CrossConnect
CONNECT. MONITOR. OPTIMIZE.
BY Cybr IQ

Data Flow Architecture

This document follows a single network fact through CrossConnect: how it enters, where it lands, how it is validated and committed to the source of truth (the records the system treats as authoritative), and how it leaves. It is written for the network and software engineers who integrate with, operate, or extend the platform, so it names the specific services, classes, ports, and ordered exchanges.

Audience: network and software engineering, platform integration, SRE

Scope: ingestion, staging, validation, source of truth, derived layer, output paths

Notation: Mermaid flowcharts (architecture / data flow) and sequence diagrams (ordered exchanges)

Document: technical reference, 21 June 2026

Contact: contact_us@cybriq.io

0 How to read this document

This reference traces a fact from the wire to a cited answer. Rather than describe behavior with adjectives, each section names the mechanism behind it: the class, the port, the table, the default. Where a behavior can be verified in the build, it is stated exactly. Where a control is a deployment option rather than a default, it is labelled as such.

GA shipped & on by default CONFIGURABLE shipped, operator-enabled

DEPLOYMENT OPTION supported integration / hardening you select at install

1 Scope & the core invariant

CrossConnect ships as one deployable Spring Boot application (Java 21, Spring Boot 3.4). It is backed by a single PostgreSQL system of record, plus an optional Batfish sidecar (a helper process) that analyzes the whole fleet's configuration. Every inbound path, whether a REST call, a Vaadin view action (REST is the API; Vaadin is the server-rendered UI), a scheduled sweep, or a passive network listener, funnels through the `@Service` layer, which owns all persistence and domain logic. This document follows the data, not the code call tree: where each fact comes from, which table holds it, what commits it to truth, and what reads it back out.

> **One rule underpins everything below.** A fact stays **observed** until it earns enough confidence to be **committed** to the source of truth. Staged observations are confidence-scored before anything is trusted, so an observation never becomes truth silently. The validation step (§8) is the trust boundary of the system, and `commit()` is the only write path that crosses it.

Navy nodes = in-app service / process Teal nodes = data store (Postgres)

Amber nodes = decision / trust gate White nodes = external actor or system

2 System context

CrossConnect reaches into the network only with outbound, read-only operations: it performs no packet capture and inspects no payloads. It accepts traffic summaries and operator input, and it emits cited answers, hash-chained audit entries (a tamper-evident log), signed webhooks, and

exports. Every edge in the diagram below is labelled with its transport and direction.

