02 OCTOBER 8, 2026 6 PP

One key, every model: the LLM gateway behind our agents

Collin Rijock · Tim Child · Felipe Arce

Every app and every engineer at Exowatt calls models through one self-hosted gateway that we can meter, upgrade and route from a single config file.

SOFTWARE-TEAM AGENTS ROUTER

The problem

Until June, every internal agent at Exowatt that used AWS Bedrock signed in as the same IAM user. If one agent burned a few hundred dollars of Opus in an afternoon, we could not say which one without reading every agent's logs. Revoking one leaked key meant rotating the credential for every agent at once. Several apps that wanted a "summarize this row" button brought their own API key and had the same blind spots.

The model menu also kept moving. An app that pinned a Claude or GPT version ID kept that version until someone remembered to change it, and the old model usually cost more per token than its replacement.

We decided to put one gateway between every caller and every provider, run it on our own servers, and make it the only thing on the network that holds a model credential.

What we built

The gateway is an open-source LiteLLM proxy running on a small container in our Austin Proxmox cluster, behind an internal hostname. It holds the provider credentials and hands out its own keys instead.

Each app declares `llm:` in its deploy manifest and the app platform mints a key for it at first deploy, with a model allowlist and a monthly budget, then deletes the key when the app retires. Each engineer gets a personal key through phi, our coding-agent harness. Every request lands in Postgres with its key, model and cost, so "which app spent what" became a SQL query.

On top of LiteLLM we wrote our own model catalog, a fallback policy, and `auto`, a virtual model that sends each request to the cheapest model that can handle it.

