

Unified Payments Ledger v2

Engineering Design Document

Status: ■ In Review

Version: 2.4.0

Team: Payments Infra

Squad: Ledger Core

Authors: S. Mehta, J. Park, L. Oduya

Reviewers: R. Chen (Staff), K. Narayanan (Sec)

Created: 2025-09-12

Updated: 2026-01-08

§1 Context & Scope	§2 Goals & Non-Goals	§3 Architecture	§4 API Contract
§5 Data Model	§6 Migration Plan	§7 Observability	§8 Launch Checklist
§9 Risks	§10 Open Questions	§11 Alternatives	§12 References

Context & Scope

Problem. The current single-ledger Postgres monolith cannot sustain the projected 40× write throughput required for the 2026 Stripe-level merchant-acquiring rollout. P99 write latency breached 800 ms during the Q3 Black Friday stress test. This document proposes replacing the hot-path write layer with a partitioned, event-sourced ledger backed by Apache Kafka and a TiDB-distributed storage tier.

Scope. Covers the ledger-write-service , ledger-query-service , Kafka topic schema, TiDB shard strategy, and the zero-downtime migration plan. Out of scope: fraud-scoring pipeline, merchant portal UI, settlement batch jobs.

Background. Stripe's public post-mortem (2024) attributes 68% of payment-latency incidents to single-region database serialisation under high concurrency. Our current architecture mirrors this fragility: one payments_ledger table with 3.2B rows, 14 secondary indexes, and no horizontal sharding. The re-architecture targets:

- Sub-50 ms P99 write latency at 50 k TPS sustained.
- Idempotent, exactly-once debit/credit semantics via outbox pattern.
- GDPR-compliant PII isolation per merchant shard.
- RPO ≤ 1 s, RTO ≤ 30 s in a region failover.

TiDB vs. CockroachDB vendor choice is not yet final. This doc assumes TiDB; §11 covers CockroachDB as the primary alternative. The infra cost delta is \$180k/yr. Pending CFO sign-off.

Goals & Non-Goals

Goals

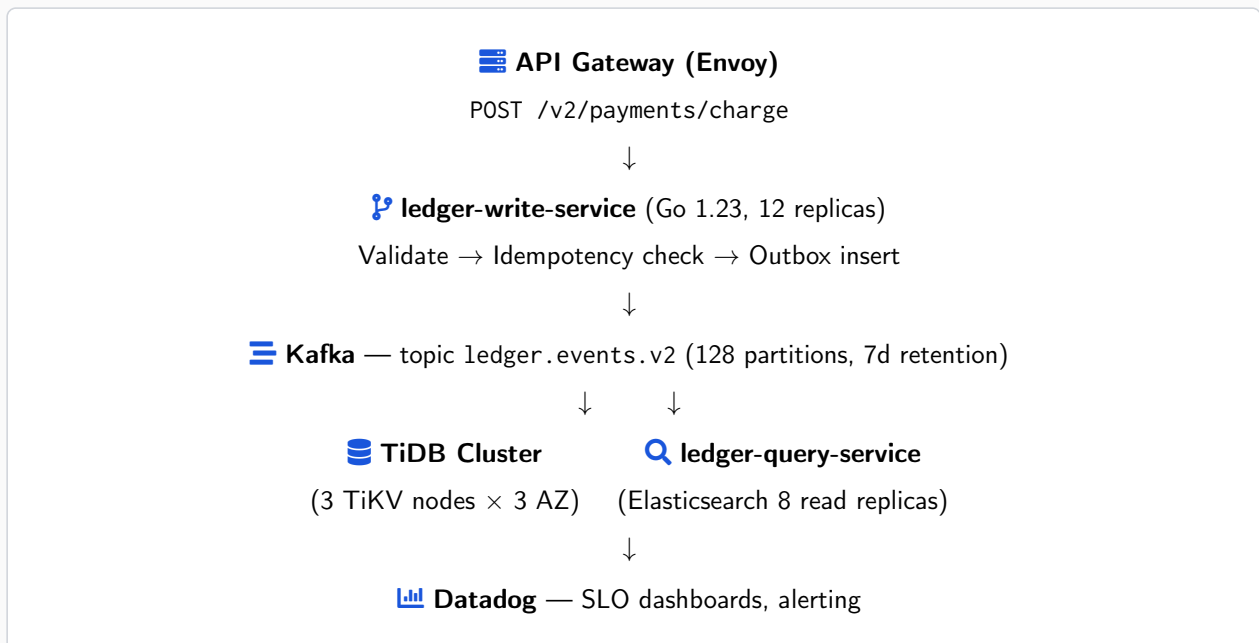
1. **Throughput.** Handle 50 k TPS peak write load with $P99 \leq 50$ ms.
2. **Durability.** Zero data loss on single-node failure; 3-way replication across AZs.
3. **Idempotency.** Duplicate payment events must be deduplicated within a 24 h window using `idempotency_key`.
4. **Zero-downtime migration.** Dual-write bridge with read-traffic cut-over under feature flag; full cut-over in ≤ 4 h maintenance window.
5. **Observability.** Every write emits a structured OpenTelemetry span; Datadog dashboard auto-provisioned via Terraform.