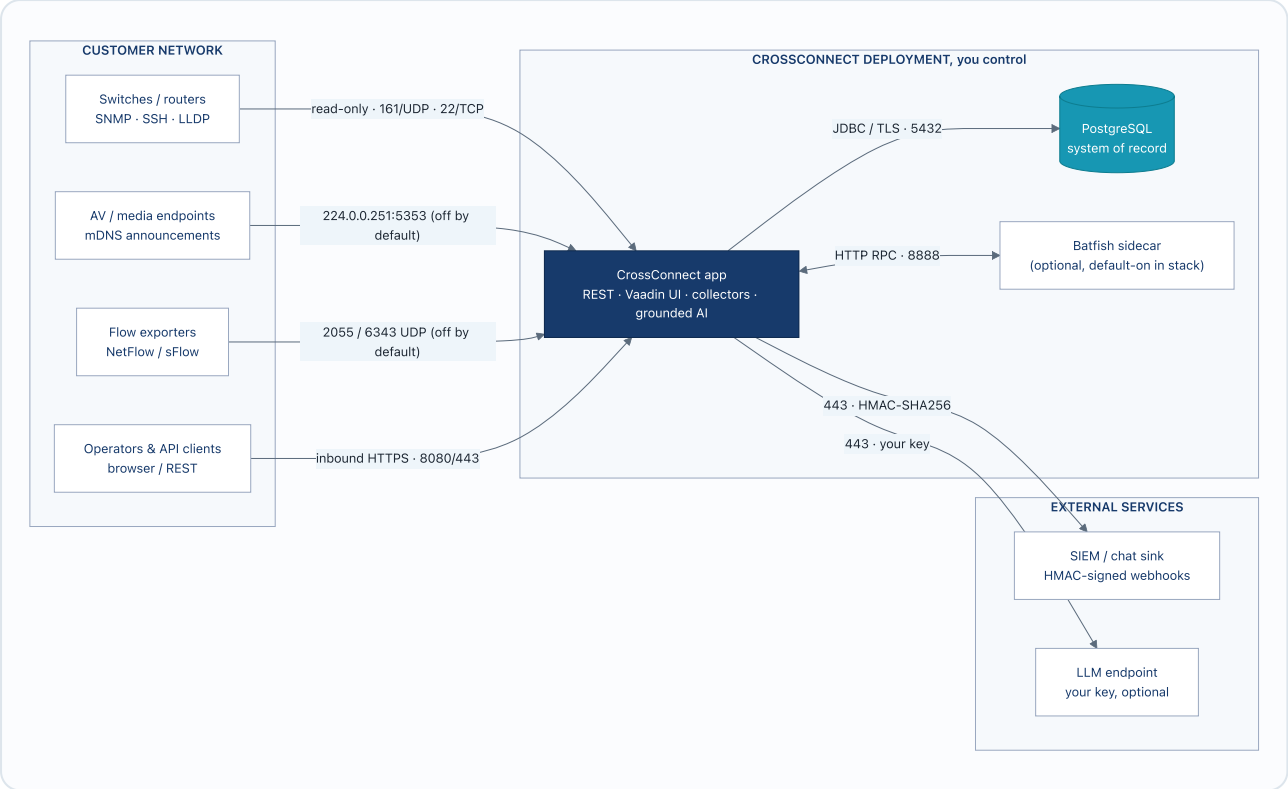

Figure 1. System context. Connections into the network are outbound and read-only. Operator and API traffic is inbound to the application port only. The source of truth, the keys, and the data never leave the deployment you control. Passive listeners (mDNS, NetFlow/sFlow) are off by default and bind no socket until enabled.

3 Runtime architecture

Inside the application, background work comes from scheduled sweeps and passive listeners, while foreground work enters through REST controllers and Vaadin views. Everything converges on the service layer, which is the sole owner of repositories and domain logic. Outbound effects then fan out through an in-process `EventBus` (a publish/subscribe channel inside the app) to the audit chain, webhooks, and sinks.

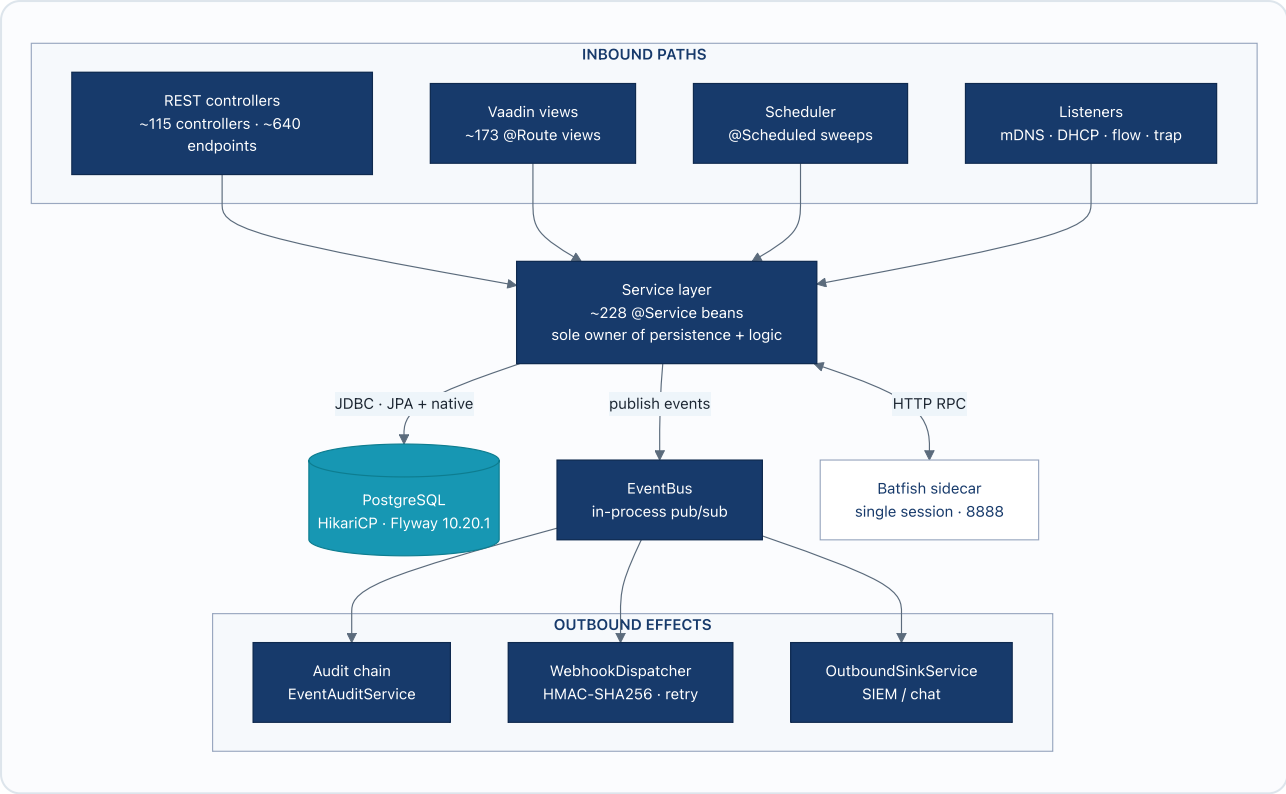

Figure 2. Runtime architecture. Every inbound path funnels through the service layer, which owns all persistence. Mutations publish to the in-process `EventBus`, which fans out synchronously to the audit chain, webhooks, and sinks.