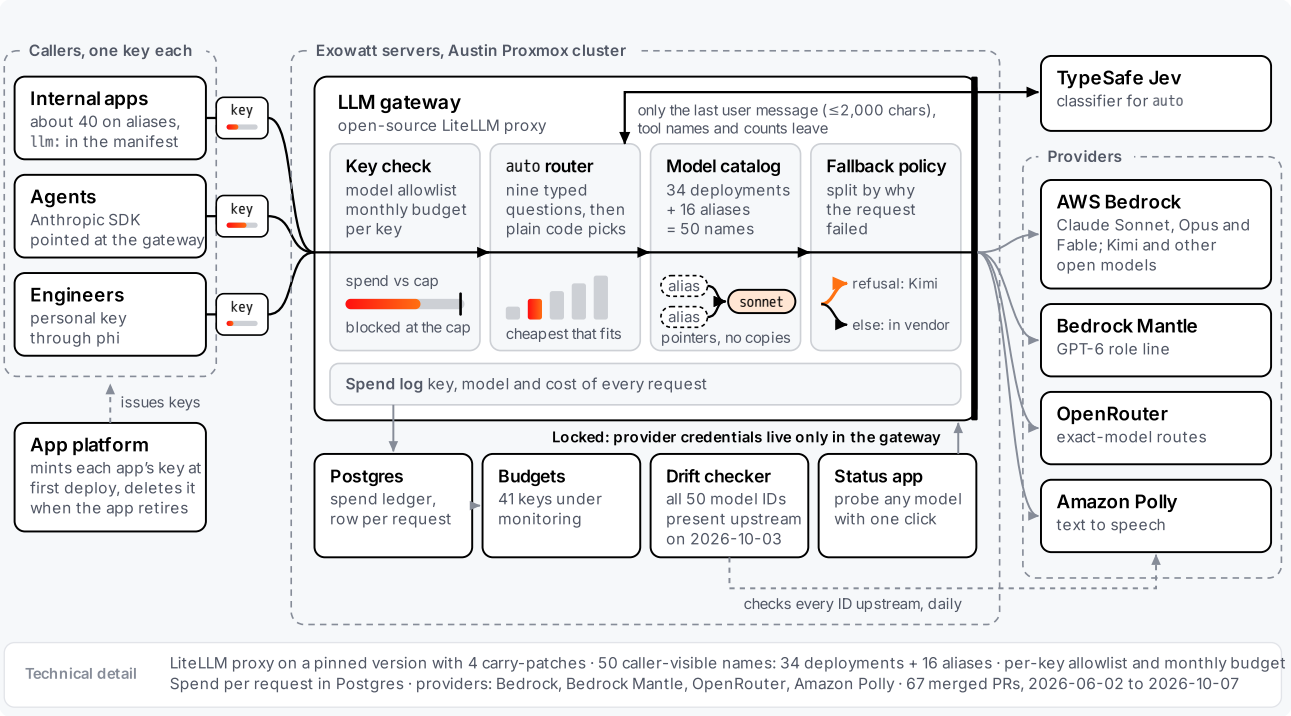


Figure 1. Every app, agent and engineer calls models with a key of its own. Inside the gateway each request passes a key check with a budget, the optional auto router, the model catalog and the fallback policy before it reaches a provider, and its cost lands in Postgres. The gateway is the only holder of provider credentials, and a drift checker and a status app watch the catalog from beside it.

How it works

The catalog has one rule: one deployment per upstream model, and every other spelling is a pointer to it. Apps call `sonnet` or `opus` or `terra` by default, and we treat a vendor ID pin as debt with an owner. When a better model ships in a family we change one line, and every app calling the bare name gets it on its next request with no redeploy. We used to keep full copies of each alias, and they drifted: `sonnet`

served Sonnet 4.6 for weeks while its own description said it tracked the latest. A pointer has no config of its own, so it cannot drift.

