

Sibuna: Foundation Architecture, Delivery Plan, and Performance Contract

STATE

published

CREATED

2026-09-07

LAST UPDATED

2026-10-03

INTENDED STATUS

Published

AUTHORS

Sibuna Contributors <team@sibuna.local>

DISCUSSION

Foundational architectural specification, zero-allocation pipeline, two-tier proof-of-work engine, keyed-hash session tokens, product surfaces, measured performance contract, and delivery record for the Sibuna pure-Zig monorepo

LABELS

architecture, firewall, pow, performance

Status of This Memo

This document is an internal Sibuna discussion record authored in Typst and tracked in git. It intentionally follows RFC-style structure so the design scope, rationale, trade-offs, and operational constraints remain explicit. Documents that still use the placeholder number XXXXX are provisional drafts. Numbered SID documents are part of the permanent project record.

Contents

Abstract	1
Introduction	1
The SID Lifecycle	1
Tooling and Numbering Workflow	1
Creating a Draft	1
Promoting a Draft	1
Listing Discussions	1
Engineering Principles and Code Standards	1
Hard Structural Limits	1
Core TigerStyle Priorities	1
Elm-Style Diagnostic and Error Reporting	1
The Three Elements of an Elm-Style Error	1

Concrete Diagnostic Example	1
Implementation Architecture	1
Recoverable Errors vs Invariant Assertions	1
Compilation Targets and Verification	1
Conclusion	1
Decision	1
Request contract	1
Authentication and work	1
Policy and memory	1
Measurement gates	1
Release toolchain	1
Native platforms and release boundaries	1
Remaining limits	1
1. Abstract	9
2. Introduction	9
3. The SID Lifecycle	9
4. Tooling and Numbering Workflow	10
4.1. Creating a Draft	10
4.2. Promoting a Draft	10
4.3. Listing Discussions	10
5. Engineering Principles and Code Standards	10
5.1. Hard Structural Limits	10
5.2. Core TigerStyle Priorities	11
6. Elm-Style Diagnostic and Error Reporting	11
6.1. The Three Elements of an Elm-Style Error	11
6.2. Concrete Diagnostic Example	11
6.3. Implementation Architecture	12
6.4. Recoverable Errors vs Invariant Assertions	12
7. Compilation Targets and Verification	12
8. Conclusion	12
9. Decision	9
10. Request contract	10
11. Authentication and work	11
12. Policy and memory	11
13. Measurement gates	11
14. Release toolchain	12
15. Native platforms and release boundaries	12
16. Remaining limits	12
Decision summary	1
Data model	1
Evaluation order	1
Built-in policy	1
Policy file	1
Dynamic policies	1
Verification	1
Milestones	1
Review corrections (2026-09-07)	1
17. Decision summary	9
18. Data model	9

19. Evaluation order	10
20. Built-in policy	10
21. Policy file	11
22. Dynamic policies	12
23. Verification	12
24. Milestones	12
25. Review corrections (2026-09-07)	12
Context and problem statement	1
Engine choice: automata and tokenizers, not regular expressions	1
Design	1
Byte-class pre-scan	1
26. Context and problem statement	9
27. Engine choice: automata and tokenizers, not regular expressions	9
28. Design	10
28.1. Byte-class pre-scan	10
Context and motivation	1
Zaxonlite foundations	1
Build and configuration	1
Schema	1
Hot-path isolation	1
Read-copy-update engine slot	1
Incident ring	1
Storage thread tick	1
Campaign clustering	1
Forensics	1
Product boundary	1
Delivery record	1
Verification	1
Open items	1
Review corrections (2026-09-07)	1
29. Context and motivation	9
30. Zaxonlite foundations	10
31. Build and configuration	10
32. Schema	10
33. Hot-path isolation	11
33.1. Read-copy-update engine slot	11
33.2. Incident ring	11
33.3. Storage thread tick	11
34. Campaign clustering	12
35. Forensics	12
36. Product boundary	12
37. Delivery record	12
38. Verification	12
39. Open items	13
40. Review corrections (2026-09-07)	13
Abstract	1
Introduction and Adversary Model	1
Setting	1
Axioms	1