Container	Responsibility	Transport / interface
REST controllers	OpenAPI surface (~115 `@RestController` / `@Controller`, ~640 mapped endpoints), tenant-scoped	`HTTPS · 8080/443`
Vaadin views	Server-rendered operator UI (~173 `@Route` views, Vaadin Flow 24.5.7), delegates to services	`HTTPS (server-rendered)`
Service layer	~228 `@Service` beans; sole owner of repositories and domain logic	`in-process`
Scheduler	`@Scheduled` sweeps: `DiscoveryWorker`, `StagingPurgeSweep`, `EventAuditPurgeSweep`, `WriteIntentSweep`	`in-process timers`
Listeners	Passive collectors: `MdnsListener`, `DhcpFingerprintListener`, `FlowListener`, `SnmpTrapListener` (see §5)	`UDP / multicast`
PostgreSQL	Single system of record; HikariCP pool; Flyway 10.20.1 migrations; JDBC 42.7.4	`JDBC · 5432 · JPA + native`
Batfish sidecar	Whole-fleet formal config model for reachability / ACL / IPAM analysis	`HTTP RPC · 8888 · single session`

4 The six-stage pipeline

End to end, every fact passes through six stages. Stages 1 and 2 are untrusted observation, stage 3 is the human trust gate, and stages 4 through 6 operate only on documented truth. Derived results (stage 5) are computed fresh from a snapshot of the source of truth, with no new facts stored, and are cached for a short window. Data never crosses the amber gate unattended.

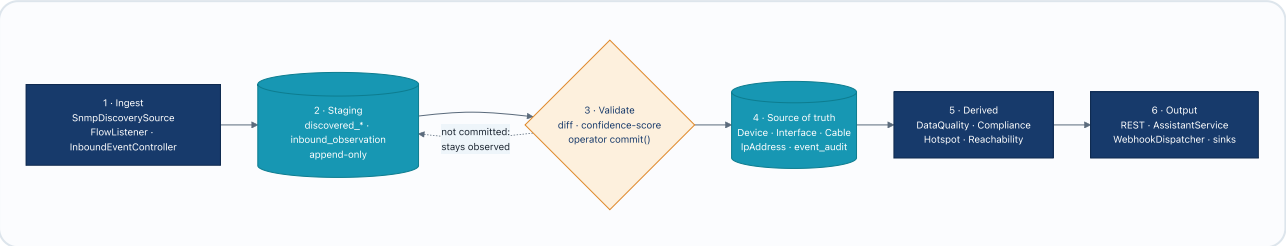

Figure 3. The six-stage pipeline. The amber node (stage 3) is the trust boundary. Observations that are not committed remain in staging until they age out; only `commit()` writes from observation into the source of truth.

Stage	Key types	Notes
1 · Ingest	`SnmpDiscoverySource`, `FlowListener`, `InboundEventController`	Decode, resolve sender to device, stamp `tenantId` + `observedAt`.
2 · Staging	`discovered_interface`, `discovered_neighbor`, … (17 tables), `inbound_observation`	Append-only; the newest row per natural key is the one in effect; auto-purged at 14 days.
3 · Validate	`ValidationService`, `InboundValidationService`	Confidence-score each observation by corroboration, then commit it. The trust gate.
4 · Source of truth	`Device`, `Interface`, `Cable`, `IpAddress`, … plus `event_audit`	The canonical model plus a hash-chained, tamper-evident audit.
5 · Derived	`DataQualityService`, `ComplianceService`, `HotspotService`, `ReachabilityService`	Computed from a snapshot, storing no new facts; single-flight coalescing plus a 90 s cache lifetime (TTL).
6 · Output	`REST`, `AssistantService`, `WebhookDispatcher`, `OutboundSinkService`	Cited answers, HMAC-signed webhooks, SIEM/chat, UI.

5 Ingestion paths

Four passive network listeners and three application paths feed staging. The listeners are dormant by default and open no network socket until they are enabled at install or toggled on at runtime through `CollectorSettingsService` (no restart needed). None of them inspects packet payloads: they read switch-derived signals and the announcements that gear already broadcasts on its own. The SNMP discovery sweep itself is an outbound poll run by `DiscoveryWorker`, not a listener. (SNMP is the standard protocol for reading device status.)

Source	Class	Bind / trigger	Captured	Default
SNMP / LLDP sweep	`SnmpDiscoverySource`	`scheduler · 161/UDP egress`	interfaces, neighbors, serial, IPs, VLANs, sensors, PTP, multicast, BGP/OSPF	OFF
NetFlow / sFlow	`FlowListener`	`2055 / 6343 UDP`	traffic tuples to top-talkers, per-app	OFF
mDNS	`MdnsListener`	`mcast 224.0.0.251:5353`	AV service announcements (Dante, NDI, AES67)	OFF
DHCP fingerprint	`DhcpFingerprintListener`	`67/UDP (relayed)`	option-55/60 device-family fingerprint	OFF
SNMP traps	`SnmpTrapListener`	`162/UDP`	link/PSU/lamp events to observations	OFF
Inbound event API	`InboundEventController`	`POST /api/v1/inbound/event`	external assertions to proposals	OFF
Manual entry	`REST / Vaadin`	`REST / UI`	operator-documented records	n/a
Config collection	`SshConfigCollector`	`22/TCP (post-sweep)`	running-config for drift / Batfish	OPT-IN

The discovery sweep is the main ingestion path and the only source on by default once a real source is wired up. Two settings gate it: `crossconnect.discovery.enabled` (default `false`) and `crossconnect.discovery.source` (default `stub`; set to `snmp` for live reads). The worker runs on the interval in `crossconnect.discovery.interval-ms` (default 300000, five minutes) after a one minute initial delay. The SNMP sweep is detailed in §6, and flow ingestion in §7.

