AUTOMATIZE · CODE QUALITY PLATFORM

Faultr.

Self-hosted code quality analysis — SonarQube-grade, without the SonarQube server.

Faultr clones your repositories, builds them, and runs real analyzer plugins inside its own process. On top of static analysis it layers AI code review, a five-dimension audit suite, two adversarial review gates, architecture diagrams and an eleven-method codebase valuation. Every run is a permanent snapshot; any two can be compared.

No analysis server | **No licence gate** | **13 model providers** | **Tamper-evident audit** | **Air-gap capable**

11	5	10	99%
PIPELINE PHASES	AI SUBSYSTEMS	DEPENDENCY ECOSYSTEMS	INSTRUCTION COVERAGE

REGULATED IT · DEFENCE · DUE DILIGENCE · PLATFORM TEAMS

PRODUCT BROCHURE 2026.1

THE PROBLEM

Code quality tooling is either a server estate or a data-egress decision.

The established option is a separate analysis server with its own database, its own upgrade cycle and a licence that scales with lines of code. The modern option sends your source to somebody else's cloud. Neither answers the question a head of engineering actually has, which is not “how many code smells” but “what is the state of this codebase, what is it worth, and can I prove it.”

Server estate

A second platform to run, patch and license — plus a scanner in every pipeline.

Source egress

Cloud analysis means your intellectual property is analysed on infrastructure you do not control.

Rules without judgement

Static rules never see the design problem — the god object, the missing boundary, the wrong abstraction.

No commercial answer

No quality tool tells you what the codebase would cost to rebuild — the number the board actually asks for.

THE SOLUTION

Faultr *is* the scanner.

There is no analysis server to deploy and no scanner command line to wire into a pipeline. Faultr implements the analyzer plugin interface itself and drives industry-standard analyzer JARs directly, in its own process, against a working tree it cloned and built. Drop a plugin JAR in, or pull one from the public catalogue, and it runs.

The build is the zeroth quality gate: if the code does not compile, that is the finding, and everything downstream is skipped rather than reported against stale artefacts.

One Spring Boot application against one PostgreSQL database, installed by a browser wizard, with no telemetry and no call home. Point the model layer at a local endpoint and the entire platform runs air-gapped.

ELEVEN PHASES

- Build and compile
- Native size and complexity metrics
- Test result ingestion
- Dependency and vulnerability ingestion
- Analyzer plugin sensors
- Custom AI sensors
- AI code review
- Five-dimension audit suite
- Two adversarial review gates
- Architecture diagrams
- Codebase valuation

Absent is not zero

A missing vulnerability report produces *no* metrics rather than zeroes — so a project cannot pass a vulnerability gate by never having been scanned.

Cancellation is real

Stop kills the entire build process tree, checkpoints at every phase boundary, and refuses the final save. A stopped run stores nothing and never becomes “the latest analysis”.

Faults are isolated

A plugin that throws becomes a recorded warning with its stack trace, and the analysis continues. One bad analyzer never costs you the run.

ANALYSE

Static analysis, the quality model, and the AI layer.

1 Repositories & scheduling

- A project is one Git repository, with its own branch, build command, test command, timeout and environment
- Per-project polling schedule; head-ahead detection when the branch moves past the last analysis
- Provider integration with GitHub, GitLab, Gitea, Forgejo and Codeberg — including self-hosted instances — with token-based cloning
- **Git Explorer**: file tree, blob viewer with linkable lines, commit log, per-commit diff, branch and tag switching, filename and content search, contribution heatmap
- Per-project exclusions, disabled plugins and configuration copied from another project
- Concurrency fairness at platform, tenant and organisation scope
- Live phase-by-phase progress on the report page

2 The quality model

- **Quality profiles** per language, inheritable, with severity and parameter overrides — snapshotted into every run so a report is always reproducible
- **Quality gates** with metric conditions on overall or new code, and three curated presets: strict, balanced and lenient
- A rule catalogue with descriptions, types, severities, tags and remediation effort
- Issue triage: open, confirmed, resolved, will-not-fix — with AI explain and AI fix per issue
- **Suppressions** on three combined axes — rule, file pattern and severity band — applied before metrics are computed, with a live preview of how many findings they match
- Suppression staleness is measured, not guessed: match counts are written back, so dead suppressions are visible
- Test results from standard report formats, and coverage from standard coverage reports
- Vulnerability findings ingested from your build's own dependency scan, with severity and score metrics

