola llamada gRPC a `inventory-service` y guarda la orden en su base; todo evento y comando sale por el outbox, y los consumidores continúan la saga con los hechos que llegan por Kafka, sin repetir efectos. En `erp-adapter`, el sondeo lee los cambios del ERP a través de un cliente con circuit breaker, los deja crudos en Kafka, y el traductor los pasa al modelo canónico; es el único componente con ruta de red al ERP. Este adaptador no usa outbox: confirma el crudo en Kafka antes de guardar su cursor y sus identificadores de evento son deterministas, así que repetir una lectura no duplica nada.

Componente	Responsabilidad	Lo que no hace
API Gateway (KrakenD)	Valida el token, enruta, aplica tope general y por cliente, propaga el `correlationId`	No transforma, no agrega respuestas, no decide negocio
Proveedor de identidad (Keycloak)	Emite tokens OAuth2 y OIDC; un cliente por partner con sus scopes, canal y nivel de cuota	No guarda datos de clientes finales en el MVP
pricing-service	Modelo de lectura de precios; resuelve lista de precios por canal y segmento de cliente	No calcula precios: los recibe del ERP
inventory-service	Stock vendible por SKU y ubicación; reservas con vencimiento	No reemplaza al WMS: el stock físico sigue siendo del WMS
order-service	Orquesta la saga, guarda el estado de la orden, garantiza idempotencia	No reemplaza al OMS: le entrega las órdenes confirmadas
payment-service	Inicia el cobro con el token de la pasarela, recibe y verifica sus webhooks	Nunca ve ni guarda datos de tarjeta
webhook-service	Entrega eventos a los partners con firma, reintentos y cola de fallidos	No decide qué pasó: solo notifica
Adaptadores ERP, WMS, OMS, CRM	Traducen entre el modelo canónico y cada sistema; protegen al legado con límites de concurrencia	No contienen reglas de negocio
Kafka	Transporte de eventos y comandos; estado reproducible del modelo de lectura	No es base de datos de consulta
Redis	Modelo de lectura de stock y precio para la ruta rápida	No es fuente de verdad de reservas
PostgreSQL	Una base por servicio: órdenes, reservas, outbox, idempotencia	Sin tablas compartidas entre servicios
Malla de servicios (Istio con Envoy)	Cifrado mutuo, reintentos, circuit breaker, cuota por partner, métricas de red	Sin lógica de dominio

Principios que gobiernan todo lo demás:

1. El dato tiene un solo dueño. La capa no reemplaza al ERP, al WMS, al OMS ni al CRM. Solo posee las reservas de stock, porque hoy nadie las tiene.
2. La ruta síncrona de las APIs externas nunca llama a un sistema existente.
3. Gateway delgado: identidad, enrutamiento y límites. Cero lógica de negocio.
4. Modelo canónico adentro; el formato de cada legado solo existe en su adaptador.
5. Todo evento se publica con outbox y se consume de forma idempotente.
6. Las capacidades transversales (telemetría, cifrado entre servicios, reintentos, secretos) las pone la plataforma, no el código de dominio.
7. Cada requisito no funcional tiene una función de aptitud definida, con su estado real: compuerta, prueba ejecutable o diseño (sección 3.8).

3.3 Decisión clave 1: qué es síncrono y qué es asíncrono

La regla: **es síncrono lo que el canal necesita saber antes de seguir y que la capa puede responder con sus propios datos. Es asíncrono todo lo que depende de un sistema que la capa no controla.**

Tramo	Modo	Por qué
Consulta de stock y precio	Sync, contra el modelo de lectura	El canal necesita la respuesta ya; la capa la tiene en Redis sin tocar el legado
Alimentar el modelo de lectura desde ERP y WMS	Async (eventos)	Desacopla el ritmo de lectura del ritmo de cambio; absorbe caídas del enlace on-prem

Tramo	Modo	Por qué
Crear orden: validar token, catálogo y precio	Sync	Errores que el canal puede corregir de inmediato
Crear orden: reservar stock	Sync (gRPC interno)	Evitar sobreventa exige una respuesta atómica antes de aceptar la orden
Crear orden: cobro	Async	Depende de la pasarela, que es externa y responde por webhook
Crear orden: registrar en OMS	Async	El OMS no debe poder tumbar ni frenar la aceptación de órdenes
Consultar estado de orden	Sync, contra la base de order-service	Dato propio de la capa
Notificar a partners	Async (webhooks)	El partner puede estar caído; se reintenta sin bloquear nada
Eventos de despacho y entrega	Async	Nacen en el WMS a su ritmo

gRPC es un protocolo de llamadas remotas con contrato binario estricto; se usa en un único punto interno, de order-service a inventory-service, donde importan la latencia y la exactitud del contrato. Fuera de ese punto, los servicios de dominio no se llaman entre sí: se comunican por eventos.

Dónde no se usa async, y por qué: la reserva podría hacerse por eventos, pero entonces la orden se aceptaría antes de saber si hay stock, que es exactamente la sobreventa del AS-IS. Dónde no se usa sync: el registro en el OMS podría ser una llamada directa, pero haría que una caída del OMS impida vender.

Trade-off: crear orden responde 202 (aceptada) y no el resultado final. El canal debe consultar el estado o recibir el webhook. Se acepta porque el cobro ya es asíncrono por naturaleza (supuesto S9). Consecuencia verificable: con los sistemas existentes apagados, las lecturas siguen respondiendo y las órdenes se siguen aceptando.

3.4 Decisión clave 2: rol del API Gateway

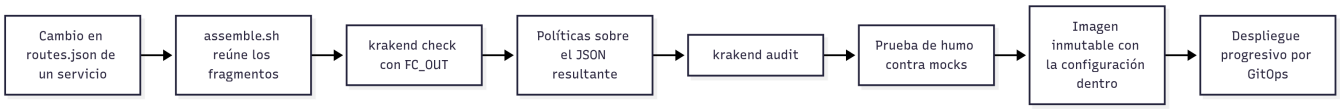
Gateway delgado. Hace: terminar TLS (Transport Layer Security, el cifrado de HTTPS), validar firma y vigencia del token, exigir el scope de la ruta, tope general por endpoint y por cliente, tamaño máximo de petición, generar o propagar `X-Correlation-Id`. No hace: transformar cuerpos, componer respuestas, guardar estado ni aplicar reglas por partner más allá del scope.

Se eligió KrakenD (edición comunitaria) frente a Kong, que el enunciado sugiere como ejemplo:

Capacidad	KrakenD comunitario	Kong código abierto
Validar tokens de Keycloak	Sí, con rotación automática de llaves	Sí, con el plugin JWT (JSON Web Token, sección 5.1); el plugin OIDC completo es de pago
Límites distintos por partner	No: misma tasa para todos los clientes de un endpoint	Sí: el plugin se configura por consumidor
Contadores compartidos entre instancias	No: memoria por instancia	Sí
Operación	Sin estado, configuración declarativa	Requiere base de datos o modo declarativo

Trade-off: las cuotas diferenciadas por partner viven un salto más adentro, en la malla (sección 6.1), y hay dos lugares donde mirar límites. A cambio el gateway es sin estado, se despliega como configuración y no puede convertirse en un ESB. Se revisaría si AndesRetail necesitara portal de desarrolladores, planes de consumo o monetización de APIs.

La configuración del gateway no es un solo archivo que todos tocan. Cada equipo declara sus rutas junto a su servicio en un formato corto (`routes.json`: método, ruta, scope, timeout y grupo de cuota) y el pipeline las ensambla, valida y empaqueta:



El formato corto ni siquiera permite expresar una transformación: la regla "el gateway no tiene lógica" la impone la herramienta, no la buena voluntad. Lo que se sacrifica: los equipos no usan funciones avanzadas del gateway, y hay un ensamblador propio que mantener (la edición comunitaria no admite directorios anidados).

3.5 Decisión clave 3: dónde se transforma y se valida

Lugar	Valida	Transforma
Gateway	Token, scope de la ruta, tamaño, tipo de contenido	Nada
Servicio de dominio	Esquema del contrato, pertenencia del recurso al partner, políticas de canal, reglas de negocio	Contrato externo a modelo canónico
Adaptador	Datos que llegan del legado (rangos, nulos, duplicados)	Modelo canónico a formato del legado y viceversa. Es el único lugar donde existe el formato del legado
Productor de eventos	Esquema del evento antes de publicar	Nada

Trade-off: la validación del esquema ocurre dentro del servicio, un salto después del gateway, así que una petición mal formada consume un poco más de recursos antes de rechazarse. A cambio el gateway no conoce el negocio y cada equipo es dueño de su contrato.

3.6 Decisión clave 4: versionado de APIs

- Versión mayor en la ruta (`/v1`). Solo cambia ante una ruptura.
- Los cambios compatibles (campos nuevos opcionales, endpoints nuevos) no cambian versión. Los clientes deben ignorar campos desconocidos.
- Una ruptura publica `/v2` y `/v1` convive al menos 6 meses, con cabeceras `Deprecation` y `Sunset`.
- Regla de diseño, con su comprobación aún sin implementar (F4, sección 3.8): el pipeline comparará el contrato contra la rama principal y bloqueará rupturas no declaradas, es decir, eliminar o renombrar un campo de respuesta, agregar un campo obligatorio a una petición o cambiar el tipo o el significado de un campo. Hoy el pipeline valida el formato de los contratos (lint) y la comparación se hace en revisión.
- Eventos: mismo criterio. Evolución compatible dentro del tópico; una ruptura crea un tópico `.v2`.

Se descartaron la versión por cabecera o por tipo de contenido (más difícil de probar, enrutar y explicar a un partner) y la versión por fecha (útil con muchos cambios pequeños; aquí sobra). Trade-off: la versión en la ruta invita a duplicar código al llegar `/v2`; se mitiga traduciendo `/v1` a `/v2` en el borde del servicio.

3.7 Disponibilidad 99.9%

99.9% al mes son 43 minutos de indisponibilidad permitida. Consecuencias de diseño:

- Ninguna API externa depende en línea del enlace on-prem ni de ERP, WMS, OMS o CRM.
- Tres zonas de disponibilidad (centros de datos independientes de una misma región de AWS) para EKS, PostgreSQL, Redis y Kafka.
- Si Redis falla, las lecturas caen a la base del servicio con concurrencia limitada.
- Si un legado cae, las lecturas siguen con el último dato y lo marcan como desactualizado; las órdenes se siguen aceptando contra las reservas de la capa.

Trade-off: la capa pasa a ser crítica. Si cae, caen todos los canales a la vez (riesgo R2); por eso las tres zonas y el presupuesto de error con alertas (sección 7).

3.8 Cómo se impide que la arquitectura se degrade

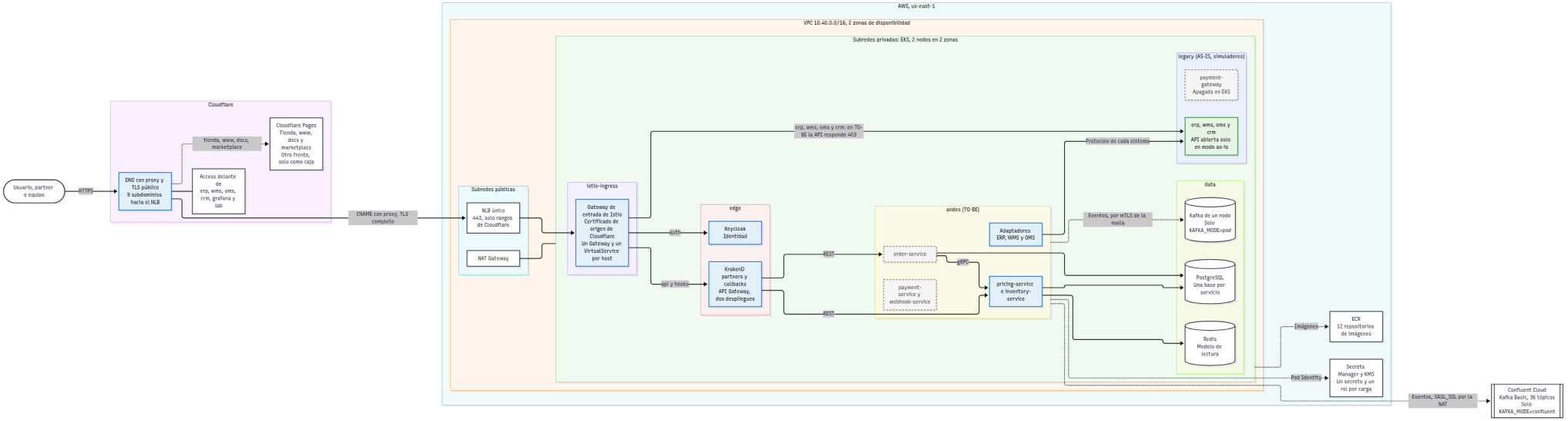
Cada requisito no funcional tiene una función de aptitud (fitness function: prueba automática que protege una característica de la arquitectura mientras el sistema cambia). Son el criterio de aceptación de cada requisito, no documentación. No todas tienen hoy el mismo estado, y la tabla lo dice con tres niveles: **compuerta hoy** (corre en el pipeline en cada cambio, con `make verify`), **prueba ejecutable, sin compuerta** (existe y pasa cuando se corre a mano, y entra al pipeline según el roadmap, sección 8.3) y **diseño, sin implementar** (la función está definida y no hay verificador).

#	Requisito	Función de aptitud	Estado hoy
F1	p95 menor a 300 ms en stock y precio	Prueba de carga con umbral	Prueba ejecutable, sin compuerta: `make f3-e2e`
F2	No tumbar ERP ni WMS	En el pico 10x, las llamadas al legado no superan la línea base más 10%	Prueba ejecutable, sin compuerta: `make f3-e2e`
F3	Agregar sistemas con mínimo impacto	Regla de dependencias: solo los adaptadores importan clientes del legado; los servicios de dominio no se llaman por REST entre sí	Diseño, sin implementar: ninguna herramienta la comprueba. Se verificó a mano que hoy se cumple (los únicos clientes HTTP o gRPC de los servicios son el de la pasarela, el de entrega a partners y el de order a inventory)
F4	Evolución y versionado	Comparación de contratos que bloquea rupturas	Diseño, sin implementar. Hoy `make verify` solo valida el formato de los tres contratos (lint), no compara contra la rama principal
F5	Seguridad consistente	Ninguna ruta del gateway sin scope declarado	Compuerta hoy: `gateway-verify`, dentro de `make verify`
F6	Auditoría de punta a punta	Una orden de prueba deja su `correlationId` en cada servicio y en cada evento	Prueba ejecutable, sin compuerta: `make f4-e2e` comprueba los logs de los cinco componentes; los eventos y la auditoría no se comprueban de punta a punta
F7	Multi-tenant	Un partner no puede leer la orden de otro; un partner que excede su cuota no afecta a otro	Prueba ejecutable, sin compuerta: `make f4-e2e` (aislamiento de órdenes) y `make mesh-kind-test` (cuota)
F8	Manejo de datos personales	Ningún dato personal de la orden de prueba aparece en logs, eventos ni trazas	Prueba ejecutable, sin compuerta: `make f4-e2e` revisa los logs de los servicios y del borde; eventos y trazas no se revisan automáticamente
F9	Idempotencia	La misma orden enviada 5 veces en paralelo crea una sola orden y una sola reserva	Prueba ejecutable, sin compuerta: `make f4-e2e`
F10	Resiliencia	Con el ERP caído, stock y precio responden 200	Prueba ejecutable, sin compuerta: `make f3-e2e`

El pipeline de hoy corre `make verify` (formato de contratos, gateway, secretos, redacción, versiones, infraestructura y malla sin clúster), `make smoke` y `make edge-test`. Las pruebas de comportamiento de F1, F2 y F6 a F10 necesitan todas las piezas corriendo y por eso se ejecutan aparte; llevarlas al pipeline es parte del roadmap (sección 8.3), igual que construir F3 y F4.

Trade-off: cada función de aptitud es código que mantener y tiempo de pipeline. Por eso son pocas, concretas y ejecutables.

3.9 Despliegue y red del demo en AWS



Despliegue y red del demo en AWS

El demo corre en AWS (región `us-east-1`) detrás de Cloudflare. Cloudflare pone el DNS con proxy, el TLS público y Cloudflare Access (control de acceso por correo) delante de `erp`, `wms`, `oms`, `crm`, `grafana` y `lab`; `api`, `hooks` y `auth` quedan abiertos porque los protege el token. Un solo balanceador de red (NLB, Network Load Balancer, el balanceador de AWS) acepta únicamente los rangos de Cloudflare en el puerto 443 y entrega al gateway de entrada de Istio, que enruta por host con un Gateway y un VirtualService de Istio por subdominio (`networking.istio.io/v1`; no se usa la Gateway API de Kubernetes en este demo, ADR-030): `api` y `hooks` a KrakenD, `auth` a Keycloak, y `erp`, `wms`, `oms` y `crm` directo a los simuladores. Dentro del clúster (EKS, dos nodos en dos zonas, subredes privadas) hay cinco espacios de nombres: `istio-ingress`, `edge`, `andes` (la capa, TO-BE), `legacy` (los sistemas de hoy, simulados, AS-IS) y `data` (PostgreSQL y Redis). La única llamada síncrona interna es el gRPC de `order-service` a `inventory-service`. Kafka depende de un interruptor (ADR-028): en un pod de `data` para desarrollo, ensayos y respaldo, o Confluent Cloud, fuera del clúster, al que las cargas llegan por la NAT (Network Address Translation: la salida a internet de las subredes privadas). Cada carga lee su secreto de Secrets Manager con un rol de Pod Identity; las imágenes salen de ECR (Elastic Container Registry, el registro de imágenes de AWS). La tienda, `www`, `docs` y `marketplace` están en Cloudflare Pages y son de otro frente; aquí solo aparecen como caja.

La tienda de demostración, el canal de comercio electrónico del demo, está publicada en `andesretail.com` (un Worker de Cloudflare), con un aviso de que es una demostración. Quedó conectada a la capa en EKS el 15/11/2026: sus lecturas de precio y stock salen de la capa, con la hora del dato y el aviso de dato desactualizado cuando el stock llega `stale`. La compra de punta a punta en el dominio publicado, con la tarjeta de prueba de Stripe, queda por probar por Andrés; la tienda no guarda datos de pago, que van a la pasarela.

Lo AS-IS y lo TO-BE conviven en el mismo clúster: los subdominios de los sistemas tienen su ruta de API abierta solo en modo `as-is`; en modo `to-be` responde 403 y solo su adaptador la llama, por dentro. La API de Kubernetes del clúster no está en internet: el endpoint es solo privado y se entra por un túnel (sección 5.8, ADR-027).

