



# VigilDQ — a guided tour

AI-native data quality with guardrails built into the architecture

A screenshot walkthrough of the real product — read it in ten minutes, no meeting required. Every screen shown is live software, not a mock-up.

# Executive summary

---

## The idea

- An LLM drafts data-quality rules from a profile of your data.
- Hard structural guardrails vet every proposal — then a human approves.
- Approved rules run as read-only SQL inside your warehouse.

## What makes it different

- Guardrails are architecture, not prompt instructions.
- The AI can never approve its own work — the lifecycle forbids it.
- PII is masked before the AI or the browser ever sees a value.

## Where it stands

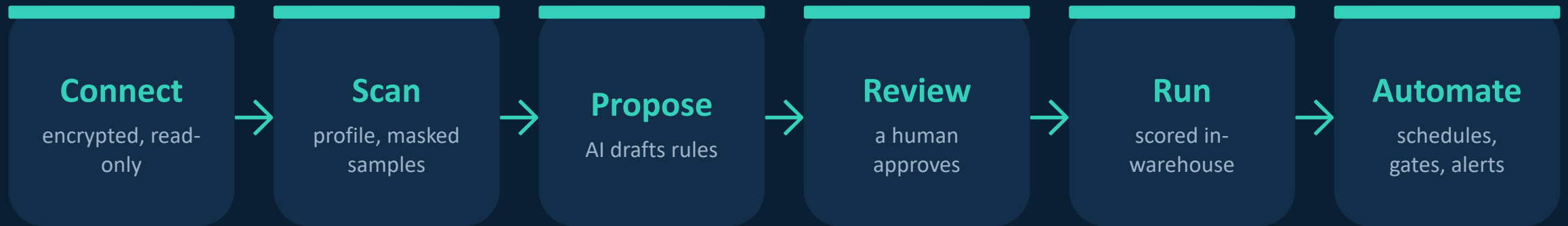
- PostgreSQL and Databricks / Unity Catalog validated live.
- 199 automated tests; full audit trail; multi-tenant service.
- SQL Server dialect implemented, live validation next.

## Who it's for

- Pharma, banking, insurance — teams that must PROVE data quality.
- Data platform owners who want AI speed without AI risk.

# The workflow

---



Six steps from raw table to governed, continuously-scored dataset — the next ten slides walk through them on the real product.

# Connect a warehouse, define the dataset

The screenshot shows the VigilIQ control room interface. The top navigation bar includes 'observe', 'datasets', 'rules', 'runs', 'trends', 'schedules', 'alerts', 'audit', 'api log', and 'admin'. The user is logged in as 'michael@acme.com' with 'admin' privileges. The 'ADMIN' section is active, showing a 'connections' tab. A 'New connection' form is visible, with a dropdown menu for 'select a database...'. Below the form is a table of existing connections:

NAME	KIND	Actions
dbx_samples	databricks	Edit Delete
pg_adventureworks	postgres	Edit Delete
pg_main	postgres	Edit Delete

The screenshot shows the VigilIQ control room interface in the 'observe' section. The dataset 'public.dimcustomer' is selected. The interface includes a 'Scan' button, 'Propose rules', and 'Run checks' buttons. Below, a bar chart displays quality metrics:

- Overall Quality: 96.0
- Completeness: 100
- Uniqueness: 100
- Validity: 88
- Consistency: -
- Timeliness: -
- Accuracy: -

A table lists rules for validity and uniqueness:

RULE	DIMENSION	WEIGHT	FAILED / TOTAL	PASS
<a href="#">title_allowed_values</a>	validity	x1	0 / 20,484	100.00%
<a href="#">datefirstpurchase_range</a>	validity	x1	0 / 20,484	100.00%
<a href="#">maritalstatus_allowed_values</a>	validity	x1	0 / 20,484	100.00%
<a href="#">totalchildren_range</a>	validity	x1	0 / 20,484	100.00%
<a href="#">customeralternatekey_unique</a>	uniqueness	x1	0 / 20,484	100.00%