> **Real first, fallback second.** `RoutingDiscoverySource` tries the live SNMP read first. Only when a device is unreachable or unauthenticated does it fall back to deterministic synthetic facts, and it attaches a note explaining why. A fallback value is never recorded as if it were measured: live values are marked `CONFIRMED`, fallbacks are not.

6 SNMP discovery sweep

The sweep uses `snmp4j 3.8.2` for read-only GET / GETBULK walks (the SNMP operations that read one value or a batch of values). `SnmpProbe.probeAll()` runs one required step first (the system group, which throws if the device does not answer) and then a set of best-effort walks. An unreachable MIB (a group of values the device may or may not expose) is logged and skipped, never aborting the sweep. The PTP, multicast, sensor, and routing walks all ride the same SNMP session as the interface and neighbor walks. Each device gets a 1500 ms timeout with one retry. Results are written to staging with `observedAt = Instant.now()` at staging time, and nothing is committed here.

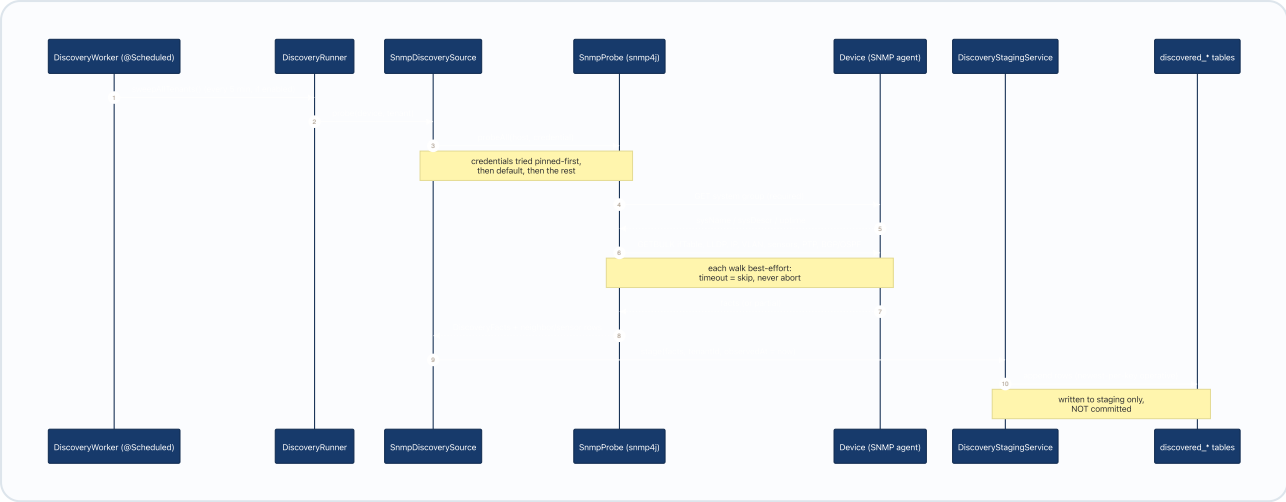

Figure 4. SNMP discovery sweep. One tenant sweep. Credentials are tried pinned-first for speed; the system group is required and the remaining walks are best-effort on the same session. Results land in `discovered_*` staging, never the source of truth.

Optional config collection runs on the same pass when `crossconnect.discovery.collect-config=true`. `SshConfigCollector` uses `sshj 0.38.0` to open an interactive shell, send the vendor paging command (default `terminal length 0`) and a read-only show command (default `show running-config`), capture the text, and hand it to `GoldenConfigService.recordRunning()` for drift detection (spotting changes from the known-good config) and Batfish analysis. No command that changes device state is ever issued.

7 Flow ingestion

Flow ingestion is a tight decode-resolve-aggregate loop: decode the data, resolve it to a known device, then add it up. `FlowListener` receives NetFlow v5/v9/IPFIX and sFlow v5 datagrams on UDP 2055 / 6343 (off by default), decodes them with `NetflowDecoder` / `SflowDecoder`, and calls `TrafficFlowService.ingest()`. The same `ingest()` method backs the push API at `POST /api/v1/flows` (`TrafficFlowController`), so a collector can post summarized flows instead of exporting raw datagrams. No payload is inspected: ingestion sees only the 5-tuple (source and destination address, port, and protocol), the byte and packet counts, and the resolved device and interface.