Kafka para los ensayos: el ADR-028, ampliado el 15/11/2026 por el ADR-034: Kafka en pod y Confluent Cloud quedan los dos arriba y un comando cambia a cuál se conectan las cargas (195 s de pod a Confluent y 28 s de vuelta, medidos). La capa corre en Confluent Cloud Basic (una zona, `us-east-1`, 36 tópicos) desde el 15/11/2026. La decisión original del 14/11/2026 es correr Kafka en un pod dentro del clúster para desarrollo y pruebas, y usar Confluent Cloud del 15 al 17/11, hasta la presentación, con el pod de respaldo. El cambio es un interruptor de configuración (`KAFKA_MODE`, `pod` o `confluent`), sin tocar código ni contratos; el diagrama muestra los dos modos. Registro: ADR-028 (anexo D).

Estado real al 15/11/2026. En AWS está aplicada la base de la cuenta (secretos, repositorios de imágenes, llaves de KMS y presupuestos), que cuesta 0,29 dólares por día y queda siempre. El 14/11/2026 el clúster se encendió una sola vez, de 17:07 a 19:32 UTC (2 h 25 min), para desplegar los simuladores del legado y medir el AS-IS en la nube (sección 6.8); se apagó con `make infra-off` y `make infra-audit` confirmó que no queda clúster, nodos, NAT, balanceador, discos ni IP elásticas del demo. El 15/11/2026 el clúster se volvió a encender con la capa TO-BE completa, la malla de Istio, el DNS (Cloudflare en modo Full strict y Access con PIN) y Kafka en Confluent Cloud, y se midieron en EKS los seis escenarios (sección 6.8). Además, desde el 15/11/2026 el despliegue a EKS (`make k8s-eks-deploy`) aplica la malla con sus salidas declaradas hacia Kafka en Confluent Cloud, la pasarela de pagos, KMS y el agente de Pod Identity, que las políticas de salida restringida (`REGISTRY_ONLY`) bloquearían sin declarar; el ingress de `auth` ya no deja alcanzar la consola de administración de Keycloak. La prueba del borde en EKS (`EDGE_ENV=eks node tests/k8s/edge-e2e.mjs`) dio 20 de 20 contra Confluent: token por `auth`, stock, precio y orden por `api`, y la orden llega a `CONFIRMED` por la saga. Defecto abierto a esa fecha: los consumidores de `wms-adapter` y `oms-adapter` se crean sin SASL_SSL, así que con Confluent el stock sale marcado `stale` y la orden se confirma sin registrarse en el OMS; con Kafka en pod no ocurre (ADR-034). Las piezas con borde discontinuo en el diagrama son diseño y no corren todavía.

4. Diseño de la capa de interoperabilidad

Conclusión. La capa se organiza por quién es dueño de cada dato. Cinco servicios de dominio cubren las preguntas que hoy nadie responde bien (¿a cuánto?, ¿cuánto puedo prometer?, ¿en qué va la orden?, ¿se cobró?, ¿a quién se le avisó?) y cuatro adaptadores aíslan a los sistemas existentes como capa anticorrupción. Los contratos son la fuente de verdad: OpenAPI para las APIs, AsyncAPI para los eventos y Protocol Buffers para gRPC. Crear orden tiene dos barreras de idempotencia y un modelo de errores único.

4.1 Bounded contexts

Un bounded context es la frontera dentro de la cual un modelo y su vocabulario significan una sola cosa; aquí define qué servicio es dueño de qué.

Contexto	Pregunta que responde	Dueño del dato	Servicio de la capa
Catálogo y precios	¿Qué se vende y a cuánto, para este canal y cliente?	ERP	pricing-service (modelo de lectura)
Disponibilidad	¿Cuánto puedo prometer?	WMS (físico), capa (reservas)	inventory-service
Órdenes	¿En qué va esta orden?	OMS (registro), capa (saga)	order-service
Pagos	¿Se cobró?	Pasarela	payment-service
Despacho	¿Salió, llegó?	WMS	Sin servicio propio: eventos del adaptador WMS
Clientes	¿Quién es y qué segmento tiene?	CRM	Sin servicio propio en el MVP: adaptador CRM
Notificaciones a partners	¿A quién se le avisa y se le avisó?	Capa	webhook-service
Identidad y acceso de partners	¿Quién llama y qué puede hacer?	Capa	Keycloak

4.2 Servicios y responsabilidades

Servicio	Expone	Consume y publica	Guarda
pricing-service	`GET /v1/prices`	Consume productos y precios del ERP y el segmento del cliente; entrega `priceVersion`, que la orden usa para validar el precio	Productos, precios, segmentos por `customerId` opaco, eventos procesados
inventory-service	`GET /v1/stock`; gRPC interno `Reserve`, `Confirm`, `Release`, `GetReservation`	Consume stock físico del WMS y despachos; publica eventos de reserva	Niveles de stock, reservas, outbox, eventos procesados
order-service	`POST /v1/orders`, `GET /v1/orders/{orderId}`	Orquesta la saga; publica eventos de orden y comandos a pagos y OMS	Órdenes, pasos de la saga, llaves de idempotencia, outbox
payment-service	Sin API para partners; recibe los webhooks de la pasarela en una ruta dedicada	Consume comandos de cobro; publica resultados de pago	Referencia de la pasarela, estado, monto y moneda. Nunca datos de tarjeta
webhook-service	`POST/GET/DELETE /v1/webhooks/subscriptions`	Consume eventos de órdenes, pagos y despacho; entrega a cada partner con firma	Suscripciones, entregas e intentos

Comunicación: del gateway a los servicios, REST; de order-service a inventory-service, gRPC; todo lo demás, Kafka. No hay llamadas REST entre servicios de dominio. Una base de datos por servicio, sin tablas compartidas.

Lo que deliberadamente no se construye, y por qué:

- Un servicio de clientes: el CRM es el dueño; la capa solo guarda el segmento.
- Un motor de precios o de promociones: el ERP calcula, la capa distribuye.
- Un reemplazo del OMS: la capa orquesta la aceptación y el cobro; el ciclo de vida posterior sigue en el OMS.

- Un bus con transformaciones centralizadas: la traducción vive en cada adaptador.

4.3 Patrón de integración: adaptadores como capa anticorrupción

Cada sistema existente queda detrás de un adaptador que actúa como ACL (anti-corruption layer, capa anticorrupción): traduce entre el modelo canónico y el del sistema, de modo que el modelo del legado no se filtre hacia adentro.

Adaptador	Del sistema hacia la capa	De la capa hacia el sistema
erp-adapter	CDC (change data capture: leer los cambios desde el log de la base de datos, sin consultar al sistema) sobre la base del ERP: productos y precios	Ninguna en el MVP
wms-adapter	Sondeo incremental de niveles de stock y de estados de despacho: publica `DispatchStarted` y `DeliveryConfirmed` por outbox, y de ahí salen el despacho y la entrega que ve el partner	Ninguna en el MVP (el OMS sigue enviando el alistamiento al WMS)
oms-adapter	Confirmación de orden registrada	Crear orden confirmada, cancelar orden
crm-adapter	Segmento del cliente (después del MVP)	Notificar compra para puntos (después del MVP)

Reglas comunes a todo adaptador:

1. Es el único componente con credenciales y ruta de red hacia su sistema.
2. Limita su concurrencia y su tasa de llamadas, con circuit breaker, timeout y reintentos solo en operaciones idempotentes (sección 6.3).
3. Publica en dos pasos: tópico crudo (`raw.<sistema>.*`) y tópico canónico. El crudo permite reprocesar si cambia la traducción.
4. No guarda estado de negocio; solo cursores de lectura y tabla de deduplicación.
5. Agregar un sistema nuevo es crear un adaptador desde la plantilla; los servicios de dominio no cambian.

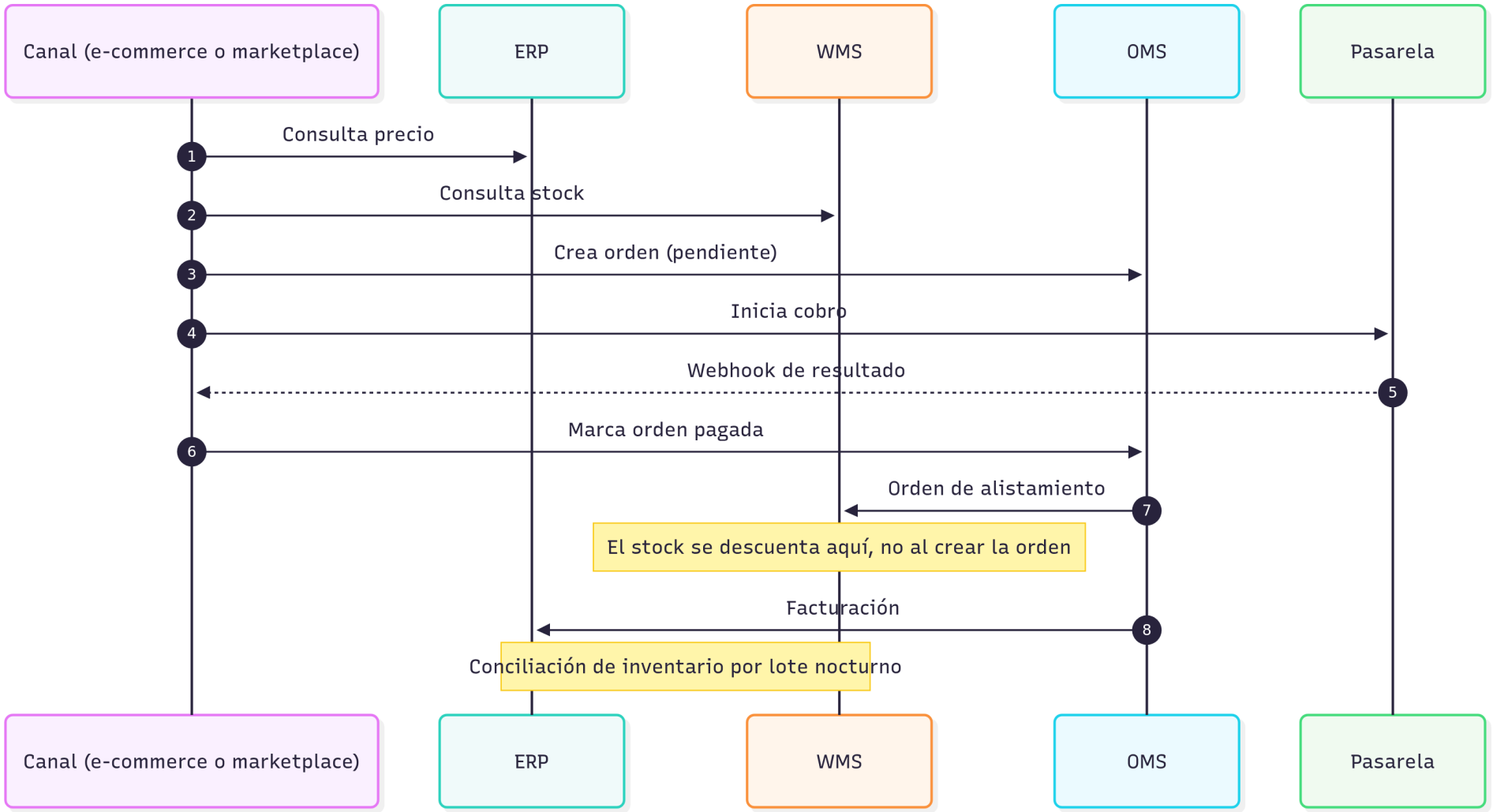
Trade-off: un adaptador por sistema es más código que un integrador central, y una traducción que cambia se toca en un solo lugar pero se debe desplegar. Se acepta porque el costo de un cambio en el legado queda confinado a una pieza.

Modelo canónico, el vocabulario único dentro de la capa: `sku`; `locationId` (tienda o bodega); `channel`; `partnerId`; `customerId` opaco, nunca nombre ni documento en eventos; `orderId` con prefijo `ord_`; dinero como decimal en texto más moneda ISO 4217 (el estándar de códigos de moneda, por ejemplo `COP`), nunca punto flotante; tiempos en ISO 8601 (el estándar de fechas) en UTC (tiempo universal coordinado).

Eventos: todos usan el sobre CloudEvents 1.0, una especificación abierta que estandariza los metadatos de un evento (`id`, `source`, `type`, `subject`, `time`) más tres extensiones propias: `correlationid`, `traceparent` y `partnerid`. La clave de partición es el agregado (`orderId` o `sku`), así que los eventos de una orden llegan en orden. Entrega al menos una vez y deduplicación por `id` en cada consumidor. Los eventos no llevan datos personales: si un consumidor necesita la dirección, la pide al dueño con el `orderId`. El catálogo de tópicos está en el anexo C.

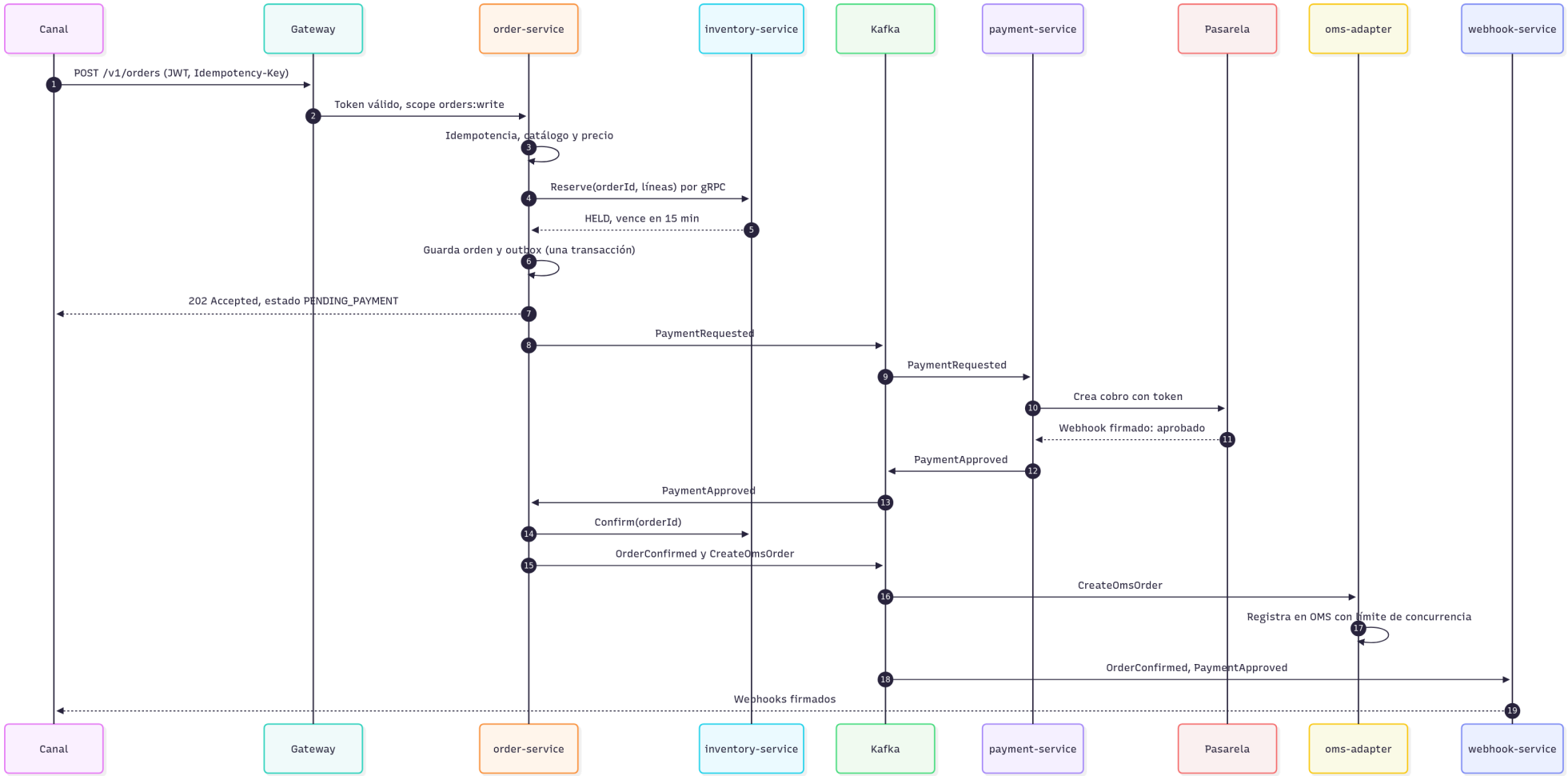
4.4 Flujo A: crear orden, antes y después

En el AS-IS no hay saga: hay una cadena de llamadas síncronas dirigida desde el canal, sin compensaciones. El stock baja en el alistamiento (paso 7), no al vender; si se pierde el webhook de pago la orden queda pagada y pendiente; si el canal reintenta, se duplica la orden. Y cada canal tiene su propia versión de la cadena: eso explica la fragilidad mejor que la cantidad de conexiones.