Cost calculus	1
Part A: The Hashcash Tier	1
Part B: Proof of Sequential Work	1
Why sequential work	1
The construction	1
Theorems	1
Why not the alternatives	1
Implementation and calibration	1
Part C: Stateless Challenges and Symmetric Tokens	1
Pseudorandom functions and message authentication	1
Stateless challenge identifiers	1
Session tokens	1
Part D: Rate Limiting by the Generic Cell Rate Algorithm	1
Part E: Robin Hood Hashing for the Spent Set	1
Part F: Linear-Time Inspection	1
Multi-pattern matching	1
The semantic layer	1
Vectorised matchers	1
Part G: Read-Copy-Update Engine Slots	1
Part H: Load-Adaptive Difficulty	1
Part I: Campaign Clustering by Feature Hashing	1
Verification and Benchmark Gates	1
Security Considerations	1
Open Problems	1
References	1
41. Abstract	9
42. Introduction and Adversary Model	10
42.1. Setting	10
42.2. Axioms	10
42.3. Cost calculus	11
43. Part A: The Hashcash Tier	11
44. Part B: Proof of Sequential Work	12
44.1. Why sequential work	12
44.2. The construction	12
44.3. Theorems	13
44.4. Why not the alternatives	14
44.5. Implementation and calibration	15
45. Part C: Stateless Challenges and Symmetric Tokens	15
45.1. Pseudorandom functions and message authentication	15
45.2. Stateless challenge identifiers	16
45.3. Session tokens	17
46. Part D: Rate Limiting by the Generic Cell Rate Algorithm	17
47. Part E: Robin Hood Hashing for the Spent Set	18
48. Part F: Linear-Time Inspection	19
48.1. Multi-pattern matching	19
48.2. The semantic layer	20
48.3. Vectorised matchers	20
49. Part G: Read-Copy-Update Engine Slots	21
50. Part H: Load-Adaptive Difficulty	21

51. Part I: Campaign Clustering by Feature Hashing	22
52. Verification and Benchmark Gates	22
53. Security Considerations	23
54. Open Problems	23
55. References	24
Abstract	1
Introduction and Motivation	1
Operator Workflows	1
Design Principles	1
Don't make me think	1
Fast judgement: System 1	1
Careful judgement: System 2	1
Elegance	1
Principle audit by page	1
Terminology and Scope	1
Goals and Non-Goals	1
Goals	1
Non-Goals	1
Architecture Overview	1
Modules and directories	1
Process model and the isolation contract	1
Startup from the command line	1
The Serve Kernel	1
Authentication and Authorization	1
The Telemetry Pipeline	1
The sampler	1
The incident tap	1
The challenge funnel	1
Traffic sampling and top-k rankings	1
Data-plane changes this record requests	1
Cluster aggregation	1
The Real-Time Protocol	1
Data Model	1
GeoIP	1
Live earth globe on the landing page	1
Cluster Management	1
The Interface Module	1
Pages	1
Wireframes	1
The Build Pipeline	1
Security Considerations	1
Performance Budget	1
Delivery Plan	1
Verification	1
Resolved Design Decisions	1
Operational Contracts and Trade-offs	1
Retained observations	1
Rule observations	1
Management consistency	1