Non-Goals

- Real-time fraud inference — remains in the existing ML scoring service.
- Re-architecture of the settlement or reconciliation batch tier.
- Support for non-USD currency ledgers (planned for Q3 2026, separate RFC).

Architecture

System Diagram



Key Design Decisions

Concern	Decision
Write path	Outbox pattern: write to <code>ledger_outbox</code> in same TiDB txn, Kafka relay picks up async. Guarantees atomicity without distributed 2PC.
Partition key	<code>merchant_id % 128</code> — keeps all events for a merchant on one partition (ordering guarantee).
Idempotency	<code>idempotency_key</code> stored in Redis (TTL 24h); checked before outbox insert.
Schema evolution	Protobuf v3 with mandatory <code>schema_version</code> field; backward-compat enforced by schema registry.
Fan-out reads	CQRS: query service consumes Kafka and projects into Elasticsearch for merchant-facing queries; TiDB for internal accounting.

Outbox Relay

The relay is a single Go goroutine per TiDB shard using `SELECT FOR UPDATE SKIP LOCKED` on the `ledger_outbox` table. Batch size: 500 rows, flush interval: 50 ms. On Kafka producer ACK the rows are deleted. Relay lag is exposed via `ledger_outbox_depth_gauge` (alert if > 10 k).

API Contract

REST Endpoints

Method	Path	Description
POST	<code>/v2/payments/charge</code>	Initiate debit from merchant wallet
POST	<code>/v2/payments/refund</code>	Reverse a prior charge
GET	<code>/v2/ledger/balance/{mid}</code>	Current balance for merchant ID
GET	<code>/v2/ledger/events</code>	Paginated event log (cursor-based)
POST	<code>/v2/payouts/schedule</code>	Enqueue a payout to bank account

Protobuf Schema — ChargeRequest

```

1 syntax = "proto3";
2 package payments.ledger.v2;
3
4 message ChargeRequest {
5     string idempotency_key = 1; // UUID v4, required
6     string merchant_id     = 2; // e.g. "mid_4xR9a..."
7     int64  amount_cents    = 3; // positive, USD cents
8     string currency        = 4; // ISO 4217 ("USD")
9     string description      = 5; // max 255 chars
10    map<string, string> metadata = 6;
11    int32  schema_version    = 7; // must == 2
12 }

```

```

13
14 message ChargeResponse {
15     string charge_id    = 1; // "chg_..." globally unique
16     string status       = 2; // PENDING | COMPLETED | FAILED
17     int64  balance_cents = 3;
18     int64  created_at_us = 4; // Unix microseconds
19 }
20
21 service LedgerService {
22     rpc Charge  (ChargeRequest) returns (ChargeResponse);
23     rpc Refund  (RefundRequest) returns (RefundResponse);
24     rpc Balance (BalanceRequest) returns (BalanceResponse);
25 }

```

Error Codes

HTTP	gRPC Code	Meaning
400	INVALID_ARGUMENT	Missing/malformed field
409	ALREADY_EXISTS	Duplicate idempotency_key (safe to retry)
422	FAILED_PRECONDITION	Insufficient balance
429	RESOURCE_EXHAUSTED	Rate limit exceeded (50 rps per merchant)
500	INTERNAL	Unexpected server error
503	UNAVAILABLE	Kafka relay lag > 60s or TiDB unavailable