Flujo A en el AS-IS

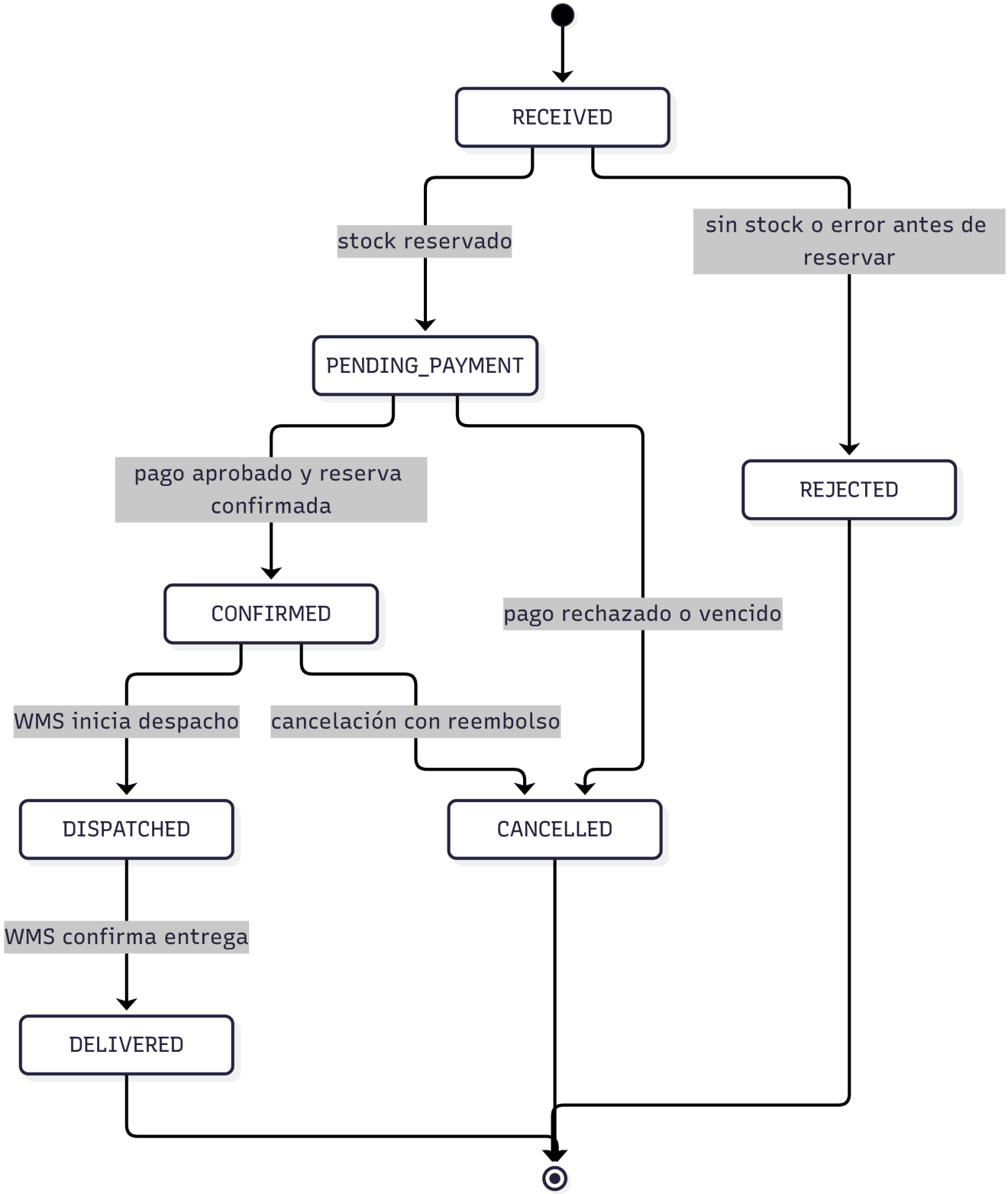
En el TO-BE la saga es orquestada por order-service: un solo lugar conoce los pasos, el estado y las compensaciones.



Flujo A en el TO-BE

#	Paso	Modo	Tipo	Si falla
1	Validar token, scope y canal	Sync	Validación	401 o 403
2	Idempotencia	Sync	Validación	Devuelve la respuesta original, o 409 o 422
3	Validar catálogo y precio contra el modelo de lectura	Sync	Validación	422 `UNKNOWN_SKU` o `PRICE_MISMATCH`
4	Reservar stock	Sync, gRPC	Compensable (liberar)	409 `INSUFFICIENT_STOCK`
5	Guardar orden y responder 202	Sync	Local	La reserva vence sola
6	Cobrar	Async	Pivote	Liberar reserva, cancelar orden
7	Confirmar reserva	Async	Reintentable	Reintento; si la reserva venció y no hay stock, reembolso
8	Registrar en OMS	Async	Reintentable	Reintento con espera creciente y alerta; no se compensa
9	Notificar al partner	Async	Reintentable	Reintentos y cola de fallidos

El cobro es el pivote: antes de él todo se puede deshacer; después de él todo se reintenta hasta lograrlo, porque el cliente ya pagó. Si el pago se rechaza, order-service libera la reserva, marca la orden `CANCELLED` y el partner recibe `payment.rejected` y `order.cancelled`. Si la respuesta del cobro se pierde, payment-service no lo da por no cobrado: lo busca en la pasarela por los metadatos de la orden antes de anularlo, y si aparece aprobado después de anular, lo reembolsa. Una tarjeta rechazada sale como cobro fallido con su referencia.



El canal ve `PENDING_PAYMENT`, `CONFIRMED`, `CANCELLED`, `DISPATCHED` y `DELIVERED`; `RECEIVED` y `REJECTED` son internos. El registro en el OMS se sigue aparte con el campo `omsSync`.

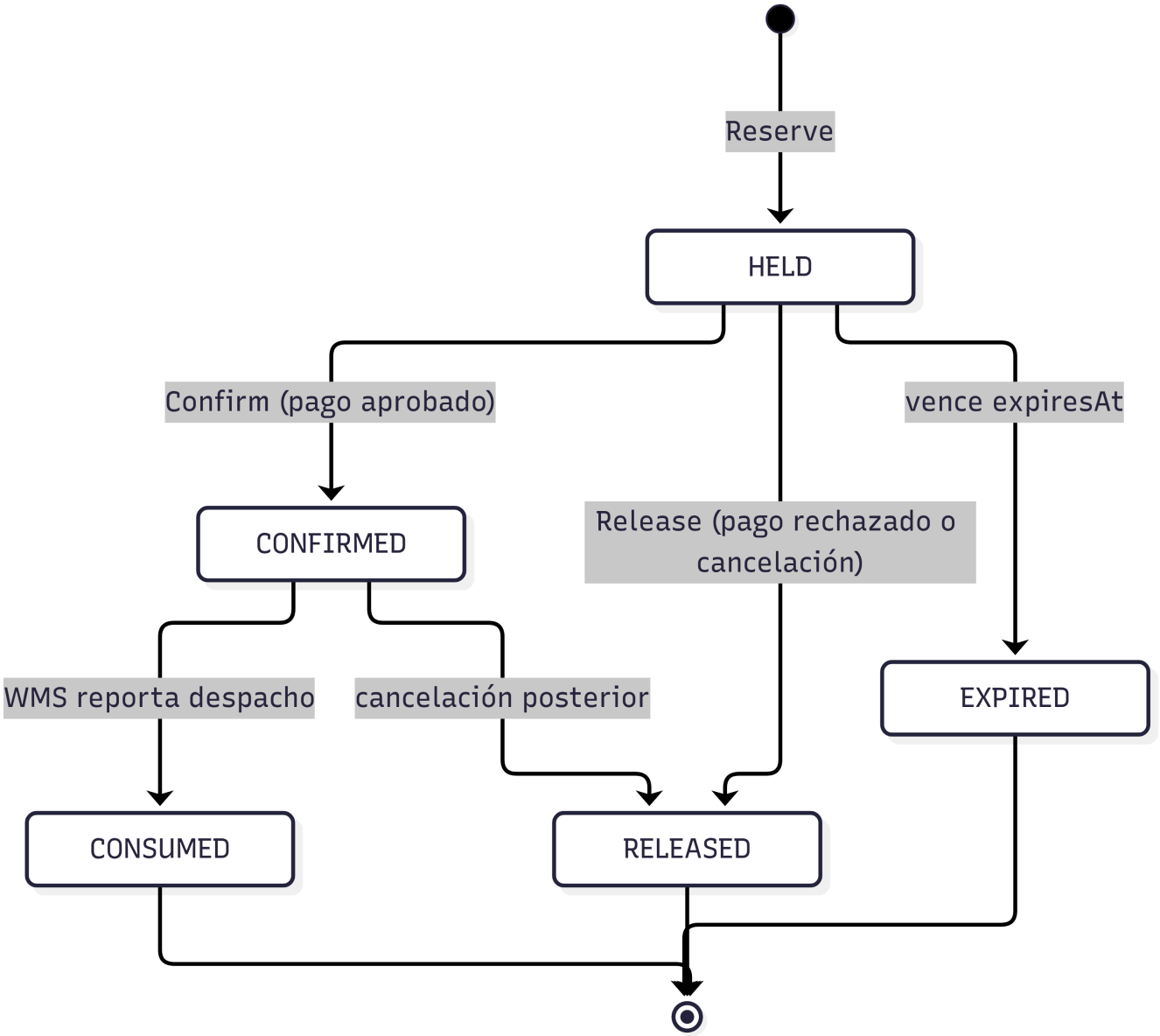
Hueco conocido del demo: el OMS necesita la dirección de entrega para despachar, y esa dirección vive solo en order-service, cifrada, sin salir a Kafka (sección 5.5). Hoy la orden se registra en el OMS sin dirección. La salida es el ADR-025 (anexo D), aceptado el 15/11/2026 y sin implementar: oms-adapter pide la dirección a order-service por gRPC con el `orderId`, como un ticket de reclamo, sin datos personales en el tópico. Costará una segunda llamada síncrona interna, es decir, una excepción explícita a la regla de la sección 3.3. Hasta que se implemente, la orden sale al OMS sin dirección de entrega.

Se descartó la coreografía (cada servicio reacciona a eventos sin coordinador) porque con cinco participantes nadie puede responder en qué va una orden, que es justo el problema del AS-IS. Se descartó un motor de flujos (Temporal o Step Functions) porque es otra pieza que operar para un flujo de nueve pasos. Trade-off: order-service conoce a los demás participantes y es el punto de mayor acoplamiento lógico, y el manejo de estado y recuperación es código propio. Se revisaría al sumar la tercera saga (devoluciones, cancelaciones), o si el supuesto S1 resulta falso.

Stock vendible y reservas. inventory-service calcula el stock vendible, lo que la industria llama ATP (available to promise: lo que se puede prometer sin riesgo de incumplir): físico según el WMS, menos lo reservado por la capa, menos un colchón de seguridad. La reserva es una actualización condicional atómica en PostgreSQL:

```
UPDATE stock_level
   SET reserved = reserved + :qty, version = version + 1
 WHERE sku = :sku AND location_id = :loc
   AND on_hand - reserved - safety_buffer >= :qty;
```

Si no actualiza fila, no hay stock y la orden se rechaza con 409. Una orden con varias líneas reserva todas en una transacción, ordenando las filas por SKU para evitar interbloqueos. Redis no participa en esta decisión. La reserva vence a los 15 minutos (configurable por canal), lo que también es la red de seguridad de la saga: si order-service falla después de reservar, la reserva se libera sola.



Límites honestos: la garantía es total entre canales que pasan por la capa y parcial frente a ventas que no pasan por ella (tiendas físicas, canales aún no migrados). Se cubre con el colchón de seguridad por SKU y canal, la frescura del WMS medida y con alerta, y la conciliación nocturna. Despacho: el físico y la reserva bajan por tópicos distintos. El descuento del físico (feed de stock del WMS, `StockLevelChanged` en `andes.inventory.stock-levels.v1`) y el consumo de la reserva (`DispatchStarted` en `andes.fulfillment.events.v1`) no tienen orden entre sí. Solo un orden es peligroso: si el despacho llega primero, el vendible sube por unos segundos lo despachado y se puede prometer una unidad que ya salió. Si el físico llega primero, el vendible baja de más por unos segundos, que es el lado seguro. Duración de la ventana: por medir (`make f4-e2e` comprueba que el vendible vuelve a su valor en menos de 30 s, no mide la ventana). En el MVP lo cubre el colchón de seguridad. Salida para producción, sin implementar: que el consumo de la reserva espere a que el nivel de stock de ese SKU y ubicación tenga `wmsAsOf` igual o posterior al `occurredAt` del despacho. Trade-off: la capa pasa a tener un dato de negocio propio, la única excepción al principio de no poseer datos. Se justifica porque no existe dueño hoy y porque reservar en el WMS metería al legado on-prem en la ruta síncrona de crear orden.

4.5 Contrato 1: consultar stock

`GET /v1/stock`, scope `stock:read`. Acepta hasta 50 SKU por llamada. El canal sale del token; el parámetro `channel` solo se usa si el partner tiene varios, y debe ser uno de los suyos.

```
GET /v1/stock?sku=SKU-1001&sku=SKU-2002&storeId=BOG-01 HTTP/1.1
Host: api.andesretail.com
Authorization: Bearer eyJhbGciOi...
X-Correlation-Id: 7f3c2a9e-51d4-4f0b-9a57-2c1e8b6d4a10
```

```
HTTP/1.1 200 OK
Content-Type: application/json
X-Correlation-Id: 7f3c2a9e-51d4-4f0b-9a57-2c1e8b6d4a10
RateLimit-Limit: 500
RateLimit-Remaining: 97
Cache-Control: private, max-age=2

{
  "items": [
    {
      "sku": "SKU-1001",
      "storeId": "BOG-01",
      "channel": "mkp-andino",
      "available": 42,
      "status": "IN_STOCK",
      "asOf": "2026-11-10T18:04:11Z",
      "stale": false
    },
    {
      "sku": "SKU-2002",
      "storeId": "BOG-01",
      "channel": "mkp-andino",
      "available": 0,
      "status": "OUT_OF_STOCK",
      "asOf": "2026-11-10T18:04:09Z",
      "stale": false
    }
  ],
  "notFound": []
}
```

`available` ya descuenta reservas, colchón y el tope visible del canal: no se exponen el físico ni lo reservado. El mensaje del `409 INSUFFICIENT_STOCK` de crear orden ("Disponible: N") aplica el mismo tope visible que la consulta de stock (corregido el 15/11/2026). `asOf` dice de cuándo es el dato y `stale` avisa si superó el umbral de desactualización (sección 6.2).

4.6 Contrato 2: crear orden

`POST /v1/orders`, scope `orders:write`. La cabecera `Idempotency-Key` es obligatoria y es un UUID (identificador único universal) que genera el cliente.

```
POST /v1/orders HTTP/1.1
Host: api.andesretail.com
Authorization: Bearer eyJhbGciOi...
Idempotency-Key: 0b8f6f0e-6a1b-4d5c-9c59-0d6f3f0a2b77
Content-Type: application/json

{
  "externalOrderId": "MKP-778899",
  "customer": {
    "customerId": "cus_8f3a21",
    "name": "Laura Gómez",
    "email": "laura@example.com",
    "phone": "+573001234567"
  },
  "fulfillment": {
    "type": "DELIVERY",
    "storeId": "BOG-01",
    "address": {
      "line1": "Calle 10 # 5-20",
      "city": "Bogotá",
      "region": "DC",
      "country": "CO"
    }
  },
  "lines": [
    {
      "sku": "SKU-1001",
      "quantity": 2,
      "unitPrice": "159900.00",
      "priceVersion": "pv_01JC8Z4T6M"
    }
  ],
  "payment": {
    "mode": "CAPTURE",
    "method": "CARD",
    "token": "tok_4f9a7c2e",
    "amount": "319800.00",
    "currency": "COP"
  }
}
```

```
HTTP/1.1 202 Accepted
Content-Type: application/json
Location: /v1/orders/ord_01JC8ZB7Q2X9
X-Correlation-Id: 5e1d0c7a-93b2-4c41-8d0e-6a7f2b9c3d15

{
  "orderId": "ord_01JC8ZB7Q2X9",
  "externalOrderId": "MKP-778899",
  "channel": "mkp-andino",
  "status": "PENDING_PAYMENT",
  "reservation": { "expiresAt": "2026-11-10T18:19:30Z" },
  "total": { "amount": "319800.00", "currency": "COP" },
  "createdAt": "2026-11-10T18:04:30Z",
  "updatedAt": "2026-11-10T18:04:30Z",
  "links": { "self": "/v1/orders/ord_01JC8ZB7Q2X9" }
}
```

202 significa: la orden está aceptada y el stock reservado; el cobro sigue en curso. El resultado llega por webhook y se consulta en `links.self`. `payment.token` es el token que entrega la pasarela. Los identificadores (`externalOrderId`, `channel`, `customer.customerId`, `fulfillment.storeId`, `lines[].sku` y `lines[].priceVersion`) tienen formato declarado en el contrato: de 1 a 64 caracteres, letras ASCII, dígitos y `.`, `_`, `:` y `-`, empezando por letra o dígito (supuesto S27); uno fuera del formato recibe `400 VALIDATION_ERROR` con el campo en `errors[].field`. El texto libre (nombre, correo, dirección) y `payment.token` pasan por un detector que rechaza un número de tarjeta enviado por error: 13 a 19 dígitos que cierren con el dígito de verificación de las tarjetas (algoritmo de Luhn). Los identificadores quedan fuera del detector a propósito (sección 5.5, ADR-026). La capa es la autoridad del precio: acepta `unitPrice` si coincide con el vigente o con uno reemplazado dentro de la ventana de pago del canal, que es la misma duración de la reserva (15 minutos por defecto). La ventana es una política comercial: protege al cliente que vio un precio y está pagando. Un cambio marcado como corrección en el ERP invalida de inmediato las versiones anteriores, para no seguir vendiendo un precio mal cargado. order-service valida contra su propia vista de precios, alimentada por el tópico de precios (ADR-015). Si no, responde `422 PRICE_MISMATCH` con el precio vigente.

4.7 Modelo de errores

Un solo formato para toda la API: Problem Details (RFC 9457; un RFC, Request for Comments, es un documento de estándar del IETF, el organismo que publica los estándares de internet, para describir errores HTTP), con tipo de contenido `application/problem+json` y tres campos propios: `code`, `correlationId` y `errors`.

```
{
  "type": "https://api.andesretail.com/problems/insufficient-stock",
  "title": "Stock insuficiente",
  "status": 409,
  "code": "INSUFFICIENT_STOCK",
  "detail": "No hay unidades suficientes para una o más líneas.",
  "instance": "/v1/orders",
  "correlationId": "5e1d0c7a-93b2-4c41-8d0e-6a7f2b9c3d15",
  "errors": [
    { "field": "lines[0].quantity", "code": "INSUFFICIENT_STOCK", "message": "Disponible: 1" }
  ]
}
```

HTTP	code	Cuándo	¿Reintentar?
400	`VALIDATION_ERROR`	El cuerpo o los parámetros no cumplen el contrato	No
401	`UNAUTHENTICATED`	Token ausente, vencido o inválido	Con token nuevo
403	`INSUFFICIENT_SCOPE`	Falta el scope de la ruta	No
403	`CHANNEL_NOT_ALLOWED`	Canal, tienda o modo de cobro no permitidos al partner	No
404	`NOT_FOUND`	El recurso no existe o es de otro partner	No
409	`INSUFFICIENT_STOCK`	No se pudo reservar	No, o con otra cantidad
409	`REQUEST_IN_PROGRESS`	Misma llave de idempotencia aún en proceso	Sí, tras `Retry-After`
413	`PAYLOAD_TOO_LARGE`	El cuerpo supera el tope de la ruta; lo responde el gateway de entrada en las operaciones con cuerpo	No, con un cuerpo más pequeño
422	`UNKNOWN_SKU`	SKU inexistente o no habilitado para el canal	No
422	`PRICE_MISMATCH`	El precio enviado no coincide con el vigente	Con el precio vigente
422	`IDEMPOTENCY_KEY_REUSED`	Misma llave con cuerpo distinto	No
429	`RATE_LIMITED`	Cuota del partner agotada	Sí, tras `Retry-After`
500	`INTERNAL_ERROR`	Falla no prevista de la capa; también cualquier otro 5xx que el borde normaliza	Sí, con la misma llave de idempotencia
503	`DEPENDENCY_UNAVAILABLE`	La capa no puede responder con seguridad	Sí, tras `Retry-After`

Reglas: un recurso de otro partner responde 404, no 403, para no revelar que existe. Los errores 5xx nunca incluyen detalles internos. El `correlationId` permite a soporte encontrar la traza completa. Trade-off frente a un formato propio: el estándar obliga a convivir con sus campos, pero los partners ya tienen herramientas para él.