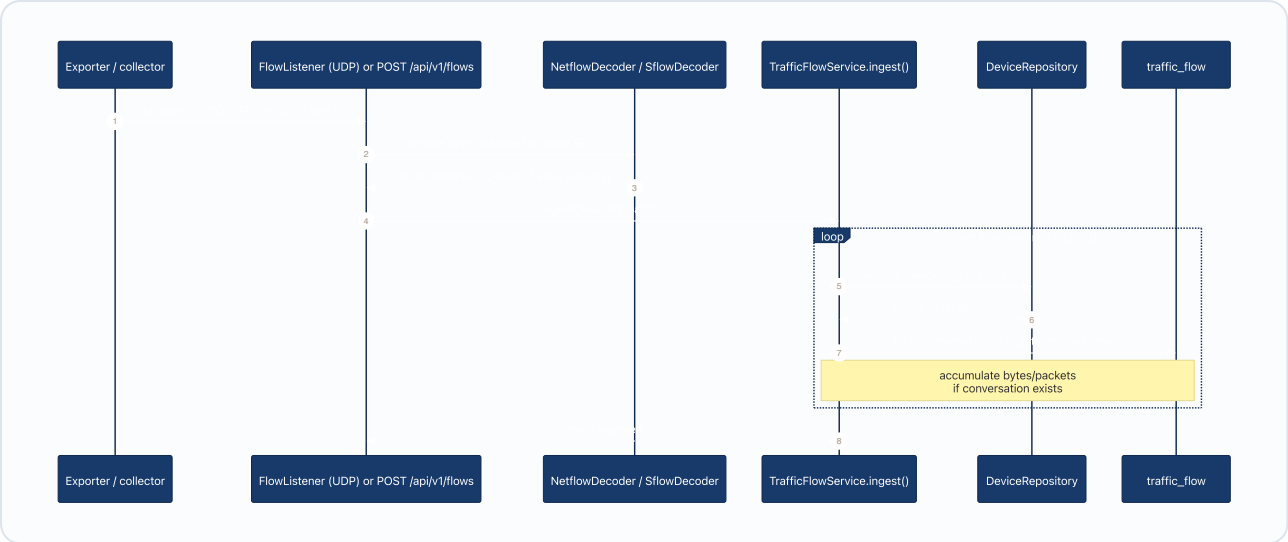

Figure 5. Flow ingestion. The UDP receiver decodes each datagram, resolves the exporter to a device, and aggregates per conversation. The push API skips the decode step and shares the same `ingest()` path. No payload is inspected.

8 Validation: the trust boundary

Staged observations are confidence-scored before anything is trusted. Nothing crosses into the source of truth unscored: each observation earns a confidence from how well it corroborates across sources, and `commit()` is the only write path. `ValidationService.report()` classifies every discovered LLDP neighbor (a link a switch reports seeing) as corroborated, single-source, or unresolved, and it collapses the two reciprocal rows for a link (A-to-Z and Z-to-A) into one physical link.

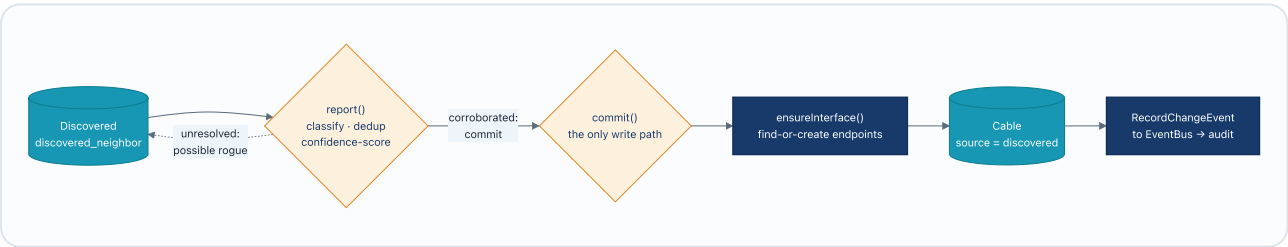

Figure 6. Validation. Each discovered link is confidence-scored and deduplicated. `commit()` is the only write path from observation into the source of truth: it finds or creates the `Interface` endpoints, writes the `Cable` through `CableService.create()`, and that in turn publishes a `RecordChangeEvent`.

Confidence is earned by corroboration: more agreeing sources mean a higher score. For inbound assertions (claims pushed in from outside), `InboundValidationService` scores a claim *high* when it resolves to a known entity and two or more distinct sources agree within a 24 hour window, *medium* for a single source that resolves, and *low* when it resolves to no known entity (a possible rogue, typo, or new device). The inbound queue records an `InboundObservation` and never reaches the source of truth without passing validation.

> **`commit()` is the only write path from observation to truth.** `ValidationService.commit()` finds or creates the `Interface` endpoints and writes the `Cable` with `source = "discovered"`. It throws an error rather than double-book an already-cabled endpoint. `commitAllPending()` commits every pending link in one atomic batch and returns a `CommitSummary(committed, skipped)`, skipping a conflicting link rather than failing the whole batch.