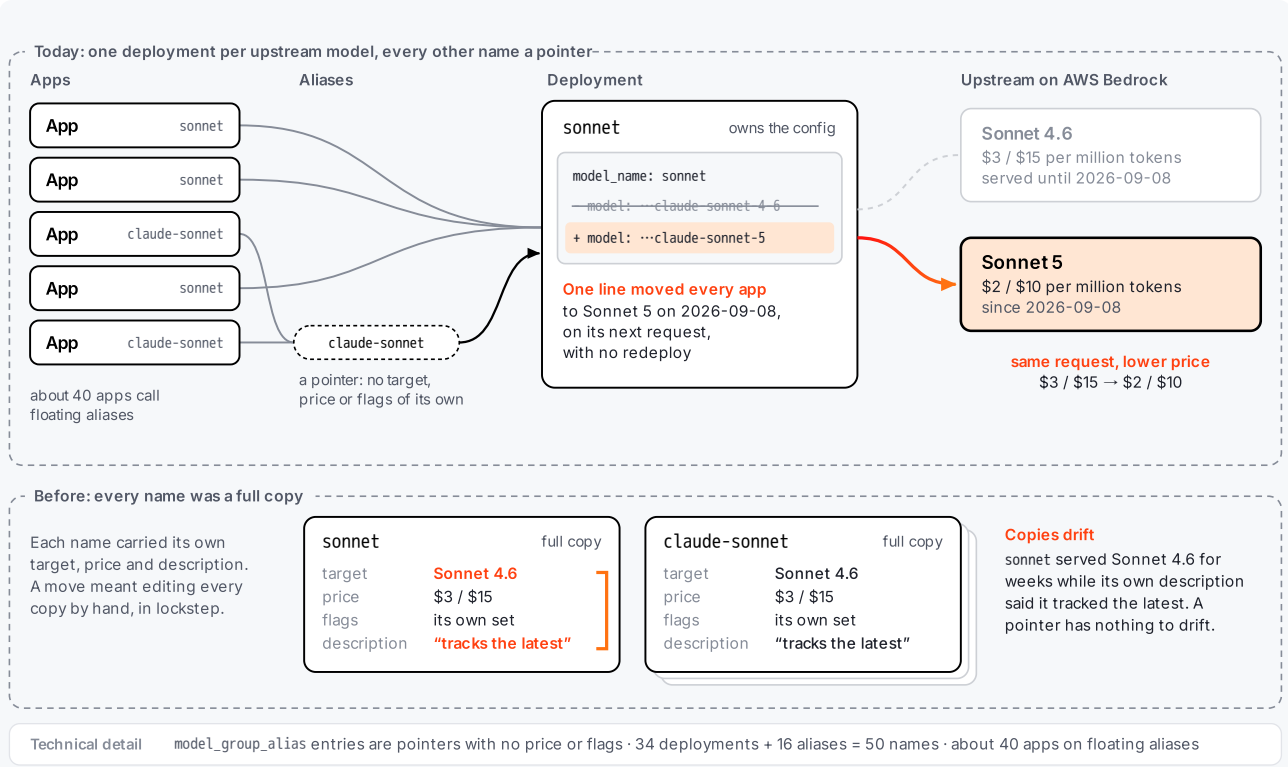


Figure 2. Apps call the family name sonnet, and every other spelling of it is a pointer. Moving sonnet from Sonnet 4.6 to Sonnet 5, as we did on 2026-09-08, is one line in its deployment: every app on the alias gets the newer model at $2/$10 instead of $3/$15 per million tokens on its next request, with no redeploy. Below is the old layout, where every name was a full copy and sonnet kept serving Sonnet 4.6 for weeks.

We refused to add `high` and `low` tier aliases. A tier can silently change what it points at, which makes last month's spend under that name meaningless. The client knows whether it is doing a mechanical pass or a high-stakes review, so tiers live there.

For about a week in August, a Claude request that Bedrock rejected for an ordinary reason could come back as a successful answer from Kimi, a different vendor. An engineer caught it because the response named the wrong model. We split the policy in two. Safety refusals may still hop vendors, since only that vendor's classifier objected. Every other fallback stays inside the vendor and fires only on timeouts, rate limits and server errors. Opus has no general fallback, so an Opus outage returns an error instead of an answer from Kimi. We carry four small patches on our pinned LiteLLM version, and two of them exist to enforce this policy.