4.8 Idempotencia en crear orden

Dos barreras, ambas con restricción única en la base:

1. `Idempotency-Key`: clave `partnerId` más la llave; se guarda con el hash del cuerpo y la respuesta, por 24 horas.
2. Llave natural: `partnerId` más `externalOrderId`.

Situación	Respuesta
Misma llave, mismo cuerpo, petición ya terminada	La respuesta original, con `Idempotent-Replayed: true`
Misma llave, mismo cuerpo, petición en curso	`409 REQUEST_IN_PROGRESS`
Misma llave, cuerpo distinto	`422 IDEMPOTENCY_KEY_REUSED`
Llave nueva, `externalOrderId` ya existente	`200` con la orden existente
Llave nueva, `externalOrderId` de una orden que otra llave todavía crea	`409 REQUEST_IN_PROGRESS`, sin guardar la respuesta
Cuerpo o canal inválido (`400 VALIDATION_ERROR`, `403 CHANNEL_NOT_ALLOWED`)	No consume la llave: se corrige el cuerpo y se reintenta con la misma llave

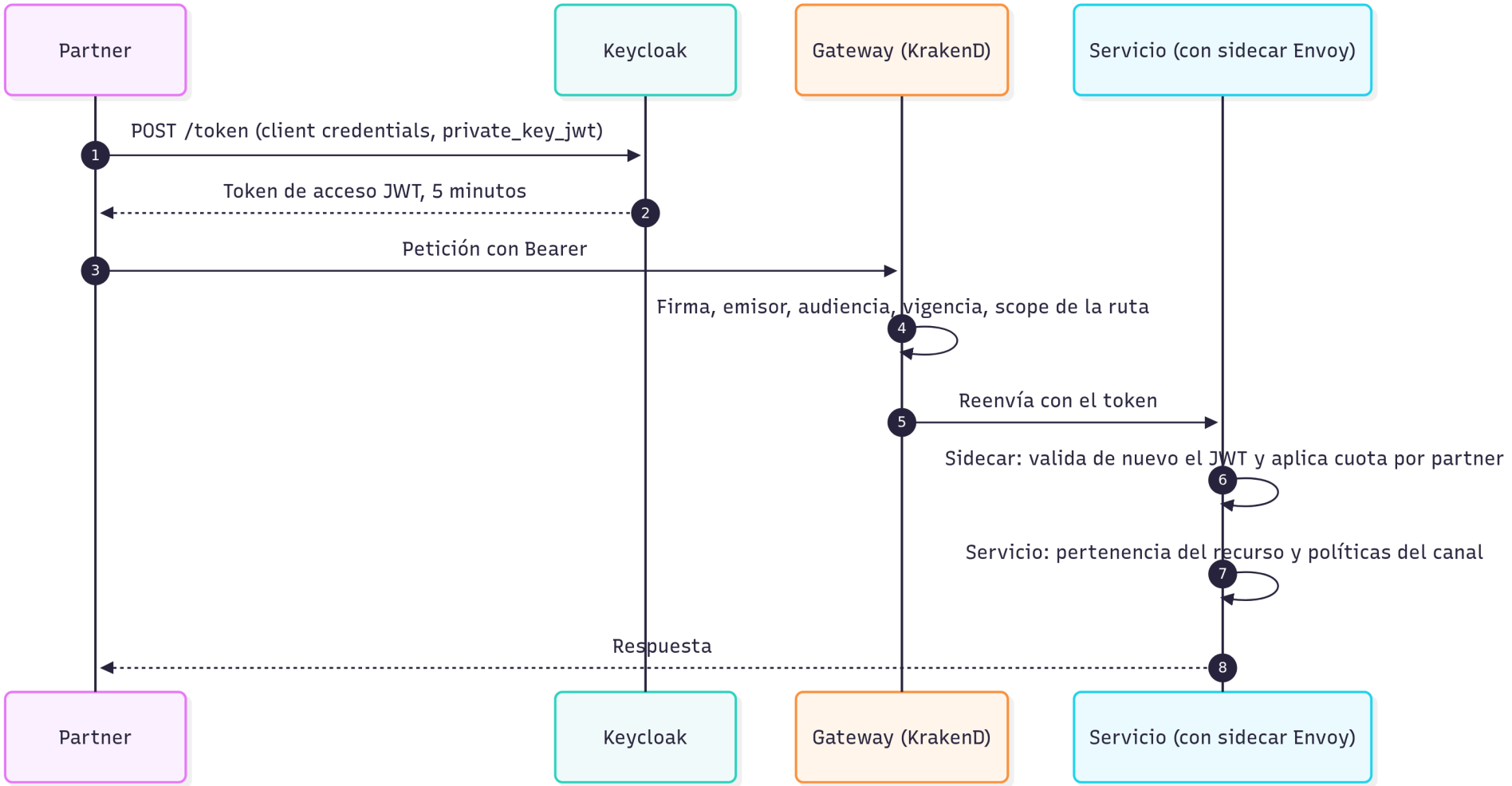
Hacia adentro cada paso es idempotente por `orderId`: reservar, confirmar, liberar, cobrar (la llave de idempotencia de la pasarela es el `orderId`) y registrar en OMS. El cambio de estado y su evento se guardan en la misma transacción (outbox: un proceso aparte lee la tabla y publica en Kafka), así que no existe orden guardada sin evento ni evento sin orden; cada consumidor deduplica por el `id` del evento. Trade-off: guardar respuestas y llaves cuesta almacenamiento y una escritura más por orden, a cambio de que un reintento del canal nunca cree una segunda orden ni un segundo cobro.

5. Seguridad

Conclusión. Una sola identidad para todos los partners (OAuth2 y OIDC con Keycloak), tokens cortos de 5 minutos, autenticación de cliente con llave privada en lugar de secretos compartidos, y autorización en tres niveles: scope por ruta, política por partner y canal, y pertenencia del recurso. El gateway valida el token, la malla lo vuelve a validar y aplica la cuota, y el servicio decide sobre el negocio. La capa no toca datos de tarjeta y guarda los datos personales en un solo servicio, cifrados.

5.1 Flujos OAuth2 y OIDC

Quién	Flujo	Detalle
Marketplaces y sistemas de partners	Client Credentials	Máquina a máquina. Un cliente de Keycloak por partner
E-commerce, a nombre de un cliente final	Authorization Code con PKCE (Proof Key for Code Exchange, la protección del flujo para aplicaciones que no pueden guardar un secreto)	El backend del e-commerce llama a la capa con su propio cliente; la identidad del comprador viaja como `customerId`
Servicios internos	Sin tokens de usuario	Identidad de carga de trabajo por mTLS (mutual TLS: ambos extremos se autentican con certificado) de la malla



Flujo OAuth2 de un partner

El partner se autentica con `private_key_jwt`: firma con su llave privada y la capa conoce solo la pública. Así no hay secreto compartido que robar. Para partners que no puedan, secreto de cliente con rotación obligatoria cada 90 días. Trade-off: más fricción de alta para el partner (supuesto S16), a cambio de eliminar la credencial que más se filtra.

El token es un JWT (JSON Web Token: token firmado que lleva los datos de quien llama, llamados claims):

Claim	Ejemplo	Uso
`iss`	`https://auth.andesretail.com/realms/andesretail`	Validación
`aud`	`interop-api`	Validación
`exp`	5 minutos	Validación
`azp`	`partner-mkp-andino`	Cliente
`partner_id`	`mkp-andino`	Aislamiento de datos y cuota
`channels`	`["mkp-andino"]`	Canales permitidos
`tier`	`gold`	Nivel de cuota
`scope`	`stock:read prices:read orders:write orders:read`	Autorización por ruta

5.2 Qué se valida en el gateway y qué en los servicios

Validación	Gateway	Sidecar (malla)	Servicio
TLS 1.2 o superior	Sí (borde)	mTLS interno	
Firma del JWT, `iss`, `aud`, `exp`	Sí	Sí, de nuevo	
Scope exigido por la ruta	Sí	Sí	
Tope general y por cliente	Sí		
Cuota por partner y nivel		Sí	
Solo el gateway puede llamar al servicio		Sí (política por identidad de carga)	
Tamaño y tipo de contenido	Sí		
Esquema del cuerpo			Sí
El recurso pertenece al `partner_id`			Sí
Canal, tienda y modo de cobro permitidos			Sí
Reglas de negocio			Sí

Un sidecar es un contenedor que acompaña a cada servicio y le aporta capacidades sin tocar su código; aquí es el proxy Envoy de Istio. Por qué validar el token dos veces: si alguien alcanza un servicio sin pasar por el gateway, el servicio no confía en cabeceras sino en el token. Los servicios leen `partner_id` del token validado por su sidecar, nunca de una cabecera puesta por el cliente. Trade-off: unos milisegundos más por salto (dentro del presupuesto de la sección 6.2) a cambio de no tener un perímetro único.

5.3 Autorización por roles, scopes, partner y canal

1. Scope por ruta: qué operaciones puede invocar (`stock:read`, `prices:read`, `orders:write`, `orders:read`, `webhooks:manage`). Ninguna ruta del gateway existe sin scope: lo exige el pipeline (F5).
2. Política por partner y canal, como configuración versionada: canales, tiendas, listas de precios, modos de cobro, vencimiento de reserva y tope visible de stock.
3. Pertenencia del recurso: toda consulta a la base filtra por `partner_id`. Una orden ajena responde 404.

Roles internos para operadores de AndesRetail en Keycloak: `support-read` (consultar órdenes y entregas de webhooks) y `ops-admin` (reprocesar colas de fallidos, cambiar cuotas). Toda acción de operador queda en auditoría.

5.4 Rotación de llaves, expiración y secretos

Elemento	Vigencia	Rotación
Token de acceso	5 minutos	No se revoca: vence. Sin tokens de refresco en Client Credentials
Llaves de firma de Keycloak	90 días	Con solapamiento: la llave nueva se publica en el JWKS (el documento público de llaves, ver abajo) antes de usarse; la vieja sigue válida hasta que venzan sus tokens
Llaves de los partners (`private_key_jwt`)	12 meses	El partner registra la nueva antes de retirar la vieja
Secreto de cliente (si aplica)	90 días	Dos secretos válidos durante el cambio
Secreto de firma de webhooks	12 meses o a demanda	Doble firma durante el cambio
Credenciales hacia ERP, WMS, OMS, CRM, pasarela, Kafka	90 días	Automática en el gestor de secretos
Certificados mTLS de la malla	24 horas	Automática por Istio

El JWKS (JSON Web Key Set) es el documento público con las llaves con las que se verifica la firma de los tokens; el gateway y los sidecars lo cachean y lo refrescan ante una llave desconocida.

Estrategia de secretos:

- Fuente: AWS Secrets Manager. Los pods los reciben como archivos montados por el Secrets Store CSI Driver (CSI, Container Storage Interface: la interfaz estándar con la que Kubernetes monta volúmenes externos).
- Acceso a servicios de AWS sin credenciales guardadas: EKS Pod Identity, que asigna un rol de IAM (Identity and Access Management, el sistema de permisos de AWS) al pod.
- Ningún secreto en el repositorio, en imágenes ni en variables de entorno. El pipeline escanea secretos en cada cambio.
- Cada adaptador solo puede leer el secreto de su sistema.

Trade-off: la rotación toma efecto en el siguiente sondeo del driver y solo si el servicio relee el archivo; por eso el contrato de todo servicio exige leer configuración y secretos de archivos. Se descartó Vault de HashiCorp para no sumar un proyecto de operación (alta disponibilidad, desellado, respaldo); el cambio a Vault solo tocaría la configuración del driver, no el servicio.

5.5 Datos personales y de pago

- Minimización: la capa solo recibe lo necesario para entregar la orden (nombre, contacto, dirección). Estos son datos personales o PII (personally identifiable information: datos que identifican a una persona).
- Viven únicamente en order-service, cifrados por campo con llaves de AWS KMS (Key Management Service, el servicio de gestión de llaves de AWS).
- No aparecen en eventos, logs, trazas ni métricas. Los eventos llevan `customerId` y `orderId`.
- Retención: se anonimizan a los 90 días de la entrega, salvo obligación legal distinta. Marco aplicable: Ley 1581 de 2012, la ley colombiana de protección de datos personales (supuesto S10).
- Pago: la capa nunca recibe, procesa ni guarda números de tarjeta. El canal tokeniza con la pasarela y envía el token. Con eso la capa queda fuera del alcance fuerte de PCI DSS (Payment Card Industry Data Security Standard, el estándar de la industria de tarjetas, que exige auditorías costosas a quien toque datos de tarjeta). Si un partner envía un número de tarjeta por error, un detector lo rechaza en el texto libre (nombre, correo, dirección) y en `payment.token`, y un filtro de logs lo enmascara.
- Los identificadores de la orden no pasan por el detector: tienen formato declarado en el contrato (sección 4.6). Un dígito de verificación acierta por azar una de cada diez veces, así que el detector adivina, y sobre un identificador adivinar rechaza órdenes válidas: un UUID de la tienda caía en él una de cada 450 compras, y el reintento con la misma llave recibía el mismo rechazo (ADR-026). Trade-off: una tarjeta puesta a propósito en un identificador no se detecta. Se acepta porque el detector protege contra el error y no contra el abuso: la capa nunca recibe tarjetas, solo el token de la pasarela. En el texto libre sigue adivinando, y un número de 13 a 19 dígitos en una dirección se rechaza aunque no sea tarjeta; ahí el cliente ve qué campo y lo corrige.

5.6 Modelo de amenazas

Se documentan 9 amenazas; aquí van las 6 principales y las otras 3 en el anexo A.

#	Amenaza	Mitigación
A1	Robo o reutilización de un token	Vigencia de 5 minutos, `private_key_jwt`, TLS, audiencia fija, detección de uso anómalo por partner
A2	Un partner accede a órdenes o precios de otro (BOLA, broken object level authorization: falla al comprobar que el objeto pedido es de quien lo pide)	Filtro por `partner_id` en toda consulta, 404 para recursos ajenos, pruebas automáticas de aislamiento entre partners (F7)
A3	Webhook de pago falsificado ("pago aprobado" inventado)	Implementado: verificación de firma de la pasarela, deduplicación, y confirmación contra la API de la pasarela antes de confirmar la orden. Diseño, sin implementar en el demo: lista de IP de origen
A4	Abuso o denegación de servicio: scraping de precios, ráfagas	Cuota por partner y nivel, tope general, microcache, Cloudflare en el borde, protección del legado por diseño
A5	Órdenes duplicadas o repetición de peticiones	Idempotencia con dos barreras, deduplicación de eventos (F9)
A6	Fuga de datos personales por logs, eventos o trazas	Sin datos personales en eventos, enmascarado en logs, cifrado por campo, pruebas que fallan si aparece un dato personal en la telemetría (F8)

5.7 Auditoría

Diseño: toda petición externa de escritura, todo cambio de estado de orden, reserva y pago, toda acción de operador y todo fallo de autenticación o autorización genera un registro en el tópico `andes.audit.v1`: quién (`partner_id`, cliente o operador), qué, cuándo, resultado y `correlationId`, y se archiva en almacenamiento de objetos con bloqueo contra modificación.

Implementado hoy: order-service publica el registro, con ese contenido y por outbox, en cuatro acciones (crear orden, confirmarla, reembolsar el pago y cancelarla). Todavía no auditan inventory-service (reservas), payment-service (pagos), webhook-service (alta y baja de suscripciones) ni el borde (fallos de autenticación o autorización), y el archivo con bloqueo contra modificación no está construido. Completar esos tres productores y el archivo es trabajo del roadmap (sección 8.3). Trade-off: costo de almacenamiento que crece sin límite, a cambio de poder reconstruir cualquier disputa con un partner.

5.8 Acceso de los operadores al clúster

La API de Kubernetes de EKS es solo privada: no tiene dirección pública, ni siquiera limitada a una IP. Se entra por un túnel de AWS Systems Manager (SSM, el servicio de AWS para administrar instancias) hacia un bastión: una máquina mínima (`t4g.nano`) dentro de la red privada, sin llave SSH, sin IP pública y sin reglas de entrada, con un rol que solo le permite hablar con SSM. El bastión no tiene permisos sobre Kubernetes: solo reenvía bytes, y quien se autentica ante la API es `kubectl`, en el equipo del operador, con su propio rol. Hay dos contextos: administrador (abre el túnel y despliega) y solo lectura, que ve todo menos los Secrets de Kubernetes y es el que se muestra en vivo. La verificación del certificado de la API se mantiene (el cliente valida el nombre del endpoint de EKS aunque se conecte a `localhost`); nunca se desactiva. CloudTrail registra quién abrió cada sesión.

Se descartaron el endpoint público limitado por IP (la API de un clúster con secretos expuesta a internet aunque esté filtrada, y una variable que cambiar cada vez que cambia la red del operador), el EC2 Instance Connect Endpoint (túneles de una hora como máximo y apuntados a una IP que en EKS cambia) y AWS Client VPN (veinte veces el costo del bastión, con certificados que administrar). Trade-off: un recurso más que cobra por hora (unos 0,005 dólares), que depende del NAT y que puede fallar; todo lo que habla con la API necesita el túnel abierto. Como respaldo, y solo por decisión explícita, el endpoint público limitado a una IP se vuelve a encender con una variable (nunca abierto a todo internet). Fuente: ADR-027, que reemplaza ese punto de ADR-022.

6. Resiliencia, performance y escalabilidad

Conclusión. El pico 10x de lecturas no llega al legado: se resuelve desde Redis, con cuotas por partner que impiden que uno consuma la capacidad de todos. Lo que sí llega al legado pasa por adaptadores con límite de concurrencia, circuit breaker y Kafka como amortiguador. Los fallos asíncronos se reintentan con espera creciente y terminan en colas de fallidos con alerta. En el demo, con el pico 10x, el p95 de stock y precio queda en 1,6 ms sin errores y las llamadas al ERP no crecen; con el ERP caído, las lecturas siguen respondiendo (sección 6.8).