9 Source of truth & the audit chain

The canonical entities (`Device`, `Interface`, `Cable`, `IpAddress`, `Prefix`, `Vlan`, `Vrf`, `Circuit`, `NetworkService`) are what every read, view, and AI tool resolves against. Every change publishes a `RecordChangeEvent` to the in-process `EventBus`. `EventAuditService` subscribes and folds each one into a hash-chained, HMAC-signed audit trail in `event_audit` (each entry cryptographically links to the one before it). The exact content hash is:

```
contentHash = SHA-256(tenantId · kind · occurredAt · actor · payload · previousHash)
```

The fields are joined with newline separators, and the first row in a tenant's chain uses the placeholder `GENESIS` in place of `previousHash`. Because each entry links to the previous one for its tenant, altering any record breaks the chain and becomes detectable. A parallel global chain in `system_event_audit` (scope id all-zero UUID) covers deployment-wide actions such as key rotation, and every AI prompt, retrieval, and output is logged to `ai_audit_entry`.

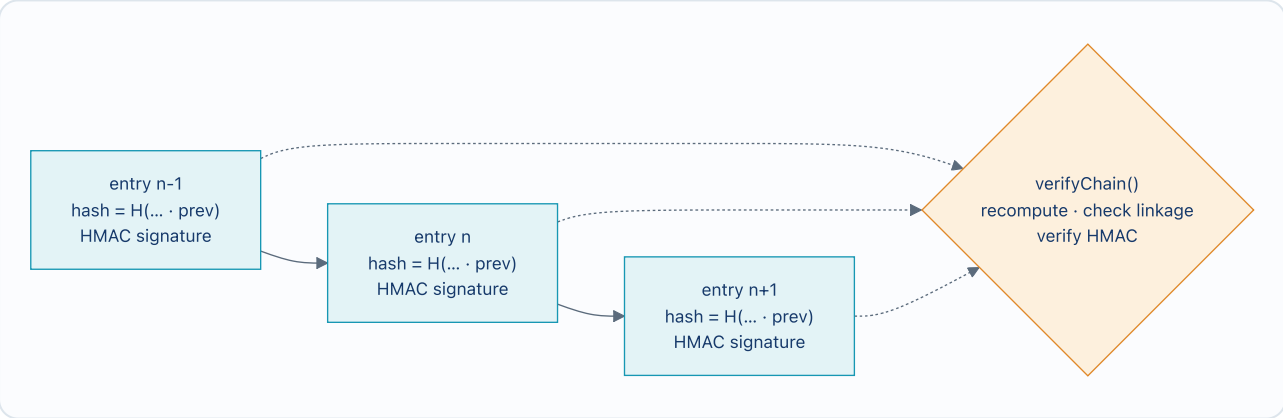

Figure 7. Hash-chained audit. Each entry folds in the previous entry's hash and is HMAC-signed under the deployment signing secret. `EventAuditService.verifyChain()` recomputes every hash, checks linkage, and verifies signatures, returning an `AuditIntegrityReport` that pinpoints any break. This is the integrity guarantee behind change-cause traces and the compliance evidence pack.

10 Derived layer & caching

The intelligence services (`DataQualityService`, `ComplianceService`, `MaturityService`, `HotspotService`, `ReachabilityService`, `ThreatDetectionService`) store no new facts. Each one reads a snapshot of the source of truth and computes a score or ranked list from it, so the same evidence backs many reports. Fleet-wide reads share a short-lived cache with single-flight coalescing: the first caller does the work, and any other callers asking the same question within the window wait for that one result instead of each running the Batfish-backed audit again.

The coalescing is built on `SingleFlight` (a `ConcurrentHashMap` of `CompletableFuture` per key). `BatfishResultCache` wraps it and lines callers up one at a time on the single Batfish session, keyed on `tenant:question:config-hash` so a cached result is discarded when the config changes. `ComplianceService` applies an explicit 90000 ms (90 s) cache lifetime for signals (`SIGNALS_TTL_MS`).

