NEXA PLATFORM ENGINEERING

Interface Design Documentation Series

Microservices Integration Interface Design Document

Nexa Commerce Platform — Service-to-Service Contract Specification

Document ID	NEXA-ARCH-INT-2024-017
Version	2.3.0
Status	Approved
Classification	Internal — Confidential
Effective Date	10 February 2026
Last Revised	15 July 2026
Next Scheduled Review	15 January 2027
Author	Priya Anand, Staff Platform Architect, Nexa Platform Engineering
Reviewers	Marcus Feldt (Synapse Financial Systems) Deja Okonkwo (Vertex Supply Chain) Ravi Chandran (Nimbus Communications)
Approved By	Helena Brandt, VP Engineering, Nexa

This document specifies the synchronous and asynchronous integration contracts between Nexa-owned and partner-owned services on the Nexa Commerce Platform. Distribution is limited to Nexa staff and named partner engineering leads under existing NDA terms.

1 Introduction and Scope

1.1 Purpose

This document is the authoritative interface specification for integration between the services that make up the Nexa Commerce Platform: the synchronous REST contracts exposed through the platform API gateway, the asynchronous message contracts published on the **Meridian** event bus, and the error-handling and resilience policies every participating service must implement. Read alongside the parent architecture overview (`NEXA-ARCH-000`), but self-contained for implementing or consuming any interface described here.

1.2 Scope

In scope: request/response schemas and status codes for the `/v2` API surface; topic names, payload schemas, and delivery guarantees for the Meridian event bus; the shared error envelope; retry, backoff, and circuit-breaker policy; and the versioning and deprecation process. Out of scope: internal implementation details of any single service, database schemas, infrastructure-as-code, and UI-facing contracts, each covered by its own document.

1.3 Intended Audience

Software architects and senior engineers at Nexa and at partner organisations (**Synapse**, **Vertex**, **Nimbus**, **Apex**) who are building against, or operating, any service described in Section 2. Familiarity with REST, JSON, and publish/subscribe messaging is assumed.

1.4 Related Documents

Document	ID	Relationship
Nexa Commerce Platform — System Architecture Overview	NEXA-ARCH-000	Parent document; describes deployment topology and ownership boundaries
Meridian Event Bus — Operations Runbook	MER-OPS-114	Referenced for broker capacity, partitioning, and on-call escalation
Synapse Payment Gateway — Security Addendum	SYN-SEC-042	Referenced for PCI-scoped fields in payment payloads
Nexa API Gateway — Authentication Guide	NEXA-SEC-031	Referenced for token issuance and mTLS certificate rotation

Table 1: Documents referenced by this specification.

1.5 Terminology

Key Terms

Consumer A service that reads from a REST endpoint or subscribes to an event topic.

Producer A service that owns and emits a REST resource or an event topic.

Idempotency key A client-supplied unique token that lets a producer safely discard duplicate retries of the same logical request.

Envelope The fixed outer JSON structure wrapping every error response and every event payload on the Meridian bus, described in Sections 4 and 5.

2 System Architecture Overview

2.1 Context

The Nexa Commerce Platform is decomposed into six services, each owned by a single team, that cooperate to take an order from checkout through fulfilment notification. Cross-service calls are either synchronous, through a shared API gateway, or asynchronous, through the Meridian event bus. No service is permitted to call another service's private endpoint or database directly; every interaction crosses one of these two boundaries.

2.2 Service Inventory

Service	Owner	Responsibility
Order Orchestration Service	Nexa	Order lifecycle, saga coordination, order state machine
Payment Gateway Adapter	Synapse	Payment authorization, capture, refund, PCI-scoped tokenisation
Inventory Ledger Service	Vertex	Stock reservation, availability, warehouse allocation
Notification Dispatch Service	Nimbus	Transactional email, SMS, and push notification delivery
Customer Analytics Service	Apex	Event ingestion for funnel, cohort, and revenue reporting
Meridian Event Bus	Meridian	Shared Kafka-compatible broker; topic governance and retention

Table 2: Services participating in the integration described by this document.

2.3 Integration Topology

Figure 1 shows the two integration paths. Solid arrows are synchronous REST calls routed through the API gateway; dashed arrows are asynchronous publish/subscribe traffic on the Meridian bus. **Customer Analytics** is a bus-only consumer and exposes no inbound API of its own.