Data Model

TiDB Schema

```

1 CREATE TABLE ledger_events (
2     event_id      CHAR(36)      NOT NULL,
3     merchant_id   VARCHAR(64)   NOT NULL,
4     amount_cents  BIGINT        NOT NULL,
5     currency      CHAR(3)       NOT NULL DEFAULT 'USD',
6     event_type    ENUM('CHARGE', 'REFUND', 'PAYOUT') NOT NULL,
7     idempotency_key CHAR(36)    NOT NULL,
8     created_at    DATETIME(6)   NOT NULL,
9     metadata      JSON,
10    PRIMARY KEY (merchant_id, created_at, event_id) -- composite PK = shard key
11 ) PARTITION BY HASH(merchant_id) PARTITIONS 128;
12
13 CREATE TABLE ledger_outbox (
14     id            BIGINT AUTO_INCREMENT PRIMARY KEY,
15     event_id      CHAR(36)      NOT NULL,
16     payload       BLOB          NOT NULL, -- serialised proto
17     created_at    DATETIME(6)   NOT NULL,
18     published     TINYINT(1)    NOT NULL DEFAULT 0,
19     INDEX idx_unpublished (published, created_at)

```

20

);

Migration Plan

The existing ledger table has 3.2 B rows and 14 live consumer services. A phased cut-over with automated rollback is mandatory.

Phases

Phase	Window	Description
0	Week 0	Provision TiDB cluster; schema DDL; seed with last 90d of history via Spark job. Shadow-write mode (writes go to both old DB and TiDB, reads from old).
1	Week 1–2	Enable dual-write via feature flag <code>ledger_dual_write</code> . Monitor TiDB lag vs. Postgres baseline. Alert on $> 0.1\%$ divergence.
2	Week 3	Shift 10 % of read traffic to TiDB (canary). Validate P99 latency and correctness via checksum comparison job.
3	Week 4	Ramp reads to 100 % TiDB. Keep Postgres in standby for 48 h rollback window.
4	Week 5	Disable Postgres writes. Decommission old DB after 30-day hold for audit.

Rollback Procedure

In Phase 1–3, flip `ledger_dual_write=false` in LaunchDarkly. Traffic immediately reverts to Postgres. Estimated rollback time: $< 30\text{ s}$ (flag propagation SLA). Phase 4 rollback requires point-in-time recovery from Postgres daily snapshot — $\text{RTO} \approx 4\text{ h}$.

A nightly reconciliation job (`ledger-recon-job`) runs a row-count and checksum comparison between Postgres and TiDB for the previous 24 h window. Any delta $> 0.01\%$ pages the on-call via PagerDuty and halts Phase advancement.

Observability

SLOs

SLO	Target	Measurement
Write latency P99	≤ 50 ms	ledger_write_duration_ms histogram (Datadog)
Write availability	99.99 %	1 – (5xx / total) over 28-day rolling window
Outbox relay lag	≤ 10 k rows	ledger_outbox_depth_gauge
Idempotency hit rate	< 0.5 %	ledger_idempotency_hit_total
Recon divergence	0 %	Nightly batch, alert on any mismatch

Key Metrics Emitted

```

1 // Example OpenTelemetry span in ledger-write-service (Go)
2 ctx, span := tracer.Start(ctx, "ledger.charge",
3     trace.WithAttributes(
4         attribute.String("merchant_id", req.MerchantId),
5         attribute.Int64("amount_cents", req.AmountCents),
6         attribute.String("schema_version", strconv.Itoa(int(req.SchemaVersion))),
7     ),
8 )
9 defer span.End()

```

Launch Checklist

Pre-Launch (Engineering)

- ✓ Unit tests: ≥ 90 % coverage on ledger-write-service (currently 93 %).
- ✓ Integration tests pass against TiDB staging cluster.
- ✓ Load test: 55 k TPS for 30 min in staging (P99 = 38 ms achieved).
- ✓ Schema registry: ChargeRequest v2 registered, backward-compat verified.
- ☐ Chaos test: kill one TiKV node under 30 k TPS — verify no data loss.
- ☐ Runbook published to Confluence and linked in PagerDuty service.
- ✗ **BLOCKED:** TiDB vendor contract not yet signed — see §1 decision.