connections – encrypted at rest, read-only credentials

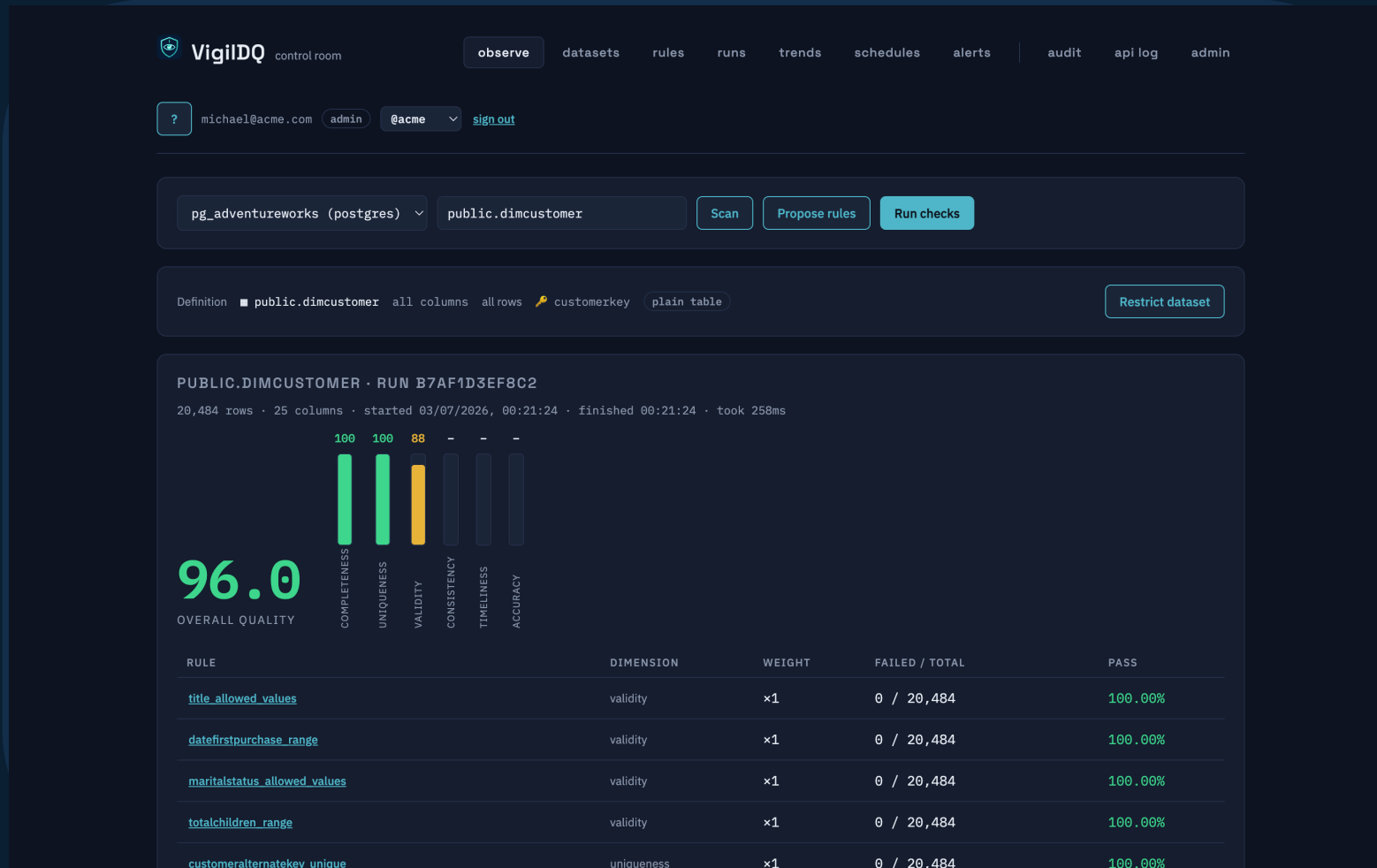
the dataset definition – one click to restrict columns/rows

- ▶ Secrets are Fernet-encrypted; rule execution needs read-only grants only.
- ▶ Logical datasets: restrict columns and rows (e.g. SCD2 current rows) — rules, profiling and drift all respect the restriction.

## WHY IT MATTERS

Your data never leaves the warehouse — VigilIQ pushes queries down and stores results, not rows.

# Score the dataset



- ▶ One 0–100 quality score, decomposed into six dimensions: completeness, uniqueness, validity, consistency, timeliness, accuracy.
- ▶ Every rule shows failed/total and its pass ratio — the score is explainable all the way down.
- ▶ An erroring rule scores zero and is surfaced — never silently skipped.

## WHY IT MATTERS

A number a steering committee can track, with the drill-down an auditor can defend.

# Review before anything runs

The screenshot shows the VigilIQ control room interface. At the top, there are navigation tabs: observe, datasets, rules (selected), runs, trends, schedules, alerts, audit, api log, and admin. Below the navigation, there is a user profile for michael@acme.com with an admin role and a sign out link. The main content area has a search bar with 'pg\_adventureworks (postgres)' and 'public.dimcustomer' selected, and buttons for 'Scan', 'Propose rules', and 'Run checks'. Below this is a '+ New rule' button with a tooltip: 'Write a rule by hand — it's vetted by the guardrails and enters review as a draft, just like a proposal.' The 'DRAFT RULES AWAITING REVIEW' section shows 'No drafts awaiting review' and a note: 'Run a scan and propose on the Observe tab to generate rules.' The 'ACTIVE RULES' section contains a table with the following data:

NAME	TYPE	WEIGHT	AUTHOR	
cust_gender_allowed	allowed_values	x1	human:michael@acme.com	<a href="#">Edit</a> <a href="#">Deactivate</a> <a href="#">Delete</a>
customeralternatekey_unique	unique	x1	agent:qwen3.6-27b-m1x	<a href="#">Edit</a> <a href="#">Deactivate</a> <a href="#">Delete</a>
customeralternatekey_not_null	not_null	x1	agent:qwen3.6-27b-m1x	<a href="#">Edit</a> <a href="#">Deactivate</a> <a href="#">Delete</a>
phone_pattern	pattern	x1	agent:qwen3.6-27b-m1x	<a href="#">Edit</a> <a href="#">Deactivate</a> <a href="#">Delete</a>
title allowed values	allowed_values	x1	agent:qwen3.6-27b-m1x	<a href="#">Edit</a> <a href="#">Deactivate</a> <a href="#">Delete</a>

- ▶ AI proposals land as drafts with a rationale grounded in the profile.
- ▶ Approve, reject, edit or delete — the agent cannot activate anything.
- ▶ The approving human is recorded on every rule (human:<email>).
- ▶ Hand-written rules go through exactly the same gates.

## WHY IT MATTERS

Change control is built in: no AI-authored check ever runs without a named human approving it.

# Every rule is inspectable SQL

ACTIVE RULES

NAME	TYPE	WEIGHT	AUTHOR	
cust_gender_allowed	allowed_values	x1	human:michael@acme.com	<a href="#">Close</a> <a href="#">Deactivate</a> <a href="#">Delete</a>

**Definition** [Raw SQL](#) [close](#)

columns:  row filter:

weight:

params (JSON): 

```
{
  "values": [
    "M",
    "F"
  ]
}
```

compiled SQL (current saved definition):  

```
SELECT COUNT(*) AS total, COALESCE(SUM(CASE WHEN "gender" IS NOT NULL AND "gender" NOT IN ('M', 'F') THEN 1 ELSE 0 END), 0) AS failed FROM "pub";
```

[Save → draft](#)

customeralternatekey_unique	unique	x1	agent:qwen3.6-27b-mlx	<a href="#">Edit</a> <a href="#">Deactivate</a> <a href="#">Delete</a>
customeralternatekey_not_null	not_null	x1	agent:qwen3.6-27b-mlx	<a href="#">Edit</a> <a href="#">Deactivate</a> <a href="#">Delete</a>
phone_pattern	pattern	x1	agent:qwen3.6-27b-mlx	<a href="#">Edit</a> <a href="#">Deactivate</a> <a href="#">Delete</a>
title_allowed_values	allowed_values	x1	agent:qwen3.6-27b-mlx	<a href="#">Edit</a> <a href="#">Deactivate</a> <a href="#">Delete</a>
maritalstatus_allowed_values	allowed_values	x1	agent:qwen3.6-27b-mlx	<a href="#">Edit</a> <a href="#">Deactivate</a> <a href="#">Delete</a>
gender_allowed_values	allowed_values	x1	agent:qwen3.6-27b-mlx	<a href="#">Edit</a> <a href="#">Deactivate</a> <a href="#">Delete</a>
yearlyincome_range	range	x1	agent:qwen3.6-27b-mlx	<a href="#">Edit</a> <a href="#">Deactivate</a> <a href="#">Delete</a>

- ▶ View the compiled SQL for any rule, per engine dialect.
- ▶ Edit the definition — or override with raw SQL for edge cases.
- ▶ Any logic change is re-vetted by the guardrails and reset to draft for re-approval.

## WHY IT MATTERS

No black boxes: your DBAs can read every control, and edits can't bypass review.

# Evidence stays in YOUR database

Failing-row evidence captured on the target database — VigiiDQ stores only this reference

[captured](#) `cust_gender_allowed` 25 rows → `dq_results.rule_failure_values` [hide rows](#)

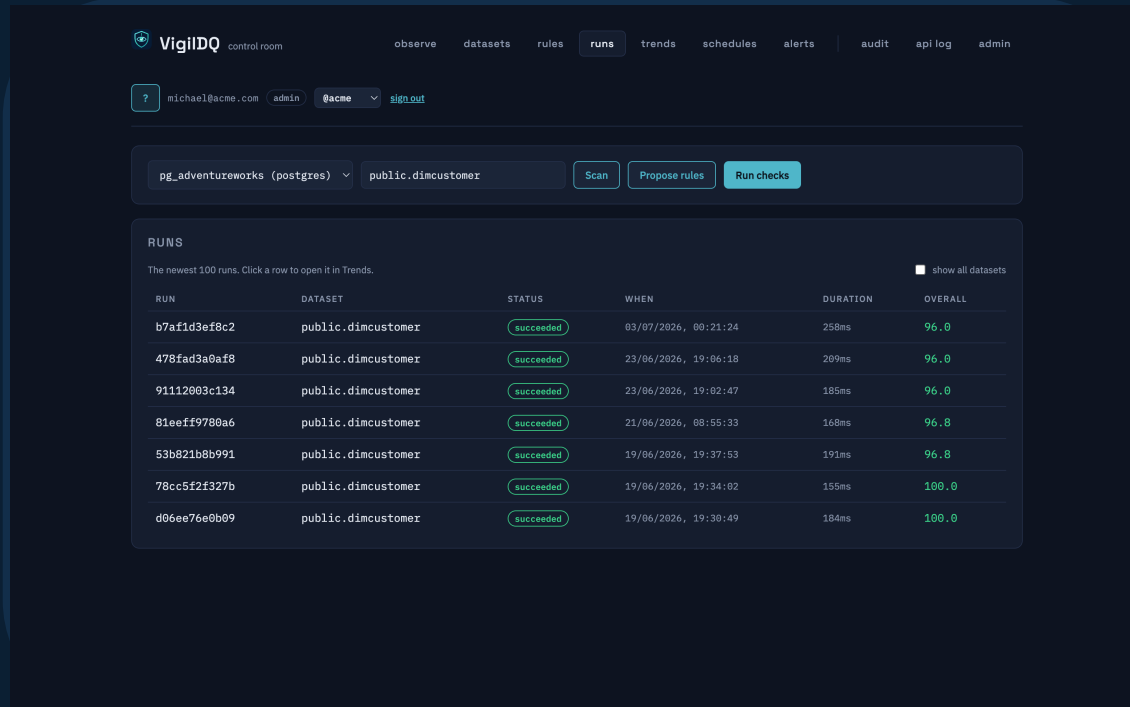
CUSTOMERKEY	GENDER	FIRSTNAME
900013	X	B***
900005	X	B***
900025	X	B***
900010	X	B***
900022	X	B***
900023	X	B***
900019	X	B***
900014	X	B***
900009	X	B***
900006	X	B***
900011	X	B***
900007	X	B***
900020	X	B***
900024	X	B***
900012	X	B***
900001	X	B***
900021	X	B***
900018	X	B***
900017	X	B***
900008	X	B***

- ▶ Failing rows are captured into a DQ schema on the target warehouse — keys plus the offending values.
- ▶ Capped, fail-soft, and written by a separate writer grant scoped to that schema only.
- ▶ The drill-down masks PII at the boundary (names show as initials).

## WHY IT MATTERS

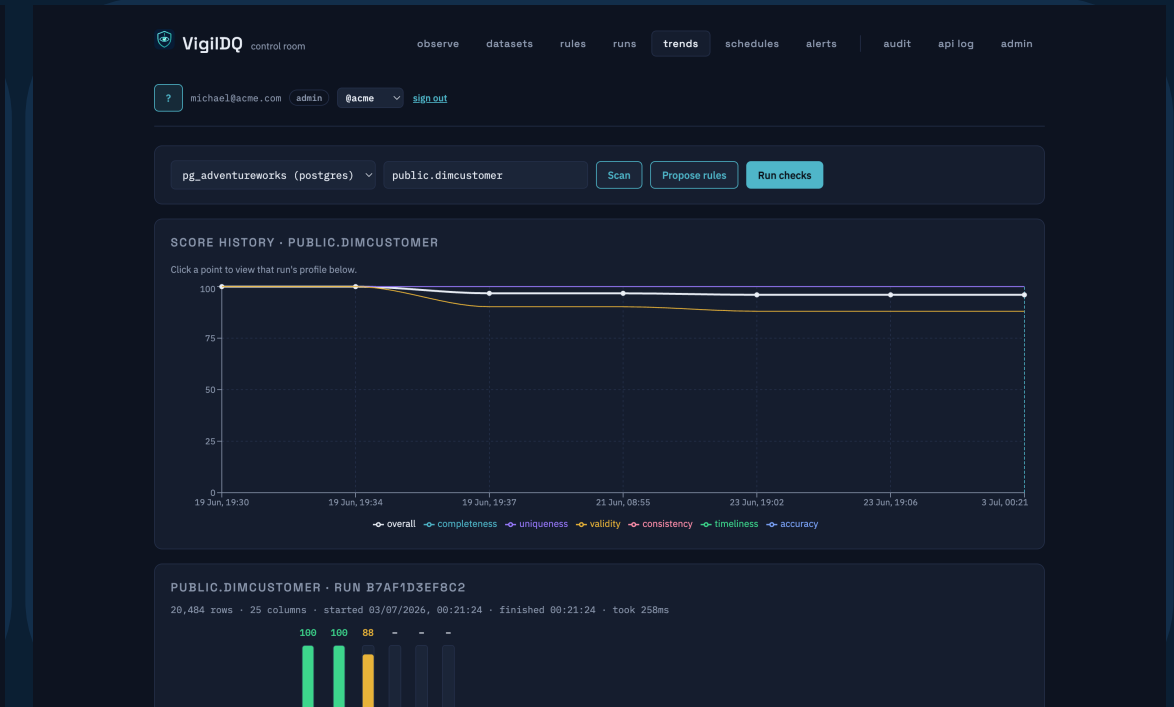
When an auditor asks "show me the failing rows", the answer lives in your own database — not in a vendor's cloud.

# Track every run, watch the trend



every run: status, duration, score

- ▶ Full run history with per-run scorecards and captured profiles.
- ▶ Drift rules (row count, null ratio, schema) judge against a median baseline — no day-one false alarms.



score history + anomaly baselines per dataset

## WHY IT MATTERS

Quality becomes a time series — regressions show up as a bend in the line, not a surprise in production.

# Automate: schedules, alerts, gates

structured cadences – interval / daily / weekly / monthly

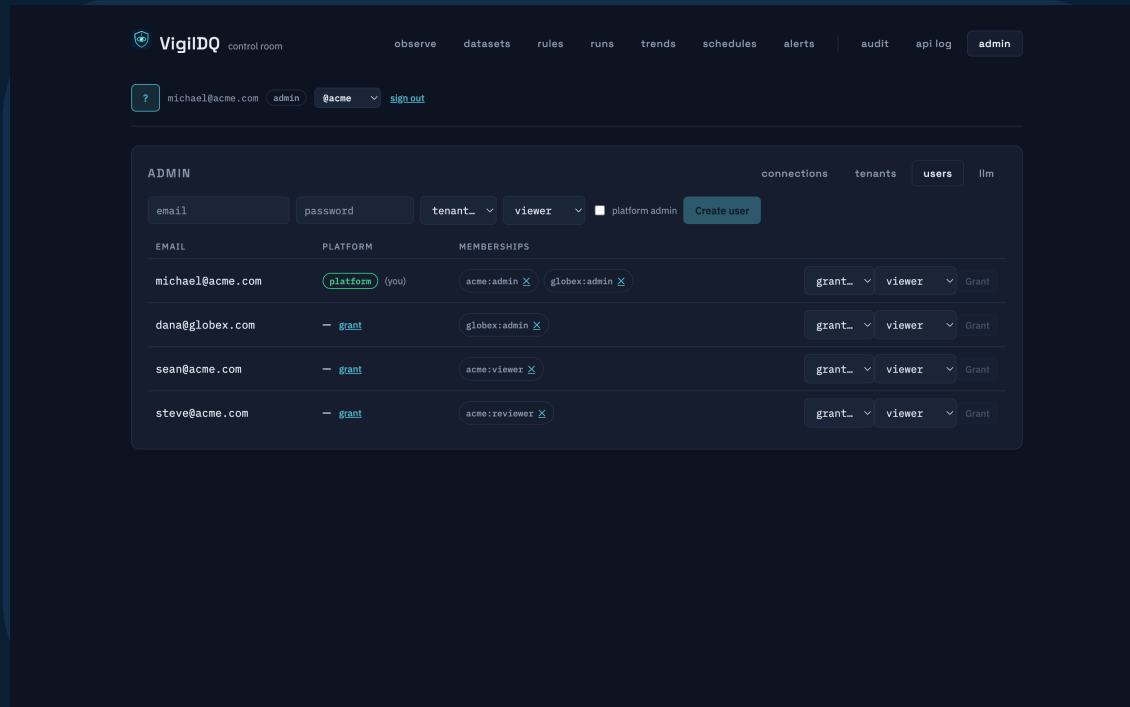
- ▶ Schedules run each dataset under its own tenant, with recorded health.
- ▶ Pipeline quarantine gates return pass/fail verdicts your orchestrator can branch on (Airflow, dbt, ADF...).

score alerts → email; webhooks for gate-fail / anomaly

## WHY IT MATTERS

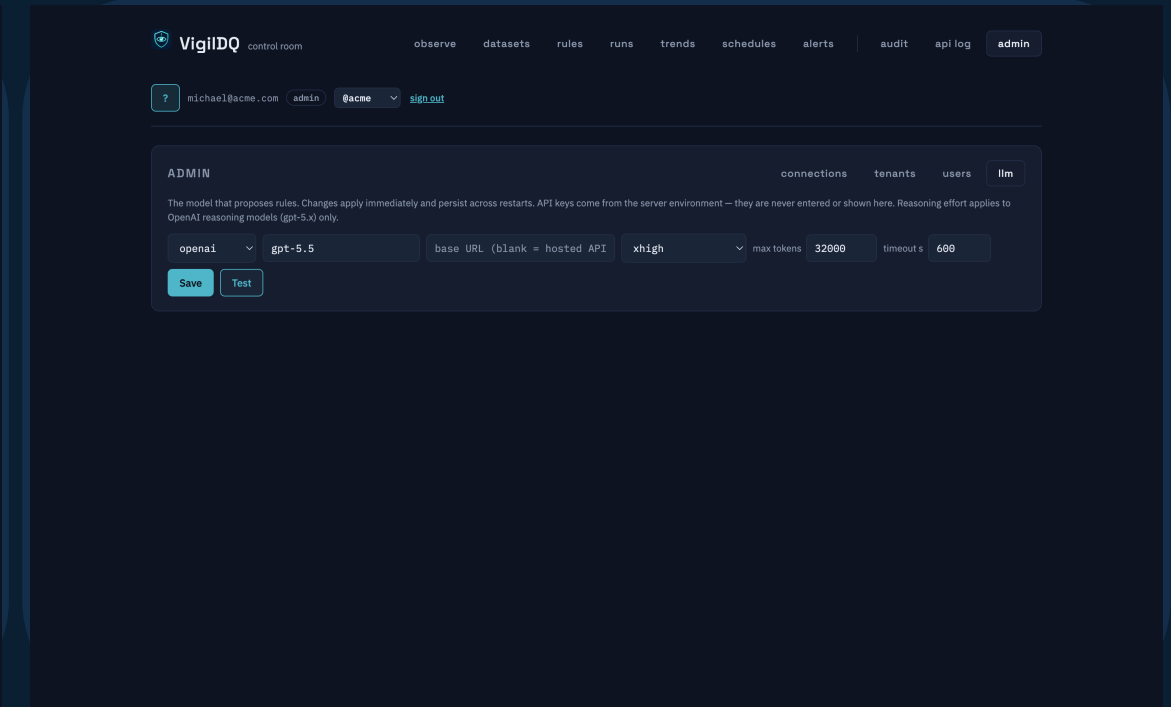
Set thresholds once; the platform watches every load and tells you — or stops the pipeline — when quality slips.

# Multi-tenant administration



tenants, users, memberships, roles

- ▶ Shared-schema multi-tenancy; roles per tenant (admin/reviewer/viewer).
- ▶ Choose the LLM in the UI: hosted (Anthropic/OpenAI incl. reasoning models) or fully LOCAL via any OpenAI-compatible server.



the AI model – switchable at runtime, audited

## WHY IT MATTERS

Bring your own model — including an on-prem LLM, so even the masked profile never leaves your estate.