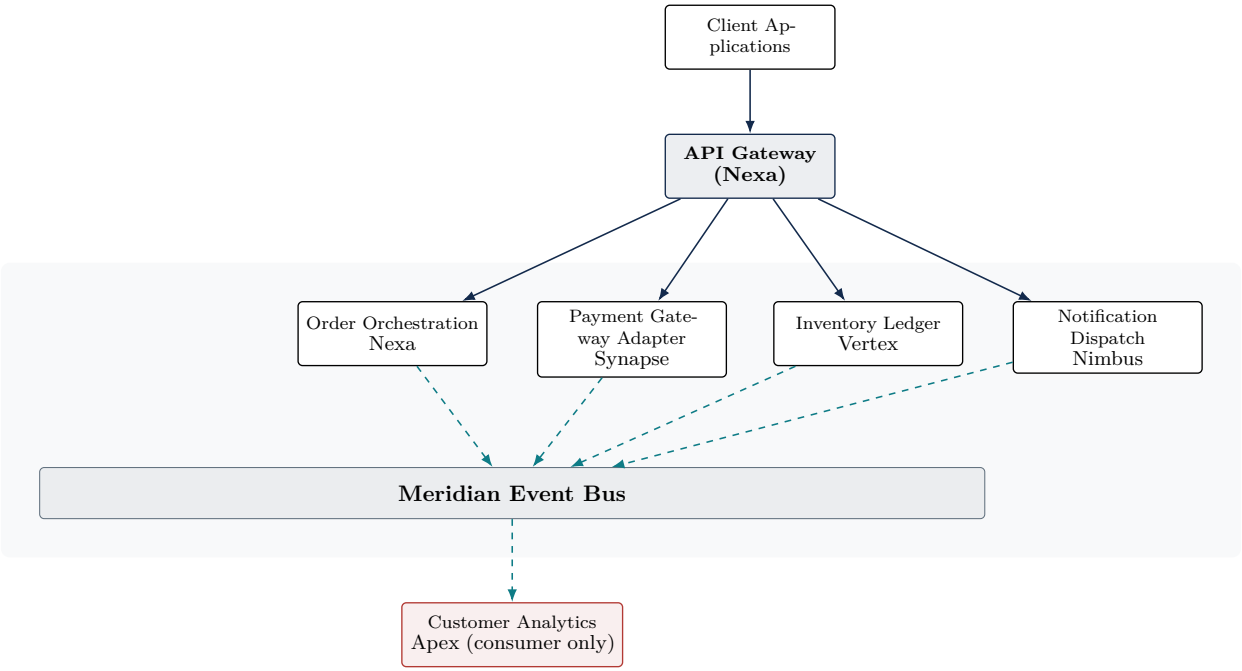

Figure 1: Synchronous (solid) and asynchronous (dashed) integration paths across the platform.

2.4 Deployment Environments

Every service is deployed independently to three environments. Base URLs below apply to the API gateway; the Meridian bus uses a separate bootstrap-server hostname per environment, listed in `MER-OPS-114`.

3 Synchronous API Contracts

Environment	Gateway Base URL	Purpose
Development	`https://api.dev.nexa.internal`	Feature branches, contract testing
Staging	`https://api.staging.letx-nexa.io`	Partner integration sign-off
Production	`https://api.nexa.io`	Live customer traffic

Table 3: Gateway base URLs by environment.

3.1 Gateway and Versioning

All synchronous traffic between services is routed through the Nexa API gateway and is versioned in the URI path, currently `/v2`. A new major version is introduced only for breaking changes; additive changes (new optional fields, new endpoints) ship within the current version. See Section 6 for the full versioning policy.

3.2 Authentication and Transport Security

Every gateway-routed call requires two layers of trust:

- **Mutual TLS** between the gateway and each downstream service, terminated at the service mesh sidecar, with certificates rotated every 90 days.
- **A short-lived JWT bearer token** (5-minute expiry) issued by the internal service-identity provider and validated by the gateway before the request is forwarded.

Partner-owned services (Synapse, Vertex, Nimbus) authenticate the same way; there is no separate "partner" auth path, which keeps the trust model uniform across ownership boundaries.

3.3 Endpoint Catalog

Method & Path	Description	Owning Service
`POST /v2/orders`	Create an order and reserve inventory atomically	Order Orchestration
`GET /v2/orders/{orderId}`	Retrieve order status and state-machine history	Order Orchestration
`POST /v2/payments/authorize`	Authorize payment for a given order	Payment Gateway Adapter
`POST /v2/payments/{paymentId}/capture`	Capture a previously authorized payment	Payment Gateway Adapter
`GET /v2/inventory/{sku}/availability`	Real-time stock availability by SKU and warehouse	Inventory Ledger
`POST /v2/notifications/dispatch`	Send a transactional notification	Notification Dispatch

Table 4: Primary gateway-routed endpoints in the `/v2` contract.

