

Magic Runtime — Enterprise Standards

February 2026 · v2.0 · magic.threadsync.io

Canonical Reference

Enterprise Standards

The six standards that define Magic Runtime's technical stack. These are policy, not preferences—all roadmap work, documentation, and procurement materials must align to this page.

1

API Plane How the runtime exposes functionality

Magic Runtime is a **governed controller runtime**, not an MVC web framework. It exposes a controller execution API with policy enforcement at every boundary.

COMPONENT	STANDARD
Runtime	Python 3.11+ — controller execution engine with sandboxing
Controller API	REST endpoints: <code>/api/execute/{name}</code> , <code>/api/health</code> , <code>/api/readyz</code>

Policy Enforcement

Every deploy and execute passes through the policy engine (deny-takes-precedence)

Contract Validation

JSON Schema validation at request and response boundaries — no unvalidated data

Transport

HTTPS/TLS required in production. Nginx terminates TLS, proxies to runtime.

Magic is not an MVC web framework. It's a governed controller runtime that can power any framework. FastAPI, Django, or Next.js sit in front; Magic controls what executes behind them.

2

Database How controllers access data

All database access goes through Magic's structured DB API by default. Raw SQL is an explicit, audited escape hatch.

ASPECT	STANDARD
Engine	PostgreSQL via <code>DATABASE_URL</code> environment variable
Recommended Workflow	Neon for dev/stage/pilot (branching, fast provisioning). Not a hard dependency.
BYO Postgres	First-class support: Aurora, RDS, Cloud SQL, on-prem — any Postgres-compatible engine
Default Access	Structured DB API: <code>self.db.query()</code> , <code>self.db.insert()</code> , <code>self.db.update()</code>

Escape Hatch

`db:rawsql` capability — must be declared in contract, denied by default in production policy

Migrations

Standard SQL migrations via `magic migrate` (Alembic-compatible)

`DATABASE_URL` is the portability layer. Swap Neon for Aurora by changing one environment variable. No code changes, no vendor lock-in.

3

Frontend How users interact with Magic

Node.js (Next.js) is the standard presentation layer for admin interfaces and optional starter scaffolding.

COMPONENT	STANDARD
Admin Console	Next.js <code>v2.3</code>
Starter Web App	Optional <code>web/</code> scaffold via <code>magic init --with-web v2.1</code>
Design System	Dark theme aligned with ThreadSync aesthetic
API Integration	Console consumes the same REST API as CLI — no special endpoints

The console is a client, not the product. Every operation available in the console is also available via CLI. Console adds visibility, not capability.

See the [Frontend Integration Guide](#) for TypeScript client, Next.js Server Actions, React hooks, and error handling patterns.

4

Controller How business logic is packaged and deployed

Controllers are contract-first, isolated, and deterministically packaged. The contract is the source of truth—not the code.

ASPECT	STANDARD
Contract	<code>contract.yaml</code> declares name, version, input/output schemas, capabilities, resource limits
Capabilities	Permission boundaries: <code>db:read</code> , <code>http:egress</code> , <code>secrets:read</code> , etc. Default: none.
Packaging	One directory per controller: <code>contract.yaml</code> + <code>controller.py</code> + optional <code>requirements.txt</code>
Manifest	<code>MANIFEST.json</code> auto-generated at deploy with content hashes for provenance
Dependencies	Default: no external deps. If present, must be pinned. Production policy can deny external deps.
Signing	Optional <code>MANIFEST.json.sig</code> (detached GPG/cosign) v2.2

Contract as "permission boundary." A controller cannot access anything not declared in its contract. The runtime enforces this at every call.

5

Audit How every action is recorded and verified

Every controller execution, deployment, and policy evaluation is recorded with `X-Request-ID` correlation, hash-chained for tamper evidence.

ASPECT	STANDARD
Correlation	<code>request_id</code> on every API call, log entry, and audit event
Integrity	SHA-256 hash chaining — each entry includes hash of previous entry
Storage	Append-only table in PostgreSQL. Optional WORM sink (S3 Object Lock, Azure Immutable Blob).
Export	JSONL with Merkle root verification for external SIEM integration
Verification	<code>magic audit verify</code> walks the chain and reports breaks <code>v2.2</code>
Retention	Configurable per environment. Default: 90 days hot, WORM sink for long-term.

Immutability is not hand-wavy. Hash chaining + WORM sink guidance means a compliance reviewer can independently verify the audit trail.

6

Licensing

How Magic Runtime is distributed

Open core model: the runtime, CLI, and contracts library are open source. Enterprise features are commercially licensed.

Apache 2.0

- Controller runtime
- CLI (init, new, validate, run, deploy)
- Contracts library
- Example controllers
- Documentation

Commercial License

- Policy engine + approval workflows
- SSO / RBAC integration
- Audit retention + WORM export
- Admin Console (Next.js)
- Helm chart + Kubernetes operator

Open core builds trust; enterprise layer pays the bills. Developers evaluate with the full runtime. Procurement buys governance, compliance, and support.

Version Roadmap

When each standard becomes fully implemented

VERSION	CODENAME	STANDARDS DELIVERED
v2.0	Foundation	API plane, database (structured API), controller contracts, basic audit

v2.1	Flask DX	Host CLI (pipx), <code>magic init</code> scaffolding, Neon-ready docs, <code>web/</code> starter
v2.2	Enterprise Governance	Policy engine, audit chain verification, RBAC, controller signing
v2.3	Admin Console	Next.js console, ThreadSync integration, dark theme
v2.4	Platform	Helm chart, Kubernetes operator, standalone binary, official images

CLI Lanes Two-lane strategy for different contexts

LANE	CONTEXT	COMMAND PREFIX	AVAILABLE
Production	Deployed environments, evaluation, CI	<code>docker compose</code> <code>exec runtime</code> <code>magic ...</code>	v2.0
Development	Local dev, prototyping, tutorials	<code>magic ...</code> (via <code>pipx install</code>) <code>magic-runtime</code>	v2.1

Pinned runtime = pinned tooling. In production, the CLI version matches the runtime version inside the container. No version drift.

Questions About Enterprise Standards?

Request a technical architecture review or discuss how these standards map to your compliance requirements

Contact Enterprise Team — enterprise@threadsyntax.io