Browser and peer state	1
Acceptance and Open Questions	1
Query and authentication observation bounds	1
Launch scope (2026-09-11)	1
Post-launch completion (2026-09-11)	1
Interface budget review (2026-09-11)	1
References	1
56. Abstract	9
57. Introduction and Motivation	10
58. Operator Workflows	10
59. Design Principles	11
59.1. Don't make me think	11
59.2. Fast judgement: System 1	12
59.3. Careful judgement: System 2	13
59.4. Elegance	13
59.5. Principle audit by page	13
60. Terminology and Scope	14
61. Goals and Non-Goals	15
61.1. Goals	15
61.2. Non-Goals	15
62. Architecture Overview	16
62.1. Modules and directories	16
62.2. Process model and the isolation contract	17
62.3. Startup from the command line	18
63. The Serve Kernel	19
64. Authentication and Authorization	20
65. The Telemetry Pipeline	21
65.1. The sampler	21
65.2. The incident tap	22
65.3. The challenge funnel	23
65.4. Traffic sampling and top-k rankings	23
65.5. Data-plane changes this record requests	24
65.6. Cluster aggregation	24
66. The Real-Time Protocol	24
67. Data Model	26
68. GeoIP	28
68.1. Live earth globe on the landing page	29
69. Cluster Management	30
70. The Interface Module	30
70.1. Pages	31
71. Wireframes	33
72. The Build Pipeline	43
73. Security Considerations	44
74. Performance Budget	45
75. Delivery Plan	46
76. Verification	47
77. Resolved Design Decisions	48
78. Operational Contracts and Trade-offs	49
78.1. Retained observations	49

78.2. Rule observations	49
78.3. Management consistency	50
78.4. Browser and peer state	50
79. Acceptance and Open Questions	51
79.1. Query and authentication observation bounds	51
79.2. Launch scope (2026-09-11)	51
79.3. Post-launch completion (2026-09-11)	52
79.4. Interface budget review (2026-09-11)	53
80. References	53
Abstract	1
Introduction and Motivation	1
Terminology and Scope	1
Problem Statement and Architectural Challenges	1
The Gap in the Current Codebase	1
The Asymmetric Challenges of AI Bot Identification	1
System Invariants and Design Principles	1
Detailed Design	1
1. The Multi-Tier Bot Classification Engine	1
1.1 Provider and Bot Taxonomy	1
1.2 Comprehensive AI Bot Signatures Database	1
1.3 Sub-Microsecond Radix CIDR Verification	1
1.4 Behavioral and Structural Heuristics	1
2. High-Efficiency Streaming Analytics: Algorithms, Mathematics, and Storage Decoupling	1
2.1 The Analytics Dilemma: Why Traditional OLAP Fails Edge WAFs	1
2.2 Axiomatic Foundations of Edge Analytics	1
2.3 Contention-Free Cache-Line Striped Accumulation	1
2.4 Bounded Stochastic Sampling and Poisson Ring Dynamics	1
2.5 Space-Saving Heavy-Hitters Algorithm for Path Scraping	1
2.6 Micro-RRD Pre-Aggregated Rollups and Zaxonlite Storage Decoupling	1
2.7 Long-Term Telemetry Egress Strategy	1
3. Operator Console Dashboard Interface	1
3.1 Operator Screen Wireframes	1
3.2 Visual Components	1
3.3 Granular Tabular Components	1
3.4 Actionable Policy Integration	1
4. Elm-Style Operator Error and Diagnostic Reporting	1
Verification and Testing Matrix	1
Security Considerations	1
Rollout and Operational Plan	1
References	1
81. Abstract	9
82. Introduction and Motivation	9
83. Terminology and Scope	10
84. Problem Statement and Architectural Challenges	11
84.1. The Gap in the Current Codebase	11
84.2. The Asymmetric Challenges of AI Bot Identification	12
85. System Invariants and Design Principles	12
86. Detailed Design	13