3 Five AI subsystems

- **Code review** of only the files changed since the last analysed commit — summary, risks, naming, design and suggestions per file, then an overall risk verdict
- **Audit suite** across security, performance, compliance, quality and functional dimensions, each with an editable prompt template, a score out of a hundred, a letter grade and individual findings with recommendations and confidence
- **Two review gates** — a constructive architectural peer review, and an adversarial red-team pass over authorisation boundaries, concurrency, database safety, data integrity and input validation
- **Custom AI sensors** you author yourself, which become first-class rules participating in profiles and gates like any static rule
- **AI-generated analyzer plugins** — describe a rule in plain language, review the generated source, compile it, and have the rule active in your profile
- Thirteen providers, seven of them local or self-hosted, with encrypted keys, connectivity tests and monthly token and cost reporting

4 Architecture visualisation

- Five diagram types — class, sequence, state, entity-relationship and dependency
- Per-project configuration of included packages, depth, detail and limits
- **Architectural rules** such as no-circular-packages, with violations listed under the dependency diagram
- Pan and zoom, source copy, SVG and PNG export, and comparison against the previous snapshot with an optional plain-English summary of what changed
- An ad-hoc sequence explorer that traces any method against the live working tree — and states explicitly what the trace could not resolve

Every finding is portable to an agent. Vulnerability rows and audit findings export as ready-to-paste remediation prompts — package coordinates, installed and fixed versions, severity, advisory link, and which manifest to edit. Issues export as a Markdown checklist. The output of the analysis is designed to become the input of the fix.

COMPARE, VALUE, PROVE

History, valuation and the audit trail.

5 Snapshots & comparison

- Every completed run is an immutable snapshot with commit, branch, author, phase timings and gate verdict
- Compare any two runs: metric deltas in absolute and percentage terms, issues classified as new, resolved or persisting, and changes to build status, gate status and the AI summary
- Chart any metric across the project's entire history, with a click through to the snapshot behind any point
- Dependency inventory across **ten package ecosystems**, including vendored directories
- A security tab listing advisories with severity, score, affected artefact, installed and fixed versions, and links to public databases

6 Codebase valuation

- **Eleven methods** — two COCOMO variants, four function-point methods, internal and outsourced reproduction cost, an AI-assisted delivery model, a documentation model, and a headline pre-revenue asset figure
- Every figure explains itself: measured inputs, formula, substituted numbers and intermediates are recorded while the number is computed, so the explanation can never drift from the result
- Two interactive financial models — a market and discounted-cash-flow view, and a partnership view — recalculated live
- Export as PDF with the full derivation appendix, and as a **spreadsheet where every derived cell is a live formula over named assumption cells** — so a reviewer can change an assumption and watch the valuation move

7 The audit trail

- Append-only by construction *and* by database trigger — an update or delete attempt is refused at the database, not merely discouraged in code
- A **SHA-256 hash chain** over a canonical encoding, with a gapless sequence guaranteed by a database lock
- Nightly verification that distinguishes tampered content from a broken chain and from a sequence gap, recording its own result back into the trail
- Entries survive the rollback of the failure they recorded, and the writer never throws
- Correlation identifiers grouping a whole analysis run, request or poll
- **Cursor-paged export** for a security information and event management system — it cannot skip or duplicate entries the way an offset can
- Passwords, tokens and keys are never written; repository URLs are stripped of credentials

8 Administration

- Two-level tenancy with a superadmin context switcher across tenants and organisations
- Plugin administration: discovery, enable, reload, update check, upgrade, redownload and a catalogue sync with checksums
- Log viewer with tailing, download and runtime logger-level changes
- Scheduled collection with heap recording; per-organisation retention with a nightly sweep that never deletes an in-flight run or a project's latest
- A six-step first-run wizard that tests the database, installs the schema, creates the first account, generates the secrets and writes a service unit — in fourteen languages

Language reach

Native metrics across eighteen file types, dependency parsing across ten ecosystems, and analyzer plugins for Java, JavaScript, TypeScript, Python and XML out of the box.

Zero network at the edge

No content delivery network anywhere. Charts, syntax highlighting, icons and the interface kit are all vendored, enforced by a strict content security policy.

Proven on itself

99% instruction and 98% branch coverage on the in-scope bundle, 243 test files, and a build that fails on any package dependency cycle.

WHO BUYS IT

Two buyers, two entirely different conversations.

The engineering buyer
A head of engineering or platform lead who wants continuous quality without running a second server estate, and who cannot send source code to a hosted analyser. They buy the analysis, the AI review layer and the fact that it installs itself.

The commercial buyer
An investor, acquirer, escrow agent or finance director who needs a defensible number for a codebase — for due diligence, for an intangible asset, for a research and development claim. They buy the eleven-method valuation with its derivations and its live spreadsheet.