6.1 Rate limiting por partner y canal

Dos niveles:

Nivel	Dónde	Qué limita	Contador
1. Tope general	KrakenD	Total por endpoint y un máximo igual para cualquier cliente	En memoria de cada instancia
2. Cuota por partner	Envoy (rate limit global de la malla)	Por `partner_id`, nivel y grupo de rutas	Compartido en Redis

Cuotas iniciales por nivel, en peticiones por segundo:

Nivel	Lecturas (stock y precio)	Órdenes	Uso previsto
`bronze`	20	2	Partner nuevo o en pruebas
`silver`	100	10	Marketplace mediano
`gold`	500	50	Marketplace principal, e-commerce propio

- La cuota es por partner; si un partner tiene varios canales, comparten cuota salvo que su política diga otra cosa.
- Al exceder: `429 RATE_LIMITED` con `Retry-After` y cabeceras `RateLimit-*`.
- Las órdenes tienen cuota separada de las lecturas: un pico de consultas no deja a un partner sin poder vender.
- Si el servicio de cuotas falla, se deja pasar (fail open) y queda el tope general del gateway.

Trade-off: dos lugares donde se configuran límites y un servicio de cuotas más que operar. Se eligió así porque KrakenD comunitario no tiene cuotas por partner (sección 3.4). Riesgo de construcción declarado: si el rate limit global de Envoy se complica, el plan B es un filtro de cuota en la plantilla de servicio con el mismo contador en Redis.

6.2 Cache: dónde, qué, TTL e invalidación

Dónde	Qué	TTL (time to live, vigencia)	Invalidación
Redis (modelo de lectura)	Stock, precio, producto, segmento	Sin vencimiento funcional; TTL de seguridad de 24 horas (stock) y 7 días (precio)	Por evento: cada cambio reescribe la llave
Gateway (microcache)	Respuestas de `GET /v1/stock` y `GET /v1/prices`, por partner y consulta	2 segundos, solo en modo promo	Por tiempo
Sidecar y gateway	JWKS de Keycloak	10 minutos	Por tiempo, con refresco ante llave desconocida
pricing-service	Política de canal y listas de precios	60 segundos en memoria	Por tiempo

No se cachea el estado de órdenes, las reservas ni nada de la ruta de crear orden.

El modelo de lectura no es una cache por vencimiento: es una vista materializada que se actualiza por evento, y la invalidación es el evento mismo. Se descartó la cache con vencimiento delante del legado porque cada vencimiento vuelve a golpearlo, y una avalancha de fallos de cache en plena promo lo tumba igual. Los tópicos de estado de Kafka son compactados (conservan el último valor por llave), así que reconstruir Redis desde cero es releerlos.

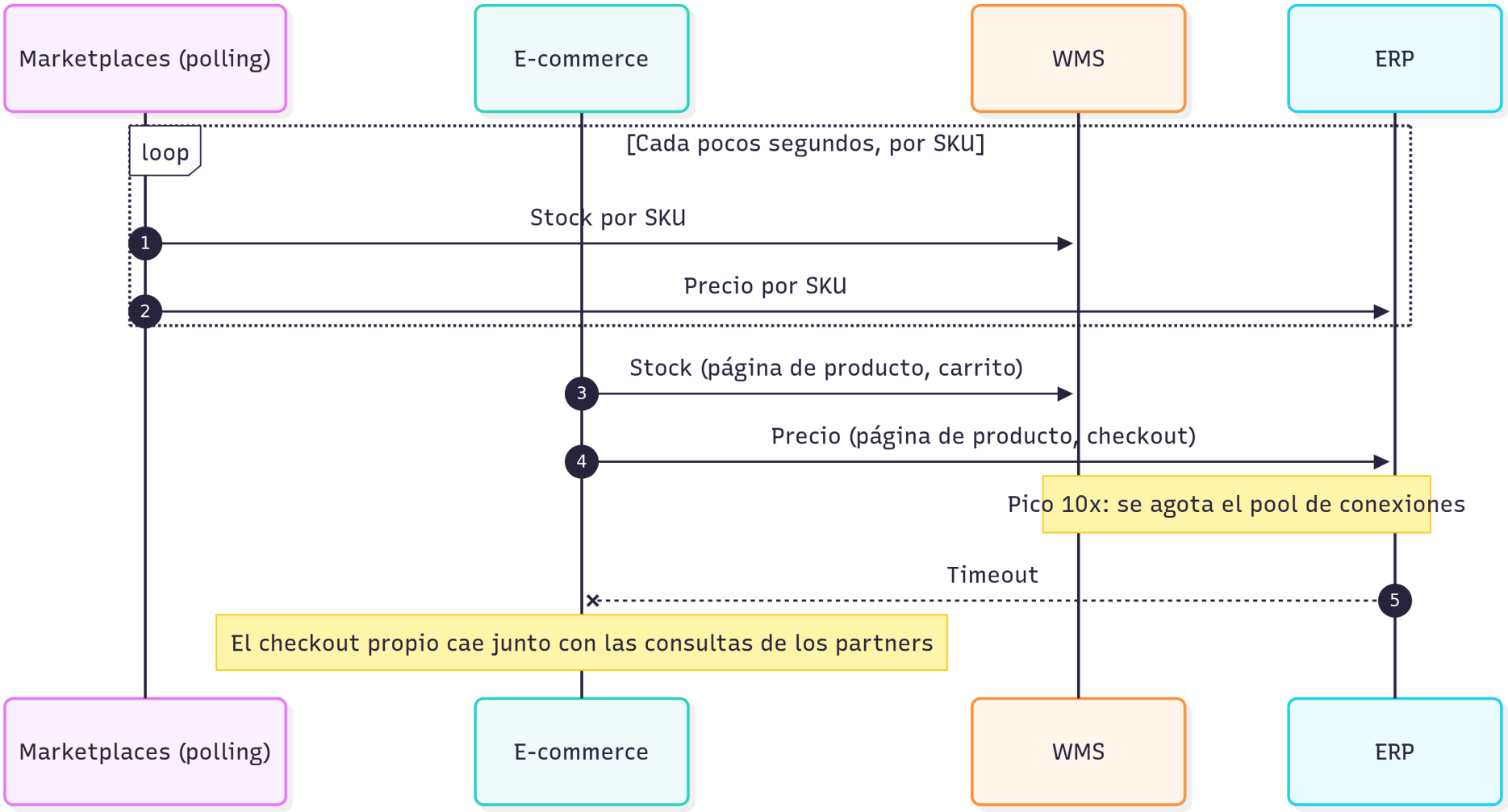
Frescura prometida, como objetivo de diseño:

Dato	Frescura objetivo (p95)	Umbral de `stale: true`
Stock	10 segundos desde el cambio en el WMS	5 minutos
Precio	60 segundos desde el cambio en el ERP	30 minutos

Trade-off: el dato puede estar levemente desactualizado. Se acepta porque se declara (`asOf`, `stale`), se mide, y porque la decisión que importa, prometer la unidad, no se toma con este dato sino con la reserva.

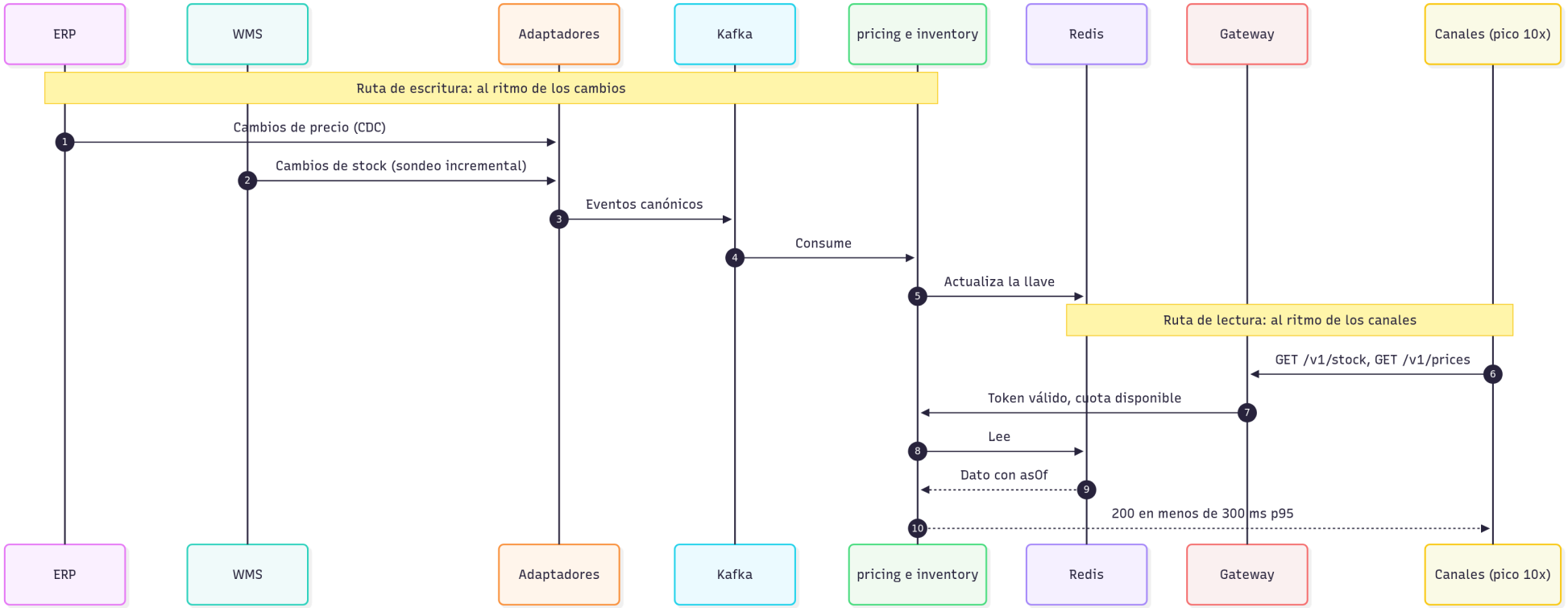
6.3 Flujo B: consulta masiva en promo, antes y después

En el AS-IS cada consulta de un canal termina en una consulta en línea al WMS o al ERP. Una promo que multiplica por 10 las consultas multiplica por 10 la carga de sistemas que no escalan, y el checkout propio cae junto con las consultas de los partners.



Flujo B en el AS-IS

En el TO-BE hay dos rutas con ritmos distintos: la de escritura va al ritmo de los cambios del legado y la de lectura al ritmo de los canales.



Flujo B en el TO-BE

Presupuesto de latencia p95, como objetivo de diseño: borde (Cloudflare y balanceador) 20 ms, gateway 10 ms, malla con dos sidecars 5 ms, servicio 15 ms, Redis 5 ms. Total interno: 55 ms, con margen amplio frente a los 300 ms. Capacidad: con el supuesto S4 de 2.000 consultas por segundo y lotes promedio de 10 SKU, son 20.000 lecturas de llave por segundo, holgado para un Redis con una réplica por zona.

Falla	Comportamiento
Llave ausente en Redis	Se lee de la base del servicio y se repone la llave
Redis caído	Lectura desde la base con concurrencia limitada; si se satura, 503 con `Retry-After`
Kafka o adaptador detenido	Se sigue respondiendo con el último dato; al superar el umbral, `stale: true` y alerta
ERP, WMS o enlace on-prem caídos	Igual. El canal no percibe la caída del legado

6.4 Protección de los legados (backpressure)

Backpressure es el mecanismo por el que un sistema lento frena a quien le envía trabajo en lugar de ahogarse.

1. Por diseño: los canales no llegan al legado. Solo los adaptadores lo hacen.
2. Bulkhead por sistema (límite de concurrencia por dependencia): máximo inicial de 10 llamadas simultáneas al ERP, 10 al WMS y 20 al OMS, más un límite de llamadas por segundo.
3. Kafka como amortiguador: los comandos hacia el OMS se consumen al ritmo que el OMS aguanta. Un pico de órdenes alarga la cola, no la carga del OMS.
4. Circuit breaker por sistema (corte automático de llamadas a una dependencia que falla): abre con 50% de fallos en una ventana de 20 llamadas y prueba de nuevo a los 30 segundos. Abierto, el adaptador pausa su consumo en vez de quemar reintentos.
5. Timeouts de 2 segundos por llamada al legado. Nunca una espera sin límite.
6. Sondeo adaptativo: si el WMS responde lento, el adaptador alarga su intervalo.

Trade-off: con el legado lento, el dato tarda más en llegar a la capa (crece la frescura) y las órdenes tardan más en registrarse en el OMS. Se prefiere eso a que el legado caiga.

6.5 Reintentos y DLQ

Una DLQ (dead letter queue) es la cola donde quedan los mensajes que fallaron tras agotar los reintentos, para revisarlos o reprocesarlos sin bloquear el flujo.

Caso	Política
gRPC de reserva	2 reintentos con espera exponencial y jitter (variación aleatoria de la espera), solo ante errores transitorios; presupuesto total de 300 ms
Llamadas de adaptadores al legado	3 intentos (200 ms, 1 s, 5 s) con jitter, solo en operaciones idempotentes
Consumo de eventos	Reintento inmediato hasta 3 veces; luego al tópico de reintento con esperas de 1 minuto, 10 minutos y 1 hora; luego DLQ
Registro en OMS	Igual, pero no va a DLQ antes de 24 horas: es un paso que debe lograrse
Webhooks a partners	8 intentos en 24 horas (10 s, 1 min, 5 min, 30 min, 1 h, 3 h, 6 h, 12 h); luego DLQ por partner

- Errores no transitorios (esquema inválido, regla de negocio) van directo a DLQ, sin reintentos.
- Los reintentos nunca bloquean la partición: el mensaje fallido sale a su tópico de reintento y el consumidor sigue. En oms-adapter, un mensaje con espera larga no frena a los demás del tópico de reintento (corregido el 15/11/2026). El `POST` que registra la orden en el OMS no se repite tras un timeout, porque podría duplicar la orden; se reintenta por el tópico de reintento.
- El mensaje de reintento y el de DLQ son el original, salvo los datos sensibles: al pasar a la DLQ se reemplazan por `--data_redacted--` y la cabecera `andes_payload_redacted` lista qué campos se enmascararon (ADR-029). El tópico de reintento conserva el original, porque el reintento lo necesita; el token de pago nunca aparece en logs.
- Cada DLQ tiene alerta y un procedimiento de reproceso con rol `ops-admin`.
- Tras varios fallos seguidos de un partner, su suscripción se pausa y se le notifica; los eventos quedan guardados para reenvío.

Trade-off: Kafka no trae reintentos ni DLQ nativos (con SNS y SQS, las colas y notificaciones de AWS, sí vienen), así que se construyen con tópicos. Se eligió Kafka porque permite releer el pasado para reconstruir el modelo de lectura y garantiza orden por llave.

6.6 Plan para promos

Antes:

1. Prueba de carga a 10x del tráfico esperado, en un entorno igual al productivo, dos semanas antes.
2. Subir el mínimo de réplicas de inventory-service, pricing-service y gateway por escalado programado. No depender del autoescalado reactivo en el primer minuto.
3. Precalentar el modelo de lectura y verificar la frescura.
4. Revisar cuotas con cada partner; subir de nivel a quien lo necesite.
5. Congelar despliegues desde 24 horas antes.

Durante:

1. Activar el modo promo: microcache de 2 segundos en el gateway.
2. Subir la frecuencia de sondeo del WMS solo para los SKU en promoción.
3. Tablero de promo a la vista: p95, frescura, cuotas agotadas, llamadas al legado, órdenes por estado.
4. Degradaciones preparadas, de menor a mayor: servir dato con `stale: true`, bajar el tope visible de stock, reducir cuotas de los niveles bajos.

Después: revisión con datos, ajuste de colchones de seguridad y de cuotas.

6.7 Escalabilidad

Componente	Cómo escala
Gateway y servicios de lectura	Horizontal, por uso de procesador (CPU) y peticiones por segundo; sin estado
order-service y payment-service	Horizontal; el orden lo da la partición de Kafka
Adaptadores	No escalan con el tráfico: su límite es el legado
Redis	Réplicas de lectura por zona; partición por llave si el catálogo crece 10x
PostgreSQL	Una instancia por servicio; réplica de lectura para el respaldo de lecturas
Kafka	Más particiones y consumidores

6.8 Evidencia

El mismo escenario de carga se corre contra los dos lados: el AS-IS (simuladores de ERP y WMS consultados directo, como hoy lo harían los canales) y el TO-BE (por la capa). La escala del demo es 20 consultas por segundo en base y 200 en pico: lo que importa es la proporción 10x. Las cifras son de un solo equipo de desarrollo con Docker Desktop, no de producción: sin red entre zonas, sin cifrado en tránsito, sin Cloudflare ni malla. Lo que se compara es el comportamiento a la misma escala y en el mismo equipo; la latencia de producción tiene su presupuesto en la sección 6.3. El umbral de `stale` del precio se bajó a 45 s para la prueba de caída (en el diseño es 30 minutos). Archivos crudos en `tests/load/results/`.

Medida	AS-IS	TO-BE
p95 de stock y precio en tráfico base	Stock 128 ms, precio 196 ms	3 ms los dos
p95 de stock y precio en pico 10x	5.001 ms los dos: el timeout del canal	1,6 ms los dos
Tasa de error en pico 10x	100% en stock y en precio	0% en stock y en precio
Llamadas por segundo al ERP en pico 10x	200 (20 en la base: crecen 10x con el tráfico); el ERP alcanza a atender 67	0,9, igual que en la base: es el sondeo del adaptador y no depende de las consultas. Igual en el WMS

Medida	AS-IS	TO-BE
ERP lento (3 s de latencia)	Precio con p95 de 5.001 ms (el timeout) y 99,6% de error	p95 de precio 1,9 ms y 0% de error; el circuit breaker del adaptador abre y las llamadas al ERP bajan de 0,9 a 0,1 por segundo
Con el ERP caído, ¿responde stock y precio?	Precio no (0,03% de respuestas correctas); stock sí, porque viene del WMS. Con el WMS caído, stock no	Sí, los dos: 100% de respuestas correctas durante 3 minutos de caída, y lo mismo con el WMS caído. El precio sale `stale: true` pasado el umbral y vuelve a `stale: false` 5 s después de que el ERP regresa
Frescura p95: del cambio en el legado a la respuesta de la API	No aplica: cada consulta lee el legado en línea	Precio 1,2 s (objetivo 60 s); stock 1,2 s (objetivo 10 s)
Orden creada dos veces por reintento	Sí: 20 de 20 órdenes reintentadas tras un timeout quedaron duplicadas	No: 20 primeros envíos vencidos por el timeout del canal y reintentados con la misma `Idempotency-Key` dieron 20 órdenes en la capa y 20 en el OMS, 0 de más

Criterio de éxito en el TO-BE para el pico 10x: p95 menor a 300 ms, errores menores a 0,1% y llamadas al ERP y al WMS que no superan la base más 10%. Los seis escenarios de falla dirigida están en el anexo F.