86.1. 1. The Multi-Tier Bot Classification Engine	13
86.1.1. 1.1 Provider and Bot Taxonomy	13
86.1.2. 1.2 Comprehensive AI Bot Signatures Database	14
86.1.3. 1.3 Sub-Microsecond Radix CIDR Verification	15
86.1.4. 1.4 Behavioral and Structural Heuristics	16
86.2. 2. High-Efficiency Streaming Analytics: Algorithms, Mathematics, and Storage	
Decoupling	16
86.2.1. 2.1 The Analytics Dilemma: Why Traditional OLAP Fails Edge WAFs	16
86.2.2. 2.2 Axiomatic Foundations of Edge Analytics	17
86.2.3. 2.3 Contention-Free Cache-Line Striped Accumulation	17
86.2.4. 2.4 Bounded Stochastic Sampling and Poisson Ring Dynamics	18
86.2.5. 2.5 Space-Saving Heavy-Hitters Algorithm for Path Scraping	19
86.2.6. 2.6 Micro-RRD Pre-Aggregated Rollups and Zaxonlite Storage Decoupling . . .	20
86.2.7. 2.7 Long-Term Telemetry Egress Strategy	21
86.3. 3. Operator Console Dashboard Interface	22
86.3.1. 3.1 Operator Screen Wireframes	22
86.3.2. 3.2 Visual Components	24
86.3.3. 3.3 Granular Tabular Components	25
86.3.4. 3.4 Actionable Policy Integration	25
86.4. 4. Elm-Style Operator Error and Diagnostic Reporting	26
87. Verification and Testing Matrix	26
88. Security Considerations	26
89. Rollout and Operational Plan	27
90. References	27
Abstract	1
Introduction	1
Terminology and Scope	1
Problem Statement	1
Goals and Non-Goals	1
Goals	1
Non-Goals	1
State of the Art	1
Protocol Rules	1
Detailed Design	1
The decoder	1
Admission: decoding into the head buffer	1
Forwarding	1
Cost	1
Verification and Testing	1
Security Considerations	1
Rollout and Operational Plan	1
Implementation Record (2026-10-02)	1
References	1
91. Abstract	9
92. Introduction	9
93. Terminology and Scope	10
94. Problem Statement	10
95. Goals and Non-Goals	10
95.1. Goals	10

95.2. Non-Goals	11
96. State of the Art	11
97. Protocol Rules	11
98. Detailed Design	12
98.1. The decoder	12
98.2. Admission: decoding into the head buffer	13
98.3. Forwarding	14
98.4. Cost	15
99. Verification and Testing	15
100. Security Considerations	15
101. Rollout and Operational Plan	16
102. Implementation Record (2026-10-02)	16
103. References	17

Implementation review — 2026-09-07

This revision follows code review and regression tests. Performance results live in `benchmarks/results/latest.json` and `distributed-latest.json`; earlier fixed latency tables and unsupported competitor cost models have been removed. Measurements are host-specific, not proofs of optimality or deployment capacity.

Whole-product measurement — 2026-09-07, later the same day

Driving the daemon with an external load generator (`benchmarks/tools.py`, results in `tools-comparison-latest.json`) exposed three limits invisible to the primitive suite: accept threads served one connection to completion (tail latency over 100 ms at 64 clients), every proxied response closed the client connection, and every proxied request opened a new origin connection (about 1,400 requests per second before ephemeral-port exhaustion). The request contract below now describes bounded per-connection threads with a 503 overload path, origin response framing, and a pooled origin connection with one safe retry.

9. Decision

Sibuna provides two product surfaces from one Zig binary. Persistent storage and distributed edge deployment are options for these surfaces, not a third product.

Capability	Gate (--gate)	Shield (--shield, default)
Admission	Hashcash or sequential-work browser challenge; bound session token	Same
Policy	Declarative rules, bot signatures, CIDR reputation	Same
Flood controls	GCRA, honeypot, temporary bans	Same
Payload inspection	Disabled	SQL injection, XSS, traversal, command-injection heuristics
Deployment	Reverse proxy or forward auth	Reverse proxy or forward auth

Capability	Gate (--gate)	Shield (--shield, default)
Optional state	Embedded persistent policy and reputation	Also WAF incident forensics
Distributed deployment	Cluster build and configured members	Same