# Audited end to end, scriptable end to end

The screenshot shows the VigilIQ control room interface. At the top, there are navigation tabs: observe, datasets, rules, runs, trends, schedules, alerts, audit (selected), api log, and admin. Below the navigation, there is a user profile for 'michael@acme.com' with an 'admin' role and a 'sign out' button. The main content area is titled 'AUDIT TRAIL' and has a dropdown menu set to 'all'. Below this is a table with the following columns: WHEN, ACTOR, ACTION, ENTITY, and DETAIL. The table contains 17 rows of login events, all performed by 'michael@acme.com' on '07/2026'.

WHEN	ACTOR	ACTION	ENTITY	DETAIL
08/07/2026, 01:08:29	michael@acme.com	login	user	michael@acme.com
07/07/2026, 20:19:01	michael@acme.com	login	user	michael@acme.com
07/07/2026, 20:14:02	michael@acme.com	login	user	michael@acme.com
07/07/2026, 20:11:53	michael@acme.com	login	user	michael@acme.com
03/07/2026, 14:30:46	michael@acme.com	login	user	michael@acme.com
03/07/2026, 14:28:39	michael@acme.com	login	user	michael@acme.com
03/07/2026, 13:40:53	michael@acme.com	login	user	michael@acme.com
03/07/2026, 13:34:02	michael@acme.com	login	user	michael@acme.com
03/07/2026, 13:32:42	michael@acme.com	login	user	michael@acme.com
03/07/2026, 13:00:31	michael@acme.com	login	user	michael@acme.com
03/07/2026, 10:25:05	michael@acme.com	login	user	michael@acme.com
03/07/2026, 10:10:08	michael@acme.com	login	user	michael@acme.com
03/07/2026, 09:58:09	michael@acme.com	login	user	michael@acme.com
03/07/2026, 09:48:47	michael@acme.com	login	user	michael@acme.com

The screenshot shows the VigilIQ REST API interface. At the top, it displays 'VigilIQ 0.15.0 OAS 3.1' and a 'Authorize' button. Below this is a list of API endpoints under the 'default' namespace. Each endpoint is shown with its method (GET, POST, DELETE), path, and description. The endpoints are: /health (Health), /auth/login (Login), /auth/me (Me), /admin/tenants (Admin List Tenants), /admin/tenants (Admin Create Tenant), /admin/users (Admin List Users), /admin/users (Admin Create User), /admin/users/{user\_id}/memberships (Admin Add Membership), /admin/users/{user\_id}/memberships/{tenant\_id} (Admin Remove Membership), and /admin/llm (Admin Get Llm).

every mutation: who, what, before/after

the full REST API – everything the UI does

- ▶ Approvals, edits, config changes — all appended to an immutable audit trail with before/after snapshots.
- ▶ The dashboard is a pure API client: anything you see here, your platform can automate.

## WHY IT MATTERS

Compliance gets its evidence; engineering gets an API. Nobody has to trade one for the other.

# Designed for regulated data

---

## Guardrails as architecture

- Schema, read-only and dry-run gates — hallucinations cannot pass
- Prompts are advisory; gates make failure modes impossible

## Fail closed

- A rule that cannot execute counts as a failed control
- A threshold with no rules is a breach, never a silent pass

## Provable masking

- PII masked inside scan() — no unmasked code path exists
- Tests capture the agent's exact prompt and assert zero raw values

## Complete audit trail

- Every control names the human who certified it
- The audit log has already been used to reconstruct a metastore

## THE NEXT STEP

---

# What a 30-minute demo adds

- ▶ The agent proposing rules live — and the review queue filling up.
- ▶ A guardrail refusing a bad rule live (a hallucinated column, a sneaky UPDATE).
- ▶ A quality gate wired into a pipeline: pass, fail, quarantine.
- ▶ All on our sandbox warehouse — nothing to install, no access to your systems.

## ■ Watch it catch bad data — live.

Everything in this pack is real and running today. A 30-minute demo adds the parts a PDF can't: watching the agent propose rules live, a guardrail refusing a bad rule, and a quality gate stopping a bad load.

Arrange a demo → [hello@vigildq.com](mailto:hello@vigildq.com) · [vigildq.com](https://vigildq.com)

The demo runs entirely on our sandbox warehouse — nothing to install, no access to your systems needed.