Distributed Storage Architecture: Zaxonlite Integration for Dynamic Policies, Replicated Reputation, and Incident Forensics

STATE

published

STATUS

Published

TYPE

Architectural Specification

AUTHORS

Sibuna Contributors <team@sibuna.local>

CREATED

2026-09-07

UPDATED

2026-10-15

SUMMARY

Specifies and records the implementation of Sibuna's Edge surface: the embedded Zaxonlite store (replicated SQLite on paxos-zig) for cluster-wide dynamic policies, IP reputation, and forensic incident search, the read-copy-update engine slot that keeps the request path free of database access, the lock-free incident ring, and campaign clustering of attack payloads.

DISCUSSION

No external discussion URL assigned

LABELS

storage, zaxonlite, paxos, sqlite, distributed

Record status

The catalog defines the review process and the meaning of each state. This numbered record forms part of the catalog.

Contents

1. Revision note 2
2. Abstract 2
3. Context and motivation 2
4. Zaxonlite foundations 2
5. Build and configuration 2
6. Schema 3
7. Hot-path isolation 3
8. Campaign clustering 4
9. Forensics 4
10. Product boundary 4
11. Delivery record 4
12. Verification 5
13. Open items 5
14. Review corrections (2026-09-07) 5
15. References 5

1. Revision note

2026-10-15: Ported from the Typst original in the Zig repository. The record describes the Zig implementation as it was decided; SID 0010 maps its packages to the Rust crates of the port. The Rust port removes the Zaxonlite dependency. SID 0011 replaces it with sqlx: SQLite for a single node and PostgreSQL for clusters. This record remains the account of the Zig design.
2026-10-15 (Rust port): beyond the storage engine, the port replaces two mechanisms this record specifies. The read-copy-update engine slot becomes an owned engine published behind an `Arc`, which is never rebuilt in place (SID 0002, SID 0006 Part G). The sqlite-vec nearest-neighbour search over every incident becomes a search over the most recent 16,384 embeddings of all nodes, kept in memory by the storage thread (SID 0011, SID 0006 Part I). The incident ring of 512 records, the batches of 32, the durable receipts and the campaign threshold are unchanged.

Revision note (2026-09-07)
Revised after the implementation review of 2026-09-07, at which point the storage layer existed only as a design. All three phases are now implemented in `apps/sibuna/src/persistent.zig` against the official `insanai/zaxonlite` release (v0.7.0 since 2026-09-14). Zaxonlite 0.7.0 fixed the cluster trim regression that wedged members under sustained replicated load (upstream issue 10) and cut its journal, TRIM, identity, and wire formats without migration, so data directories from earlier builds are recreated rather than upgraded. This record describes the code as built and lists what remains unexercised.

2. Abstract

Specifies and records the implementation of Sibuna's Edge surface: the embedded Zaxonlite store (replicated SQLite on paxos-zig) for cluster-wide dynamic policies, IP reputation, and forensic incident search, the read-copy-update engine slot that keeps the request path free of database access, the lock-free incident ring, and campaign clustering of attack payloads.

3. Context and motivation

A single Sibuna instance loads policy from a file and keeps its challenge, rate-limit, and ban state in local memory. Operating a fleet of edge nodes adds three requirements: policy changes that take effect everywhere without restarts, threat intelligence that propagates from the node that observed an attack to every other node within milliseconds, and durable, searchable forensics for the payloads that were blocked. Existing designs delegate this to external PostgreSQL and Redis deployments or to a proprietary edge platform. Sibuna embeds **Zaxonlite**, a replicated SQLite built on the `paxos-zig` Multi-Paxos library, and keeps all of it out of the request path.

4. Zaxonlite foundations

1. **Replicate the bytes, not the SQL.** A transaction executes once on the leader; the SQLite write-ahead-log page images are what consensus replicates, so nondeterministic SQL is safe and replicas converge to byte-identical files.
2. **The journal is the truth.** Every write is framed, checksummed, and fsynced before consensus proceeds; the SQLite file is a materialised cache rebuildable from the durable anchor plus the journal suffix.
3. **Embedded.** Zaxonlite compiles into the Sibuna binary from `build.zig.zon` (`https://github.com/insanai/zaxonlite/archive/refs/tags/v0.7.0.tar.gz`, hash pinned). The single-node `Node` needs no transport; the cluster `Embedded` facade adds a TCP listener, peers, and either mTLS or the loopback development PSK.