10. Request contract

The server runs bounded blocking accept threads (--workers, one per CPU by default). Each accepted connection is served on its own thread with a one-megabyte stack, so a slow or idle peer never delays another connection; the number of connection threads is bounded by --max-connections (1,024 by default), beyond which the accept loop answers 503 and closes the socket without spawning, counting the event in `sibuna_overloaded_total`. Pre-parse refusals render into a bounded buffer and share a 100 ms send/lingering-close budget, draining at most 64 KiB of late input so ordinary clients receive the response before closure. A peer that withholds input or close cannot hold the accept loop indefinitely. A finished connection thread still owns its stack until it is joined, and it is joined when its slot is next taken, so slots are taken lowest first and the number of allocated stacks follows the connections currently open rather than the number served. The idle reaper closes connections silent for longer than --idle-timeout. Each active connection owns a 64 KB read buffer, with a 16 KB head limit, and bounded write buffers. HTTP/1.0 and HTTP/1.1 are supported. Chunked request bodies are decoded in place and re-framed for the origin (SID 0009); other transfer codings are refused with 501. Invalid header names, request-line control bytes, conflicting lengths, and invalid decimal lengths are refused. Truncated bodies fail closed. Requests whose declared body exceeds the buffered prefix do not reuse the client connection after a local response.

The response path applies these checks in order:

1. Local ban table.
2. Reserved internal routes (assets, challenge issuance, verification, honeypot, health, metrics).
3. Local GCRA admission.
4. Active policy snapshot: Shield inspection, reputation, ordered rules, scoring, static paths, bot signatures and default admission.
5. A valid session satisfies an admission challenge; it never overrides a denial or WAF finding.
6. An admitted reverse-proxy request streams to the origin; forward auth returns a verdict. The proxy reads the origin's response head (at most 16 KB), classifies the body framing by RFC 9112 (no body, Content-Length, chunked, or close-delimited), relays it exactly, and keeps the client connection open unless the response was close-delimited. Origin sockets left at a clean boundary by a persistent response return to a fixed pool of 256; a pooled socket the origin has closed is retried once on a fresh connection before any byte reaches the client, and only when the request body was fully buffered.

The server's primitive parsing, evaluation and verification routines take no allocator. This is a source/API contract, not a claim that the networking runtime, thread creation, startup, or persistent storage perform no allocation. Benchmark instrumentation is linked only into the standalone benchmark executable. Production metrics and snapshot reader counts still have real atomic costs; they are not benchmark overhead.

11. Authentication and work

A 32-byte master seed derives independent token, challenge, fingerprint and signing keys. The normal token is a 40-byte payload (version, the work level the holder solved, timestamp, expiry, rule hash, fingerprint) plus a 16-byte keyed BLAKE3 tag, URL-safe base64 encoded; a session admits only routes whose demanded work level it reaches (amended 2026-10-01). Ed25519 is an optional asymmetric token format. Tokens are bound to client address and User-Agent and expire. Trust forwarded identity only behind an ingress that overwrites it.

Issuing a challenge stores no entry. Successful verification inserts the authenticated tag into a 16-shard spent set. Failed bounded insertions leave every live tag intact and refuse admission. The set is per process and disappears on restart. In cluster mode, challenge keys are additionally derived from the node id at startup, while token keys remain shared. Challenge issuance and solution submission must reach the same node; the resulting token works on every member with the same seed. Distinct cluster node ids are required. This avoids cross-node replay without a request-path consensus transaction. It does not make replay state durable across restarts.

Tier 1 verifies `SHA256(challenge || ":" || decimal(nonce))` with bit-level difficulty. Tier 2 labels the sequential-work graph in `libs/crypto/src/posw.zig`; both native verification and the WASM prover import the same source. SID 0006 separates graph/hash evaluations from SHA-256 compression counts and states the research assumptions. Tests against a reference labeller are correctness evidence, not an independent cryptographic audit.

12. Policy and memory

The engine owns fixed-capacity rules, a radix trie, and dense Aho–Corasick tables. Signature capacity is one plus the sum of pattern lengths: no trie can require more states. This retains one transition lookup per input byte while avoiding the former oversized tables. Exact engine and table sizes are emitted by the benchmark using `@sizeOf`.