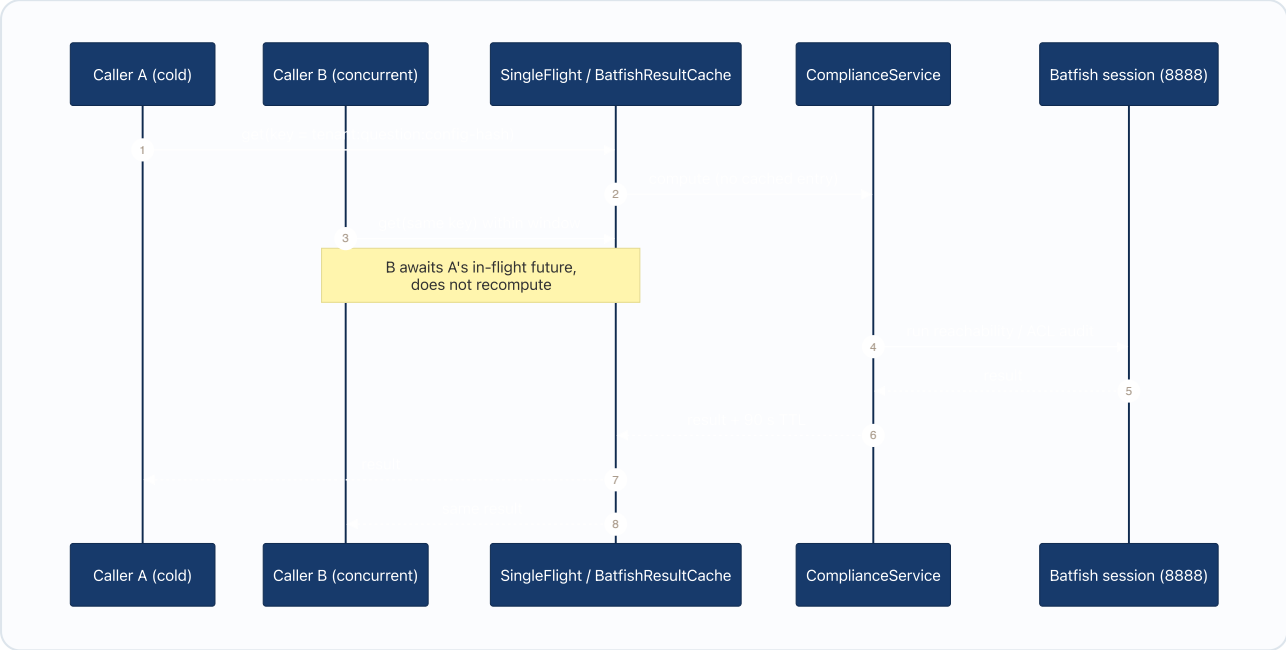

Figure 8. Cached fleet read. Single-flight coalescing: concurrent cold callers within the 90 s window await one computation instead of each running the Batfish-backed audit, which the single session would otherwise serialize.

11 AI query path

The assistant reads from the source of truth and cites what it reads. `AssistantService` drives one turn: it invokes read-only tools on `CrossConnectTools` (every result carries a `kind` and a `primaryKey` so it can be cited), hands the prompt to an `LlmClient` (the large language model client), and then filters the answer through `CitationValidator`, which rejects any answer that references a record the tools did not actually return. The AI is scoped to one tenant and respects role-based access control (RBAC), so it cannot reveal data the user is not allowed to see. Write actions are never applied on the spot: `WriteIntentDetector` queues a `WriteIntent` for confirm-before-commit (it expires after 900 s, cleaned up by `WriteIntentSweep`). Every prompt, retrieval, and output is logged to `ai_audit_entry` .

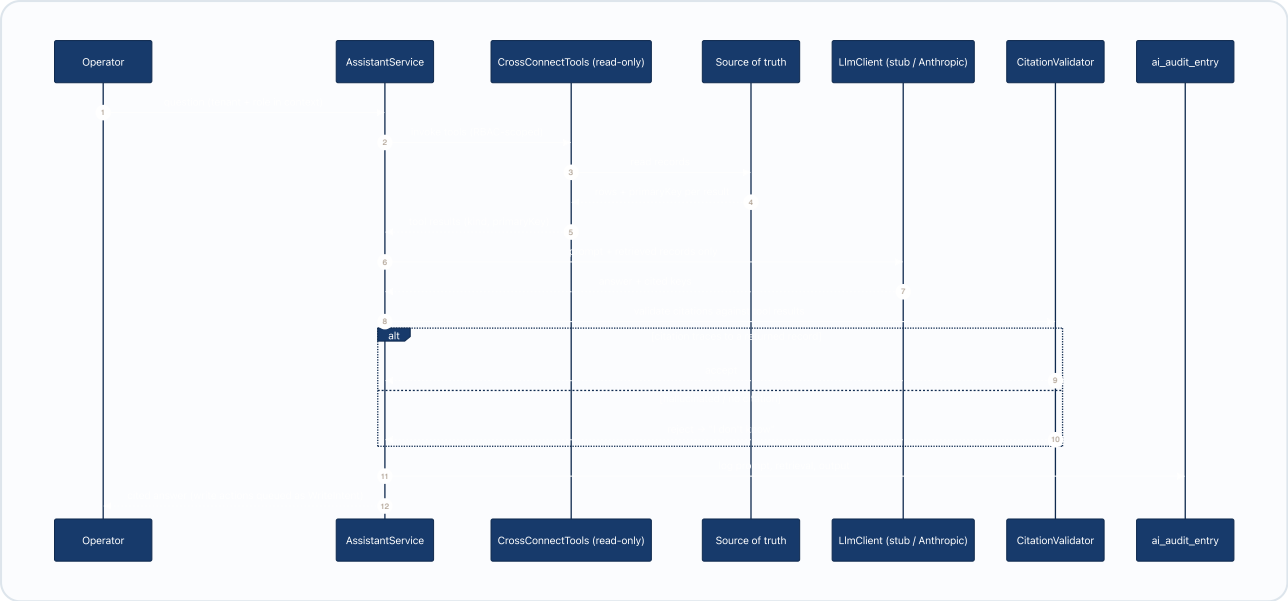

Figure 9. AI query path. Grounded and RBAC-scoped; every tool result carries a citable key, `CitationValidator` rejects unsupported claims, and any write action is queued as a `WriteIntent` for confirm-before-commit, never applied inline. The default provider is `stub` (deterministic, no external call) until an LLM endpoint and key are configured.

12 Output paths & egress

Output is the only place data leaves the deployment, and every external path is operator-configured and stays off until a destination is set. The audit-and-event stream drives three sinks (destinations data is sent to): synchronous in-process subscribers, the webhook dispatcher, and the SIEM/chat sink. (A SIEM is a security monitoring system.)

Path	Class	Mechanism	Default
REST / GraphQL reads	controllers	Tenant-scoped, RBAC-enforced cited reads over the source of truth	ON
Webhooks	`WebhookDispatcher`	Async delivery, `WebhookSigner` HMAC-SHA256 over the raw body, header `X-CrossConnect-Signature`; exponential backoff (6 attempts, 1 s to 1 h, then dead-letter)	ON URL
SIEM / chat sink	`OutboundSinkService`	Fire-and-forget export of the activity stream to a SIEM HTTP collector or Slack/Teams webhook; subscribes to `RecordChangeEvent`	OFF UNTIL URL SET
AI answers	`AssistantService`	Cited, RBAC-scoped; only the question and retrieved records reach the model, never secrets	PROVIDER STUB BY DEFAULT

Each subscription's signing secret is stored encrypted at rest with AES-256-GCM. The receiver verifies a delivery by recomputing `HMAC-SHA256(secret, body)` and constant-time-comparing it to the `X-CrossConnect-Signature` header. Inbound webhooks the platform accepts (presence, external assertions) are staged as proposals and never applied without passing the trust gate in §8.

13 Stores, retention & cross-cutting concerns

Concern	Mechanism
Tenancy	Every row belongs to one tenant (one customer); every query filters on `tenant_id`; a request filter binds every `/api/v1/*` call to an active tenant. The tenant is the isolation boundary that keeps customers' data apart.
Staging retention	Append-only `discovered_*` rows (17 tables) are dropped once they age past the window (default 14 days, `crossconnect.discovery.staging.retention-days`) by `StagingPurgeSweep`, which runs daily after a one-hour initial delay.
Audit retention	`EventAuditPurgeSweep` trims `event_audit` rows past retention (default 90 days, `crossconnect.audit.retention-days`) while keeping the chain links intact; runs daily.
Idempotency	Staging is keyed on a natural key (for example device + ifIndex), and the newest `observedAt` wins, so re-running a sweep is safe and never creates duplicates.
Caching	Single-flight coalescing plus a 90 s cache lifetime on fleet reads (`SingleFlight`, `BatfishResultCache`); topology, import, and committing are specific to each request and are not cached.
Back-pressure	A per-device read timeout (1500 ms, one retry) and a request rate limit (default 100 per 60 s window); the single Batfish session handles reachability callers one at a time through the coalescing cache.
Egress safety	Webhooks are HMAC-SHA256 signed, with a bounded number of retries and a dead-letter queue for failures; sinks are fire-and-forget and stay off until a destination URL is set; every external call has a fallback so the app keeps working (a Batfish error returns a heuristic result instead).

14 Configuration reference

The data-flow surface is driven by environment variables. The defaults favor a working out-of-the-box install: a fresh deployment opens no listening ports toward the network and makes no external calls. Representative keys:

Control	Key	Default
Server port	`SERVER_PORT`	`8080`
Discovery enabled	`CROSSCONNECT_DISCOVERY_ENABLED`	`false`
Discovery source	`CROSSCONNECT_DISCOVERY_SOURCE`	`stub` (set `snmp` for live)
Sweep interval	`CROSSCONNECT_DISCOVERY_INTERVAL_MS`	`300000` (5 min)
Config collection	`CROSSCONNECT_DISCOVERY_COLLECT_CONFIG`	`false`
mDNS listener	`crossconnect.discovery.mdns.enabled` / `.port`	`false` / `5353`
DHCP fingerprint	`crossconnect.discovery.dhcp.enabled` / `.port`	`false` / `67`

Control	Key	Default
Flow receiver	`CROSSCONNECT_FLOW_ENABLED` / NetFlow / sFlow port	`false` / `2055` / `6343`
SNMP trap receiver	`CROSSCONNECT_SNMPTRAP_ENABLED` / `_PORT`	`false` / `162`
Inbound event API	`CROSSCONNECT_INBOUND_ENABLED`	`false`
Staging retention	`crossconnect.discovery.staging.retention-days`	`14`
Audit retention	`crossconnect.audit.retention-days`	`90`
Batfish sidecar URL	`CROSSCONNECT_BATFISH_URL`	`http://localhost:8888`
AI provider / model	`CROSSCONNECT_AI_PROVIDER` / `_MODEL`	`stub` / `claude-opus-4-20250514`
AI write-intent TTL	`CROSSCONNECT_AI_INTENT_TTL_SECONDS`	`900`

Where to look next. The reality-stream thesis (cross-referencing the discovery, flow, and health streams against the source of truth) is documented in `docs/reality-stream-triangulation-strategy.md` . The security and trust-boundary view is in the companion *Security & Architecture Reference*.

CrossConnect by CybrIQ · Data Flow Architecture · Technical reference · 21 June 2026 · Behavior described reflects the shipped operator-preview build; items marked *deployment option* are listeners or integrations enabled at install. · contact_us@cybriq.io