Banking & fintech
Tamper-evident audit, event-management export and full self-hosting for organisations whose code is a regulated asset.

Insurance
Long-lived core systems where an architecture diagram and a dependency inventory are as valuable as the issue list.

Government & defence
Air-gapped deployment with local models: no call home, no content delivery network, every dependency inside the artefact.

Healthcare software
Compliance-dimension auditing over the customer's own code, alongside a hash-chained record of who analysed what and when.

Critical infrastructure
Vulnerability ingestion and gate conditions that cannot be satisfied by simply never running a scan.

Managed service providers
Tenant and organisation hierarchy with per-tenant concurrency fairness and per-organisation retention — a service-provider shape, not a single-company one.

Software due diligence
Technical diligence for mergers and acquisitions delivered as a printed dossier with every figure derived and every assumption adjustable.

Escrow & IP valuation
An independent, reproducible, method-plural valuation of a codebase as an intangible asset.

Cost-driven migrations
Organisations reconsidering a per-line-of-code analysis licence, who want the same analyzer plugins without the server or the meter.

WHO USES IT

Platform operator · **Organisation administrator** · **Analyst / developer** · **Security auditor** · **Due-diligence reviewer**

Three enforced roles, plus non-human actors the audit trail distinguishes by name: the scheduler, the system and the anonymous caller. An external auditor is served directly by the export endpoint and the event stream, without needing an account.

LANGUAGE & ECOSYSTEM COVERAGE

AREA	COVERAGE
Source languages measured	18 file types
Dependency ecosystems	10
Diagram types	5
Valuation methods	11
Model providers	13
Setup wizard languages	14

TECHNICAL SPECIFICATION

One JAR. One database. No second platform.

Runtime	Java 21 on Spring Boot 4, single JAR or system service
Database	PostgreSQL 15 or later, 60 versioned migrations
Analysis	Analyzer plugin interface implemented in-process; one isolated class loader per plugin, cached across runs
Builds	Any build tool — Maven, Gradle, npm, dotnet, Python and anything else that runs in a shell
Front end	Server-rendered with vanilla ES6; no framework, no build step, no remote asset
API	REST with OpenAPI, a uniform response envelope, and typed request and response objects throughout
Concurrency	Separate pools for analysis, post-analysis fan-out and repository work
Install	Six-step browser wizard, no account required, ending in a running service unit
Scale	~66 000 lines of application code, 55 entities, 243 test files

SECURITY & GOVERNANCE

- Three roles with method-level authorisation across the whole surface
- Tenant and organisation isolation checked in every service — with cross-organisation reads answered as **not found rather than forbidden**, so an identifier is never confirmed by a refusal
- **AES-256-GCM encryption at rest** for repository credentials, provider tokens and model keys
- Passwords hashed with bcrypt; sessions persisted so logins survive a restart, capped per user
- Production boot refuses a shipped default encryption key or signing secret
- Strict content security policy, frame denial, cross-site request protection, trusted-proxy allow list
- Outbound request validation blocking private and loopback ranges
- Hash-chained, trigger-protected audit trail with nightly verification and event-management export
- Dependency vulnerability scanning wired into the build and gated at score seven
- Build fails on any package dependency cycle

> Single sign-on and multi-factor authentication are not implemented in the current release, and there is no data-subject export or erasure tooling. We would rather you heard that from us than found it in a proof of concept.

COMMERCIAL

No licence server. No line-of-code meter.

Faultr is licensed per installation or per tenant. Analyse a hundred repositories or a thousand; the price does not move. The analysis runs on your machines against your own model keys, so there is no gate to call home to and no code leaving your network.

REVENUE	PER YEAR
Up to €500k	€900
€1M	€1,500
€5M	€2,500
€25M	€5,500
€100M	€13,000
€500M	€37,000
€1B	€52,000

Annual licence on your own turnover, ex VAT. The schedule is marginal like income tax — each slice of turnover is charged at its own rate, so there are no cliffs — with a €900 minimum and any yearly increase capped at 25%. Every tranche carries unlimited users, projects and organisations inside one tenant, self-hosting and support that rises with the tranche. Hosting and hardware are yours, AI runs on your own provider keys and is billed by them, and consultancy, setup, migration and training are quoted separately. Illustrative: a per-developer analysis subscription at €30 a month bills a sixty-person team €21,600 a year; at €5M turnover this is €2,500.

NEXT STEP

Give us one repository.

We will build it, analyse it, review it, diagram it and value it — and hand you the dossier, whatever it says.

Sales tim@automatize.eu
Partners luc@automatize.eu
Web automatize.eu