Dynamic storage owns two engine slots and an arena dedicated to each slot. A slot’s arena is reset only when that slot is inactive and prior readers have drained. Response rule names are copied into a bounded 128-byte buffer, then the reader releases its slot before any proxy I/O. Pointer publication and reader-counter operations use sequential consistency so the cross-atomic ordering argument is valid. Storage polling, SQL, vector search and persistence stay on the storage thread.

13. Measurement gates

```
zig build fmt
zig build test
zig build test -Dstorage=false
zig build test -Dcluster=true
sh benchmarks/run-all.sh
python3 benchmarks/distributed.py
zig build sid
```

The primitive harness warms inputs, resets mutable stores outside the timer before each of seven batches, and prevents pure work from being hoisted out of loops. No operation reads a clock. Its bot implementations search identical pattern sets with equivalent any-match semantics. Allocation values are source-audited expectations, not allocator measurements.

The distributed harness starts three actual daemon processes and six external client processes. It checks status codes, sessions across nodes, authenticated WAF denial, issuer-bound challenge rejection, reputation propagation, and continued HTTP service with one member down. Batch

timings include client and loopback costs. The harness tests loopback PSK and mutual TLS. WAN behavior needs separate measurements.

14. Release toolchain

Version 0.2.0 uses checksum-pinned Zig 0.17.0 on every release platform. macOS packages explicitly target version 15, matching this compiler's minimum OS contract. Build configuration uses lazy paths and deferred command arguments; source code uses the compiler's field names, field types and attributes directly. The console exports only its declared browser ABI, excluding compiler runtime globals. Compatibility changes preserve the HTTP, challenge, console and storage wire formats. The primitive benchmark record identifies its own source revision and compiler; platform qualification and performance acceptance are separate gates.

Zaxonlite and Paxos 0.7.0 do not yet publish Zig 0.17 packages. Their MIT library sources are included in `vendor/`, with original archive hashes and file digests. Library-only build entry points retain SQLite, `sqlite-vec` and optional OpenSSL configuration. No generated source or compiler-cache file is patched. Replace these snapshots with upstream compatible packages only after storage replay, cluster and native-platform qualification.

15. Native platforms and release boundaries

The packaged engine targets Linux x86-64 and ARM64, macOS Intel and Apple Silicon, and native Windows x86-64. Every platform runs the same HTTP parser, inspection and proof protocol. Platform code supplies bounded socket operations, process resource gauges and termination notification; it does not change policy or challenge semantics.

The application-independent socket library supplies native lifetime operations to both the HTTP service and streaming relay. Windows uses AFD handles owned by Zig's native Io backend, not Winsock socket identifiers. Each bounded operation owns a completion event. A timeout cancels and joins the outstanding driver request before any caller buffer can be reused. Shutdown and idle expiry use abortive Windows disconnects to release already pending reads; handles remain owned until their worker closes them, and shared state remains alive until the worker joins. Ordinary reply delivery and upgrade half-closes remain graceful. The upgraded byte relay keeps its existing bounded buffers and one connection worker. Ctrl+C and Ctrl+Break request ordered shutdown; forcibly terminating a process is not that shutdown contract. Private credential files use owner-only POSIX permissions or Windows DACLs permitting the owner, SYSTEM and Administrators. Unknown grant forms and absent ACLs fail closed.

The default release includes storage and the opt-in console but no cluster transport. Cluster deployments use a separate build with OpenSSL 3. Release archives carry the target, compiler, source commit and binary digest; native functional qualification is required before a platform archive is published. Cross-compilation alone does not qualify a native port.

16. Remaining limits

Shield is a bounded heuristic inspector, not a full SQL/HTML parser. Only the first 8 KB of body is inspected, and encoded fields longer than the canonical buffer are inspected raw. There is no HTTP/2, native TLS termination, global rate quota, durable replay set, native service manager integration, or measured volumetric network mitigation. Forward-auth inspection sees only bytes the ingress sends. Blocking workers and slow origins limit concurrency. No claim of full managed edge-service equivalence follows from a local benchmark.