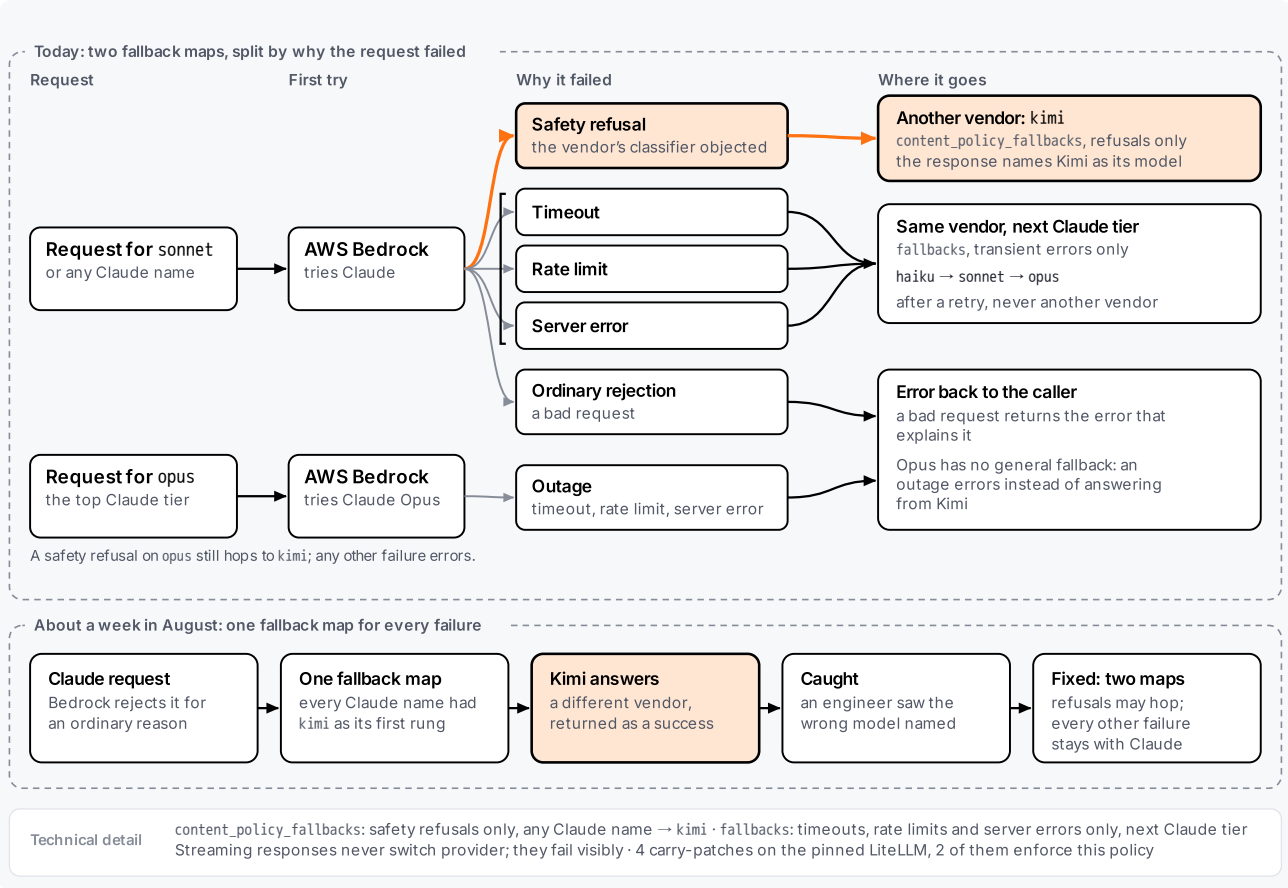


Figure 3. Where a failed Claude request goes depends on why it failed. A safety refusal may hop to Kimi, because only that vendor's classifier objected; timeouts, rate limits and server errors stay with Claude and move up a tier; an ordinary bad request returns its own error. Opus has nothing above it, so an Opus outage errors instead of answering from Kimi. Below is the single fallback map that let Kimi answer Claude requests for about a week in August.

`auto` asks TypeSafe Jev, a classifier service, nine typed questions about each request, covering things like difficulty and task type. Plain code maps those answers to a model, from the cheapest GPT-6 tier up to Fable. Only the last user message, capped at 2,000 characters, plus tool names and a few counts ever leave the gateway; system prompts, tool calls and earlier turns stay home. If the classifier is unsure or fails, the request goes to Sonnet, so routing never causes a user-visible error.