La misma medición en la nube. El AS-IS se repitió en AWS el 14/11/2026 (17:30 a 17:48 UTC): simuladores del ERP, WMS y OMS en el clúster EKS (`us-east-1`, dos nodos `m7i-flex.xlarge` en dos zonas) y el generador de carga k6 dentro del mismo clúster, con los mismos scripts, la misma imagen y las mismas perillas que en el equipo de desarrollo. Sin internet, sin Cloudflare, sin gateway de entrada y sin malla: lo que cambia frente a la corrida local es la red entre nodos y el tope de procesador. Archivos crudos en `tests/load/results/as-is-nube/`.

Medida	AS-IS en el equipo	AS-IS en la nube	TO-BE en la nube
p95 de stock y precio en tráfico base	Stock 128 ms, precio 196 ms	Stock 129 ms, precio 196 ms	Stock 10,7 ms, precio 8,6 ms
p95 de stock y precio en pico 10x	5.001 ms los dos	5.000 ms los dos: el timeout del canal	Stock 12,0 ms, precio 10,3 ms
Tasa de error en pico 10x	100% en stock y en precio	100% en stock y en precio	0% (36.001 consultas)
Llamadas por segundo al ERP en pico 10x	200; el ERP atiende 67	200 (20 en la base); el ERP atiende 66,6	0,9, igual que en la base
ERP lento (3 s de latencia)	Precio con p95 de 5.001 ms y 99,6% de error	Precio con p95 de 5.000 ms y 99,5% de error	Precio con p95 de 8,6 ms y 0% de error
Con el ERP o el WMS caído	Precio no (0,03%) con el ERP caído; stock no con el WMS caído	Igual: precio 0,03% con el ERP caído, stock 0,03% con el WMS caído	Stock y precio responden el 100% con el ERP caído y con el WMS caído
Orden creada dos veces por reintento	20 de 20	20 de 20 (40 órdenes en el OMS para 20 órdenes lógicas)	0 de más en 20
Un partner excede su cuota (escenario 6)	No aplica	No aplica	Bronze a 5 veces su cuota: 20,2 aceptadas por segundo de 100 pedidas, 9.580 respuestas 429 (80%) con `Retry-After`; gold sin errores, p95 13,2 ms

El AS-IS da en la nube lo mismo que en el equipo de desarrollo: su límite no es la máquina, es que cada consulta llega en línea al legado. El TO-BE se midió en la nube el 15/11/2026 (16:49 a 17:11 UTC, y el escenario 6 repetido a las 17:40 UTC), con los mismos scripts, la misma imagen de k6 y las mismas perillas, entrando por el gateway de entrada de Istio con HTTPS. Los seis escenarios cumplen. Sus p95 son mayores que en el equipo de desarrollo por el TLS del gateway, el mTLS de la malla y la red entre nodos. La primera corrida del escenario 6 incumplió: a bronze se le aceptaron las 100 lecturas por segundo, sin ningún 429, porque el servicio de cuotas no tenía usuario en Redis; se arregló y se repitió, y el detalle está en el README de la corrida. Archivos crudos en `tests/load/results/to-be-nube/`.

El flujo de órdenes (flujo A) se verificó de punta a punta el 11/11/2026, con todas las piezas reales detrás del borde y el token de cada partner (`make f4-e2e`). Son pruebas de comportamiento, que pasan o fallan, y no de carga; no tienen columna AS-IS porque el AS-IS no tiene saga ni compensación que probar (sección 4). Algunos tiempos de reintento se acortaron para que la prueba dure minutos (el reintento hacia el OMS, por ejemplo, de 1 minuto a 3 s); la saga, los vencimientos y la consulta a la pasarela usan los

valores del diseño. Límite de la medición del OMS: el simulador no puede reproducir un OMS real y lento que confirme una orden después de que el adaptador ya la buscó. oms-adapter consulta si la orden existe antes de cada intento, pero no envía una llave de idempotencia al OMS; el "0 de más" vale para el simulador y queda por probar contra un OMS real.

Prueba	Resultado en el TO-BE
Escenario 4: la pasarela rechaza el pago	Orden `CANCELLED` con `PAYMENT_REJECTED`, reserva `RELEASED`, el vendible baja con la reserva y vuelve a su valor; el partner recibe `payment.rejected` y `order.cancelled` firmados
Escenario 5: webhook de pago duplicado	Un solo evento registrado, un solo cambio de estado y un solo `payment.approved` al partner. Un webhook con firma inválida recibe 401 en el borde
Escenario 5: webhook de pago perdido	payment-service consulta a la pasarela y la orden se resuelve sola al minuto
OMS caído	La orden queda `CONFIRMED` con `omsSync` pendiente; cuando el OMS vuelve se registra una sola vez
F9: la misma orden 5 veces en paralelo	Una orden y una reserva: un 202 y cuatro 409 `REQUEST_IN_PROGRESS`; el sexto envío recibe la respuesta original
100 pedidos simultáneos por la última unidad	Gana uno; los 99 restantes reciben `INSUFFICIENT_STOCK`; sin sobreventa
F7: un partner pide la orden de otro	404 `NOT_FOUND`, igual que por una orden inexistente
F6 y F8: trazabilidad y datos personales	El `correlationId` de la orden aparece en los logs de los cinco componentes por los que pasa (order, inventory, payment, oms-adapter y webhook-service); ningún dato personal de la corrida aparece en los logs de los servicios ni del borde
Camino feliz	`PENDING_PAYMENT`, `CONFIRMED`, `DISPATCHED`, `DELIVERED`. El despacho y la entrega salen del WMS simulado por wms-adapter (`DispatchStarted`, `DeliveryConfirmed`), ya no se publican a mano, y la reserva pasa a `CONSUMED`; el receptor del partner verifica la firma de cada webhook y recibe los eventos de la orden en su secuencia. Un evento de pago viaja por otro tópico y puede cruzarse con uno de orden: el partner ordena por `time`

La malla se probó en vivo el 14/11/2026 en un clúster local de Kubernetes (kind, Kubernetes in Docker: un clúster que corre en contenedores del equipo), con Istio y sus sidecars, el proveedor de identidad, el gateway y el gateway de entrada (`make mesh-kind-test`). El clúster se crea para la prueba y se borra al final.

Prueba	Resultado en el TO-BE
Escenario 6: un partner excede su cuota	El partner bronze pide 100 lecturas por segundo, 5 veces su cuota: recibe 20,1 por segundo y el resto (79,9%) en 429 `RATE_LIMITED` con `Retry-After`. El partner gold, al mismo tiempo: 0 errores y p95 de 6,6 ms (8,2 ms sin el exceso del otro)
mTLS estricto	Una carga sin sidecar no logra conectarse a un servicio, ni con un token válido
Llamada directa a un servicio, sin pasar por el gateway	403 dentro de la malla
Salida al legado	Solo el adaptador de cada sistema llega a él
Circuit breaker de la malla	Expulsa al ERP lento, responde 503 de inmediato y se recupera a los 30 s

En el AS-IS el escenario 6 no aplica: no hay cuotas por partner y cada canal llega al legado sin límite propio. Archivo crudo: `tests/load/results/to-be/escenario-6-cuota-20261114T202635Z.json`.

7. Observabilidad y operación

Conclusión. Logs, métricas y trazas con OpenTelemetry, unidos por dos identificadores: el `traceId` técnico y el `correlationId` de negocio, que el partner puede enviar y que viaja en cada respuesta, evento y registro de auditoría. Con un `correlationId` se reconstruye la historia completa de una orden. Hay 17 métricas, 10 alertas ligadas a objetivos de servicio y un runbook por alerta. Aquí van las 10 primeras métricas, las 6 primeras alertas y el runbook de la falla de stock y precio; el resto de métricas y alertas está en el anexo B, y los demás runbooks en el diseño (`docs/to-be/10-observabilidad.md`).

OTel (OpenTelemetry) es el estándar abierto para generar y exportar logs, métricas y trazas. Los servicios no implementan esto a mano: la malla aporta las métricas de red, la autoinstrumentación aporta las trazas, el recolector lee las métricas de negocio que cada servicio expone en `/metrics` y un recolector por nodo toma los logs de la salida estándar. Trade-off: dependencia fuerte de la plataforma; a cambio, ningún equipo reinventa la telemetría y el contrato no depende del lenguaje.

7.1 Campos mínimos de logging

JSON por línea, a la salida estándar.

Campo	Ejemplo
`timestamp`	`2026-11-10T18:04:30.123Z`
`level`	`INFO`
`service`, `version`, `env`	`order-service`, `1.4.2`, `prod`
`message`	`Orden aceptada`
`correlationId`	`5e1d0c7a-...`
`traceId`, `spanId`	Contexto de traza
`partnerId`, `channel`	`mkp-andino`
`route`, `method`, `status`, `durationMs`	En logs de acceso
`orderId`, `sku`, `eventId`	Cuando aplica
`errorCode`	El `code` del modelo de errores

Prohibido en logs: nombres, correos, teléfonos, direcciones, tokens, secretos y cualquier dato de pago. Un filtro enmascara patrones conocidos y la prueba F8 falla si aparecen en los logs de los servicios y del borde (hoy se ejecuta con `make f4-e2e`, sin compuerta; sección 3.8).

7.2 Métricas

#	Métrica	Dimensiones	Para qué
M1	Peticiones por segundo	Ruta, partner, código	Tráfico y mezcla
M2	Tasa de error (5xx y 4xx por separado)	Ruta, partner	Disponibilidad
M3	Latencia p50, p95, p99	Ruta, partner	Objetivo de 300 ms
M4	Frescura del modelo de lectura (`now - asOf`)	Stock o precio	Calidad del dato servido
M5	Aciertos del modelo de lectura frente a respaldo en base	Servicio	Salud de Redis
M6	Retraso de consumidores de Kafka (lag)	Grupo, tópico	Acumulación
M7	Llamadas al legado por segundo y concurrencia en uso	Sistema	Protección del legado
M8	Estado del circuit breaker	Sistema	Legado degradado
M9	Peticiones rechazadas por cuota (429)	Partner, nivel	Cuotas mal dimensionadas o abuso
M10	Órdenes por estado y antigüedad en estado	Estado, canal	Órdenes atascadas

Las métricas se ligan a SLO (service level objective: meta medible de calidad, contra la que se alerta):

SLO	Meta	Ventana
Disponibilidad de APIs externas	99.9% de peticiones sin 5xx	30 días
Latencia de stock y precio	95% bajo 300 ms	30 días
Frescura de stock	95% de lecturas con dato de menos de 10 s	7 días
Confirmación de orden	99% en menos de 60 s desde el pago aprobado	7 días

7.3 Trazas distribuidas y correlación

- `traceparent`, la cabecera del estándar W3C Trace Context (el formato del W3C, el consorcio que publica los estándares de la web, para propagar el contexto de una traza), se propaga por HTTP, gRPC y cabeceras de Kafka. Una traza une la petición del canal con la reserva, el cobro, el registro en OMS y el webhook.
- El consumo de un evento abre un tramo enlazado al que lo produjo, así el flujo asíncrono se ve como una sola historia.
- `correlationId`: lo envía el partner en `X-Correlation-Id` o lo genera el gateway; vuelve en toda respuesta y viaja en cada evento.
- Muestreo (diseño; el demo no lo tiene configurado todavía): 100% de errores y de órdenes; 5% de lecturas de stock y precio. Trade-off: una lectura lenta que no cae en el 5% no tiene traza, pero sí métrica y log; se acepta para no pagar el almacenamiento de 2.000 trazas por segundo en pico.
- Auditoría de punta a punta: con un `correlationId` se obtienen la traza, los logs de todos los servicios, los eventos publicados y los registros de auditoría (sección 5.7). Hoy F6 comprueba el `correlationId` en los logs de los cinco componentes de la orden (sección 3.8); los eventos y la auditoría completa son diseño.

7.4 Alertas

#	Alerta	Condición	Severidad
AL1	Consumo acelerado del presupuesto de error de disponibilidad	Ritmo 14 veces el permitido durante 1 hora, o 6 veces durante 6 horas	Página
AL2	Latencia de stock y precio	p95 mayor a 300 ms por 5 minutos	Página
AL3	Frescura de stock	Mayor a 60 s por 5 minutos	Página en horario de promo; aviso fuera
AL4	Circuit breaker abierto	ERP, WMS u OMS abierto por más de 2 minutos	Aviso
AL5	Órdenes atascadas	Más de 10 órdenes en `PENDING_PAYMENT` con más de 20 minutos	Página
AL6	Registro en OMS atrasado	Alguna orden con `omsSync` pendiente por más de 10 minutos	Aviso; página a los 30

Toda alerta enlaza a su tablero y a su runbook. La alerta AL1 se basa en el ritmo de consumo del presupuesto de error y no en un umbral fijo: avisa rápido ante una caída fuerte y no despierta a nadie por un pico breve.

7.5 Runbook: falla stock o precio

Síntoma: alerta AL2 o AL3, o partners reportando errores o datos viejos en `GET /v1/stock` o `GET /v1/prices`.

1. Clasificar en el tablero de lecturas, en menos de 2 minutos: ¿errores 5xx o latencia? Paso 2. ¿Responde bien pero con dato viejo (`stale: true`, frescura alta)? Paso 3.
2. La ruta de lectura falla:
   - Redis caído o saturado: confirmar que el respaldo a base está activo; promover réplica.
   - Servicio: revisar reinicios y saturación; escalar réplicas.
   - Gateway o cuotas: si los 429 se dispararon para todos, revisar el servicio de cuotas (falla abierta) y el tope general.
   - Despliegue reciente: si coincide en el tiempo, devolver a la versión anterior antes de investigar.
3. La ruta de alimentación falla (el canal sigue recibiendo 200):
   - Lag de Kafka alto: revisar el consumidor del servicio.

- Adaptador con circuit breaker abierto o cursor detenido: el problema está en el legado o en el enlace on-prem. Confirmar con el equipo del ERP o del WMS.
- Mientras dure, se sirve el último dato con `stale: true`. Si es stock y supera 15 minutos, subir el colchón de seguridad de los SKU de alta rotación para reducir el riesgo de sobreventa.

4. Comunicar: estado a soporte y a los partners afectados si supera 10 minutos.
5. Recuperación: al volver la fuente, el adaptador retoma desde su cursor; verificar que la frescura baje al objetivo. Si hubo pérdida de cambios, disparar la conciliación completa.
6. Cierre: revisión sin culpables en 5 días hábiles; si el aprendizaje es repetible, se convierte en alerta, función de aptitud o paso de este runbook.

Lo que no se hace: apuntar los canales directo al ERP o al WMS "mientras tanto". Eso recrea el AS-IS en el peor momento.

7.6 Herramientas

En producción: OpenTelemetry Collector, Amazon Managed Service for Prometheus (métricas), Loki u OpenSearch (logs), Tempo o AWS X-Ray (trazas) y Amazon Managed Grafana (tableros y alertas). En el demo, las mismas piezas de código abierto dentro del clúster.

8. Plan de implementación (roadmap)

Conclusión. Migración por estrangulamiento: la capa crece alrededor de las integraciones existentes y las reemplaza una por una, primero lo que más duele y menos arriesga, con vuelta atrás en cada paso. El MVP es una tajada vertical completa para un marketplace piloto en 6 semanas: los cuatro endpoints mínimos, seguridad completa y observabilidad. Lo que recorta es alcance funcional, nunca seguridad ni operación.

8.1 Estrategia y orden de migración

El patrón estrangulador (strangler fig) hace crecer lo nuevo alrededor de lo viejo y lo reemplaza por partes, sin un corte único.

Paso	Qué se mueve	Integraciones del AS-IS que retira	Riesgo
1	Lecturas de stock y precio de un marketplace piloto	7 y 8 (para ese partner)	Bajo: solo lectura, reversible cambiando una URL
2	Lecturas de todos los marketplaces	7 y 8	Bajo
3	Crear orden del marketplace piloto	6 (para ese partner)	Medio: entra la saga
4	Crear orden de todos los marketplaces	6	Medio
5	Lecturas del e-commerce	1 y 2	Medio: es el canal de más tráfico
6	Órdenes y pago del e-commerce	3 y 4	Alto: es el canal propio
7	Clientes y puntos	5 y 12	Bajo
8	Integraciones internas (OMS con WMS y ERP, lote nocturno)	9, 10, 11	Se evalúa; puede no justificarse

Los números de integración corresponden a las 12 flechas del C4 AS-IS: 1 y 2, e-commerce a ERP y WMS; 3, e-commerce a OMS; 4, e-commerce y pasarela; 5, e-commerce a CRM; 6, 7 y 8, marketplaces a OMS, WMS y ERP; 9 y 10, OMS a WMS y ERP; 11, lote nocturno de WMS a ERP; 12, OMS a CRM.

Cada paso pasa por cuatro etapas: sombra (la capa corre en paralelo y se comparan respuestas, sin afectar al canal), canario (un porcentaje pequeño del tráfico real), corte (todo el tráfico; la integración vieja apagada pero disponible dos semanas) y retiro (se eliminan credenciales y accesos directos al legado; sin este paso la integración vieja sigue siendo una puerta abierta).

Trade-off: durante la convivencia los canales no migrados venden sin reservar en la capa (riesgo R8). Se mitiga con un colchón mayor y migrando primero los canales de más órdenes.

8.2 MVP: 4 a 6 semanas

Semana	Entrega
1	Plataforma base: clúster, pipeline, plantilla de servicio, Keycloak, gateway, observabilidad mínima. Contratos v1 acordados con el partner piloto. Validación del supuesto S1
2	Adaptadores de ERP y WMS; modelo de lectura alimentado y comparado en sombra contra el legado
3	`GET /v1/stock` y `GET /v1/prices` en producción para el piloto; cuotas; prueba de carga 10x
4	Reservas e inventory-service con gRPC; order-service con idempotencia; payment-service contra el entorno de pruebas de la pasarela
5	Saga completa, adaptador del OMS, webhooks de orden confirmada y pago; órdenes del piloto en canario
6	Estabilización: runbooks, alertas, prueba de escenarios de falla, corte del piloto

Lo que el MVP sí entrega:

- Los 4 endpoints mínimos del enunciado, para un partner.
- Webhooks: orden confirmada, pago aprobado y rechazado. Despacho y entrega si el WMS los expone sin trabajo adicional.
- Seguridad completa: no hay versión reducida de seguridad.
- Observabilidad: tableros, 6 alertas, runbook de stock y precio.

- Evidencia: p95 y carga al legado, antes y después.

Lo que el MVP no entrega, a propósito:

- Los demás marketplaces y el e-commerce.
- Integración con el CRM: el precio por segmento de cliente usa la lista por defecto del canal.
- Devoluciones, cancelaciones iniciadas por el cliente, órdenes con varias ubicaciones de despacho.
- Portal de desarrolladores y alta autónoma de partners.
- Multirregión.
- Cambios en las integraciones internas entre OMS, WMS y ERP.

Trade-off: un MVP de un solo partner demora el beneficio para el resto de los canales, a cambio de probar los dos flujos completos en producción con riesgo acotado.

8.3 A 3 meses

- Todos los marketplaces migrados en lecturas y órdenes; integraciones 6, 7 y 8 retiradas.
- E-commerce leyendo stock y precio por la capa.
- Adaptador del CRM: precio por segmento y aviso de compras para puntos.
- Webhooks de despacho y entrega para todos.
- Alta de partners guiada: plantilla de configuración, entorno de pruebas (sandbox), documentación publicada desde los contratos.
- Herramienta de reproceso de DLQ y reenvío de webhooks.
- Escenarios de falla y pruebas de comportamiento (F1, F2 y F6 a F10) corriendo en el pipeline como funciones de aptitud, y construidas las dos que hoy son diseño: la regla de dependencias (F3) y la comparación de contratos (F4), sección 3.8.
- Auditoría completa: registro en inventory, payment, webhook y el borde, y archivo con bloqueo contra modificación (sección 5.7).
- Primera promo real atendida con el plan de promos y revisión posterior.

8.4 A 6 meses

- Órdenes y pago del e-commerce por la capa; integraciones 1 a 5 retiradas.
- Devoluciones y cancelaciones como sagas propias (punto en que se revisa el motor de flujos, sección 4.4).
- Portal de desarrolladores, llaves autogestionadas, planes de consumo.
- Eventos ofrecidos a consumidores internos (analítica, fidelización) como producto.
- Evaluación de las integraciones internas 9 a 11: se migran solo si los datos de incidentes lo justifican.
- Recuperación ante desastres probada: restauración en una segunda región.
- Revisión de costos y de dimensionamiento con datos reales.
- Plantilla de adaptador probada con un sistema nuevo, para medir cuánto cuesta de verdad agregar uno.

8.5 Equipo y criterio de salida

Equipo supuesto: en el MVP, 1 arquitecto, 4 desarrolladores, 1 de plataforma, 1 de calidad y medio tiempo de seguridad (supuesto S17). De 3 a 6 meses, dos equipos de producto (lecturas y órdenes) sobre un equipo de plataforma.

Para que eso escale, el camino dorado (golden path: ruta estándar y ya resuelta para crear y operar un servicio) es parte del roadmap desde la semana 1: plantilla de servicio con health, logs JSON, lectura de secretos, pipeline, manifiestos y alertas base; plantilla de adaptador con bulkhead y circuit breaker; formato corto de rutas del gateway. Despliegue por GitOps (el clúster se sincroniza con lo que declara el repositorio), con imágenes inmutables y despliegue progresivo con vuelta atrás.

Una etapa termina cuando sus funciones de aptitud pasan en producción, no cuando se despliega el código: p95 bajo 300 ms con tráfico real, cero llamadas directas del canal migrado al legado y cero órdenes atascadas sin alerta.

9. Cultura y liderazgo técnico

Conclusión. Las cinco respuestas comparten un hilo: hacer explícito lo que suele quedar implícito (supuestos, decisiones, costos, criterios) y convertir los acuerdos en mecanismos automáticos para que no dependan de la memoria ni de la autoridad de nadie. Cada respuesta se sostiene con la evidencia que deja este mismo trabajo.

9.1 Ambigüedad: cómo reducir riesgo sin frenar el avance

No espero a tener todas las respuestas: convierto cada duda en un par de pregunta y supuesto, avanzo sobre el supuesto y diseño para que el error sea barato. Lo importante es saber qué decisión cambia si el supuesto resulta falso, y aislar esa decisión.

Evidencia en este trabajo: el AS-IS no venía dado. Lo formulé como hipótesis y cada duda quedó registrada como pregunta, supuesto y decisión que cambia (sección 2). El riesgo R4 dice qué pasa si el supuesto principal es falso (que el OMS ya orquesta) y por qué el daño está acotado: la saga vive en un solo servicio y se puede reducir a un pase directo. La validación de ese supuesto está agendada en la semana 1 del MVP.

Dos casos de la construcción, con la misma mecánica. El OMS necesita la dirección de entrega y la regla es que ningún dato personal salga por Kafka; el contrato decía que quien la necesite la pida a order-service, pero ninguna operación lo permitía, y cada salida chocaba con una regla de arquitectura. En lugar de torcer una regla en silencio o frenar el avance, se escribieron cuatro opciones con lo que cada una sacrifica y una propuesta (ADR-025); el ADR se aceptó el 15/11/2026 y no está implementado; hasta entonces el límite queda dicho en este documento: la orden se registra en el OMS sin dirección (sección 4.4). Y el formato de los identificadores de la orden, que el diseño no fijaba, se registró como supuesto explícito (S27) y se verificó contra los datos que hoy se envían (1.143 SKU de la semilla, 5 ubicaciones, 4 canales y 200 clientes del CRM) antes de convertirlo en regla del contrato (ADR-026).

9.2 Cómo documentar decisiones para escalar el equipo

Una decisión que solo vive en una conversación se vuelve a discutir cada vez que entra alguien nuevo. Documento cada decisión de arquitectura en un ADR (architecture decision record: nota corta con contexto, opciones, decisión y costo), junto al código, y mantengo los contratos como fuente de verdad de la que se genera la documentación para terceros.

Evidencia en este trabajo: 27 ADR con contexto, opciones, decisión, qué se sacrifica y cuándo se revisaría (anexo D); contratos OpenAPI, AsyncAPI y Protocol Buffers que el pipeline valida en cada cambio; diagramas en texto versionados junto al código; y una plantilla de servicio que hace que un equipo nuevo no tenga que decidir nada de plataforma. Antecedente externo: el caso publicado por KrakenD sobre Linex Travel, donde la configuración del gateway como código deja el historial de cambios como registro de quién cambió qué.

Tres hechos del último tramo. El ADR-027 trae la fecha de consulta de cada una de sus nueve fuentes y, donde AWS no publicaba el precio de una opción (el EC2 Instance Connect Endpoint), dice "por medir" en lugar de estimarlo. El ADR-026 deja escrito cómo se encontró el defecto, con los números de la simulación, para que quien lo lea no tenga que reconstruir el razonamiento. Y una falla de herramienta que se repitió cinco veces el 10/11/2026 (editar archivos con comandos de una línea que terminaban sin error y sin escribir nada) quedó como regla en el archivo de reglas del repositorio (`CLAUDE.md`), junto con una segunda falla de la misma familia que ocurrió dos veces ese día: la regla es la memoria del equipo, no la conversación en la que se descubrió.

9.3 "Rápido" frente a seguridad y operación: cómo se negocia

Negocio el alcance funcional, no los mínimos de seguridad ni de operación. Cuando hay presión de tiempo, propongo entregar menos funciones completas en lugar de todas las funciones a medias, y hago visible lo que se pospone y su fecha.

Evidencia en este trabajo: el MVP recorta alcance (un partner, sin CRM, sin devoluciones, sin portal) pero no seguridad ni observabilidad: "no hay versión reducida de seguridad" (sección 8.2). La lista de lo que el MVP no entrega está escrita para que nadie la descubra tarde.

Dos hechos de la construcción, uno de costo y uno de seguridad. El costo en la nube se trata como un requisito: el clúster se enciende por horas y se apaga. El 14/11/2026 estuvo encendido 2 h 25 min (17:07 a 19:32 UTC), cerca de 1,30 dólares estimados a 0,54 la hora; el costo real queda por medir en Cost Explorer, la herramienta de AWS que muestra el gasto. El apagado no se da por hecho: `make infra-audit` comprobó que no quedaban "clúster, nodos, NAT, balanceador, discos ni IP elásticas del demo", y lo

que sigue encendido, la base de la cuenta, cuesta 0,29 dólares por día (`docs/plan.md`, F7). En seguridad, la API de Kubernetes se dejó solo privada, con un bastión de unos 0,005 dólares por hora, aunque el endpoint público limitado por IP era gratis y más cómodo; la pérdida de comodidad y el costo están en el ADR-027, y el respaldo para el día de la presentación está escrito.

9.4 Desacuerdos técnicos fuertes: cómo se gestionan

Saco la discusión del terreno de las preferencias: acordamos primero los criterios, verificamos los hechos en la documentación oficial o con una medición, decidimos, y registramos lo que se sacrifica y bajo qué condición se revisaría. Quien no gana la discusión sabe exactamente qué evidencia la reabriría.

Evidencia en este trabajo: Kong frente a KrakenD se comparó con criterios escritos y verificados en documentación, y el ADR registra lo que se sacrifica y cuándo se revisaría (sección 3.4). Igual con SNS y SQS frente a Kafka, y con agregar un servicio en Go o Rust por rendimiento: decide una medición (la función de aptitud F1), no una preferencia.

Dos casos con el mismo método. En el detector de tarjetas (ADR-026) había cuatro opciones, y la que parecía más simple, quitar el detector, era también la que sacaría a la capa del alcance reducido de PCI DSS; se decidió con una medición y no con una opinión: de 2.000.000 de llaves al azar, 4.315 (una de cada 463) caían en el detector y rechazaban una compra válida. Quedó registrado lo que se sacrifica (una tarjeta puesta a propósito en un identificador no se detecta) y por qué se acepta. Para entrar a Kubernetes (ADR-027) se compararon cuatro caminos con su costo por hora y con lo que cada uno sacrifica, y la elección más cara que la gratuita se justificó por escrito, con el camino anterior como respaldo.

9.5 Calidad sin volver lento el delivery

La calidad que depende de revisiones manuales escala mal y frena. La convierto en compuertas automáticas y en un camino dorado: lo correcto es lo más fácil de hacer, y lo incorrecto lo detiene una prueba sin esperar a una persona. Ese es el criterio de diseño; la sección 3.8 dice cuáles ya son compuerta y cuáles todavía no.

Evidencia en este trabajo: diez funciones de aptitud definidas y clasificadas con honestidad (sección 3.8): una compuerta hoy en el pipeline (F5, más el lint de contratos, los secretos y la redacción), siete pruebas ejecutables que pasan cuando se corren y entran al pipeline según el roadmap, y dos todavía por construir (F3 y F4); plantilla de servicio y de adaptador; y el gateway modular, que deja a cada equipo cambiar sus rutas sin esperar a plataforma, con políticas que el pipeline exige (ninguna ruta sin scope, sin transformaciones, timeouts acotados). Antecedente externo: en el caso de Linex Travel publicado por KrakenD, el tiempo para lanzar una variante de producto pasó de meses a semanas.

Cuatro hechos de la construcción, con sus fuentes:

- Una prueba intermitente que destapó un defecto real. La prueba F9 (la misma orden enviada 5 veces en paralelo) fallaba de vez en cuando con `400, 409, 409, 409, 409` y cero órdenes. En lugar de repetirla hasta que pasara, se reprodujo a voluntad y se halló un defecto de la capa: el detector de tarjetas rechazaba una de cada ~450 compras válidas de la tienda, y el reintento con la misma llave recibía el mismo 400, guardado por la idempotencia. Terminó en el ADR-026, con una prueba de 100.000 llaves al azar sin un solo rechazo (antes, entre 225 y 246) y una comprobación en `make verify` que falla si la lista de campos y el contrato difieren. Después, `make f4-e2e` tres veces seguidas en 105 de 105 y la orden duplicada por reintento en 0 de más en 20 (commits `3460176` y `9d006f6`; `tests/load/results/to-be/orden-duplicada-20261115T033312Z.json`).
- Una prueba que pasaba por suerte de tiempos. `make f4-webhooks-test` exigía que `payment.approved` llegara primero, aunque el pago viaja por otro tópico que las órdenes: pasaba si el consumidor de pagos llegaba antes, y solo 2 de 4 corridas salieron limpias. Antes de corregir se midió: los registros de una corrida fallida mostraron que el consumidor de pagos entró al grupo 3 s antes que el de órdenes, y en otra, reproducida en local, las transacciones de un evento de orden y de un pago empezaron con 2 ms de diferencia. En los dos casos el orden de entrega era el que el diseño ya describía. Se corrigió la prueba, no el servicio, porque ordenar por `time` dentro del servicio obligaría a retener entregas; cinco corridas seguidas dieron 45 de 45 (commit `a540942`; `services/webhook-service/README.md`, hueco 4).
- Un hueco de despliegue encontrado al revisar qué llegaba a la nube: Keycloak y KrakenD, el borde de la capa, no tenían todavía paquete de despliegue para EKS, y la malla, que pasaba todas las validaciones sin clúster, falló en vivo en varios puntos (una ruta de llaves bloqueada, cuotas sin descriptor, una llamada que el sidecar cerraba). De ahí salió una prueba en vivo, `make mesh-kind-test`, que hoy cubre la cuota por partner, el mTLS estricto y el circuit breaker. El hueco del borde está registrado en el plan y su cierre está en curso (`docs/plan.md`; `tests/e2e/mesh/README.md`).
- Ensayar gratis antes de gastar. Al probar el despliegue en un clúster local apareció un choque entre el volumen de credenciales de EKS Pod Identity y la ruta de secretos del contrato de plataforma: el contenedor no arrancaba. Se encontró y se resolvió antes

de encender nada en AWS (ADR-024, contexto).

Anexos

Anexo A. Amenazas 7 a 9

#	Amenaza	Mitigación
A7	Un partner registra una URL de webhook interna para alcanzar la red privada (SSRF, server-side request forgery: lograr que el servidor haga peticiones a destinos que elige el atacante)	Implementado: solo HTTPS, y resolución con bloqueo de rangos privados al crear la suscripción y antes de cada envío. Diseño, sin implementar en el demo: salida por un egress controlado (hoy webhook-service es el único servicio de la capa sin restricción de salida en la malla) y verificación de propiedad de la URL
A8	Compromiso de un servicio y movimiento lateral hacia el legado	mTLS estricto, políticas de mínimo privilegio por identidad de carga, solo los adaptadores tienen ruta y credenciales al legado, secretos por adaptador
A9	Manipulación del precio o del monto en la orden	La capa es autoridad del precio y recalcula el total; `PRICE_MISMATCH` si no coincide

Anexo B. Métricas y alertas complementarias

#	Métrica	Dimensiones	Para qué
M11	Duración de la saga (de 202 a confirmada)	Canal	Experiencia del canal
M12	Reservas activas, vencidas por minuto, rechazos por falta de stock	Canal	Sobreventa evitada, cobros que no llegan
M13	Tasa de aprobación de pagos	Canal, medio	Negocio y pasarela
M14	Órdenes con `omsSync` pendiente y su antigüedad		Salud del OMS
M15	Entregas de webhooks: éxito, latencia, intentos	Partner	Integración del partner
M16	Mensajes en DLQ	Tópico	Trabajo manual pendiente
M17	Deriva de inventario en la conciliación	Ubicación (los SKU van al log)	Calidad del dato del WMS

#	Alerta	Condición	Severidad
AL7	DLQ con mensajes	Cualquier DLQ mayor a 0	Aviso
AL8	Lag de consumidores	Creciente por 10 minutos	Aviso
AL9	Webhooks de un partner fallando	Más de 20% de fallos en 15 minutos	Aviso, y notificación al partner
AL10	Pagos	Tasa de aprobación cae 20 puntos frente a la misma hora de la semana anterior	Página

Tiempos objetivo de la saga: `POST /v1/orders` hasta el 202, p95 menor a 500 ms; reserva gRPC, p95 menor a 50 ms; de pago aprobado a orden confirmada, p95 menor a 5 s; de evento a webhook entregado, p95 menor a 10 s.

Anexo C. Catálogo de eventos

Tópico	Clave	Tipo	Productor	Consumidores	Eventos
`raw.erp.changes`	Tabla y llave	Crudo, 7 días	erp-adapter	erp-adapter	Cambios sin traducir
`raw.wms.changes`	SKU y ubicación	Crudo, 7 días	wms-adapter	wms-adapter	Cambios sin traducir
`andes.catalog.products.v1`	`sku`	Estado, compactado	erp-adapter	pricing, order, inventory	`ProductChanged`
`andes.pricing.prices.v1`	`sku|channel|priceList`	Estado, compactado	erp-adapter	pricing, order	`PriceChanged`
`andes.inventory.stock-levels.v1`	`sku|locationId`	Estado, compactado	wms-adapter	inventory	`StockLevelChanged`

Tópico	Clave	Tipo	Productor	Consumidores	Eventos
`andes.sources.sync.v1`	`system`	Estado, compactado	erp-adapter, wms-adapter	pricing, inventory	`SourceSynced`: hasta dónde está al día cada sistema de origen
`andes.inventory.reservations.v1`	`orderId`	Hechos, 14 días	inventory	order	`StockReserved`, `ReservationConfirmed`, `ReservationReleased`, `ReservationExpired`
`andes.orders.events.v1`	`orderId`	Hechos, 30 días	order	webhook, oms-adapter, crm-adapter (después del MVP; no existe todavía)	`OrderCreated`, `OrderConfirmed`, `OrderCancelled`, `OrderDispatched`, `OrderDelivered`
`andes.payments.commands.v1`	`orderId`	Comandos, 7 días	order	payment	`PaymentRequested`, `RefundRequested`
`andes.payments.events.v1`	`orderId`	Hechos, 30 días	payment	order, webhook	`PaymentApproved`, `PaymentRejected`, `PaymentRefunded`
`andes.oms.commands.v1`	`orderId`	Comandos, 7 días	order	oms-adapter	`CreateOmsOrder`, `CancelOmsOrder`
`andes.oms.events.v1`	`orderId`	Hechos, 14 días	oms-adapter	order	`OmsOrderCreated`, `OmsOrderFailed`
`andes.fulfillment.events.v1`	`orderId`	Hechos, 30 días	wms-adapter	order, inventory	`DispatchStarted`, `DeliveryConfirmed`
`andes.audit.v1`	`correlationId`	Hechos, archivo permanente	Todos	Archivo de auditoría	Registro de auditoría

Además, cada grupo consumidor de un tópico de hechos o de comandos tiene su `<tópico>.<grupo>.retry` y su `<tópico>.<grupo>.dlq` (11 pares). Particiones, réplicas, limpieza y retención de los 36 tópicos se declaran en el contrato AsyncAPI y de ahí se crean (ADR-017).

Dimensionamiento inicial: 12 particiones para los tópicos de estado de stock y precio, 6 para órdenes, pagos y despacho, 3 para comandos y crudos. Factor de replicación 3, `min.insync.replicas` 2, productores con `acks=all` e idempotencia activada.

Eventos que salen a los partners por webhook:

Requisito del enunciado	Tipo del evento
Orden confirmada	`com.andesretail.order.confirmed.v1`
Pago aprobado	`com.andesretail.payment.approved.v1`
Pago rechazado	`com.andesretail.payment.rejected.v1`
Despacho iniciado	`com.andesretail.fulfillment.dispatch-started.v1`
Entrega confirmada	`com.andesretail.fulfillment.delivered.v1`
Orden cancelada (adicional)	`com.andesretail.order.cancelled.v1`

Ejemplo de webhook:

```
POST /andes/webhooks HTTP/1.1
Host: partner.example.com
Content-Type: application/cloudevents+json
X-Andes-Event-Id: evt_01JC8ZD3K7
X-Andes-Signature: t=1794334000,v1=5d41402abc4b2a76b9719d911017c592...

{
  "specversion": "1.0",
  "id": "evt_01JC8ZD3K7",
  "type": "com.andesretail.order.confirmed.v1",
  "source": "/order-service",
  "subject": "ord_01JC8ZB7Q2X9",
  "time": "2026-11-10T18:05:12Z",
  "datacontenttype": "application/json",
  "correlationid": "5e1d0c7a-93b2-4c41-8d0e-6a7f2b9c3d15",
  "data": {
    "orderId": "ord_01JC8ZB7Q2X9",
    "externalOrderId": "MKP-778899",
    "status": "CONFIRMED"
  }
}
```

La firma es HMAC-SHA256 (HMAC: firma con secreto compartido que permite verificar origen e integridad; SHA-256: el algoritmo de resumen criptográfico que usa) de `t + "." + cuerpo` con el secreto de la suscripción; el partner debe rechazar marcas de tiempo de más de 5 minutos. El partner responde 2xx en menos de 5 segundos o cuenta como fallo. La entrega es al menos una vez y el partner deduplica por `id`.

Anexo D. Registro de decisiones (ADR)

ADR	Estado	Decisión	Qué se sacrifica
001	Aceptado el 10/11/2026	KrakenD como gateway delgado, en lugar de Kong	Las cuotas por partner viven un salto más adentro
002	Aceptado el 10/11/2026	Kafka como transporte de eventos, en lugar de SNS con SQS	Reintentos y DLQ se construyen; más costo y operación
003	Aceptado el 10/11/2026	La capa es dueña del stock vendible y de las reservas	Única excepción a no poseer datos de negocio; garantía parcial frente a ventas fuera de la capa
004	Aceptado el 10/11/2026	Saga orquestada en order-service	order-service es el punto de mayor acoplamiento lógico; estado y recuperación son código propio
005	Aceptado el 10/11/2026	Modelo de lectura por eventos para stock y precio	Dato con segundos de desactualización; más piezas en la alimentación
006	Aceptado el 10/11/2026	Rate limit en dos niveles: KrakenD y Envoy	Dos lugares de configuración; un servicio de cuotas más
007	Aceptado el 10/11/2026	Sync solo lo que la capa responde con sus datos; async lo que depende de terceros	Crear orden responde 202, no el resultado final
008	Aceptado el 10/11/2026	Istio con sidecars para capacidades transversales	Memoria y latencia por sidecar; complejidad de la malla
009	Aceptado el 10/11/2026	Secretos en AWS Secrets Manager con Secrets Store CSI Driver	El manifiesto nombra al proveedor; la rotación no es instantánea
010	Aceptado el 10/11/2026	Java con Spring Boot en dominio, Node.js con Express en adaptadores, gRPC en un punto, y en dos por el ADR-025 (aceptado, sin implementar)	Memoria y arranque de la JVM (la máquina virtual de Java); dos cadenas de herramientas
011	Aceptado el 10/11/2026	Versión mayor en la ruta y errores con Problem Details	Tentación de duplicar código al llegar `/v2`
012	Aceptado el 10/11/2026	Configuración modular del gateway, ensamblada por el pipeline	Los equipos no usan funciones avanzadas; ensamblador propio

ADR	Estado	Decisión	Qué se sacrifica
013	Aceptado el 10/11/2026	Pasarela del demo: Stripe en modo de prueba más pasarela simulada	Tarjetas con autenticación adicional no soportadas en el demo
014	Aceptado el 10/11/2026	Contrato gRPC de inventario alineado con el estándar de estilo de buf	Tres mensajes de respuesta casi idénticos
015	Aceptado el 10/11/2026	order-service valida el precio con su propia vista, alimentada por el tópico de precios; ventana igual al vencimiento de la reserva	Dos copias derivadas del precio; durante la ventana se vende a un precio que ya no rige, salvo corrección
016	Aceptado el 10/11/2026	Todo error del borde sale en Problem Details: el gateway de entrada de Istio normaliza lo que KrakenD y Envoy emiten sin cuerpo	Un filtro Lua propio en el borde, con su prueba
017	Aceptado el 10/11/2026	Particiones, réplicas, limpieza y retención de cada tópico, y sus tópicos de reintento y DLQ, viven en el contrato AsyncAPI	El contrato cambia con cada ajuste operativo
018	Aceptado el 10/11/2026	Cada adaptador publica una marca de sincronización de su sistema de origen; `asOf` significa "vigente hasta" y `stale` detecta la caída del legado aunque nada cambie	Un canal y un mensaje más; `asOf` deja de ser la fecha del último cambio
019	Aceptado el 11/11/2026	inventory-service consume el catálogo y responde en `notFound` los SKU fuera del surtido del canal, igual que precio	Un consumidor y una tabla más en inventory
020	Aceptado el 11/11/2026	El evento de precio lleva el calendario de vigencias del alcance y el precio se elige al leer: la promoción programada entra y sale sola	El adaptador guarda las ventanas de cada alcance; la regla se implementa en pricing y en order
021	Aceptado el 11/11/2026	Métricas de negocio en formato Prometheus en `/metrics`, leídas por la plataforma; sin etiquetas de valores sin límite	Dos caminos de telemetría: trazas por envío y métricas por lectura
022	Aceptado el 15/11/2026; reemplazado en parte por 027 (endpoint de Kubernetes)	Infraestructura del demo en AWS con OpenTofu: raíces separadas, estado local cifrado, secretos de solo escritura, un rol por carga, EKS mínimo y apagado por interruptores	Un solo NAT; la llave del certificado de origen en el estado cifrado
023	Aceptado el 11/11/2026 (en implementación)	Cada servicio tiene su espacio de llaves en Redis: prefijo con su nombre y un usuario que solo accede a ese prefijo	Sin aislamiento de rendimiento entre servicios; un usuario y un secreto más por servicio
024	Aceptado el 15/11/2026	Cargas en Kubernetes con un chart genérico ensayado en kind, secretos en `/etc/andes/secrets` por Pod Identity, y Kafka en Confluent Cloud con tópicos y permisos sacados del AsyncAPI	Ruta de secretos distinta de Docker Compose; llaves de Confluent en el estado cifrado; las 151 particiones del contrato cuestan 0,70 dólares por hora
025	Aceptado el 15/11/2026 (sin implementar)	La dirección de entrega llega al OMS por un ticket de reclamo: oms-adapter la pide a order-service por gRPC, sin pasar por Kafka	Una segunda llamada síncrona interna
026	Aceptado el 14/11/2026	Los identificadores de la orden tienen formato declarado en el contrato y salen del detector de tarjetas, que queda solo en el texto libre y en `payment.token`	Una tarjeta puesta a propósito en un identificador no se detecta; el contrato es más estricto en esos campos
027	Aceptado el 14/11/2026; reemplaza en parte el 022	La API de Kubernetes es solo privada y se entra por un túnel de Session Manager por un bastión, con el endpoint público limitado por IP como respaldo	Un recurso más que cobra por hora (unos 0,005 dólares) y depende del NAT; todo lo que habla con la API necesita el túnel abierto
028	Aceptado el 14/11/2026	Kafka en un pod del clúster para desarrollo, ensayos y respaldo, y Confluent Cloud del 15/11/2026 al demo, con un interruptor de configuración (`KAFKA_MODE`, con valores `pod` o `confluent`)	En modo `pod`, un solo nodo sin réplicas y sin permisos por servicio; lo propio de Confluent se prueba con poco margen antes del demo
029	Aceptado el 15/11/2026	Los datos sensibles se enmascaran con `--data_redacted--` al pasar a la DLQ; el resto del mensaje se conserva como el original	La DLQ no se reprocesa tal cual: el valor enmascarado se recupera del tópico de reintento o del origen
030	Aceptado el 15/11/2026	La entrada sigue con `Gateway` y `VirtualService` de Istio (`networking.istio.io/v1`), sin la Gateway API de Kubernetes	El estándar portable entre implementaciones; queda como evolución
031	Aceptado el 15/11/2026	Recorrido en vivo de una petición sobre el diagrama, reconstruido desde las trazas de Tempo, en `monitor.<zona>/recorrido`	Una carga más en el borde; el mapeo depende de nombres de servicio y de tópico, y la búsqueda por `correlationId` necesita una cabecera aún sin capturar en Java

ADR	Estado	Decisión	Qué se sacrifica
032	Aceptado el 15/11/2026	Panel de monitoreo de solo lectura (`monitor.andesretail.com`) sobre una pila de observabilidad en el clúster: Prometheus, Loki, Tempo, Grafana, OpenTelemetry y Kiali	Una réplica de cada pieza, 3 días de métricas y 2 de logs y trazas, sin Alertmanager; los almacenes sin mTLS dentro del clúster
033	Aceptado el 15/11/2026	Consola del demo en `control.<zona>`: cambia entre AS-IS y TO-BE, lanza los seis escenarios de k6 y apaga o degrada los simuladores, con escritura acotada en el clúster y auditoría	La consola puede crear políticas de autorización en `legacy` (Kubernetes no limita `create` por nombre); el correo del operador viene de Cloudflare Access sin verificar su firma
034	Aceptado el 15/11/2026	Kafka en pod y Confluent Cloud siempre arriba, con un comando de cambio rápido (`make kafka-switch`)	Memoria del pod siempre ocupada y Confluent cobrando aunque no se use; el cambio no lleva mensajes en vuelo, y se drena antes de salir del pod
035	Aceptado el 15/11/2026	La tienda ofrece dos caminos para escribir la tarjeta en Stripe, con un interruptor: el componente incrustado de Stripe y la página de pago alojada por Stripe; la saga, el contrato y las llaves de `payment-service` no cambian	Alojado: un salto fuera del sitio y una segunda llave secreta, acotada (no cobra ni reembolsa); incrustado: Stripe.js se carga en la página de pago

Estado de los registros: del 001 al 030, el 031, el 032, el 033, el 034 y el 035 están decididos o aceptados. El 022, el 024 y el 025 los aceptó Andrés el 15/11/2026; el 025 está aceptado y sin implementar. El 023 se decidió durante el desarrollo, con aviso, por un detalle que solo aplica el diseño, y está en implementación. Lo aplicado hasta hoy en AWS (la base y un clúster por horas, sección 3.9) se aprobó paso a paso, cada vez.

El texto completo de cada ADR (contexto, opciones, decisión, costo y cuándo se revisaría) está en `docs/adr/`.

Anexo E. Contratos

Método y ruta	Scope	Servicio	Requisito del enunciado
`GET /v1/stock`	`stock:read`	inventory-service	Stock por SKU, tienda y canal
`GET /v1/prices`	`prices:read`	pricing-service	Precio por SKU, canal y cliente
`POST /v1/orders`	`orders:write`	order-service	Crear orden
`GET /v1/orders/{orderId}`	`orders:read`	order-service	Consultar estado de orden
`POST /v1/webhooks/subscriptions`	`webhooks:manage`	webhook-service	Notificaciones
`GET /v1/webhooks/subscriptions`	`webhooks:manage`	webhook-service	Notificaciones
`DELETE /v1/webhooks/subscriptions/{id}`	`webhooks:manage`	webhook-service	Notificaciones

Cabeceras comunes: `Authorization: Bearer <JWT>` (obligatoria); `X-Correlation-Id` (opcional en la petición, siempre en la respuesta); `Idempotency-Key` (obligatoria en `POST /v1/orders`); `RateLimit-Limit`, `RateLimit-Remaining` y `RateLimit-Reset` en la respuesta; `Retry-After` en 429 y 503.

Contratos completos en el repositorio: `contracts/openapi/interop-api.v1.yaml` (OpenAPI), `contracts/asyncapi/eventos.v1.yaml` (AsyncAPI) y `contracts/proto/andesretail/inventory/v1/inventory.proto` (gRPC). Los tres pasan lint en cada cambio.

Anexo F. Escenarios de falla dirigida

Cada escenario es un script de carga (k6) más una configuración de los simuladores de ERP, WMS, OMS, CRM y pasarela, con criterios de éxito que hacen fallar la prueba. Hoy se ejecutan aparte; entran al pipeline a los 3 meses (sección 8.3). Resultados en la sección 6.8: los escenarios 1, 2 y 3 medidos contra los dos lados, y el 4 y el 5 verificados en el TO-BE. El escenario 6 depende de la cuota por partner en Envoy y se midió con la malla en un clúster local.

#	Escenario	Criterio de éxito en el TO-BE	Demuestra
1	Pico 10x de consultas	p95 menor a 300 ms; errores menores a 0,1%; llamadas al ERP y al WMS no superan la base más 10%	Modelo de lectura (flujo B)
2	ERP lento (3 s de latencia)	p95 de precio menor a 300 ms; se abre el circuit breaker del adaptador	Aislamiento

#	Escenario	Criterio de éxito en el TO-BE	Demuestra
3	ERP o WMS caído 3 minutos	Stock y precio responden 200; pasado el umbral llevan `stale: true`; dispara la alerta de frescura; al volver, la frescura se recupera sola	Degradación y runbook
4	La pasarela rechaza el pago	Orden `CANCELLED`, reserva `RELEASED`, el stock vuelve a su valor, llega el webhook `payment.rejected`	Compensación (flujo A)
5	Webhook de pago duplicado o perdido	Duplicado: un solo cambio de estado. Perdido: payment-service consulta a la pasarela y la orden se resuelve sola	Idempotencia y vencimiento
6	Un partner excede su cuota	El partner `bronze` recibe 429 con `Retry-After`; el p95 y los errores del partner `gold` no cambian	Cuotas por partner

Los escenarios 1, 2 y 3 se corren también contra el AS-IS para llenar la tabla de la sección 6.8; el AS-IS se midió además en AWS (14/11/2026) y el TO-BE en la nube está por medir.

Anexo G. Siglas y términos

Sigla o término	Significado
ACL (anti-corruption layer)	Capa anticorrupción: traduce entre el modelo propio y el de otro sistema
ADR (architecture decision record)	Nota corta de una decisión: contexto, opciones, decisión y costo
AS-IS y TO-BE	Situación actual y situación propuesta
ATP (available to promise)	Stock vendible: físico menos reservado menos colchón
AWS (Amazon Web Services)	La nube de Amazon
BOLA (broken object level authorization)	Falla al comprobar que el objeto pedido es de quien lo pide
C4	Modelo de diagramas de arquitectura en cuatro niveles
CDC (change data capture)	Lectura de cambios desde el log de una base de datos
CSI (Container Storage Interface)	Interfaz con la que Kubernetes monta volúmenes externos
DLQ (dead letter queue)	Cola de mensajes que agotaron sus reintentos
ECR (Elastic Container Registry)	Registro de imágenes de contenedor de AWS
EKS (Elastic Kubernetes Service)	El Kubernetes administrado de AWS
ERP (Enterprise Resource Planning)	Sistema central de gestión: catálogo, precios, inventario contable
ESB (Enterprise Service Bus)	Bus de integración centralizado
gRPC	Protocolo de llamadas remotas con contrato binario estricto
HMAC	Firma de un mensaje con un secreto compartido
IAM (Identity and Access Management)	Sistema de permisos de AWS
IETF (Internet Engineering Task Force)	Organismo que publica los estándares de internet (los RFC)
ISO 4217 e ISO 8601	Estándares de códigos de moneda y de fechas
JVM (Java Virtual Machine)	Máquina virtual en la que corre Java
JWKS (JSON Web Key Set)	Documento público con las llaves para verificar tokens
JWT (JSON Web Token)	Token firmado con los datos de quien llama
kind (Kubernetes in Docker)	Clúster de Kubernetes que corre en contenedores del equipo; ensaya el despliegue y la malla sin nube
KMS (Key Management Service)	Servicio de gestión de llaves de cifrado de AWS
Luhn	Suma de verificación con la que todo número de tarjeta cierra su último dígito; uno de cada diez números al azar la pasa
mTLS (mutual TLS)	Cifrado en el que ambos extremos se autentican con certificado
MVP (minimum viable product)	Primera entrega útil en producción
NAT (Network Address Translation)	Salida a internet de las subredes privadas de AWS

Sigla o término	Significado
NLB (Network Load Balancer)	Balanceador de red de AWS
OAuth2 y OIDC (OpenID Connect)	Estándares de autorización y de identidad
OMS (Order Management System)	Sistema de órdenes
OTel (OpenTelemetry)	Estándar abierto de logs, métricas y trazas
p95	Percentil 95
PCI DSS (Payment Card Industry Data Security Standard)	Estándar de seguridad de la industria de tarjetas
PII (personally identifiable information)	Datos que identifican a una persona
PKCE (Proof Key for Code Exchange)	Protección del flujo de autorización para aplicaciones sin secreto
RFC (Request for Comments)	Documento de estándar publicado por el IETF
SKU (stock keeping unit)	Código de cada producto vendible
SLO (service level objective)	Meta medible de calidad de un servicio
SHA-256	Algoritmo de resumen criptográfico; con HMAC firma los webhooks
SNS y SQS (Simple Notification Service y Simple Queue Service)	Servicios de notificaciones y colas de AWS
SSM (AWS Systems Manager)	Servicio de AWS para administrar instancias; su Session Manager abre sesiones y reenvía puertos sin abrir entradas a la red
SSRF (server-side request forgery)	Ataque que hace que el servidor llame a destinos del atacante
TI	Tecnología de la información: el área de sistemas
TLS (Transport Layer Security)	Cifrado de las conexiones, base de HTTPS
TTL (time to live)	Vigencia de un dato antes de vencer
UTC	Tiempo universal coordinado
UUID (universally unique identifier)	Identificador único universal
W3C (World Wide Web Consortium)	Consorcio que publica los estándares de la web
WMS (Warehouse Management System)	Sistema de bodega: stock físico y despachos