Provider keys

The Provider Vault: managed keys, Azure OpenAI, Gemini.

Overview

Provider keys (Provider Vault)

Some tools can only carry an API key (Cursor, Copilot, Gemini CLI, Codex CLI). For these, the Gateway injects **your organization's** provider key server-side — developers never see or hold it.

Supported providers

- **OpenAI** and **Anthropic** — standard API keys.
- **Azure OpenAI** — key + your Azure resource endpoint; the Gateway routes to your own Azure deployment.
- **Google Gemini** — standard API key.

How keys are stored

Keys are encrypted at rest (AES-256-GCM) with strict database-role separation; they are decrypted only in memory, per request, and never logged or returned by any API. The console shows only the last 4 characters.

Model allow-lists

Each key can be restricted to specific models. With multiple keys for one provider, the requested model selects the matching key.

Connection modes

Per tool, you choose: **subscription** (the developer's own account passes through — Claude Code's default) or **managed key** (vault injection). Tools whose mechanism cannot carry a subscription credential are locked to managed mode. The Activity row records which mode actually served each request.

Add a provider key

Provider keys are your organization's own API keys — OpenAI, Anthropic, Azure OpenAI, Gemini. They let tools that can't use a personal subscription (Cursor's managed mode, the Copilot CLI) run on your account instead.

Providers ? Add Provider, choose the type, paste the key.

How the key is handled

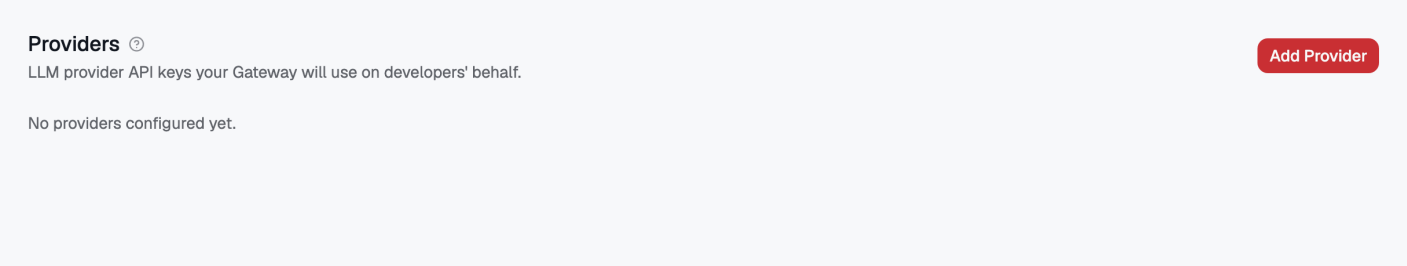
It is encrypted before it reaches storage and **never shown again** — the screen only displays the last four characters afterwards. If you lose the original, issue a new key at the provider and replace it here.

Azure OpenAI and regions

If the Azure region you enter differs from your Sentilai region, the screen warns you. It's a warning rather than a block: routing EU traffic through a US deployment may be exactly what you intended, but it should be a decision rather than an accident.

Removing a key

Remove stops any tool relying on it. Developers on subscription mode are unaffected — their own plan pays for their requests, which is the point of subscription-safe governance.



Providers — where a key is added. The list stays empty until you add one, and only the last four characters are ever shown again.

Which provider key you need

A provider key is what lets Sentilai make the actual AI request on your behalf. You only need one for tools running in **managed API key** mode — tools on a developer subscription do not touch it.

Matching keys to tools

- **Claude Code** in managed mode needs an **Anthropic** key.
- **Cursor** always needs an **OpenAI** key — it cannot run in subscription mode at all.
- **GitHub Copilot** through the custom-endpoint route needs an **OpenAI** key.

Supported types

OpenAI, **Anthropic**, **Azure OpenAI** and **Gemini**.

Azure OpenAI additionally needs your **Azure endpoint** and the **Azure region**. The region matters: if it is outside your organization's region, the console shows a warning badge on the key, because your AI traffic would be leaving the region you chose Sentilai to keep it in. It warns rather than blocks — sometimes that is a deliberate decision — but it should be a decision.

Enabled models

Each key carries a list of models it is allowed to serve. This is how you stop an expensive model from being used organization-wide, and it is also how Sentilai tells two keys of the same type apart.

If you have no key

Tools in subscription mode work fine. Tools in managed mode fail with a clear error saying no provider key is configured, and the Endpoint Suite warns developers before they connect a tool that would be affected.

Two keys of the same type

You can have several keys of the same provider type — a cheap one for general use, an expensive one for a specific team. Sentilai picks between them using the **enabled models** list.

The rule

Sentilai looks at the model the request asks for and finds the key whose enabled-models list contains it.

The failure

If a model appears in **more than one** key's list, the request is ambiguous and is refused with a conflict error rather than a guess. Guessing which of your keys to bill would be the wrong behaviour for something that costs money.

Fixing it

Make the lists disjoint. Each model should appear in exactly one key's enabled-models list.

If you genuinely want two teams on the same model with separate billing, that is not something the enabled-models mechanism can express today — tell us, because it is a reasonable thing to want.

Keys are write-only

After saving, a key is shown only as `••••` and its last four characters. Editing a key and leaving the field blank keeps the stored value; you never need to retrieve the original just to change the model list.

Removing a key stops every tool that depends on it, and the confirmation says so.