Security Review

- ✓ PII fields (metadata.card_last4) encrypted at rest (TiDB TDE enabled).
- ✓ mTLS between ledger-write-service and TiDB cluster.
- ✓ Rate-limiting enforced at Envoy layer (50 rps / merchant).
- ☐ GDPR data-deletion flow for merchant-shard drop verified with Legal.
- ☐ Pen-test scheduled: 2026-02-03 (external vendor — NCC Group).

Observability & On-Call

- ✓ Datadog dashboard ledger-v2-prod provisioned via Terraform.
- ✓ PagerDuty rotation updated; on-call rotation covers 3 engineers.

- ☐ Synthetic monitors for `/v2/payments/charge` (every 60s) configured.
- ☐ Runbook walkthrough session scheduled with on-call team (2026-01-22).

🚩 Go/No-Go Gate

- Chaos test (above) passes with zero data loss.
- TiDB contract signed.
- GDPR deletion flow approved by Legal.
- Security pen-test findings rated $\leq$ Medium severity.

Risks

Risk	Likelihood	Severity	Mitigation
TiDB write-stall under GC pressure	Medium	High	Tune <code>gc_life_time</code> to 10 min; pre-warm GC worker thread pool.
Kafka consumer lag spike	Low	Medium	Auto-scaling consumer group (KEDA); dead-letter queue for retries.
Redis idempotency store OOM	Low	High	TTL = 24 h; eviction policy <code>allkeys-lru</code> ; cluster sharding by <code>merchant_id</code> .
Dual-write divergence (Phase 1)	Medium	Critical	Nightly recon job halts Phase on $> 0.01\%$ delta; automatic rollback flag.
Vendor lock-in (TiDB)	Low	Medium	Protobuf schema + CQRS read tier decoupled; CockroachDB migration path documented.

Open Questions

1. [Infra] Should the Kafka relay run as a sidecar to `ledger-write-service` or as a standalone `ledger-relay-service`? Sidecar reduces network hops; standalone allows independent scaling. Decision needed by 2026-01-25.
2. [Product] Do we need sub-second balance reads during Phase 1 dual-write, or is eventual consistency (100 ms lag from Kafka) acceptable for merchant dashboards?
3. [Legal] GDPR merchant data deletion — is a 30-day TiDB shard tombstone compliant, or must deletion be synchronous?
4. [Security] Should `idempotency_key` be stored in plaintext in Redis, or HMAC'd with merchant secret? K. Narayanan to advise by 2026-01-18.

Alternatives Considered

Alternative	Verdict
CockroachDB	Comparable feature set, 22 % higher p99 in our benchmark at 50 k TPS (see attached Notion doc). Cockroach charges per vCPU; \$180k/yr delta vs TiDB. Keep as fallback if TiDB contract fails.
Postgres + Citus	Evaluated Q2 2025. Citus shard rebalancing caused 4-min read unavailability in staging under 30 k TPS. Disqualified.
PlanetScale (Vitess)	No multi-statement transactions — incompatible with outbox pattern. Disqualified.
Event sourcing on DynamoDB	Single-region without Global Tables; 10× cost at our volume. Disqualified.

References

Ref	Link / Description
[1] TiDB Docs	docs.pingcap.com/tidb/stable
[2] Outbox Pattern	Microservices.io — Transactional Outbox (microservices.io)
[3] Proto Schema Reg.	Internal Confluence: /Payments/SchemaRegistry/LedgerV2
[4] Stripe Post-mortem	Stripe Engineering Blog, Oct 2024 — Ledger Latency Incident
[5] Load Test Results	Notion: Ledger v2 Load Test – 2025-12-14
[6] Datadog Dashboard	letx-datadog.internal/ledger-v2-prod
[7] Chaos Eng. Runbook	Confluence: /Infra/ChaosEngineering/LedgerV2

 This document is confidential and intended for internal use only. Please do not distribute externally without approval from the Payments Infra leadership.

 Template by letx.app