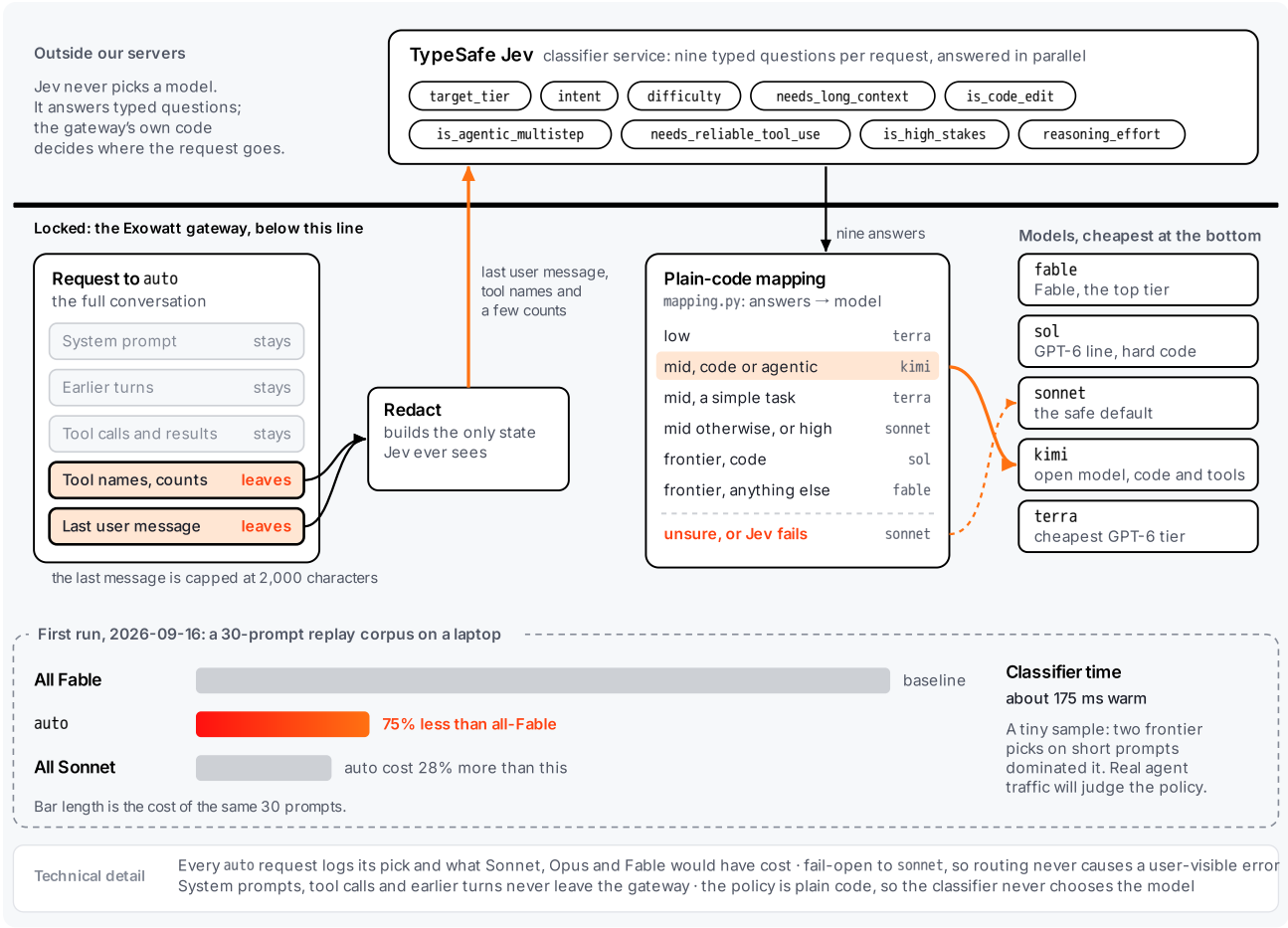


Figure 4. For each auto request, only the last user message, capped at 2,000 characters, plus tool names and a few counts go to TypeSafe Jev, which answers nine typed questions. Plain code turns the answers into a model, from the cheapest GPT-6 tier up to Fable, and anything unsure or failed goes to Sonnet. The bars show the first run on 2026-09-16, a 30-prompt corpus on a laptop, which is too small to judge the policy.

Results

Numbers are as of 2026-10-08, from the gateway's runbook, config and merged PRs.

Measure	Value	Basis
Merged PRs touching the gateway	67	2026-06-02 to 2026-10-07, five human authors, no bots
Caller-visible model names	50	34 deployments plus 16 aliases in `config.yaml`, across Bedrock, Bedrock Mantle, OpenRouter and Amazon Polly

Measure	Value	Basis
Internal apps on floating aliases	about 40	runbook count
Keys under budget monitoring	41	budget collector at deploy, 2026-08-25
Hourly request volume	278 to 2,914	one day sampled during an alert review, 2026-08-10
Sonnet peak throughput	about 1.3M tokens/min	across ten production apps, noted 2026-09-30

The family-alias rule paid off on 2026-09-08, when we moved `sonnet` from Sonnet 4.6 to Sonnet 5. Every app on that alias got the newer model and a price cut from $3/$15 to $2/$10 per million tokens on the same request.

The first `auto` run, on 2026-09-16, was a 30-prompt replay corpus on a laptop. It cost 75% less than sending everything to Fable and 28% more than sending everything to Sonnet, because two frontier picks on short prompts dominated a tiny sample. The classifier added about 175 ms warm. That proves the plumbing; real agent traffic will judge the policy. Every `auto` request now logs what it chose and what Sonnet, Opus and Fable would have cost.

On 2026-10-03 the governance work went live: a drift checker found all 50 configured model IDs present upstream, and a status app lets an operator probe any model with one click. Some identity evidence in that report is still marked unknown.

What’s next

Sonnet 5.5 sits on the gateway as an opt-in pin, and `sonnet` moves to it once Bedrock publishes quotas and early callers have tested its stricter request rules. The same one-key pattern is also spreading past models: Hypergoose gives each agent one key for internal APIs and MCP tools.

About Exowatt — Exowatt is a next-generation renewable energy company providing modular dispatchable energy solutions for energy-intensive applications. Inquiries: