

Magogi Foundation — federation visual model

Atlas v0.2 · the **mgf-*** ecosystem — 31 sibling Python libraries, one standards corpus, end-to-end at four zoom levels, with four detail plates

Source of truth: `mgf-standard` · companion to the standards corpus & library registry · June 2026

This is the visual model for the Magogi Foundation federation. It presents the system end-to-end at four zoom levels — from its place in the world, through its layers and its release pipeline, down into the conformance system that holds it together. Every library is a sibling under the `mgf.*` namespace; one cornerstone (`mgf-common`) sits under all of them, and one corpus (`mgf-standard`) governs them all.

COLOUR KEY — READS ACROSS EVERY LEVEL

- federation library (released)
- presentation / interface
- foundation (corpus · infra)
- artifacts (wheels · registry)
- gate refuses
- gate warns
- planned / gated

A note on the dashes. Solid components are built and released today — versioned wheels on the self-hosted index, gated by live CI. Dashed components are designed but *gated*: the self-hosted infrastructure programme (Vault, central logging, the IaC for the CI hosts) is staged but not yet stood up. When a gate opens, the matching dashed element activates with no re-architecture. Nothing here claims a capability before it exists.

WHAT'S INSIDE

- L0 · the whole federation in context** — one orchestrator, one cornerstone, one corpus
- L1 · the layered architecture** — presentation → cornerstone → families → foundation
- Plate A · **the dependency graph** — how the libraries actually wire together
- Plate B · **the library families** — all 31, by family, with purpose
- L2 · the release pipeline** — scaffold → gate → conformance → registry
- Plate C · **the CI gate up close** — every step the pipeline runs
- L3 · inside the conformance system** — the heart: rules, checks, the Finding
- Plate D · **the operating model** — how cross-repo work actually happens

The whole federation in context

A developer drives the federation through one orchestrator — `mgf-fed` — and every library it touches imports the cornerstone, `mgf-common`. There is one place rules live (`mgf-standard`) and one place tests are driven from (`mgf-test-supervisor`), so behaviour never drifts between siblings.

Each library is built, conformance-gated, and published as a versioned wheel to a **self-hosted devpi** index; CI runs on a **self-hosted Woodpecker**, and both live on **Infomaniak**, provisioned by the federation's own `mgf-cloud-provision`. The remaining infrastructure — Vault and central logs — sits beyond the gated frontier until its IaC lands.

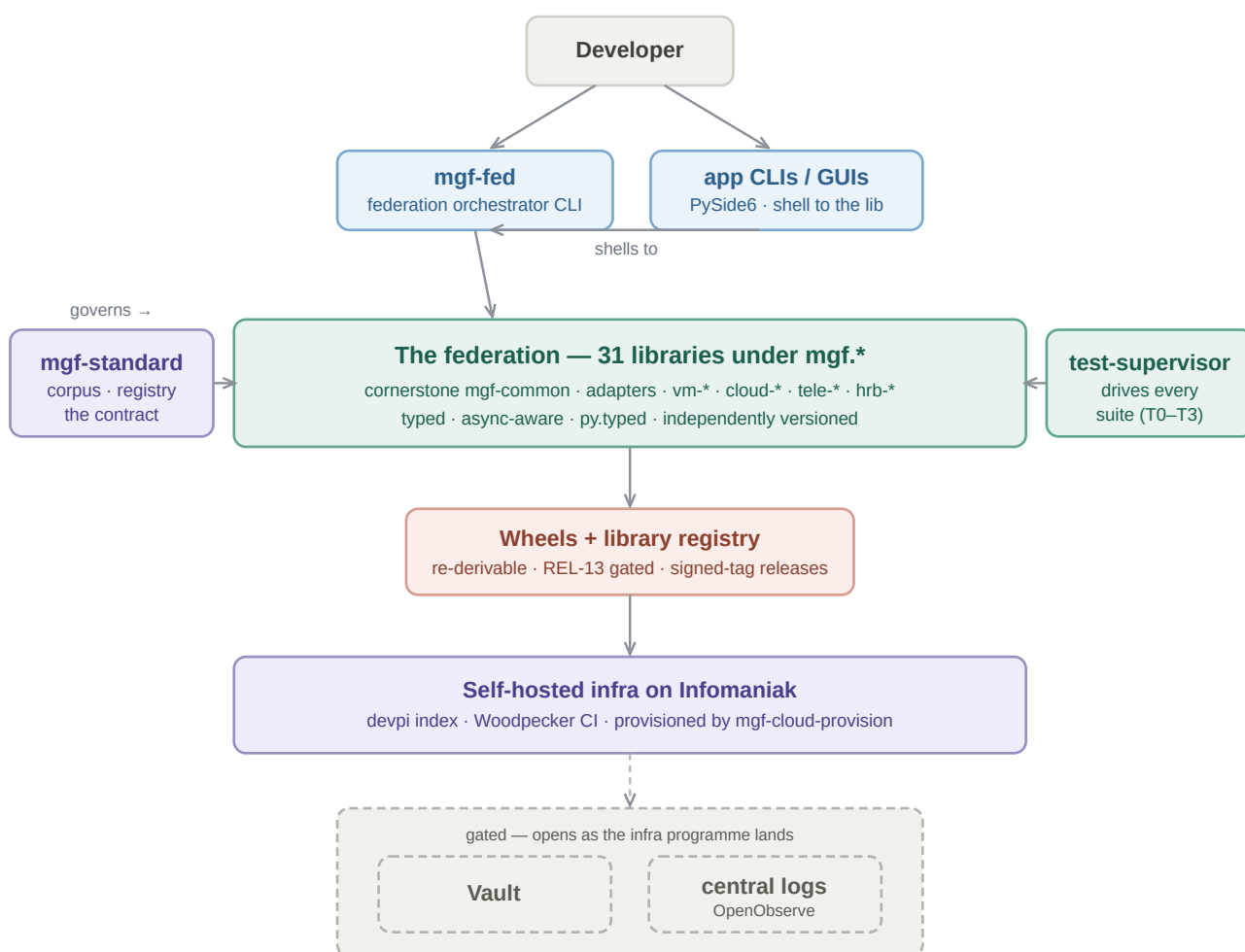


Figure 0 — the federation in context, and the built-vs-gated frontier

The layered architecture

The same system, opened into five layers. Presentation is the orchestrator and the app front-ends. Beneath it, the cornerstone `mgf-common` gives every sibling the same exceptions, config, and structured logging/tracing. The breadth of the federation lives in the next layer — four families of domain libraries — and a single test harness drives them all.

At the foundation sits `mgf-standard` (the corpus, registry, and federation contract) alongside the self-hosted infrastructure and a pinned toolchain. Colour marks what is released today versus what is gated.

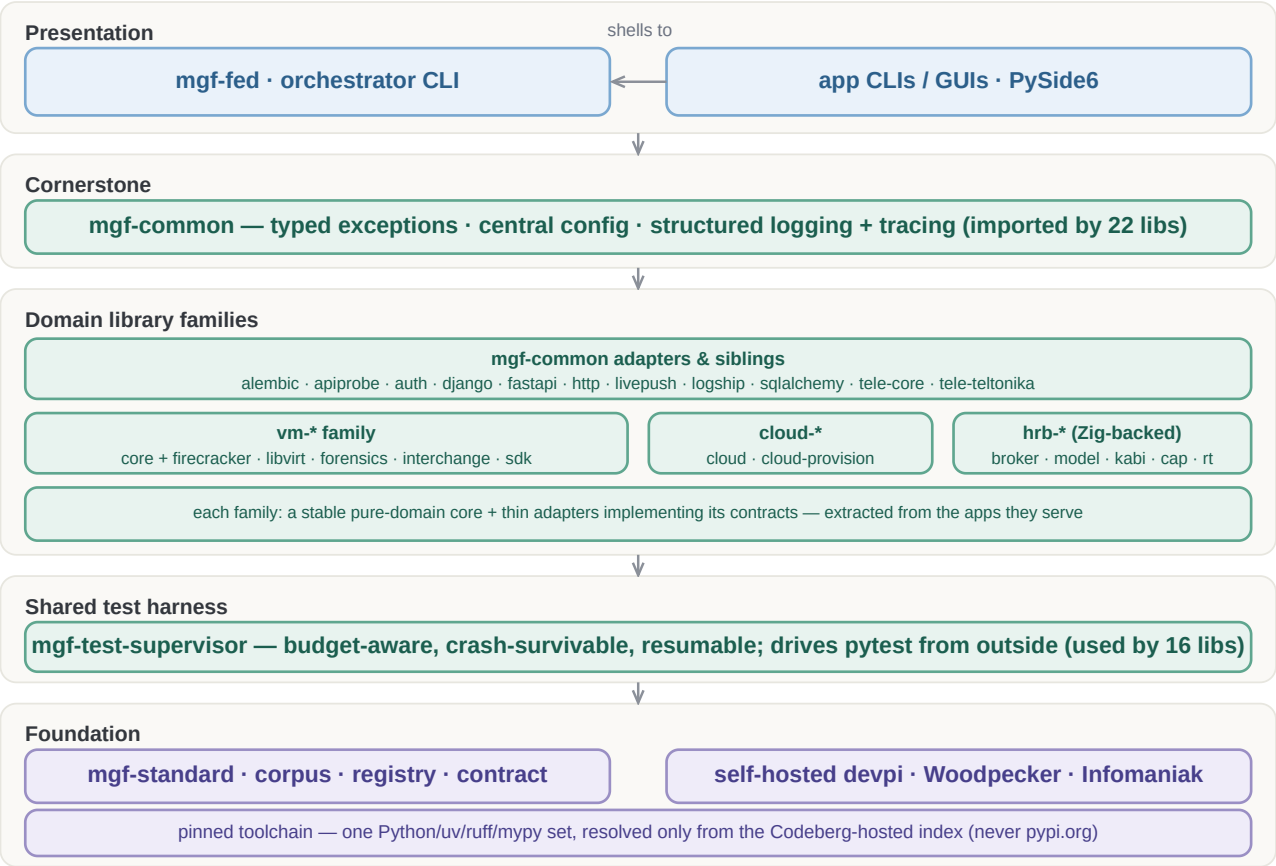


Figure 1 — presentation, cornerstone, library families, test harness, and the foundation

The dependency graph

How the libraries actually wire together. `mgf-standard` is the root — it depends on nothing; it ships the corpus, the registry, and the scaffold. The cornerstone `mgf-common` and the test harness `mgf-test-supervisor` are a co-foundational pair, and **every domain library depends on both** (those universal edges are summarised, not drawn, to keep the graph legible).

Each family then hangs off a stable hub — `vm-core`, `tele-core`, `hrb-model`, `cloud` — that its adapters implement against. The few **cross-family** edges are the interesting ones: `cloud` reuses `fastapi`, the orchestrator `mgf-fed` reaches into `cloud` to provision hosts, and within the adapters `fastapi` builds on `sqlalchemy` + `livepush`.

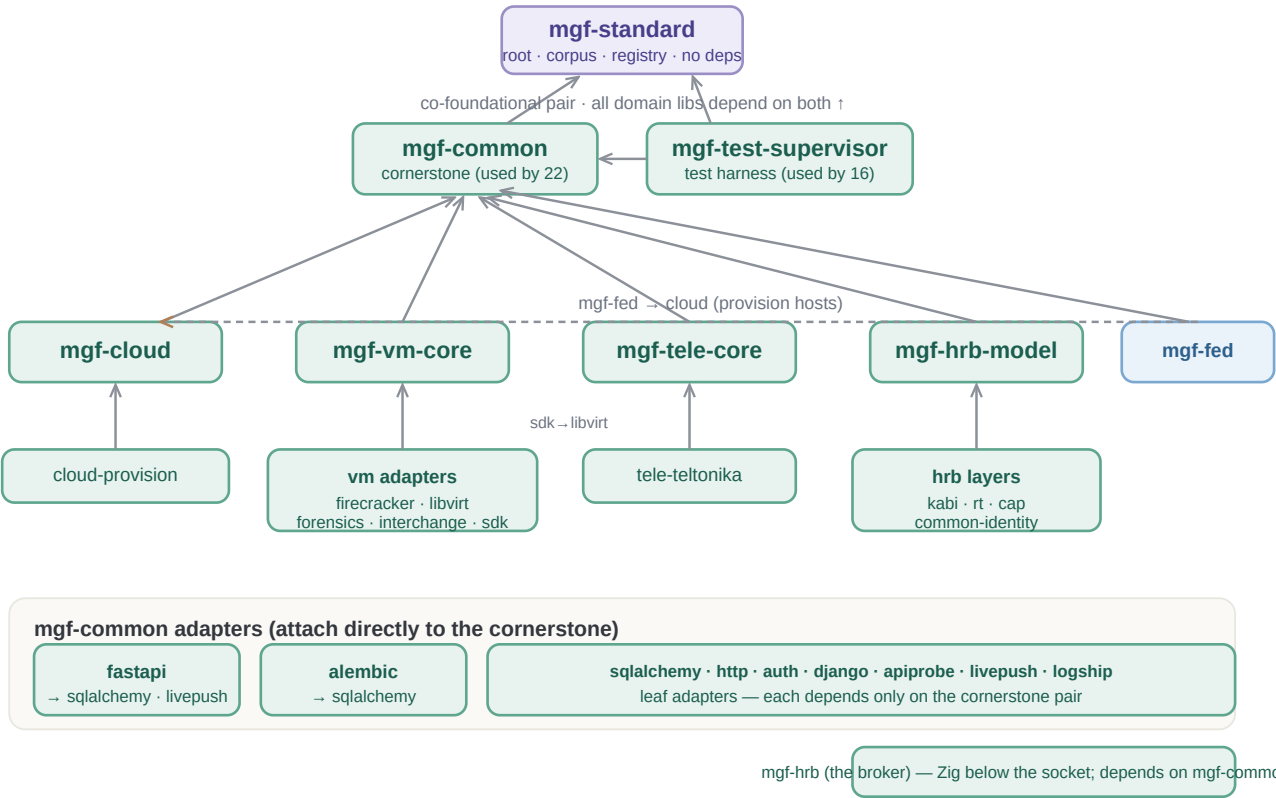


Plate A — the intra-federation dependency graph (universal cornerstone edges summarised)

The library families

All 31 libraries, grouped by family. Versions are as of the registry generated 2026-06-21. The four core/infra libraries hold the federation up; the rest are domain siblings that an application composes.

Core & infrastructure

the spine

mgf-standard 0.1.58	the engineering-standards corpus, federation contract, and library registry — the source of truth
mgf-common 0.47.0	shared infrastructure: typed exceptions, central config, structured logging + tracing (the cornerstone)
mgf-test-supervisor 0.1.16	budget-aware, crash-survivable, resumable test supervisor that drives pytest from the outside
mgf-fed 0.5.0	federation orchestrator — gates, local/remote release, status, drift-check, scaffold, provisioning

cloud-*

self-hosted infra

mgf-cloud 0.1.1	cloud-resource interface (Infomaniak first, AWS-ready); ships a CLI + a local cost estimator
mgf-cloud-provision 0.1.0	declarative, idempotent provisioning of the self-hosted services (devpi, CI agents, Vault) on mgf-cloud

mgf-common adapters & siblings

the breadth

mgf-sqlalchemy 0.5.8	async SQLAlchemy helpers — typed engine factory, sessionmaker, Postgres RLS tenant-scoping
mgf-alembic 0.4.9	async-aware Alembic env.py helper — one-call <code>configure_env</code> replaces ~40 lines of boilerplate
mgf-fastapi 0.6.11	FastAPI adapters — request-id, exception translation, lifespan, Svix webhooks, ip-allowlist
mgf-django 0.4.8	Django adapters — AppConfig + middleware (request-id, context) + JsonFormatter + settings bridge
mgf-http 0.4.10	async HTTP client — typed httpx wrapper with retries, timeouts, header redaction, OTel-aware logging
mgf-apiprobe 0.5.8	REST API verification — typed probes, checks, and findings
mgf-auth 0.1.7	ISP-split auth verifier seams — SessionVerifier + WebhookVerifier + a Clerk adapter
mgf-livepush 0.1.7	realtime push — SSE streaming, a pluggable pub/sub broker seam, per-key connection-slot caps
mgf-logship 0.1.5	centralized log shipping — a fail-open handler that batches canonical LogRecord JSON and POSTs it

vm-* VM management

mgf-vm-core 0.1.12	pure VM-domain library — state machine, VmSpec, contracts, fakes (the stable hub)
mgf-vm-libvirt 0.1.9	libvirt / KVM / QEMU HypervisorBackend + StorageBackend
mgf-vm-firecracker 0.1.3	Firecracker / jailer microVM backend
mgf-vm-forensics 0.1.7	pcap / dpkt analyzers, case management, top-talkers, DNS, TLS-SNI
mgf-vm-interchange 0.2.3	OVA / OVF import & export
mgf-vm-sdk 0.1.8	vm-vmanager-app client SDK — VmManagerClient with in-process and gRPC transports

tele-* telematics · **hrb-*** hardware broker · early

mgf-tele-core 0.1.7	vendor-neutral telematics seam — the TelematicsProvider ABC + codec-agnostic AVL frame types
mgf-tele-teltonika 0.1.7	Teltonika Codec 8 / 8 Extended decoder + TelematicsProvider implementation
mgf-hrb 0.1.6	hardware resource broker — userspace arbitration of PCIe/USB/GPU/CPU/memory; Zig below the socket
mgf-hrb-model 0.1.0	the HRB family hub; with kabi · cap · rt (v0.1.0) — the kernel-ABI, capability, and runtime layers (early)
mgf-common-identity 0.1.0	identity primitives built on hrb-kabi + hrb-model (early)
mgf-common-zig ·	with mgf-website-template — early scaffolds (a Zig common base and a website starter, pre-release)

The library lifecycle pipeline

This is how a library goes from a scaffold to a released, registry-tracked wheel. A new repo is generated from one cookiecutter, then driven by `mgf-fed`. Every change runs the full CI gate on the self-hosted Woodpecker — and a failure at any step **refuses with no release**.

On pass, conformance is checked twice: `app-doctor --standards` audits the repo against the corpus, and `drift-check` proves every generated file still matches its template. Only then does the wheel publish to devpi, after which the registry is **re-derived** from the fleet and gated for consistency and currency (REL-13). Verify-before, re-derive-after — the same spine that makes the federation auditable.

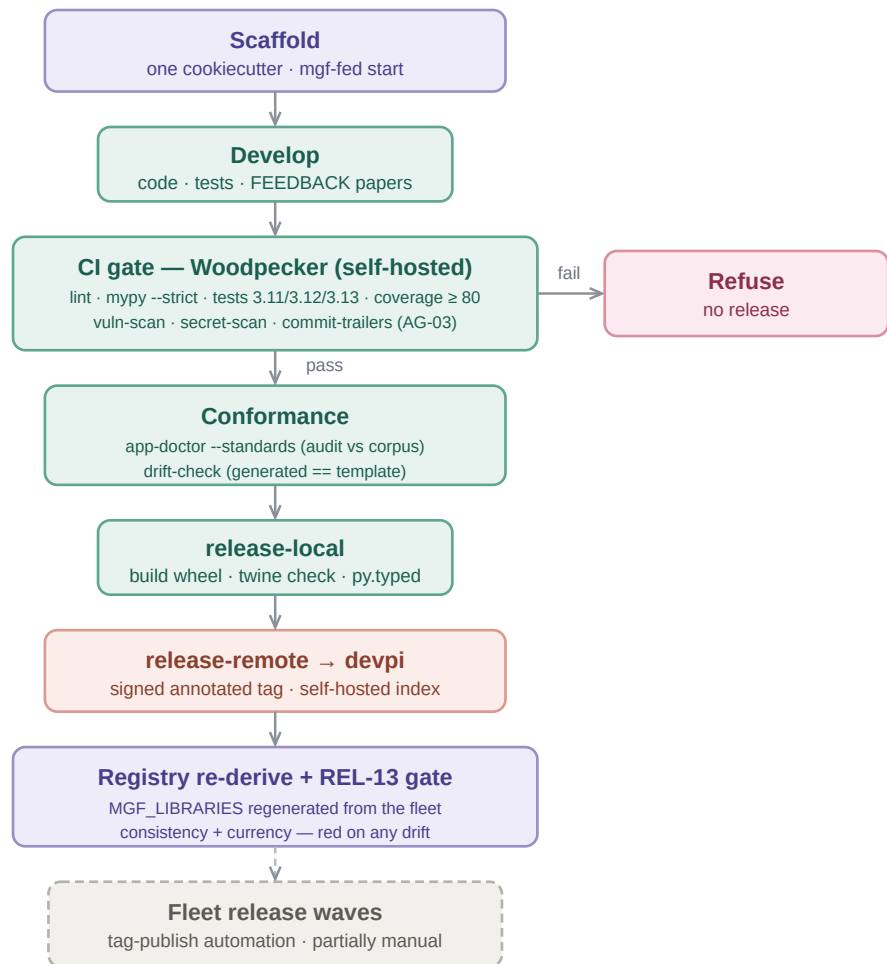


Figure 2 — from scaffold to a re-derived, registry-tracked release

The CI gate, up close

Every generated repo runs one canonical `.woodpecker.yml`, rendered from the corpus template by `mgf-fed`. On every commit it runs the merge gate below — **any red step refuses the merge**. A separate group runs only on a release tag. Each step maps to a rule family in the corpus; CI is how the standards stop being prose.

On every commit — the merge gate		red ⇒ no merge
clone	full (non-partial) git checkout — pinned plugin-git, no treeless fetch	
lint	ruff + import-linter (layering) + a placeholder-URL guard · DP-01 · WF-02 · OP-03	
commit-trailers	every commit carries Signed-off-by + Co-Authored-By · AG-03	
typecheck	mypy --strict over the package · TM-*	
test-3.11 / 3.12 / 3.13	pytest on each Python; coverage ≥ 80 gated on 3.12 · TS-06 · WF-10	
vuln-scan	pip-audit against the dependency tree; carve-outs declared in [tool.mgf-fed] · SC-10	
secret-scan	gitleaks over working tree + history with the project allowlist · SC-01	
supervisor-self-check	mtest verifies the test supervisor itself reports failures correctly · TS-24	
supervisor-tier-0	drive the suite through mtest — structured JSONL run record + failure bundle · TS-23/25/27	
build	build the wheel, twine check, assert py.typed is shipped · PKG-* · AP-13	
sbom	emit a software bill of materials for the build · SC-*	

On a release tag only		v* tags
tag-coherence	the tag matches the pyproject version · REL-*	
smoke	install the freshly-built wheel into a clean venv; the console script reports --version · TS-21	
publish	upload to the self-hosted devpi index (never pypi.org) · PKG-06	
verify-devpi	re-resolve the just-published wheel from the index to prove it installs · REL-*	

One template, no drift. No repo hand-edits its pipeline — `mgf-fed drift-check` fails CI if a generated file diverges from the corpus template, so a fix to the gate propagates fleet-wide from a single source.

Inside the conformance system — the heart

The corpus is **271 MUSTs** across 22 rule families — testing (TS), agent/governance (AG), security (SC), releasing (REL), workflows (WF), packaging (PKG), and more. They are enforced two ways: a **test harness** that runs tier-0 through tier-3 suites, and an **app-doctor** that audits the repo's shape against the corpus.

Every audit result is exactly one **Finding** — carrying its locus, the rule, why it matters, a suggested fix, and a severity. The doctor's checks are **corpus-aware**: each stays dormant until its rule is in the pinned Index of MUSTs, so a repo on an older corpus is never failed by a rule it predates. Severity decides the outcome: a **fail** reds the gate, a **warn** informs (and tightens to fail when the rule is promoted), and the whole thing stands on one stance — single source of truth, generated-not-edited, pin everything, fail-closed.

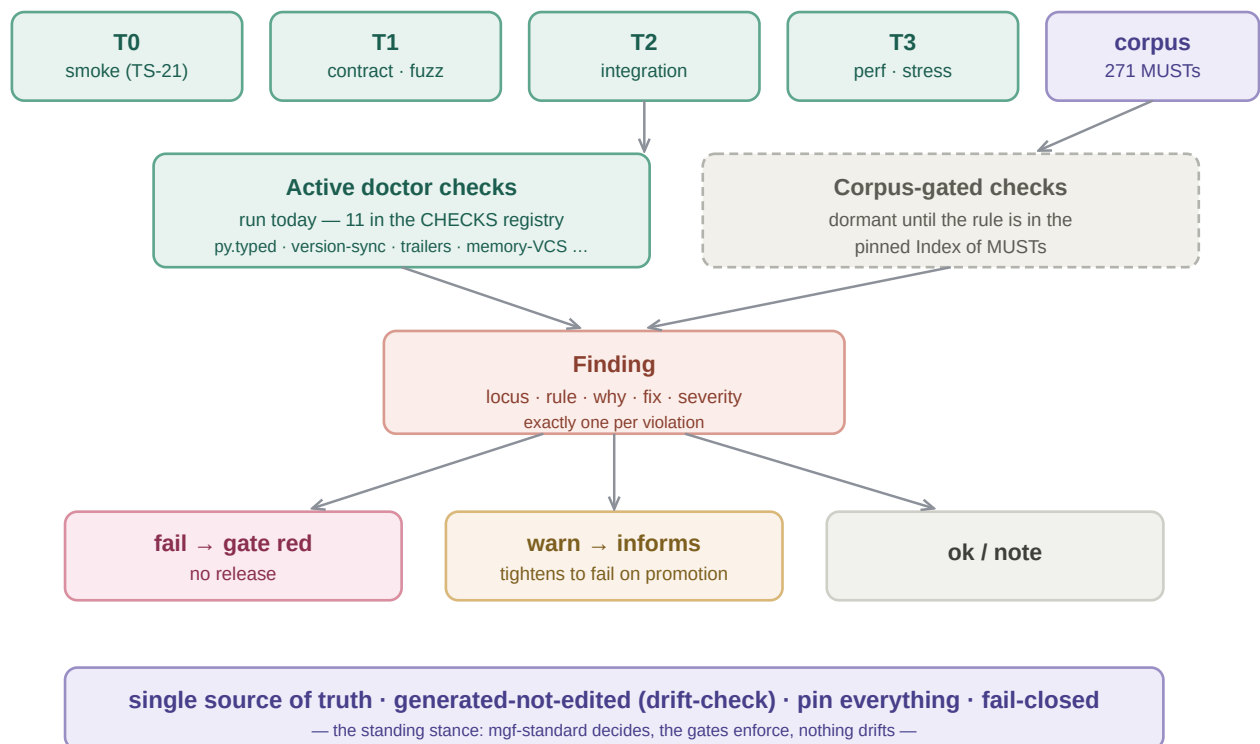


Figure 3 — test tiers, the corpus, the doctor's check registry, the Finding, and the standing stance

The operating model

The federation is not 31 repos worked by hand — it is one orchestrator, **mgf-fed**, that runs the recurring cross-repo work consistently and trackably. The same verb (**gates**, **drift-check**, **release-remote**, ...) runs identically over every member, so there is no per-repo improvisation to drift.

Learning flows back the other way: a fix or a lesson in one repo becomes a **FEEDBACK paper** that is relayed into the others and, when it generalises, promoted into the corpus as a new rule — which the gates then enforce everywhere. The irreversible, outward-facing steps — publishing a release, provisioning a host — stay **human-gated**: the orchestrator proposes and stages; only the maintainer enacts.

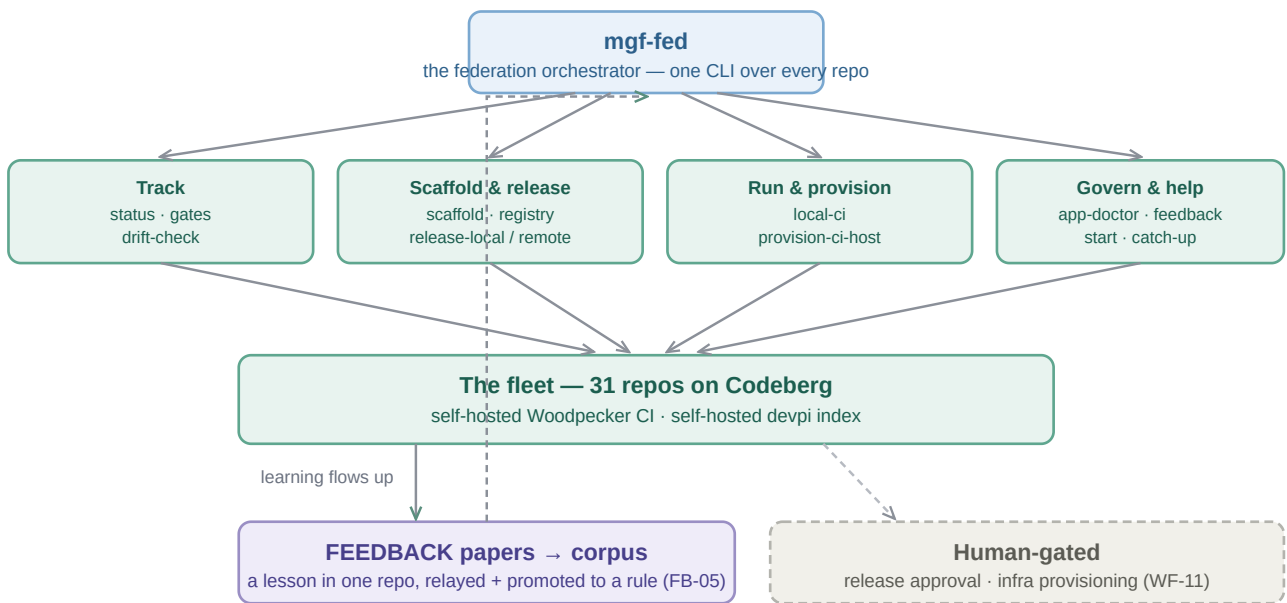


Plate D — one orchestrator over many repos; learning flows up into the corpus; irreversible steps stay human-gated