3.4 Sample Request and Response

The example below authorizes payment for an order. Every write endpoint requires an `Idempotency-Key` header; **Payment Gateway Adapter** stores the key for 24 hours and returns the original response, unchanged, for any retry that reuses it.

```
POST /v2/payments/authorize
Idempotency-Key: 8f14e45f-ceea-4c9a-b1e4-0d7e2f9a5b21

{
  "orderId": "ord_7f3c9a21",
  "amount": { "value": "129.00", "currency": "USD" },
```

```
  "paymentMethod": { "type": "card", "token": "tok_9c2e1a" },
  "capture": false
}
```

```
200 OK

{
  "paymentId": "pay_4b8d2e77",
  "orderId": "ord_7f3c9a21",
  "status": "authorized",
  "amount": { "value": "129.00", "currency": "USD" },
  "expiresAt": "2026-08-03T14:22:00Z"
}
```

3.5 Pagination and Rate Limiting

List endpoints use cursor-based pagination via a `cursor` query parameter and return a `nextCursor` field; page size is fixed at 50 and is not client-configurable. Every authenticated client is limited to 300 requests per minute per service, enforced at the gateway; the response headers `X-RateLimit-Remaining` and `X-RateLimit-Reset` report the current window. Exceeding the limit returns `429` with the error envelope described in Section 5.

4 Event-Driven Integration

4.1 Meridian Event Bus Overview

The Meridian event bus is a Kafka-compatible broker operated as a shared platform service. It carries all cross-service traffic that does not require an immediate response: state transitions, fulfilment events, and analytics telemetry. Producers own the schema of every topic they publish; consumers may not assume undocumented fields are stable.

4.2 Topic Catalog

Topic	Producer	Consumers	Retention
`order.created.v1`	Order Orchestration	Inventory Ledger, Analytics	7 days
`payment.authorized.v1`	Payment Gateway Adapter	Order Orchestration, Analytics	14 days
`inventory.reserved.v1`	Inventory Ledger	Order Orchestration	7 days
`notification.failed.v1`	Notification Dispatch	Analytics, Ops Alerting	30 days

Table 5: Topics on the Meridian bus relevant to order fulfilment. All topics use `orderId` or `sku` as the partition key, noted per topic in `MER-OPS-114`.

4.3 Payload Schema

Every event on the bus is wrapped in the same envelope as REST error responses (Section 5.1), with `data` carrying the topic-specific payload. The example below is a `payment.authorized.v1` event:

```
{
  "eventId": "evt_2a91cd6e",
  "eventType": "payment.authorized.v1",
  "occurredAt": "2026-08-03T14:20:11Z",
  "producer": "payment-gateway-adapter",
  "data": {
    "paymentId": "pay_4b8d2e77",
    "orderId": "ord_7f3c9a21",
    "amount": { "value": "129.00", "currency": "USD" },
```

```
    "authorizationExpiresAt": "2026-08-03T14:22:00Z"
  }
}
```

4.4 Delivery Semantics and Idempotency

The bus guarantees at-least-once delivery within a partition; it does not guarantee exactly-once delivery across consumer restarts. Every consumer is therefore required to be idempotent with respect to `eventId`: the recommended pattern is an upsert keyed on `eventId` into a local processed-events table with a 30-day retention window, matching the widest topic retention in Table 2. Consumers must not assume ordering across partitions; ordering is only guaranteed within a single partition key (`orderId` or `sku`).

Schema Evolution Rule

New fields may be added to `data` as optional at any time. Removing or renaming a field, or changing its type, requires a new topic version (e.g. `order.created.v2`) published alongside the old one for at least one full deprecation cycle, per Section 6.

5 Error Handling and Resilience Policy

5.1 Standard Error Envelope

Every non-2xx REST response, and every failure event published to the bus, uses the same envelope so that a client can write one error-handling path for the whole platform.

```
{
  "error": {
    "code": "ERR_RATE_LIMITED",
    "message": "Client exceeded 300 requests/minute for this service.",
    "retryable": true,
    "requestId": "req_c81f2e90",
    "details": { "retryAfterMs": 4000 }
  }
}
```

5.2 Error Code Registry

HTTP	Internal Code	Meaning	Retryable
400	`ERR_VALIDATION`	Request failed schema validation	No
401	`ERR_AUTH_INVALID`	Missing or expired bearer token	No
404	`ERR_NOT_FOUND`	Resource does not exist	No
409	`ERR_CONFLICT`	Idempotency key reused with a different payload	No
429	`ERR_RATE_LIMITED`	Client exceeded its rate limit	Yes
503	`ERR_UPSTREAM_UNAVAILABLE`	A required downstream dependency is unavailable	Yes

Table 6: Canonical error codes shared across every service in Section 2.2.

5.3 Retry and Backoff Policy

Clients retrying a `retryable: true` error must use exponential backoff with jitter, starting at 200 ms, doubling each attempt, capped at 5 s, with up to 20% random jitter added on top of the nominal delay. Figure 2 shows the resulting envelope over six attempts.