5. Build and configuration

Table 4 Build options and daemon flags

Option	Effect
`-Dstorage=true` (default)	Links SQLite (with FTS5 and sqlite-vec), paxos-zig, and Zaxonlite; the binary links libc.
`-Dstorage=false`	Pure in-memory daemon; `--data-dir` is refused with `StorageDisabled`.
`-Dcluster=true`	Enables the `Embedded` facade and links OpenSSL 3 for Zaxonlite's mutual TLS.
`--data-dir <path>`	Opens (or creates) the node data directory and starts the storage thread.
`--cluster-node <id>`	This member's id; non-zero switches from `Node` to `Embedded`.
`--cluster-listen <host:port>`	This member's cluster endpoint.
`--cluster-peer id@host:port[/role]`	A peer, repeatable up to 8; role defaults to `data-voter`.
`--cluster-secret-file <path>`	Shared PSK for the loopback development transport (`allow_psk_only_loopback`).
`--cluster-tls-cert/key/ca <path>`	Per-node certificate, key, and cluster CA for production TCP.
`--storage-poll-ms <ms>`	Storage thread cadence (default 500).

6. Schema

```
CREATE TABLE sibuna_meta (key TEXT PRIMARY KEY, value TEXT NOT NULL);
CREATE TABLE policies (
    id TEXT PRIMARY KEY, name TEXT NOT NULL, priority INTEGER NOT NULL DEFAULT 100,
    path_pattern TEXT, ua_pattern TEXT, action TEXT NOT NULL, difficulty INTEGER, algorithm TEXT,
    header_matchers TEXT, cidr_matchers TEXT,      -- JSON object / JSON array
    weight INTEGER NOT NULL DEFAULT 0, enabled INTEGER NOT NULL DEFAULT 1,
    created_at INTEGER NOT NULL, updated_at INTEGER NOT NULL);
CREATE TABLE ip_reputation (
    ip_or_cidr TEXT PRIMARY KEY, reputation_score INTEGER NOT NULL, banned_until INTEGER,
    trigger_rule TEXT, hits INTEGER NOT NULL DEFAULT 1, last_seen INTEGER NOT NULL);
CREATE TABLE security_incidents (
    id INTEGER PRIMARY KEY,                        -- node_id << 40 | sequence
    node_id INTEGER NOT NULL, client_ip TEXT NOT NULL, user_agent TEXT NOT NULL,
    method TEXT NOT NULL, path TEXT NOT NULL, violation_category TEXT NOT NULL,
    offending_payload TEXT NOT NULL, campaign_id INTEGER, recorded_at INTEGER NOT NULL);
CREATE VIRTUAL TABLE incidents_fts USING fts5(path, offending_payload,
    content='security_incidents', content_rowid='id');
CREATE VIRTUAL TABLE incidents_vec USING vec0(item_id INTEGER PRIMARY KEY,
    embedding float[64] distance_metric=cosine, embedding_coarse bit[64]);
```

Incident ids are node-scoped (`node_id << 40 | sequence`) so members insert concurrently with no id coordination; the sequence is recovered from the table at startup.

7. Hot-path isolation

7.1. Read-copy-update engine slot

Workers never touch the database. `AppState` holds an atomic pointer to an `EngineSlot` (`engine`, cache-line-aligned reader count). A request pins the slot with `acquireEngine`: load the pointer, increment the slot's reader count, and re-load the pointer; if it changed, decrement and retry. The re-check closes the window in which a writer could have swapped and observed zero readers between the load and the increment. The storage thread rebuilds the *spare* slot (database policies followed by file/default rules, then reputation prefixes into the trie), publishes it with an atomic swap, and spins until the old slot's reader count reaches zero before it becomes the new spare. Two engine buffers, sized in the benchmark metadata, therefore serve an unbounded sequence of policy changes.

7.2. Incident ring

WAF denials and honeypot hits copy a fixed-size record (address, User-Agent, method, path, category, first 512 bytes of the payload, timestamp) into a Vyukov bounded MPSC queue of 512 slots; a full queue drops the newest record rather than blocking a response. The storage thread is the single consumer.

7.3. Storage thread tick

Every `--storage-poll-ms`, persist at most 32 records in one transaction, including incident, FTS, vector, and honeypot reputation writes. The nearest-campaign query executes inside that transaction and sees earlier records in the same batch. A per-issuer cursor in `sibuna_meta` guards every effect; the cursor advances atomically with the data. The exact pending SQL is retained after an uncertain reply, making retries idempotent without one network read per incident. Queue admission itself is not durable.

The thread then reads the trigger-maintained policy revision and rebuilds when it changes or a loaded reputation entry expires, even if incident persistence failed. Each slot owns its own arena until its readers drain. A committed ban propagates when a healthy member next reloads; quorum recovery and backlog can increase latency beyond the polling interval.

Production counters expose persisted incidents, committed batches, dropped queue entries and failed write attempts. The queue holds 512 records plus one pending batch of at most 32. Incident issuer IDs are limited to 23 bits so `issuer << 40 | sequence` stays within SQLite's positive signed integer range; sequence exhaustion fails explicitly rather than wrapping.

8. Campaign clustering

Each payload is embedded as a 64-dimensional unit vector by the hashing trick over byte trigrams (digits folded, case folded, signed feature hashing, L2 normalisation). Before insert, a cosine nearest-neighbour query against `incidents_vec` returns the closest earlier incident; if its distance is at most 0.35 the new incident joins that incident's `campaign_id`, otherwise it starts a campaign of its own. Two SQL injection variants that differ only in target column and literal values land in one campaign; a cross-site scripting payload does not. No model is downloaded and nothing allocates on a request thread.

9. Forensics

`Persistent.searchIncidents(match, limit)` runs an FTS5 `MATCH` over path and payload joined to the incident rows, ordered by rank. There is no HTTP administration endpoint yet; operators query the data directory with `zaxon sql --data <dir>` or read `current.db` directly.

10. Product boundary

Gate provides admission, policy and local flood controls. Shield adds payload inspection. Distributed edge deployment adds replicated policy and reputation to either surface. No vendor feature-parity claim is made; coverage and limitations are defined by the code and tests.

11. Delivery record

Phase 1: Standalone embedded mode (delivered)

`Node` opened from `--data-dir`, schema migration, dynamic policy reload, reputation bans, incident persistence with FTS5, forensic search API.

Phase 2: Consensus replication (delivered)

`Embedded` facade behind `-Dcluster=true`, member list from `--cluster-*` flags, loopback PSK or mTLS transport, node-scoped incident ids, cluster-wide bans through `ip_reputation`. The external `benchmarks/distributed.py` harness exercises three real members, shared sessions, issuer-bound challenge verification, replicated bans, and one-member loss.

Phase 3: Payload clustering (delivered)

Feature-hashed trigram embeddings stored in `incidents_vec`, cosine nearest-neighbour campaign assignment at insert time.

12. Verification

The storage test (`apps/sibuna/src/persistent.zig`, run by `zig build test` when storage is enabled) opens a node in a temporary directory, verifies that a file policy survives the initial rebuild, inserts a `policies` row with header and CIDR matchers and a `banAddress` call, ticks once, and checks that the rebuilt engine applies the rule with its difficulty and algorithm and denies the banned address; it then pushes two SQL-injection variants and a honeypot hit through the hook, ticks, and checks the FTS search, the campaign assignment (two campaigns for three incidents), and the reputation upsert. Unit tests cover the ring under concurrent producers, the embedding, peer-spec parsing, and SQL quoting. A manual run of the release daemon recorded WAF denials and a honeypot ban in `current.db` with 12 MB resident memory.

13. Open items

- Automated multi-node tests (leader failover, ban propagation latency).
- An authenticated HTTP administration API for policies and incident search.
- Retention and archival policy for `security_incidents`.

14. Review corrections (2026-09-07)

Startup retries idempotent initialization while a cluster leader settles; incident writes are not blindly retried because an ambiguous transport failure may already have committed them. INSERT/UPDATE/DELETE triggers maintain a revision for both dynamic tables, including multiple updates with the same timestamp. A failed rebuild does not acknowledge the revision. Expiry of a loaded reputation entry independently triggers a rebuild, so time-based bans disappear without another write. Dynamic rules precede generic file/default rules; capacity exhaustion fails the reload rather than silently dropping security rules. Each engine slot has its own arena, and all publication/pinning operations use sequential consistency.

`--cluster-node` binds challenge authentication to an issuer; token keys remain shared. Route challenge fetching and verification to that member. Rate limits and spent sets remain local, and spent state is not durable across restarts. Replication is off the request path, but snapshot pinning, incident copying/enqueue, local limits and production metrics are not free. The external load harness adds no server-side timing instrumentation. See `benchmarks/results/distributed-latest.json` for local results and their limits. A healthy HTTP endpoint alone does not prove quorum health. WAN partitions and sustained forensic-write saturation are separate test targets.

Response rule names are copied into a bounded buffer before the snapshot is released, so a slow origin does not hold the reader counter. Persistent rule strings remain owned by the corresponding slot arena until readers drain.

15. References

The Typst original has no bibliography. This list collects the sources and records that the text names.

- Zaxonlite v0.7.0, the official `insanai/zaxonlite` release: https://github.com/insanai/zaxonlite/archive/refs/tags/v0.7.0.tar.gz (upstream issue 10 records the cluster trim regression fixed in 0.7.0).
- `paxos-zig`, the Multi-Paxos library on which Zaxonlite is built.
- SID 0010, the mapping from the Zig packages to the Rust crates of the port.
- SID 0011, the sqlx storage design that replaces Zaxonlite in the Rust port.

Lifecycle event

```
2026-10-15: prediscussion -> discussion. Ported from the Zig repository's
Typst record
```

Lifecycle event

```
2026-10-15: discussion -> published. State carried over from the Zig
repository's registry
```