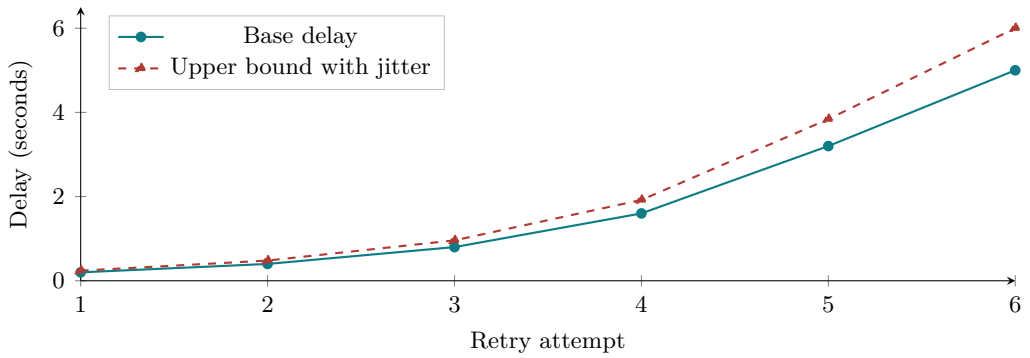

Figure 2: Exponential backoff envelope: nominal delay and worst-case delay with jitter, capped at attempt 6.

5.4 Circuit Breaker Configuration

Each service maintains a circuit breaker per downstream dependency it calls synchronously. Thresholds are evaluated over a rolling request window, not a fixed time interval.

Dependency	Failure Threshold	Open Duration	Half-Open Trials
Payment Gateway Adapter	50% of 20-req window	30s	3
Inventory Ledger Service	60% of 20-req window	20s	3
Notification Dispatch Service	70% of 30-req window	45s	5

Table 7: Circuit breaker thresholds by downstream dependency.

Policy

While a circuit is open, the calling service must fail fast with `ERR_UPSTREAM_UNAVAILABLE` rather than queue requests. Queuing during an open circuit is the single most common cause of cascading timeouts observed in past incident reviews and is explicitly disallowed.

5.5 Dead-Letter Handling

An event that fails processing after 5 consumer retries is moved to a `{topic}.dlq` topic with the original envelope plus a `failureReason` field, and retained for 30 days. Moving an event to a dead-letter topic does not remove it from the source topic; consumers must track their own offset to avoid reprocessing on restart.

6 Versioning and Change Management

6.1 Semantic Versioning Policy

Both REST endpoints and event topics are versioned independently using a major-version suffix (`/v2`, `order.created.v1`). A new major version is required for any breaking change: removing a field, changing a field's type, tightening validation, or changing the meaning of an existing status code. All other changes are additive and ship in place.

6.2 Deprecation Timeline

6.3 Change Notification Process

A breaking change proposal is opened as an RFC against this document at least four weeks before the target release, circulated to every owning team listed in Section 2.2, and requires sign-off from each consumer team before the new version is published. Non-breaking changes are announced in the `#platform-integration` channel and recorded in the changelog appendix of the parent architecture

Version	Status	Sunset Date	Notes
v1	Deprecated	2026-09-30	Superseded by the v2 idempotency-key model
v2	Current	–	Active; described throughout this document
v3 (draft)	Preview	–	Adds gRPC transport for internal-only calls; REST unaffected

Table 8: Contract version status as of the effective date of this document.

document (`NEXA-ARCH-000`); no sign-off is required.

6.4 Further Reading

The patterns applied throughout this specification, in particular the error envelope, the circuit breaker configuration, and the event schema evolution rule, follow established distributed-systems practice rather than platform-specific invention. See Fowler [2] for the service-decomposition rationale, Newman [4] for API and event contract design more broadly, Nygard [6] for the circuit breaker and bulkhead patterns applied in Section 5.4, Kleppmann [3] for the delivery-semantics discussion in Section 4.4, and [5] and [1] for the standards that informed the error envelope and future patch-style update endpoints, respectively.

References

[1] Paul C. Bryan and Mark Nottingham. Javascript object notation (json) patch. Technical Report RFC 6902, IETF, 2013.

[2] Martin Fowler and James Lewis. Microservices: A definition of this new architectural term, 2014.

[3] Martin Kleppmann. *Designing Data-Intensive Applications*. O'Reilly Media, 2017.

[4] Sam Newman. *Building Microservices: Designing Fine-Grained Systems*. O'Reilly Media, 2nd edition, 2021.

[5] Mark Nottingham and Erik Wilde. Problem details for http apis. Technical Report RFC 7807, IETF, 2016.

[6] Michael T. Nygard. *Release It! Design and Deploy Production-Ready Software*. Pragmatic Bookshelf, 2nd edition, 2018.